Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-06

What is This Course About? *What Not?*

- Calendar description:

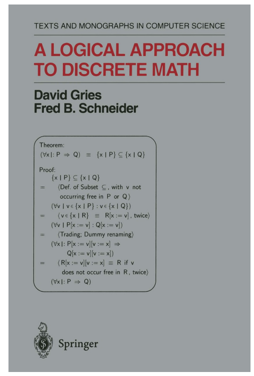
  Introduction to logic and proof techniques for practical reasoning: propositional logic, predicate logic, structural induction; rigorous proofs in discrete mathematics and programming.

- *Calculus is the mathematics of* ***continuous*** *phenomenaphysical sciences, traditional engineering — used for specifying bridges; used for justifying bridge designs.*
- **Discrete Mathematics** is
  - the math of data— **whether complex or big**
  - the math of reasoning— **logic**
  - the math of some kinds of AI— **machine reasoning**
  - the math of **specifying software**
- **Logical Reasoning** is
  - **used for justifying software designs**
  - **used for proving software implementations correct**

Goals and Rough Outline

- Understand the mechanics of mathematical expressions and proof — starting in a familiar area: **Reasoning about integers**
- Develop skill in **propositional calculus**
  - "**propositional**": statements that can be true or false, not numbers
  - "**calculus**": formalised reasoning, calculation — $\mathbb{B}, \neg, \wedge, \vee, \Rightarrow, \ldots$
- Develop skill in **predicate calculus**
  - "predicate": statement about some subjects. — $\forall, \exists$
- Develop skill in using **basic theories of "data mathematics"**
  - Sets, Functions, Relations
  - Sequences, Trees, Graphs
- *. . . skill development takes time and effort . . .*
- Introduction to **reasoning about (imperative) programs**
- Encounter mechanised discrete mathematics
- Introduction to mechanised software correctness tools — **Formal Methods**: increasingly important in industry

Textbook: "LADM"



"This is a rather extraordinary book, and deserves to be read by everyone involved in computer science and — perhaps more importantly — software engineering. I recommend it highly [. . .]. If the book is taken seriously, the rigor that it unfolds and the clarity of its concepts could have a significant impact on the way in which software is conceived and developed."

— Peter G. Neumann
(Founder of ACM SIGSOFT)

The Importance of Proof in CS

ACM's Computer Science Curricula recognize proofs as one of several areas of mathematics that are integral to a wide variety of sub-fields of computer science:

> *. . . an ability to create and understand a proof — either a formal symbolic proof or a less formal but still mathematically rigorous argument — is important in virtually every area of computer science, including (to name just a few) formal specification, verification, databases, and cryptography.*
>
> ACM/IEEE: Computer Science Curricula 2013, p. 79

"Mathematically rigorous" — "if I really needed to formalise it, I could."

- **Rigorous** (informal) proofs (e.g. in LADM) strive to "make the eventual formalisation effort minimal".
- There is value to **readable proofs**, no matter whether formal or informal.
- There is value to **formal, machine-checkable proofs**, especially in the software context, where the world of mathematics is not watching.

Strive for readable formal proofs!

COMPSCI 1DM3 Final 1(a)

Lemma "F1(a)": $(\neg q \wedge (p \Rightarrow q)) \Rightarrow \neg p$

Proof:

$(\neg q \wedge (p \Rightarrow q)) \Rightarrow \neg p$
$\equiv$ ⟨ "Material implication" ⟩
$(\neg q \wedge (\neg p \vee q)) \Rightarrow \neg p$
$\equiv$ ⟨ "Absorption" ⟩
$(\neg q \wedge \neg p) \Rightarrow \neg p$
$\equiv$ ⟨ "De Morgan" ⟩
$\neg (q \vee p) \Rightarrow \neg p$
$\equiv$ ⟨ "Contrapositive" ⟩
$p \Rightarrow q \vee p$
$\equiv$ ⟨ "Weakening" ⟩
true

Lemma "F1(a)": $(\neg q \wedge (p \Rightarrow q)) \Rightarrow \neg p$

Proof:

$(\neg q \wedge (p \Rightarrow q)) \Rightarrow \neg p$
$\equiv$ ⟨ "Material implication" ⟩
$\neg (\neg q \wedge (\neg p \vee q)) \vee \neg p$
$\equiv$ ⟨ "De Morgan" ⟩
$\neg\neg q \vee (\neg\neg p \wedge \neg q) \vee \neg p$
$\equiv$ ⟨ "Double negation" ⟩
$q \vee (p \wedge \neg q) \vee \neg p$
$\equiv$ ⟨ "Absorption" ⟩
$q \vee p \vee \neg p$
$\equiv$ ⟨ "Excluded middle" ⟩
$q \vee$ true
$\equiv$ ⟨ "Zero of $\vee$" ⟩
true

COMPSCI 1DM3 Final 1(b)

Lemma "F1(b)": $(\exists x \bullet P \Rightarrow Q) \equiv (\forall x \bullet P) \Rightarrow (\exists x \bullet Q)$

Proof:

$(\exists x \bullet P \Rightarrow Q)$
$\equiv$ ⟨ "Material implication" ⟩
$(\exists x \bullet \neg P \vee Q)$
$\equiv$ ⟨ "Distributivity of $\exists$ over $\vee$" ⟩
$(\exists x \bullet \neg P) \vee (\exists x \bullet Q)$
$\equiv$ ⟨ "Generalised De Morgan" ⟩
$\neg (\forall x \bullet P) \vee (\exists x \bullet Q)$
$\equiv$ ⟨ "Material implication" ⟩
$(\forall x \bullet P) \Rightarrow (\exists x \bullet Q)$

First Tool: CalcCheck

- CalcCheck: A proof checker for the textbook logic
- CalcCheck analyses textbook-style presentations of proofs
- $\text{CalcCheck}_{\text{Web}}$: A notebook-style web-app interface to CalcCheck
- **You can check your proofs before handing them in!**
- **Will be used in exams!**

  — initially with proof checking turned off. . .
  . . . but syntax checking left on
- **Will be used in exams**

  — as far as possible. . .

  You need to be able to do both:
  - Write formalisations and proofs using CalcCheck
  - Write formalisations and proofs **by hand on paper**

(Firefox and Chrome can be expected to work with $\text{CalcCheck}_{\text{Web}}$. Safari, Edge, IE not necessarily.)

From the LADM Instructor's Manual

Emphasis on skill acquisition:

- "a course taught from this text will give students a solid understanding of what constitutes a proof and a skill in developing, presenting, and reading proof."
- "We believe that teaching a skill in formal manipulation makes learning the other material easier."
- "Logic as a tool is so important to later work in computer science and mathematics that students must understand the use of logic and be sure in that understanding."
- "One benefit of our new approach to teaching logic, we believe is that students become more effective in communicating and thinking in other scientific and engineering disciplines."
- "Frequent but shorter homeworks ensure that students get practice"

Consciously departing from existing mechanised logics:

- "Our equational logic is a "People Logic", instead of a "Machine Logic"."
  - CalcCheck mechanises this "People Logic"

CalcCheck: A Recognisable Version of the Textbook Proof Language

(11.5) $S = \{x \mid x \in S : x\}$.
According to axiom Extensionality (11.4), it suffices to prove that $v \in S \equiv v \in \{x \mid x \in S : x\}$, for arbitrary v. We have,

$v \in \{x \mid x \in S : x\}$
$=$ ⟨ Definition of membership (11.3) ⟩
$(\exists x \mid x \in S : v = x)$
$=$ ⟨ Trading (9.19), twice ⟩
$(\exists x \mid x = v : x \in S)$
$=$ ⟨ One-point rule (8.14) ⟩
$v \in S$

```
Theorem (11.5):  S = { x ❙ x ∈ S • x }
Proof:
  Using “Set extensionality” (11.4):
    For any `v`:
        v ∈ { x ❙ x ∈ S • x }
      =   ⟨ “Set membership” (11.3) ⟩
        (∃ x ❙ x ∈ S • v = x)
      =   ⟨ “Trading for ∃” (9.19) ⟩
        (∃ x ❙ x = v • x ∈ S)
      =   ⟨ “One-point rule for ∃” (8.14), substitution ⟩
        v ∈ S
```

Note:

1. The calculation part is transliterated into Unicode plain text (only minimal notation changes).
2. The prose top-level of the proof is formalised into Using and For any structures in the spirit of LADM

From the LADM Instructor's Manual: "Some Hints on Mechanics"

- "We have been successful (in a class of 70 students) with occasionally writing a few problems on the board and walking around the class as the students work on them."
  - COMPSCI&SFWRENG 2DM3: ≈240 students in 2016, 360 in 2020
  - COMPSCI 2LC3: Over 180 students in 2021; over 200 in 2023
  - Tutorials normally have 20–40 students and use this approach, with students working on their computers
    — this still worked with online course delivery
- "Frequent short homework assignments are much more effective than longer but less frequent ones. Handing out a short problem set that is due the next lecture forces the students to practice the material immediately, instead of waiting a week or two."
  - Since 2018, giving homework up to twice per week
  - Only feasible due to online submission and autograding
  - **Clear improvement in course results**

From the LADM Instructor's Manual: "Some Hints on Mechanics" (ctd.)

- "There is no substitute for practice accompanied by ample and timely feedback"
  - Most "timely feedback" is provided by interaction with $\text{CalcCheck}_{\text{Web}}$
  - Autograding for homework and assignments produces some additional feedback
  - CalcCheck is intentionally a proof checker, not a proof assistant
  - Providing ample TA office hours (and now a "Course Help" channel) helps students overcome roadblocks.
- "We tell the students that they are all capable of mastering the material (for they are)."
  - . . . and CalcCheck homework makes more of them actually master the material.

Organisation

- Schedule
- Grading
- Exams
- Avenue
- Course Page: `http://www.cas.mcmaster.ca/~kahl/CS2LC3/2023/`

— check in case of Avenue and MSTeams outage!

— See the Outline (on course page and on Avenue)

— Read the Outline!

Schedule

	Mon	Tue	Wed	Thu	Fri
8:30–10:20	T3	T5	T1		
10:30–11:20					T2
11:30–12:20	Lecture		Lecture		T2
13:30–14:20	Office hour				Lecture
14:30–16:20					
16:30–18:20				T4	

- **Lectures: attend!, take notes!**
- **2-hour Tutorials** (starting **Thursday, September 7**):
  – Discuss student approaches to "Exercise" questions.
- **TA office hours**: TBA
- **Studying and Homework:** About 2–3 hours per lecture
  — **reading the textbook , writing proofs in $\text{CalcCheck}_{\text{Web}}$**

Grading

- **Homework**, from one lecture to the next — in total: **10%**
  - The weakest 2 or 3 homeworks are dropped (see outline)
  - MSAFs for homework are not processed
- **Roughly-weekly assignments** — in total: **16%**
  - The weakest 1 or 2 assignments are dropped (see outline)
  - MSAFs for assignments are not processed
- **2 Midterm Tests**, closed book, **on $\text{CalcCheck}_{\text{Web}}$ / on paper, each:**
  - **15%** if not better than your final
  - **20%** if better than your final
    — in total at least: **30%**
    — in total up to: **40%**
  - Deferred midterms may be oral
- **Final** (closed book, 2.5 hours, **on $\text{CalcCheck}_{\text{Web}}$ / . . .**) **34%–44%**
  = **100%**
- Possible **bonus assignments** and other **bonus marks**
  — only count if you passed the course

Exams

- Exercise questions, assignment questions, and the questions on midterm tests, and on the final —
  — will be somewhat similar. . .
- All tests and exams are **closed-book**.
  – The main difference to open-book lies in how you prepare. . .
  – **Knowledge is important:**
  Without the right knowledge, you would not even know what to look up where!
- **You need to be able and prepared to do both:**
  - Write formalisations and proofs using CalcCheck
  - Write formalisations and proofs by hand on paper
- **Know your stuff!**
  — . . . and not only in the exams . . .
  — . . . and not only for this term . . .
  — . . . similar to learning a new language

The Language of Logical Reasoning

The mathematical foundations of Computing Science involve **language skills and knowledge**:

- **Vocabulary:** Commonly known concepts and technical terms
- **Syntax/Grammar:** How to produce complex statements and arguments
- **Semantics:** How to relate complex statements with their meaning
- **Pragmatics:** How people actually use the features of the language

> Conscious and fluent use of the
> **language of logical reasoning**
> is the foundation for
> **precise specification and rigorous argumentation**
> in **Computer Science and Software Engineering**.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-06

Part 2: Expressions and Calculations

The Answer

H1 Starting Point

```
  7 · 8
= ⟨ Fact `8 = 7 + 1` ⟩
  7 · (7 + 1)
= ⟨ Fact `7 = 10 − 3` ⟩
  (10 − 3) · (7 + 1)
= ⟨ "Distributivity of · over +" ⟩
  (10 − 3) · 7 + (10 − 3) · 1
= ⟨ "Distributivity of · over −" ⟩
  10 · 7 − 3 · 7 + 10 · 1 − 3 · 1
= ⟨ "Identity of ·" — twice ⟩
  10 · 7 − 3 · 7 +  10  −  3
= ⟨ Fact `3 · 7 = 21` ⟩
  10 · 7 −  21  +  10  −  3
= ⟨ Fact `10 · 7 = 70` ⟩
  70   −  21  +  10  −  3
= ⟨ Fact `10 − 3 = 7` ⟩
  70   −  21  +     7
= ⟨ Fact `21 + 7 = 28` ⟩
  70   −     28
= ⟨ Fact `70 − 28 = 42` ⟩
  42
```

Calculational Proof Format

$$
\begin{array}{ll}
 & E_0 \\
= & \langle \text{ Explanation of why } E_0 = E_1 \ \rangle \\
 & E_1 \\
= & \langle \text{ Explanation of why } E_1 = E_2 \ \rangle \\
 & E_2 \\
= & \langle \text{ Explanation of why } E_2 = E_3 \ \rangle \\
 & E_3
\end{array}
$$

This is a proof for:

$$E_0 = E_3$$

Calculational Proof Format

E_0
= ⟨ Explanation of why $E_0 = E_1$ ⟩
E_1
= ⟨ Explanation of why $E_1 = E_2$ ⟩
E_2
= ⟨ Explanation of why $E_2 = E_3$ ⟩
E_3

The calculational presentation as such is conjunctional: This reads as:

$$E_0 = E_1 \quad \wedge \quad E_1 = E_2 \quad \wedge \quad E_2 = E_3$$

Because = is **transitive**, this justifies:

$$E_0 = E_3$$

Syntax of Conventional Mathematical Expressions

LADM 1.1, p. 7

- A **constant** (e.g., 231) or **variable** (e.g., x) is an expression
- If E is an expression, then (E) is an expression
- If $\circ$ is a **unary prefix operator** and E is an expression, then $\circ E$ is an expression, with operand E.

  For example, the negation symbol $-$ is used as a unary prefix operator, so -5 is an expression.
- If $\otimes$ is a **binary infix operator** and D and E are expressions, then $D \otimes E$ is an expression, with operands D and E.

  For example, the symbols + and · are binary infix operators, so $1 + 2$ and $(-5) \cdot (3 + x)$ are expressions.

Syntax of Conventional Mathematical Expressions

- A **constant** (e.g., 231) or **variable** (e.g., x) is an expression
- If E is an expression, then (E) is an expression
- If $\circ$ is a **unary prefix operator** and E is an expression, then $\circ E$ is an expression, with operand E.
- If $\otimes$ is a **binary infix operator** and D and E are expressions, then $D \otimes E$ is an expression, with operands D and E.

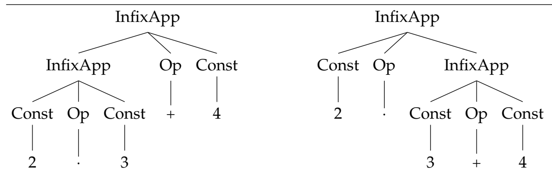
The intention of this is that each expression is **at least one** of the following alternatives:

- **either some constant**
- **or some variable**
- **or some simpler expression** in parentheses
- **or** the application of **some unary prefix operator** to **some simpler expression**
- **or** the application of **some binary infix operator** to **two simpler expressions**

Why is this an expression?

$2 \cdot 3 + 4$

- If $\otimes$ is a **binary infix operator** and D and E are expressions, then $D \otimes E$ is an expression, with operands D and E.
- **or** the application of **some binary infix operator** to **two simpler expressions**



Which expression is it? Why?

⟹ The multiplication operator · has higher **precedence** than the addition operator +.

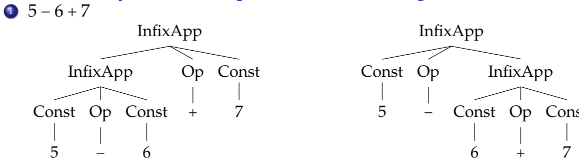
Table of Precedences

- $[x := e]$ (textual substitution) **(highest precedence)**
- . (function application)
- unary prefix operators $+, -, \neg, \#, \sim, \mathcal{P}$
- $**$
- $\cdot \quad / \quad \div \quad$ mod $\quad$ gcd
- $+ \quad - \quad \cup \quad \cap \quad \times \quad \circ \quad \bullet$
- $\downarrow \quad \uparrow$
- $\#$
- $\triangleleft \quad \triangleright \quad \hat{}$
- $= \quad < \quad > \quad \in \quad \subset \quad \subseteq \quad \supset \quad \supseteq \quad |$ (conjunctional)
- $\vee \quad \wedge$
- $\Rightarrow \quad \Leftarrow$
- $\equiv$ **(lowest precedence)**

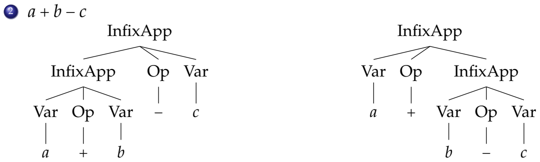
All non-associative binary infix operators associate to the left, except $**$, $\triangleleft$, $\Rightarrow$, $\rightarrow$, which associate to the right.

Why are these expressions? Which expressions are these?

1. $5 - 6 + 7$



2. $a + b - c$



The operators + and − **associate to the left**, also mutually.

Associativity versus Association

- If we write $a + b + c$, there appears to be no need to discuss whether we mean $(a + b) + c$ or $a + (b + c)$, because they evaluate to the same values:

  $(a + b) + c = a + (b + c)$ **"+" is associative**
- If we write $a - b - c$, we mean $(a - b) - c$:

  "−" associates to the left $\quad 9 - (5 - 2) \neq (9 - 5) - 2$
- If we write a^{b^c}, we mean $a^{(b^c)}$:

  exponentiation associates to the right $\quad 2^{(3^2)} \neq (2^3)^2$
- If we write $a ** b ** c$, we mean $a ** (b ** c)$:

  "$**$" associates to the right
- If we write $a \Rightarrow b \Rightarrow c$, we mean $a \Rightarrow (b \Rightarrow c)$:

  "⇒" associates to the right $\quad \mathsf{F} \Rightarrow (\mathsf{T} \Rightarrow \mathsf{F}) \neq (\mathsf{F} \Rightarrow \mathsf{T}) \Rightarrow \mathsf{F}$

An Equational Theory of Integers — Axioms (LADM Ch. 15)

(15.1)	**Axiom, Associativity:**	$(a + b) + c = a + (b + c)$ $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
(15.2)	**Axiom, Symmetry:**	$a + b = b + a$ $a \cdot b = b \cdot a$
(15.3)	**Axiom, Additive identity:**	$0 + a = a$ $a + 0 = a$
(15.4)	**Axiom, Multiplicative identity:**	$1 \cdot a = a$ $a \cdot 1 = a$
(15.5)	**Axiom, Distributivity:**	$a \cdot (b + c) = a \cdot b + a \cdot c$ $(b + c) \cdot a = b \cdot a + c \cdot a$
(15.13)	**Axiom, Unary minus:**	$a + (-a) = 0$
(15.14)	**Axiom, Subtraction:**	$a - b = a + (-b)$

An Equational Theory of Integers — Axioms (CalcCheck)

Declaration: $\mathbb{Z}$: Type
Declaration: _+_ : $\mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$
Declaration: _·_ : $\mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$
Axiom (15.1) (15.1*a*) "Associativity of +": $(a + b) + c = a + (b + c)$
Axiom (15.1) (15.1*b*) "Associativity of ·": $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
Axiom (15.2) (15.2*a*) "Symmetry of +": $a + b = b + a$
Axiom (15.2) (15.2*b*) "Symmetry of ·": $a \cdot b = b \cdot a$
Axiom (15.3) "Additive identity" "Identity of +": $0 + a = a$
Axiom (15.4) "Multiplicative identity" "Identity of ·": $1 \cdot a = a$
Axiom (15.5) "Distributivity of · over +": $a \cdot (b + c) = a \cdot b + a \cdot c$
Axiom (15.9) "Zero of ·": $a \cdot 0 = 0$
Declaration: −_ : $\mathbb{Z} \to \mathbb{Z}$
Declaration: _−_ : $\mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$
Axiom (15.13) "Unary minus": $a + (-a) = 0$
Axiom (15.14) "Subtraction": $a - b = a + (-b)$

Calculational Proofs of Theorems — (15.17) $-(-a) = a$

(15.3) **Identity of +** $\quad 0 + a = a$	(15.13) **Unary minus** $\quad a + (-a) = 0$

LADM:

Theorem (15.17): $-(-a) = a$

Proof:

$-(-a)$
= ⟨ Identity of + (15.3) ⟩
$0 + -(-a)$
= ⟨ Unary minus (15.13) ⟩
$a + (-a) + -(-a)$
= ⟨ Unary minus (15.13) ⟩
$a + 0$
= ⟨ Identity of + (15.3) ⟩
a

CalcCheck:

Theorem (15.17) "Self-inverse of unary minus": $-(-a) = a$

Proof:

$-(-a)$
= ⟨ "Identity of +" ⟩
$0 + -(-a)$
= ⟨ "Unary minus" ⟩
$a + (-a) + -(-a)$
= ⟨ "Unary minus" ⟩
$a + 0$
= ⟨ "Identity of +" ⟩
a

H1 Starting Point

The Answer

$7 \cdot 8$
$= \langle$ Fact `8 = 7 + 1` $\rangle$
$7 \cdot (7 + 1)$
$= \langle$ Fact `7 = 10 − 3` $\rangle$
$(10 - 3) \cdot (7 + 1)$
$= \langle$ "Distributivity of · over +" $\rangle$
$(10 - 3) \cdot 7 + (10 - 3) \cdot 1$
$= \langle$ "Distributivity of · over −" $\rangle$
$10 \cdot 7 - 3 \cdot 7 + 10 \cdot 1 - 3 \cdot 1$
$= \langle$ "Identity of ·" — twice $\rangle$
$10 \cdot 7 - 3 \cdot 7 + 10 - 3$
$= \langle$ Fact `3 · 7 = 21` $\rangle$
$10 \cdot 7 - 21 + 10 - 3$
$= \langle$ Fact `10 · 7 = 70` $\rangle$
$70 - 21 + 10 - 3$
$= \langle$ Fact `10 − 3 = 7` $\rangle$
$70 - 21 + 7$
$= \langle$ Fact `21 + 7 = 28` $\rangle$
$70 - 28$
$= \langle$ Fact `70 − 28 = 42` $\rangle$
42

- Work through Homework 1
- Submit by 12:30 on Friday, Sept. 8
- **Tutorials start tomorrow, Thursday, Sept. 7!**
- If you are in the Thursday tutorial, work through H1 before that!
- Get started working on Exercises 1.*
- Go to your tutorial to continue working on Ex1 — bring your laptop!

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-08

Expressions and Substitution

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

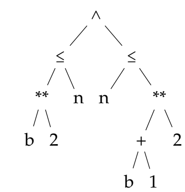
2023-09-08

Part 1: Syntax of Mathematical Expressions (ctd.)

Term Tree Presentation of Mathematical Expression

$b^2 \le n \le (b+1)^2$

$b^2 \le n \quad \wedge \quad n \le (b+1)^2$



We write strings, but we think trees.

All the rules we have for implicit parentheses only serve to encode the tree structure.

Recall: Syntax of Conventional Mathematical Expressions

Textbook 1.1, p. 7

- A **constant** (e.g., 231) or **variable** (e.g., x) is an expression
- If E is an expression, then (E) is an expression
- If $\circ$ is a **unary prefix operator** and E is an expression, then $\circ E$ is an expression, with operand E.

  For example, the negation symbol − is used as a unary prefix operator, so -5 is an expression.
- If $\otimes$ is a **binary infix operator** and D and E are expressions, then $D \otimes E$ is an expression, with operands D and E.

  For example, the symbols + and · are binary infix operators, so $1 + 2$ and $(-5) \cdot (3 + x)$ are expressions.

Recall: Syntax of Conventional Mathematical Expressions

- A **constant** (e.g., 231) or **variable** (e.g., x) is an expression
- If E is an expression, then (E) is an expression
- If $\circ$ is a **unary prefix operator** and E is an expression, then $\circ E$ is an expression, with operand E.
- If $\otimes$ is a **binary infix operator** and D and E are expressions, then $D \otimes E$ is an expression, with operands D and E.

The intention of this is that each expression is **at least one** of the following alternatives:

- **either** **some constant**
- **or** **some variable**
- **or** **some simpler expression** in parentheses
- **or** the application of **some unary prefix operator** to **some simpler expression**
- **or** the application of **some binary infix operator** to **two simpler expressions**

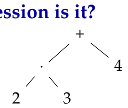
Why is this an expression?

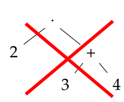
$2 \cdot 3 + 4$

- If $\otimes$ is a **binary infix operator** and D and E are expressions, then $D \otimes E$ is an expression, with operands D and E.

- **or** the application of **some binary infix operator** to **two simpler expressions**

Which expression is it?





Why?

⟹ The multiplication operator · has **higher precedence** than the addition operator +.

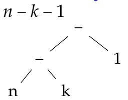
Table of Precedences

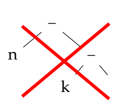
- $[x := e]$ (textual substitution) **(highest precedence)**
- . (function application)
- unary prefix operators +, −, ¬, #, ~, $\mathcal{P}$
- **
- · / ÷ mod gcd
- + − ∪ ∩ × ∘ •
- ↓ ↑
- #
- ◁ ▷ ^
- = < > ∈ ⊂ ⊆ ⊃ ⊇ | (conjunctional)
- ∨ ∧
- ⇒ ⇐
- ≡ **(lowest precedence)**

All non-associative binary infix operators associate to the left, except **, ◁, ⇒, →, which associate to the right.

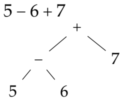
Why are these expressions? Which expressions are these?

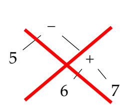
1. $n - k - 1$



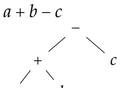


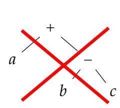
2. $5 - 6 + 7$





3. $a + b - c$





The operators + and − **associate to the left**, also mutually.

Precedences and Association — *We write strings, but we think trees*

All the rules we have for implicit parentheses only serve to encode the tree structure.

(We use underscores to denote operator argument positions.
So _⊗_ is a binary infix operator, and ⊟_ is a unary prefix operator.)

_⊗_ has higher precedence than _⊙_	means	$a \otimes b \odot c = (a \otimes b) \odot c$ $a \odot b \otimes c = a \odot (b \otimes c)$
_⊗_ has higher precedence than ⊟_	means	$\boxminus a \otimes b = \boxminus (a \otimes b)$
⊟_ has higher precedence than _⊗_	means	$\boxminus a \otimes b = (\boxminus a) \otimes b$
_⊗_ associates to the left	means	$a \otimes b \otimes c = (a \otimes b) \otimes c$
_⊗_ associates to the right	means	$a \otimes b \otimes c = a \otimes (b \otimes c)$
_⊗_ mutually associates to the left with (same prec.) _⊙_	means	$a \otimes b \odot c = (a \otimes b) \odot c$
_⊗_ mutually associates to the right with (same prec.) _⊙_	means	$a \otimes b \odot c = a \otimes (b \odot c)$

Associativity versus Association

- If we write $a + b + c$, there is no need to discuss whether we mean $(a + b) + c$ or $a + (b + c)$, because they are the same:

  $(a + b) + c = a + (b + c)$ **"+" is associative**
- If we write $a - b - c$, we mean $(a - b) - c$:

  "−" associates to the left $9 - (5 - 2) \neq (9 - 5) - 2$
- If we write a^{b^c}, we mean $a^{(b^c)}$:

  exponentiation associates to the right $2^{(3^2)} \neq (2^3)^2$
- If we write $a ** b ** c$, we mean $a ** (b ** c)$:

  "" associates to the right**
- If we write $a \Rightarrow b \Rightarrow c$, we mean $a \Rightarrow (b \Rightarrow c)$:

  "⇒" associates to the right $\mathsf{F} \Rightarrow (\mathsf{T} \Rightarrow \mathsf{F}) \neq (\mathsf{F} \Rightarrow \mathsf{T}) \Rightarrow \mathsf{F}$

Conjunctional Operators

Chains can involve different conjunctional operators:

$1 < i \leq j < 5 = k$

$\equiv$ ⟨ "Reflexivity of =" `$x = x$` — conjunctional operators ⟩

$1 < i \quad \wedge \quad i \leq j \quad \wedge \quad j < 5 \quad \wedge \quad 5 = k$

$\equiv$ ⟨ "Reflexivity of =" — $\wedge$ has lower precedence ⟩

$(1 < i) \quad \wedge \quad (i \leq j) \quad \wedge \quad (j < 5) \quad \wedge \quad (5 = k)$

$x < 5 \in S \subseteq T$

$\equiv$ ⟨ "Reflexivity of =" — conjunctional operators ⟩

$x < 5 \quad \wedge \quad 5 \in S \quad \wedge \quad S \subseteq T$

$\equiv$ ⟨ "Reflexivity of =" — $\wedge$ has lower precedence ⟩

$(x < 5) \quad \wedge \quad (5 \in S) \quad \wedge \quad (S \subseteq T)$

Remember this!!!

Mathematical Expressions, Terms, Formulae ...

"Expression" is not the only word used for this kind of concept.

Related terminology:

- Both "term" and "expression" are frequently used names for the same kind of concept.
- The textbook's "expression" subsumes both "term" and "formula" of conventional first-order predicate logic.

Remember:

- Expressions are **understood** as tree-structures

  — *"abstract syntax"*
- Expressions are **written** as strings

  — *"concrete syntax"*
- Parentheses, precedences, and association rules **only serve to disambiguate the encoding of trees in strings.**

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-08

Part 2: Substitution

Plan for Part 2

- **Substitution as such:** Replaces variables with expressions in expressions, e.g.,

  $(x + 2 \cdot y)[x, y := 3 \cdot a, b + 5]$

  $=$ ⟨ Substitution ⟩

  $3 \cdot a + 2 \cdot (b + 5)$

- **Applying substitution instances of theorems** and making the substitution explicit:

  $2 \cdot y \ + \ -(2 \cdot y)$

  $=$ ⟨ "Unary minus" `$a + -a = 0$` with `$a := 2 \cdot y$` ⟩

  0

Textual Substitution

Let E and R be expressions and let x be a variable. We write:

$E[x := R]$ or E^x_R

to denote an expression that is the same as E but with all occurrences of x replaced by (R).

Example 1:

$(x + y)[x := z + 2]$

$=$ ⟨ Substitution — performing substitution ⟩

$((z + 2) + y)$

$=$ ⟨ "Reflexivity of =" — removing unnecessary parentheses ⟩

$z + 2 + y$

Textual Substitution

Let E and R be expressions and let x be a variable. We write:

$E[x := R]$

to denote an expression that is the same as E but with all occurrences of x replaced by (R).

Example 2:

$(x \cdot y)[x := z + 2]$

$=$ ⟨ Substitution ⟩

$((z + 2) \cdot y)$

$=$ ⟨ "Reflexivity of =" — removing unnecessary parentheses ⟩

$(z + 2) \cdot y$

Textual Substitution

Let E and R be expressions and let x be a variable. We write:

$E[x := R]$

to denote an expression that is the same as E but with all occurrences of x replaced by (R).

Example 3:

$(0 + a)[a := -(-a)]$

$=$ ⟨ Substitution ⟩

$(0 + (-(-a)))$

$=$ ⟨ "Reflexivity of =" — removing (some) unnecessary parenth. ⟩

$0 \ + \ -(-a)$

Textual Substitution

Let E and R be expressions and let x be a variable. We write:

$E[x := R]$

to denote an expression that is the same as E but with all occurrences of x replaced by (R).

Example 4:

$x + y[x := z + 2]$

$=$ ⟨ "Reflexivity of =" — adding parentheses for clarity ⟩

$x + (y[x := z + 2])$

$=$ ⟨ Substitution ⟩

$x + (y)$

$=$ ⟨ "Reflexivity of =" — removing unnecessary parentheses ⟩

$x + y$

Note: Substitution $[x := R]$ is a **highest precedence** postfix operator

Textual Substitution

Let E and R be expressions and let x be a variable. We write:

$E[x := R]$ or E^x_R

to denote an expression that is the same as E but with all occurrences of x replaced by (R).

Examples:

Expression	Result	Unnecessary parentheses removed
$x[x := z + 2]$	$(z + 2)$	$z + 2$
$(x + y)[x := z + 2]$	$((z + 2) + y)$	$z + 2 + y$
$(x \cdot y)[x := z + 2]$	$((z + 2) \cdot y)$	$(z + 2) \cdot y$
$x + y[x := z + 2]$	$x + y$	$x + y$

Note: Substitution $[x := R]$ is a **highest precedence** postfix operator

Sequential Substitution

$(x + y)[x := y - 3][y := z + 2]$

$=$ ⟨ "Reflexivity of =" — adding parentheses for clarity ⟩

$((x + y)[x := y - 3])[y := z + 2]$

$=$ ⟨ Substitution — performing inner substitution ⟩

$(((y - 3) + y))[y := z + 2]$

$=$ ⟨ Substitution — performing outer substitution ⟩

$((((z + 2) - 3) + (z + 2)))$

$=$ ⟨ "Reflexivity of =" — removing unnecessary parentheses ⟩

$z + 2 - 3 + z + 2$

On CalcCheck$_{\text{Web}}$: **Exercise 2.2: *Substitutions***

Simultaneous Textual Substitution

If R is a **list** $R_1, \ldots, R_n$ of expressions
and x is a **list** $x_1, \ldots, x_n$ of **distinct variables**, we write:

$$E[x := R]$$

to denote the **simultaneous** replacement of the variables of x
by the corresponding expressions of R,
each expression being enclosed in parentheses.

Example:

$(x + y)[x, y := y - 3, z + 2]$

$=$ ⟨ Substitution — performing substitution ⟩

$((y - 3) + (z + 2))$

$=$ ⟨ "Reflexivity of =" — removing unnecessary parentheses ⟩

$y - 3 + z + 2$

Simultaneous Textual Substitution

If R is a **list** $R_1, \ldots, R_n$ of expressions
and x is a **list** $x_1, \ldots, x_n$ of **distinct** variables, we write:

$$E[x := R]$$

to denote the **simultaneous** replacement of the variables of x
by the corresponding expressions of R,
each expression being enclosed in parentheses.

Examples:

Expression	Result	Unnecessary parentheses removed
$x[x, y := y - 3, z + 2]$	$(y - 3)$	$y - 3$
$(y + x)[x, y := y - 3, z + 2]$	$((z + 2) + (y - 3))$	$z + 2 + y - 3$
$(x + y)[x, y := y - 3, z + 2]$	$((y - 3) + (z + 2))$	$y - 3 + z + 2$
$x + y[x, y := y - 3, z + 2]$	$x + (z + 2)$	$x + z + 2$

Simultaneous Substitution:

$(x + y)[x, y := y - 3, z + 2]$

$=$ ⟨ Substitution — performing substitution ⟩

$((y - 3) + (z + 2))$

$=$ ⟨ "Reflexivity of =" — removing unnecessary parentheses ⟩

$y - 3 + z + 2$

Sequential Substitution:

$(x + y)[x := y - 3][y := z + 2]$

$=$ ⟨ "Reflexivity of =" — adding parentheses for clarity ⟩

$((x + y)[x := y - 3])[y := z + 2]$

$=$ ⟨ Substitution — performing inner substitution ⟩

$(((y - 3) + y))[y := z + 2]$

$=$ ⟨ Substitution — performing outer substitution ⟩

$((((z + 2) - 3) + (z + 2)))$

$=$ ⟨ "Reflexivity of =" — removing unnecessary parentheses ⟩

$z + 2 - 3 + z + 2$

Recall: An Equational Theory of Integers — Axioms (LADM Ch. 15)

(15.1) **Axiom, Associativity:** $(a + b) + c = a + (b + c)$
$(a \cdot b) \cdot c = a \cdot (b \cdot c)$

(15.2) **Axiom, Symmetry:** $a + b = b + a$
$a \cdot b = b \cdot a$

(15.3) **Axiom, Additive identity:** $0 + a = a$
$a + 0 = a$

(15.4) **Axiom, Multiplicative identity:** $1 \cdot a = a$
$a \cdot 1 = a$

(15.5) **Axiom, Distributivity:** $a \cdot (b + c) = a \cdot b + a \cdot c$
$(b + c) \cdot a = b \cdot a + c \cdot a$

(15.13) **Axiom, Unary minus:** $a + (-a) = 0$

(15.14) **Axiom, Subtraction:** $a - b = a + (-b)$

Calculational Proofs of Theorems — (15.17) $-(-a) = a$

(15.3) **Identity of +** $0 + a = a$	(15.13) **Unary minus** $a + (-a) = 0$

Theorem (15.17) "Self-inverse of unary minus": $-(-a) = a$
Proof:

$-(-a)$

$=$ ⟨ Identity of + (15.3) ⟩

$0 + -(-a)$

$=$ ⟨ Unary minus (15.13) ⟩

$a + (-a) + -(-a)$

$=$ ⟨ Unary minus (15.13) ⟩

$a + 0$

$=$ ⟨ Identity of + (15.3) ⟩

a

Three different variables named "a"!

Calculational Proofs of Theorems — (15.17) — Renamed Theorem Variables

(15.3x) **Identity of +** $0 + x = x$	(15.13y) **Unary minus** $y + (-y) = 0$

Theorem (15.17) "Self-inverse of unary minus": $-(-a) = a$
Proof:

$-(-a)$

$=$ ⟨ Identity of + (15.3x) ⟩

$0 + -(-a)$

$=$ ⟨ Unary minus (15.13y) ⟩

$a + (-a) + -(-a)$

$=$ ⟨ Unary minus (15.13y) ⟩

$a + 0$

$=$ ⟨ Identity of + (15.3x) ⟩

a

Three different variables "x", "y", "a".

Details of Applying Theorems — (15.17) with Explicit Substitutions I

(15.3x) **Identity of +** $0 + x = x$	(15.13y) **Unary minus** $y + (-y) = 0$

Theorem (15.17) "Self-inverse of unary minus": $-(-a) = a$
Proof:

$-(-a)$

$=$ ⟨ Identity of + (15.3x) with $x := -(-a)$ ⟩ $\quad$ $(0 + x = x)[x := -(-a)] \quad = \quad (0 + -(-a) = -(-a))$

$0 + -(-a)$

$=$ ⟨ Unary minus (15.13y) with $y := a$ ⟩ $\quad$ $(y + (-y) = 0)[y := a] \quad = \quad (a + (-a) = 0)$

$a + (-a) + -(-a)$

$=$ ⟨ Unary minus (15.13y) with $y := -a$ ⟩ $\quad$ $(y + (-y) = 0)[y := -a] \quad = \quad (-a + (-(-a)) = 0)$

$a + 0$

$=$ ⟨ Identity of + (15.3x) with $x := a$ ⟩ $\quad$ $(0 + x = x)[x := a)] \quad = \quad (0 + a = a$

a

Details of Applying Theorems — (15.17) with Explicit Substitutions II

(15.3) **Identity of +** $0 + a = a$	(15.13) **Unary minus** $a + (-a) = 0$

Theorem (15.17) "Self-inverse of unary minus": $-(-a) = a$
Proof:

$-(-a)$

$=$ ⟨ Identity of + (15.3) with $a := -(-a)$ ⟩

$0 + -(-a)$

$=$ ⟨ Unary minus (15.13) with $a := a$ ⟩

$a + (-a) + -(-a)$

$=$ ⟨ Unary minus (15.13) with $a := -a$ ⟩

$a + 0$

$=$ ⟨ Identity of + (15.3) with $a := a$ ⟩

a

Three different variables named "a"!

Specifying Substitutions for Theorem Application in CalcCheck

Theorem (15.19) "Distributivity of unary minus over +": $-(a + b) = (-a) + (-b)$
Proof:

$-(a + b)$

$=$ ⟨ (15.20) with `a := a + b` ⟩

$(-1) \cdot (a + b)$

$=$ ⟨ "Distributivity of · over +" with `a, b, c := - 1, a, b` ⟩

$(-1) \cdot a + (-1) \cdot b$

$=$ ⟨ (15.20) with `a := b` ⟩

$(-1) \cdot a + -b$

$=$ ⟨ (15.20) with `a := a` ⟩

$(-a) + (-b)$

Theorem (15.20):
$-a = (-1) \cdot a$

- Backquotes enclose math embedded in English. (Markdown convention)
- Substitution notation as in LADM: *variables* := *expressions*
- ":=" reads "becomes" or "is/are replaced with"
- ":=" is entered by typing "\:=" or "\becomes"!
- The variable list has the same length as the expression list.
- No variable occurs twice in the variable list.
- CalcCheck$_{\text{Web}}$ notebooks "with rigid matching" **require** all theorem variables to be substituted.
"Rigid matching" means: The theorems you specify need to match without substitution.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-11

Part 1: Foundations of Applying Equations in Context

Plan for Today

- **Anatomy of calculation** based on **Substitution** (LADM 1.3–1.5):
  - **Inference rule Substitution:** Justifies applying instances of theorems:

    $2 \cdot y + -(2 \cdot y)$
    $=$ ⟨ "Unary minus" $a + -a = 0$ with '$a := 2 \cdot y$' ⟩
    0

  - **Inference rule Leibniz:** Justifies applying (instances of) **equational** theorems deeper inside expressions:

    $2 \cdot x + 3 \cdot (y - 5 \cdot (4 \cdot x + 7))$
    $=$ ⟨ "Subtraction" $a - b = a + -b$ with '$a, b := y, 5 \cdot (4 \cdot x + 7)$' ⟩
    $2 \cdot x + 3 \cdot (y + -(5 \cdot (4 \cdot x + 7)))$

- LADM Chapter 2: Boolean Expressions
  - Meaning of Boolean Operators
  - Equality versus Equivalence
  - Satisfiability and Validity
- Starting with LADM Chapter 3: Propositional Calculus
  - Equivalence, Negation, Inequivalence

What is an Inference Rule?

$$\frac{\text{premise}_1 \quad \ldots \quad \text{premise}_n}{\text{conclusion}}$$

- **If all the premises are theorems, then the conclusion is a theorem.**
- A theorem is a "proved truth"
  — either an axiom,
  — or the result of an inference rule application.
- *Inference rules are the building blocks of proofs.*
- The premises are also called hypotheses.
- The conclusion and each premise all have to be Boolean.
- **Axioms** are inference rules with zero premises

Inference Rule: Substitution

(1.1) **Substitution:** $\dfrac{E}{E[x := R]}$ "If E is a theorem, then $E[x := R]$ is a theorem as well"

Example:

If $a + 0 = a$ is a theorem, — "Identity of +"

then $3 \cdot b + 0 = 3 \cdot b$ is also a theorem. — "Identity of +" with '$a := 3 \cdot b$'

$$\frac{a + 0 = a}{(a + 0 = a)[a := 3 \cdot b]} \qquad \frac{a + 0 = a}{3 \cdot b + 0 = 3 \cdot b}$$

Inference Rule Scheme: Substitution

(1.1) **Substitution:** $\dfrac{E}{E[x := R]}$ "If E is a theorem, then $E[x := R]$ is a theorem as well"

Really an **inference rule scheme**:
works for **every combination** of

- expression E,
- variable x, and
- expression R.

Example:

If $a + 0 = a$ is a theorem,
then $3 \cdot b + 0 = 3 \cdot b$ is also a theorem.

$$\frac{a + 0 = a}{3 \cdot b + 0 = 3 \cdot b}$$

- expression E is $a + 0 = a$
- the variable x substituted into is a
- the substituted expression R is $3 \cdot b$

Inference Rule Scheme: Substitution — Also for Simultaneous Substitution

(1.1) **Substitution:** $\dfrac{E}{E[x := R]}$

Really an **inference rule scheme**:
works for **every combination** of

- expression E,
- variable **list** x, and
- **corresponding** expression **list** R.

Example:

If $x + y = y + x$ is a theorem,
then $b + 3 = 3 + b$ is also a theorem.

- expression E is $x + y = y + x$
- variable list x is x, y
- corresponding expression list R is $b, 3$

Logical Definition of Equality

Two **axioms** (i.e., postulated as theorems):

- (1.2) **Reflexivity of =:** $x = x$
- (1.3) **Symmetry of =:** $(x = y) = (y = x)$

Two **inference rule schemes**:

- (1.4) **Transitivity of =:** $\dfrac{X = Y \qquad Y = Z}{X = Z}$
- (1.5) **Leibniz:** $\dfrac{X = Y}{E[z := X] = E[z := Y]}$

— the rule of "replacing equals for equals"

Using Leibniz' Rule in (15.21)

Given: (15.20) $-a = (-1) \cdot a$

$$\frac{X = Y}{E[z := X] = E[z := Y]}$$

Proving (15.21) $(-a) \cdot b = a \cdot (-b)$**:**

$(-a) \cdot b$
$=$ ⟨ (15.20) — **via Leibniz (1.5) with E chosen as** $z \cdot b$ ⟩
$((-1) \cdot a) \cdot b$
$=$ ⟨ Associativity (15.1) and Symmetry (15.2) of $\cdot$ ⟩
$a \cdot ((-1) \cdot b)$
$=$ ⟨ (15.20) ⟩
$a \cdot (-b)$

Using Leibniz together with Substitution in (15.21)

Given: (15.20) $-a = (-1) \cdot a$

$$\frac{X = Y}{E[z := X] = E[z := Y]}$$

Proving (15.21) $(-a) \cdot b = a \cdot (-b)$**:**

$(-a) \cdot b$
$=$ ⟨ (15.20) — via Leibniz (1.5) with E chosen as $z \cdot b$ ⟩
$((-1) \cdot a) \cdot b$
$=$ ⟨ Associativity (15.1) and Symmetry (15.2) of $\cdot$ ⟩
$a \cdot ((-1) \cdot b)$
$=$ ⟨ (15.20) with $a := b$ — via Leibniz (1.5) with E chosen as $a \cdot z$ ⟩
$a \cdot (-b)$

Combining Leibniz' Rule with Substitution

(1.5) **Leibniz:** $\dfrac{X = Y}{E[z := X] = E[z := Y]}$ (15.20) $-a = (-1) \cdot a$

(1.1) **Substitution:** $\dfrac{F}{F[v := R]}$

Using Leibniz:	**Using them together:**	Example:
$E[z := X]$	$E[z := X[v := R]]$	$a \cdot ((-1) \cdot b)$
$=$ ⟨ $X = Y$ ⟩	$=$ ⟨ $X = Y$ ⟩	$=$ ⟨ (15.20) with $a := b$ — E is $a \cdot z$ ⟩
$E[z := Y]$	$E[z := Y[v := R]]$	$a \cdot (-b)$

Justification:

$$\dfrac{\dfrac{X = Y}{X[v := R] = Y[v := R]}\ \text{Substitution (1.1)}}{E[z := X[v := R]] = E[z := Y[v := R]]}\ \text{Leibniz (1.5)}$$

Automatic Application of Associativity and Symmetry Laws

Axiom (15.1) (15.1*a*) "Associativity of +": $(a + b) + c \;=\; a + (b + c)$

Axiom (15.1) (15.1*b*) "Associativity of ·": $(a \cdot b) \cdot c \;=\; a \cdot (b \cdot c)$

Axiom (15.2) (15.2*a*) "Symmetry of +": $a + b \;=\; b + a$

Axiom (15.2) (15.2*b*) "Symmetry of ·": $a \cdot b \;=\; b \cdot a$

- You have been trained to reason "up to symmetry and associativity"
- Making symmetry and associativity steps explicit is
  - **always allowed**
  - sometimes **very useful for readability**
- CALCCHECK allows selective activation of symmetry and associativity laws

$\Longrightarrow$ "Exercise … / Assignment …: […] **without automatic associativity and symmetry**"

$\Longrightarrow$ Having to make symmetry and associativity steps explicit can be tedious…

(15.17) with Explicit Associativity and Symmetry Steps

(15.3) **Identity of +** $0 + a = a$	(15.13) **Unary minus** $a + (-a) = 0$

Proving (15.17) $-(-a) = a$:

$$
\begin{array}{ll}
 & -(-a) \\
= & \langle \text{Identity of + (15.3)} \rangle \\
 & 0 + -(-a) \\
= & \langle \text{Unary minus (15.13)} \rangle \\
 & (a + (-a)) + -(-a) \\
= & \langle \text{Associativity of + (15.1)} \rangle \\
 & a + ((-a) + -(-a)) \\
= & \langle \text{Unary minus (15.13)} \rangle \\
 & a + 0 \\
= & \langle \text{Symmetry of + (15.2)} \rangle \\
 & 0 + a \\
= & \langle \text{Identity of + (15.3)} \rangle \\
 & a
\end{array}
$$

Some Property Names

Let $\odot$ and $\oplus$ be binary operators and $\square$ be a constant.

*($\odot$ and $\oplus$ and $\square$ are **metavariables** for operators respectively constants.)*

- "**$\odot$ is symmetric**": $x \odot y \;=\; y \odot x$
- "**$\odot$ is associative**": $(x \odot y) \odot z \;=\; x \odot (y \odot z)$
- "**$\odot$ is mutually associative with $\oplus$** (from the left)": $(x \odot y) \oplus z \;=\; x \odot (y \oplus z)$

For example:

- $+$ **is** mutually associative with $-$: $(x + y) - z \;=\; x + (y - z)$
- $-$ **is not** mutually associative with $+$: $(5 - 2) + 3 \neq 5 - (2 + 3)$

Some Property Names (ctd.)

Let $\odot$ and $\oplus$ be binary operators and $\square$ be a constant.

*($\odot$ and $\oplus$ and $\square$ are **metavariables** for operators respectively constants.)*

- "**$\odot$ is idempotent**": $x \odot x \;=\; x$
- "**$\square$ is a left-identity (or left-unit) of $\odot$**": $\square \odot x \;=\; x$
- "**$\square$ is a right-identity (or right-unit) of $\odot$**": $x \odot \square \;=\; x$
- "**$\square$ is identity (or unit) of $\odot$**": $\square \odot x \;=\; x \;=\; x \odot \square$
- "**$\square$ is a left-zero of $\odot$**": $\square \odot x \;=\; \square$
- "**$\square$ is a right-zero of $\odot$**": $x \odot \square \;=\; \square$
- "**$\square$ is a zero of $\odot$**": $\square \odot x \;=\; \square \;=\; x \odot \square$
- "**$\odot$ distributes over $\oplus$ from the left**": $x \odot (y \oplus z) \;=\; (x \odot y) \oplus (x \odot z)$
- "**$\odot$ distributes over $\oplus$ from the right**": $(y \oplus z) \odot x \;=\; (y \odot x) \oplus (z \odot x)$
- "**$\odot$ distributes over $\oplus$**": $\odot$ distributes over $\oplus$ from the left **and** $\odot$ distributes over $\oplus$ from the right

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-11

Part 2: Boolean Expression

Truth Values

Boolean constants/values: *false*, *true*

The type of Boolean values: $\mathbb{B}$

— This is the type of propositions, for example: $(x = 1) : \mathbb{B}$

— For any type t, equality _=_ can be used on expressions of that type: $\_=\_ : t \to t \to \mathbb{B}$

Boolean operators:

- $\neg\_ : \mathbb{B} \to \mathbb{B}$ — negation, complement, "logical not", `\lnot`
- $\_\land\_ : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$ — conjunction, "logical and", `\land`
- $\_\lor\_ : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$ — disjunction, "logical or", "inclusive or", `\lor`
- $\_\Rightarrow\_ : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$ — implication, "implies", "if … then …", `\=>`, `\implies`
- $\_\equiv\_ : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$ — equivalence, "if and only if", "iff", `\==`, `\equiv`
- $\_\not\equiv\_ : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$ — inequivalence, "exclusive or", `\nequiv`

Table of Precedences

- $[x := e]$ (textual substitution) **(highest precedence)**
- . (function application)
- unary prefix operators $+, -, \neg, \#, \sim, \mathcal{P}$
- $**$
- $\cdot \quad / \quad \div \quad$ mod $\quad$ gcd
- $+ \quad - \quad \cup \quad \cap \quad \times \quad \circ \quad \bullet$
- $\downarrow \quad \uparrow$
- $\#$
- $\triangleleft \quad \triangleright \quad \hat{}$
- $= \quad \neq \quad < \quad > \quad \in \quad \subset \quad \subseteq \quad \supset \quad \supseteq \quad |$ (conjunctional)
- $\lor \quad \land$
- $\Rightarrow \quad \not\Rightarrow \quad \Leftarrow \quad \not\Leftarrow$
- $\equiv \quad \not\equiv$ **(lowest precedence)**

All non-associative binary infix operators associate to the left, except $**$, $\triangleleft$, $\Rightarrow$, $\rightarrow$, which associate to the right.

Binary Boolean Operators: Conjunction

Args.		$\land$	
F	*F*	*F*	The moon is green, and 2 + 2 = 7.
F	*T*	*F*	The moon is green, and 1 + 1 = 2.
T	*F*	*F*	1 + 1 = 2, and the moon is green.
T	*T*	*T*	1 + 1 = 2, and the sun is a star.

Binary Boolean Operators: Disjunction

Args.		$\lor$	
F	*F*	*F*	The moon is green, or 2 + 2 = 7.
F	*T*	*T*	The moon is green, or 1 + 1 = 2.
T	*F*	*T*	1 + 1 = 2, or the moon is green.
T	*T*	*T*	1 + 1 = 2, or the sun is a star.

This is known as "inclusive or" — see textbook p.34.

Binary Boolean Operators: Implication

Args.		$\Rightarrow$	
F	*F*	*T*	If the moon is green, then 2 + 2 = 7.
F	*T*	*T*	If the moon is green, then 1 + 1 = 2.
T	*F*	*F*	If 1 + 1 = 2, then the moon is green.
T	*T*	*T*	If 1 + 1 = 2, then the sun is a star.

$$
\begin{array}{rcl}
p \Rightarrow q & \equiv & \neg p \lor q \\
\neg p \Rightarrow q & \equiv & \neg\neg p \lor q \\
\neg p \Rightarrow q & \equiv & p \lor q
\end{array}
$$

If you don't eat your spinach, I'll spank you.	$\equiv$	You eat your spinach, or I'll spank you.

Binary Boolean Operators: Consequence

Args.		$\Leftarrow$	
F	F	T	The moon is green **if** $2 + 2 = 7$.
F	T	F	The moon is green **if** $1 + 1 = 2$.
T	F	T	$1 + 1 = 2$ **if** the moon is green.
T	T	T	$1 + 1 = 2$ **if** the sun is a star.

$$p \Leftarrow q \quad \equiv \quad p \vee \neg q$$

Binary Boolean Operators: Equivalence

Equality of Boolean values is also called **equivalence** and written $\equiv$
(In some other places: $\Leftrightarrow$)

$p \equiv q$ can be read as: p is equivalent to q
or: p exactly when q
or: p if-and-only-if q
or: p iff q

p	q	$p \equiv q$	
false	*false*	*true*	The moon is green **iff** $2 + 2 = 7$.
false	*true*	*false*	The moon is green **iff** $1 + 1 = 2$.
true	*false*	*false*	$1 + 1 = 2$ **iff** the moon is green.
true	*true*	*true*	$1 + 1 = 2$ **iff** the sun is a star.

Binary Boolean Operators: Inequivalence ("exclusive or")

Args.		$\not\equiv$	
F	F	F	Either the moon is green, or $2 + 2 = 7$.
F	T	T	Either the moon is green, or $1 + 1 = 2$.
T	F	T	Either $1 + 1 = 2$, or the moon is green.
T	T	F	Either $1 + 1 = 2$, or the sun is a star.

Table of Precedences

- $[x := e]$ (textual substitution) **(highest precedence)**
- $.$ (function application)
- unary prefix operators $+$, $-$, $\neg$, $\#$, $\sim$, $\mathcal{P}$
- $**$
- $\cdot$ $/$ $\div$ mod gcd
- $+$ $-$ $\cup$ $\cap$ $\times$ $\circ$ $\bullet$
- $\downarrow$ $\uparrow$
- $\#$
- $\triangleleft$ $\triangleright$ $\hat{}$
- $=$ $\neq$ $<$ $>$ $\in$ $\subset$ $\subseteq$ $\supset$ $\supseteq$ $|$ (conjunctional)
- $\vee$ $\wedge$
- $\Rightarrow$ $\not\Rightarrow$ $\Leftarrow$ $\not\Leftarrow$
- $\equiv$ $\not\equiv$ **(lowest precedence)**

All non-associative binary infix operators associate to the left,
except $**$, $\triangleleft$, $\Rightarrow$, $\rightarrow$, which associate to the right.

Expression Evaluation (LADM 1.1 end)

- $2 \cdot 3 + 4$
- $2 \cdot (3 + 4)$
- $2 \cdot y + 4$

A **state** is a "list of variables with associated values". E.g.:

$s_1 = [\ (x, 5),\ (y, 6)\ ]$ — (using Haskell notation for informal lists)

Evaluating an expression in a state:
"Replace variables with their values; then evaluate":

- $x - y + 2$ in state s_1
  $\longrightarrow \ 5 - 6 + 2 \ \longrightarrow \ (5 - 6) + 2 \ \longrightarrow \ (-1) + 2 \ \longrightarrow \ 1$
- $x \cdot 2 + y$
- $x \cdot (2 + y)$
- $x \cdot (z + y)$

Evaluation of Boolean Expressions

Example: Using the state $\langle (p, \mathit{false}), (q, \mathit{true}), (r, \mathit{false}) \rangle$:

$$
\begin{aligned}
& p \vee (q \wedge \neg r) \\
= \ & \langle \text{ replace variables with state values } \rangle \\
& \mathit{false} \vee (\mathit{true} \wedge \neg \mathit{false}) \\
= \ & \langle \quad \neg \mathit{false} = \mathit{true} \quad \rangle \\
& \mathit{false} \vee (\mathit{true} \wedge \mathit{true}) \\
= \ & \langle \quad \mathit{true} \wedge \mathit{true} = \mathit{true} \quad \rangle \\
& \mathit{false} \vee \mathit{true} \\
= \ & \langle \quad \mathit{false} \vee \mathit{true} = \mathit{true} \quad \rangle \\
& \mathit{true}
\end{aligned}
$$

			$\wedge$					$\not\equiv$	$\vee$	nor	$\equiv$		$\Leftarrow$		$\Rightarrow$	nand	
F	F	F	F	F	F	F	F	F	F	T	T	T	T	T	T	T	T
F	T	F	F	F	F	T	T	T	T	F	F	F	F	T	T	T	T
T	F	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T
T	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T

Evaluation of Boolean Expressions Using Truth Tables

p	q	$\neg p$	$q \wedge \neg p$	$p \vee (q \wedge \neg p)$
F	F	T	F	F
F	T	T	T	T
T	F	F	F	T
T	T	F	F	T

- Identify variables
- Identify subexpressions
- Enumerate possible states (of the variables)
- Evaluate (sub-)expressions in all states

Validity and Satisfiability

p	q	$\neg p$	$q \wedge \neg p$	$p \vee (q \wedge \neg p)$
F	F	T	F	F
F	T	T	T	T
T	F	F	F	T
T	T	F	F	T

- A boolean expression is **satisfied** in state s
  iff it evaluates to *true* in state s.
- A boolean expression is **satisfiable**
  iff there is a state in which it is satisfied.
- A boolean expression is **valid**
  iff it is satisfied in every state.
- A valid boolean expression is called a **tautology**.
- A boolean expression is called a **contradiction**
  iff it evaluates to *false* in every state.
- Two boolean expressions are called **logically equivalent**
  iff they evaluate to the same truth value in every state.

These definitions rely on states / truth tables: **Semantic concepts**

Modeling English Propositions 1

- Henry VIII had one son and Cleopatra had two.

  Henry VIII had one son and Cleopatra had two sons.

  Declarations:

  h :≡ Henry VIII had one son

  c :≡ Cleopatra had two sons

  Formalisation:

  $h \wedge c$

Modeling English Propositions — Recipe

- Transform into shape with clear subpropositions
- Introduce Boolean variables to denote subpropositions
- Replace these subpropositions by their corresponding Boolean variables
- Translate the result into a Boolean expression, using (no perfect translation rules are possible!) **for example**:

and, but	becomes	$\wedge$
or	becomes	$\vee$
not	becomes	$\neg$
it is not the case that	becomes	$\neg$
if p then q	becomes	$p \Rightarrow q$

Ladies or Tigers

Raymond Smullyan provides, in **The Lady or the Tiger?**, the following context for a number of puzzles to follow:

> [...] the king explained to the prisoner that each of the two rooms contained either a lady or a tiger, but it *could* be that there were tigers in both rooms, or ladies in both rooms, or then again, maybe one room contained a lady and the other room a tiger.

In the first case, the following signs are on the doors of the rooms:

1	2
In this room there is a lady, and in the other room there is a tiger.	In one of these rooms there is a lady, and in one of these rooms there is a tiger.

We are told that one of the signs is true, and the other one is false.

"Which door would you open (assuming, of course, that you preferred the lady to the tiger)?"

Ladies or Tigers — The First Case — Starting Formalisation

Raymond Smullyan provides, in **The Lady or the Tiger?**, the following context for a number of puzzles to follow:

> [...] the king explained to the prisoner that each of the two rooms contained either a lady or a tiger, but it *could* be that there were tigers in both rooms, or ladies in both rooms, or then again, maybe one room contained a lady and the other room a tiger.

$R1L$:= There is a lady in room 1

$R1T$:= There is a tiger in room 1

$R2L$:= There is a lady in room 2

$R2T$:= There is a tiger in room 2

> [...] We are told that one of the signs is true, and the other one is false.

S_1 := Sign 1 is true

S_2 := Sign 2 is true

Equality "=" versus Equivalence "≡"

The operators = (as Boolean operator) and ≡

- have the **same meaning** (represent the same function),
- but **are used with different notational conventions:**
  - different precedences (≡ has lowest)
  - different **chaining behaviour**:
    - ≡ is associative:

      $$(p \equiv q \equiv r) \quad = \quad ((p \equiv q) \equiv r) \quad = \quad (p \equiv (q \equiv r))$$

    - = is **conjunctional**:

      $$(x = y = z) \quad \equiv \quad ((x = y) \quad \wedge \quad (y = z))$$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-11

Part 3: LADM Propositional Calculus: ≡, ¬, ≢

Propositional Calculus

Calculus: method of reasoning by calculation with symbols

Propositional Calculus: calculating

- with Boolean expressions
- containing propositional variables

The Textbook's Propositional Calculus: **Equational Logic E**

- a set of **axioms** defining operator properties
- **four inference rules**:
  - (1.5) **Leibniz**: $\dfrac{X = Y}{E[z := X] = E[z := Y]}$ — We can apply equalities inside expressions.
  - (1.4) **Transitivity**: $\dfrac{X = Y \qquad Y = Z}{X = Z}$ — We can chain equalities.
  - (1.1) **Substitution**: $\dfrac{E}{E[x := R]}$ — We can can use substitution instances of theorems.
  - **Equanimity**: $\dfrac{X = Y \qquad X}{Y}$ — This is …

Theorems — Remember!

A **theorem** is

- **either** an **axiom**
- **or the conclusion of an inference rule** where the premises are theorems
- **or** a Boolean expression **proved** (using the inference rules) **equal** to an axiom or a previously proved **theorem**. ("— This is …")

  Such proofs will be presented in the **calculational style**.

Note:

- **The theorem definition does not use evaluation/validity**
- But:
  - All theorems in **E** are valid
  - All valid Boolean expressions are theorems in **E**
- **Important:**
  - **We will prove theorems without using validity!**
  - This trains an **essential mathematical skill!**

Equivalence Axioms

(3.1) **Axiom, Associativity of ≡**: $((p \equiv q) \equiv r) \equiv (p \equiv (q \equiv r))$

(3.2) **Axiom, Symmetry of ≡**: $p \equiv q \equiv q \equiv p$

Can be used as:

- $(p \equiv q) = (q \equiv p)$
- $p = (q \equiv q \equiv p)$
- $(p \equiv q \equiv q) = p$

Example theorem — shown differently in the textbook:

Proving $p \equiv p \equiv q \equiv q$:

$$\begin{array}{ll} & p \equiv p \equiv q \equiv q \\ = & \langle \text{(3.2) Symmetry of } \equiv \text{, with } p, q := p, q \equiv q \rangle \\ & p \equiv q \equiv q \equiv p \quad \text{— This is (3.2) Symmetry of } \equiv \end{array}$$

Equivalence Axioms — Example Proof with Parentheses

(3.1) **Axiom, Associativity of ≡**: $((p \equiv q) \equiv r) \equiv (p \equiv (q \equiv r))$

(3.2) **Axiom, Symmetry of ≡**: $p \equiv q \equiv q \equiv p$

Can be used as:

- $(p \equiv q) = (q \equiv p)$
- $p = (q \equiv q \equiv p)$
- $(p \equiv q \equiv q) = p$

Example theorem — shown differently in the textbook:

Proving $p \equiv p \equiv q \equiv q$:

$$\begin{array}{ll} & p \equiv (p \equiv (q \equiv q)) \\ \equiv & \langle \text{(3.2) Symmetry of } \equiv \text{, with } p, q := p, (q \equiv q) \rangle \\ & p \equiv ((q \equiv q) \equiv p) \quad \text{— This is (3.2) Symmetry of } \equiv \end{array}$$

Equivalence Axioms — Introducing *true*

(3.1) **Axiom, Associativity of ≡**: $((p \equiv q) \equiv r) \equiv (p \equiv (q \equiv r))$

(3.2) **Axiom, Symmetry of ≡**: $p \equiv q \equiv q \equiv p$

Can be used as:

- $(p \equiv q) = (q \equiv p)$
- $p = (q \equiv q \equiv p)$
- $(p \equiv q \equiv q) = p$

(3.3) **Axiom, Identity of ≡**: $true \equiv q \equiv q$

Can be used as:

- $(true \equiv q) = q$
- $true = (q \equiv q)$

Equivalence Axioms, and Theorem (3.4)

(3.1) **Axiom, Associativity of ≡**: $((p \equiv q) \equiv r) \equiv (p \equiv (q \equiv r))$

(3.2) **Axiom, Symmetry of ≡**: $p \equiv q \equiv q \equiv p$

(3.3) **Axiom, Identity of ≡**: $true \equiv q \equiv q$

Can be used as: $true = (q \equiv q)$

The least interesting theorem:

Proving (3.4) *true*:

$$\begin{array}{ll} & true \\ = & \langle \text{Identity of } \equiv \text{ (3.3), with } q := true \rangle \\ & true \equiv true \\ = & \langle \text{Identity of } \equiv \text{ (3.3), with } q := q \rangle \\ & true \equiv q \equiv q \quad \text{— This is Identity of } \equiv \text{ (3.3)} \end{array}$$

Equivalence Axioms and Theorems

(3.1) **Axiom, Associativity of ≡:** $((p \equiv q) \equiv r) \equiv (p \equiv (q \equiv r))$

(3.2) **Axiom, Symmetry of ≡:** $p \equiv q \equiv q \equiv p$

(3.3) **Axiom, Identity of ≡:** $true \equiv q \equiv q$

Theorems and Metatheorems:

(3.4) *true*

(3.5) **Reflexivity of ≡:** $p \equiv p$

(3.6) **Proof Method**: To prove that $P \equiv Q$ is a theorem, transform P to Q or Q to P using Leibniz.

(3.7) **Metatheorem**: Any two theorems are equivalent.

Negation Axioms

(3.8) **Axiom, Definition of *false*:** $false \equiv \neg true$

(3.9) **Axiom, Commutativity of ¬ with ≡:** $\neg(p \equiv q) \equiv \neg p \equiv q$

(LADM: "Distributivity of ¬ over ≡")

Can be used as:

- $\neg(p \equiv q) \quad = \quad (\neg p \equiv q)$
- $(\neg(p \equiv q) \equiv \neg p) \quad = \quad q$
- $(\neg(p \equiv q) \equiv q) \quad = \quad \neg p$

(3.10) **Axiom, Definition of ≢:** $(p \not\equiv q) \equiv \neg(p \equiv q)$

(3.23) Heuristic of Definition Elimination

To prove a theorem concerning an operator ∘ that is defined in terms of another, say •, expand the definition of ∘ to arrive at a formula that contains •; exploit properties of • to manipulate the formula, and then (possibly) reintroduce ∘ using its definition.

Textbook, p. 48

"Unfold-Fold strategy"

Inequivalence Theorems: Symmetry

(3.16) **Symmetry of ≢:** $(p \not\equiv q) \equiv (q \not\equiv p)$

Proving (3.16) **Symmetry of ≢:**

$$
\begin{array}{ll}
 & p \not\equiv q \\
= & \langle \text{(3.10) Definition of } \not\equiv \rangle \qquad \text{...... Unfold} \\
 & \neg(p \equiv q) \\
= & \langle \text{(3.2) Symmetry of } \equiv \rangle \\
 & \neg(q \equiv p) \\
= & \langle \text{(3.10) Definition of } \not\equiv \rangle \qquad \text{...... Fold} \\
 & q \not\equiv p
\end{array}
$$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-13

Part 1: Correctness of Assignment Commands

Plan for Today

- **Reasoning about Assignment Commands in Imperative Programs** (≈ LADM 1.6):
  - Correctness of programs with respect to pre-/post-condition specifications
  - Reasoning using "Hoare logic"
- **Continuing Propositional Calculus (LADM Chapter 3)**
  - Negation, Inequivalence
  - Disjunction
  - Conjunction

States as Program States

LADM 1.1: A **state** is a "list of variables with associated values". E.g.:

$s_1 = [\ (x, 5),\ (y, 6)\ ]$ — (using Haskell notation for informal lists)

Evaluating an expression in a state:

"Replace variables with their values; then evaluate"

- In logic, "states" are usually called "variable assignments"
- States can serve as a mathematical model of **program states**
- Execution of imperative programs induces **state transformation**:

$$
\begin{array}{ll}
 & [\ (x,5),\ (y,6)\ ] \\
\rightsquigarrow & \langle\quad x := x + y \quad\rangle \\
 & [\ (x,11),\ (y,6)\ ] \\
\rightsquigarrow & \langle\quad y := x - y \quad\rangle \\
 & [\ (x,11),\ (y,5)\ ]
\end{array}
$$

State Predicates

- Execution of imperative programs induces **state transformation**:

$$
\begin{array}{lll}
 & [\ (x,5),\ (y,6)\ ] & \text{...... `} x < y \text{` holds} \\
\rightsquigarrow & \langle\quad x := x + y \quad\rangle & \\
 & [\ (x,11),\ (y,6)\ ] & \text{...... `} x < y \text{` does not hold} \\
\rightsquigarrow & \langle\quad y := x - y \quad\rangle & \\
 & [\ (x,11),\ (y,5)\ ] & \text{...... `} x < y \text{` does not hold}
\end{array}
$$

- Boolean expressions containing variables can be used as **state predicates**:

P "holds in state s" iff P evaluates to *true* in state s

Precondition-Postcondition Specifications

- Program correctness statement in LADM (and much current use):

$$\{\ P\ \}\ C\ \{\ Q\ \}$$

This is called a "Hoare triple".

- **Meaning:** If command C is started in a state in which the **precondition** P holds, then it will terminate only in a state in which the **postcondition** Q holds.
- Hoare's original notation:

$$P\ \{\ C\ \}\ Q$$

- **Dynamic logic** notation (will be used in CalcCheck):

$$P \Rightarrow\!\![\ C\ ]\ Q$$

Correctness of Assignment Commands

- *Recall:* Hoare triple: $\{\ P\ \}\ C\ \{\ Q\ \}$
- **Dynamic logic** notation (will be used in CalcCheck): $P \Rightarrow\!\![\ C\ ]\ Q$
- **Meaning:** If command C is started in a state in which the **precondition** P holds, then it will terminate only in a state in which the **postcondition** Q holds.
- **Assignment Axiom:** $\{\ Q[x := E]\ \}\ x := E\ \{\ Q\ \}$ $\qquad Q[x := E] \quad \Rightarrow\!\![\ x := E\ ] \quad Q$
- **Example:**
  - $(x = 5)[x := x + 1] \quad \Rightarrow\!\![\ x := x + 1\ ] \quad x = 5$
  - $(x + 1 = 5) \quad \Rightarrow\!\![\ x := x + 1\ ] \quad x = 5$
  - $$
\begin{array}{ll}
 & x + 1 = 5 \\
\equiv & \langle \text{Substitution} \rangle \\
 & (x = 5)[x := x + 1] \\
\Rightarrow\!\![\ x := x + 1\ ] & \langle \text{Assignment} \rangle \\
 & x = 5
\end{array}
$$
- **Substitution ":=": One Unicode character; type "\:="** — **Assignment ":=": Two characters; type ":="**

Correctness of Assignment Commands — Longer Example

- *Recall:* Hoare triple: $\{ P \}\ C\ \{ Q \}$
- **Dynamic logic** notation (will be used in CalcCheck):
  $P \Rrightarrow [\ C\ ]\ Q$
- **Meaning:** If command C is started in a state in which the **precondition** P holds, then it will terminate only in a state in which the **postcondition** Q holds.
- **Assignment Axiom:** $\{ Q[x := E] \}\ x := E\ \{ Q \}$ $\quad Q[x := E] \Rrightarrow [\ x := E\ ]\ Q$
- **Longer example** (**these proofs are developed from the bottom to the top!**):

$true$
$\equiv$ ⟨ Zero of $\vee$ ⟩
$1 = 0 \vee true$
$\equiv$ ⟨ Reflexivity of $=$ ⟩
$1 = 0 \vee 1 = 1$
$\equiv$ ⟨ Substitution ⟩
$(x = 0 \vee x = 1)[x := 1]$
$\Rrightarrow [\ x := 1\ ]$ ⟨ Assignment ⟩
$x = 0 \vee x = 1$

Example Proof for a Sequence of Assignments

Lemma (4):
$x = 5$
$\Rrightarrow [\ y := x + 1\ ;$
$\quad x := y + y$
$]$
$x = 12$

Read and write such "$\_ \Rrightarrow [\_]\ \_$" proofs from the bottom to the top!

Proof:
$x = 5$
$\equiv$ ⟨ "Cancellation of $+$" ⟩
$x + 1 = 5 + 1$
$\equiv$ ⟨ Fact \`$5 + 1 = 6$\` ⟩
$x + 1 = 6$
$\equiv$ ⟨ Substitution ⟩
$(y = 6)[y := x + 1]$
$\Rrightarrow [\ y := x + 1\ ]$ ⟨ "Assignment" ⟩
$y = 6$
$\equiv$ ⟨ "Cancellation of $\cdot$" with Fact \`$2 \neq 0$\` ⟩
$2 \cdot y = 2 \cdot 6$
$\equiv$ ⟨ Evaluation ⟩
$(1 + 1) \cdot y = 12$
$\equiv$ ⟨ "Distributivity of $\cdot$ over $+$" ⟩
$1 \cdot y + 1 \cdot y = 12$
$\equiv$ ⟨ "Identity of $\cdot$" ⟩
$y + y = 12$
$\equiv$ ⟨ Substitution ⟩
$(x = 12)[x := y + y]$
$\Rrightarrow [\ x := y + y\ ]$ ⟨ "Assignment" ⟩
$x = 12$

Sequential Composition of Commands

Primitive inference rule "SEQ":

$$\frac{\{ P \}\ C_1\ \{ Q \}, \quad \{ Q \}\ C_2\ \{ R \}}{\{ P \}\ C_1 ; C_2\ \{ R \}}$$

Primitive inference rule "Sequence":

$$\frac{P \Rrightarrow [\ C_1\ ]\ Q, \quad Q \Rrightarrow [\ C_2\ ]\ R}{P \Rrightarrow [\ C_1 ; C_2\ ]\ R}$$

- Activated as transitivity rule
- Therefore used implicitly in calculations, e.g., proving $P \Rrightarrow [\ C_1 ; C_2\ ]\ R$ by:

P
$\Rrightarrow [\ C_1\ ]$ ⟨ … ⟩
Q
$\Rrightarrow [\ C_2\ ]$ ⟨ … ⟩
R

- No need to refer to this rule explicitly.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-13

Part 2: Propositional Calculus: $\neg$, $\not\equiv$, $\vee$, $\wedge$

Equivalence Axioms and Theorems

(3.1) **Axiom, Associativity of $\equiv$:** $((p \equiv q) \equiv r) \equiv (p \equiv (q \equiv r))$

(3.2) **Axiom, Symmetry of $\equiv$:** $p \equiv q \equiv q \equiv p$ —

(3.3) **Axiom, Identity of $\equiv$:** $true \equiv q \equiv q$

Can be used as:
- $(p \equiv q) = (q \equiv p)$
- $p = (q \equiv q \equiv p)$
- $(p \equiv q \equiv q) = p$

Theorems and Metatheorems:

(3.4) $true$

(3.5) **Reflexivity of $\equiv$:** $p \equiv p$

(3.6) **Proof Method**: To prove that $P \equiv Q$ is a theorem, transform P to Q or Q to P using Leibniz.

(3.7) **Metatheorem**: Any two theorems are equivalent.

Proof Method Equanimity: To prove P, prove $P \equiv Q$ where Q is a theorem. (Document via "— This is …".)

Special case: To prove P, prove $P \equiv true$.

Negation Axioms

(3.8) **Axiom, Definition of** $false$: $false \equiv \neg true$

(3.9) **Axiom, Commutativity of $\neg$ with $\equiv$:** $\neg(p \equiv q) \equiv \neg p \equiv q$

(LADM: "Distributivity of $\neg$ over $\equiv$")

Can be used as:
- $\neg(p \equiv q) \quad = \quad (\neg p \equiv q)$
- $(\neg(p \equiv q) \equiv \neg p) \quad = \quad q$
- $(\neg(p \equiv q) \equiv q) \quad = \quad \neg p$

(3.10) **Axiom, Definition of $\not\equiv$:** $(p \not\equiv q) \equiv \neg(p \equiv q)$

Negation Axioms and Theorems

(3.8) **Axiom, Definition of** $false$: $false \equiv \neg true$

(3.9) **Axiom, Commutativity of $\neg$ with $\equiv$:** $\neg(p \equiv q) \equiv \neg p \equiv q$

(3.10) **Axiom, Definition of $\not\equiv$:** $(p \not\equiv q) \equiv \neg(p \equiv q)$

Theorems:

(3.11) $\neg p \equiv q \equiv p \equiv \neg q$

— can be used as "**$\neg$ connection**": $(\neg p \equiv q) \equiv (p \equiv \neg q)$

— can be used as "**Cancellation of $\neg$**": $(\neg p \equiv \neg q) \equiv (p \equiv q)$

(3.12) **Double negation**: $\neg\neg p \equiv p$

(3.13) **Negation of** $false$: $\neg false \equiv true$

(3.14) $(p \not\equiv q) \equiv \neg p \equiv q$

(3.15) **Definition of $\neg$ via $\equiv$:** $\neg p \equiv p \equiv false$

Inequivalence Theorems

(3.16) **Symmetry of $\not\equiv$:** $(p \not\equiv q) \quad \equiv \quad (q \not\equiv p)$

(3.17) **Associativity of $\not\equiv$:** $((p \not\equiv q) \not\equiv r) \quad \equiv \quad (p \not\equiv (q \not\equiv r))$

(3.18) **Mutual associativity:** $((p \not\equiv q) \equiv r) \quad \equiv \quad (p \not\equiv (q \equiv r))$

(3.19) **Mutual interchangeability:** $p \not\equiv q \equiv r \quad \equiv \quad p \equiv q \not\equiv r$

Note: Mutual associativity is not (yet…) automated!

(But omission of parentheses is implemented, similar to
- $k - m + n$
- $k + m - n$
- $k - m - n$

— None of these has $m - n$ as subexpression!
— But the second one is equal to $k + (m - n)$ …)

(3.23) Heuristic of Definition Elimination

To prove a theorem concerning an operator $\circ$ that is defined in terms of another, say $\bullet$, expand the definition of $\circ$ to arrive at a formula that contains $\bullet$; exploit properties of $\bullet$ to manipulate the formula, and then (possibly) reintroduce $\circ$ using its definition.

Textbook, p. 48

"Unfold-Fold strategy"

Inequivalence Theorems: Symmetry

(3.16) **Symmetry of $\not\equiv$:** $(p \not\equiv q) \equiv (q \not\equiv p)$

Proving (3.16) **Symmetry of $\not\equiv$:**

$p \not\equiv q$
$=$ ⟨ (3.10) Definition of $\not\equiv$ ⟩ ⋯⋯ **Unfold**
$\neg(p \equiv q)$
$=$ ⟨ (3.2) Symmetry of $\equiv$ ⟩
$\neg(q \equiv p)$
$=$ ⟨ (3.10) Definition of $\not\equiv$ ⟩ ⋯⋯ **Fold**
$q \not\equiv p$

Disjunction Axioms

(3.24) **Axiom, Symmetry of $\vee$**: $p \vee q \equiv q \vee p$

(3.25) **Axiom, Associativity of $\vee$**: $(p \vee q) \vee r \equiv p \vee (q \vee r)$

(3.26) **Axiom, Idempotency of $\vee$**: $p \vee p \equiv p$

(3.27) **Axiom, Distributivity of $\vee$ over $\equiv$**: $p \vee (q \equiv r) \equiv p \vee q \equiv p \vee r$

(3.28) **Axiom, Excluded Middle**: $p \vee \neg p$

The Law of the Excluded Middle (LEM)

Aristotle:

> . . . there cannot be an **intermediate** between contradictories, but of one subject we must either affirm or deny any one predicate. . .

Bertrand Russell in "The Problems of Philosophy":

> Three "Laws of Thought":
> 1. Law of identity: "Whatever is, is."
> 2. Law of noncontradiction: "Nothing can both be and not be."
> 3. Law of excluded middle: "Everything must either be or not be."
>
> These three laws are samples of self-evident logical principles. . .

(3.28) **Axiom, Excluded Middle**: $p \vee \neg p$

— this will often be used as: $p \vee \neg p \equiv \mathit{true}$

Disjunction Axioms and Theorems

(3.24)	**Axiom, Symmetry of $\vee$**:	$p \vee q \equiv q \vee p$
(3.25)	**Axiom, Associativity of $\vee$**:	$(p \vee q) \vee r \equiv p \vee (q \vee r)$
(3.26)	**Axiom, Idempotency of $\vee$**:	$p \vee p \equiv p$
(3.27)	**Axiom, Distr. of $\vee$ over $\equiv$**:	$p \vee (q \equiv r) \equiv p \vee q \equiv p \vee r$
(3.28)	**Axiom, Excluded Middle**:	$p \vee \neg p$

Theorems:

(3.29) **Zero of $\vee$**: $p \vee \mathit{true} \equiv \mathit{true}$

(3.30) **Identity of $\vee$**: $p \vee \mathit{false} \equiv p$

(3.31) **Distrib. of $\vee$ over $\vee$**: $p \vee (q \vee r) \equiv (p \vee q) \vee (p \vee r)$

(3.32) **(3.32)** $p \vee q \;\equiv\; p \vee \neg q \;\equiv\; p$

Heuristics of Directing Calculations

(3.33) **Heuristic:** To prove $P \equiv Q$, transform the expression with the most structure (either P or Q) into the other.

Proving (3.29) $p \vee \mathit{true} \equiv \mathit{true}$:

$p \vee \mathit{true}$
$\equiv$ ⟨ Identity of $\equiv$ (3.3) ⟩
$p \vee (q \equiv q)$
$\equiv$ ⟨ Distr. of $\vee$ over $\equiv$ (3.27) ⟩
$p \vee q \equiv p \vee q$
$\equiv$ ⟨ Identity of $\equiv$ (3.3) ⟩
true

Proving (3.29) $p \vee \mathit{true} \equiv \mathit{true}$:

true
$\equiv$ ⟨ Identity of $\equiv$ (3.3) ⟩
$p \vee p \equiv p \vee p$
$\equiv$ ⟨ Distr. of $\vee$ over $\equiv$ (3.27) ⟩
$p \vee (p \equiv p)$
$\equiv$ ⟨ Identity of $\equiv$ (3.3) ⟩
$p \vee \mathit{true}$

?

(3.34) **Principle:** Structure proofs to minimize the number of rabbits pulled out of a hat — make each step seem obvious, based on the structure of the expression and the goal of the manipulation.

(3.21) Heuristic

Identify applicable theorems by matching the structure of expressions or subexpressions. The operators that appear in a boolean expression and the shape of its subexpressions can focus the choice of theorems to be used in manipulating it.

Obviously, the more theorems you know by heart and the more practice you have in pattern matching, the easier it will be to develop proofs.

Textbook, p. 47

The Conjunction Axiom: The "Golden Rule"

(3.35) **Axiom, Golden rule**: $p \wedge q \;\equiv\; p \equiv q \;\equiv\; p \vee q$

Can be used as:

- $p \wedge q \;=\; (p \equiv q \;\equiv\; p \vee q)$ — **Definition of $\wedge$**
- $(p \equiv q) \;=\; (p \wedge q \;\equiv\; p \vee q)$
- . . .

Theorems:

(3.36) **Symmetry of $\wedge$**: $p \wedge q \;\equiv\; q \wedge p$

(3.37) **Associativity of $\wedge$**: $(p \wedge q) \wedge r \;\equiv\; p \wedge (q \wedge r)$

(3.38) **Idempotency of $\wedge$**: $p \wedge p \;\equiv\; p$

(3.39) **Identity of $\wedge$**: $p \wedge \mathit{true} \;\equiv\; p$

(3.40) **Zero of $\wedge$**: $p \wedge \mathit{false} \;\equiv\; \mathit{false}$

(3.41) **Distributivity of $\wedge$ over $\wedge$**: $p \wedge (q \wedge r) \equiv (p \wedge q) \wedge (p \wedge r)$

(3.42) **Contradiction**: $p \wedge \neg p \;\equiv\; \mathit{false}$

Conjunction Theorems: Symmetry

(3.36) **Symmetry of $\wedge$**: $(p \wedge q) \equiv (q \wedge p)$

Proving (3.36) **Symmetry of $\wedge$:**

$p \wedge q$
$\equiv$ ⟨ (3.35) Definition of $\wedge$ (Golden rule) ⟩ — **Unfold**
$p \equiv q \;\equiv\; p \vee q$
$\equiv$ ⟨ (3.2) Symmetry of $\equiv$, (3.24) Symmetry of $\vee$ ⟩
$q \equiv p \;\equiv\; q \vee p$
$\equiv$ ⟨ (3.35) Definition of $\wedge$ (Golden rule) ⟩ — **Fold**
$q \wedge p$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-15

- **Natural Induction**
- **Propositional Calculus: $\wedge$**

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-15

Part 1: Natural Numbers, Natural Induction

What is a natural number?

How is the set $\mathbb{N}$ of all natural numbers defined?

(Without referring to the integers)

(From first principles. . .)

Natural Numbers — ℕ

- The set of all **natural numbers** is written ℕ.
- In Computing, zero "0" is a natural number.
- If n is a natural number, then its successor "*suc n*" is a natural number, too.
- We write
  - "1" for "*suc* 0"
  - "2" for "*suc* 1"
  - "3" for "*suc* 2"
  - "4" for "*suc* 3"
  - ...
- **In Haskell** (data constructors start with upper-case letters):

```
data Nat = Zero | Suc Nat
```

Natural Numbers — Rigorous Definition

- The set of all **natural numbers** is written ℕ.
- Zero "0" is a natural number.
- If n is a natural number, then its successor "*suc n*" is a natural number, too.
- Nothing else is a natural number.
- Two natural numbers are equal **if and only if** they are constructed in the same way.

  Example: *suc suc suc* 0 $\neq$ *suc suc suc suc* 0

This is an inductive definition.

(Like the definition of expressions...)

Every inductive definition gives rise to an induction principle
— a way to prove statements about the inductively defined elements

Natural Numbers — Induction Principle

- The set of all **natural numbers** is written ℕ.
- Zero "0" is a natural number.
- If n is a natural number, then its successor "*suc n*" is a natural number, too.

Induction principle for the natural numbers:

- if $P(0)$ — If P holds for 0
- and if $P(m)$ implies $P(suc\ m)$, — and whenever P holds for m, it also holds for *suc m*,
- then for all $m : ℕ$ we have $P(m)$. — then P holds for all natural numbers.

Natural Numbers — Induction Proofs

Induction principle for the natural numbers:

- if $P[m := 0]$ — If P holds for 0
- and if we can obtain $P[m := suc\ m]$ from P, — and whenever P holds for m, it also holds for *suc m*,
- then P holds. — then P holds for all natural numbers.

An **induction proof** using this looks as follows:

Theorem: P
Proof:
  By induction on $m : ℕ$:
    Base case:
      Proof for $P[m := 0]$
    Induction step:
      Proof for $P[m := suc\ m]$
        using **Induction hypothesis** P

$$\frac{P[m := 0] \qquad \begin{matrix}\ulcorner P \urcorner \\ \vdots \\ P[m := suc\ m]\end{matrix}}{P}$$

Factorial — Inductive Definition

- The set of all **natural numbers** is written ℕ.
- zero "0" is a natural number.
- If n is a natural number, then its successor "*suc n*" is a natural number, too.
- Nothing else is a natural number.
- Two natural numbers are only equal if constructed in the same way.

ℕ is an inductively-defined set.

The factorial operator "_!" on ℕ can be defined as follows:

- The factorial of a natural number is a natural number again:
  $\_! : ℕ \to ℕ$
- $0\,! = 1$
- For every $n : ℕ$, we have:
  $$(suc\ n)\,! = (suc\ n) \cdot (n\,!)$$

_! is an inductively-defined function.

Proving properties about inductively-defined functions on ℕ frequently requires use of the induction principle for ℕ.

Even Natural Numbers — Inductive Definition

- The predicates even and odd are declared as Boolean-valued **functions**:

  Declaration: *even*, odd : ℕ → 𝔹

- Function application of function f to argument a is written as **juxtaposition**: $f\ a$
- The definitions provided in Homework 5.1 are **inductive definitions**:

  Axiom "Zero is even": even 0 ▪▪▪▪▪▪ read this as: `even 0 ≡ true`
  Axiom "Even successor": even (suc n) ≡ ¬ (even n)

even is an inductively-defined function.

Why does this define even for all possible arguments?
Because:

- *even* takes **one** argument of type ℕ
- This argument is **always** either 0, or *suc k* for some **smaller** $k : ℕ$
- Each clause covers one case completely.
- The second clause "builds up" the domain of definition of *even* from smaller to larger n.

Proving "Odd is not even"

Theorem "Odd is not even": odd n ≡ ¬ (even n)

Axiom "Zero is even": even 0 ▪▪▪▪▪▪ read this as: `even 0 ≡ true`
Axiom "Even successor": even (suc n) ≡ ¬ (even n)
Axiom "Zero is not odd": ¬ odd 0
Axiom "Odd successor": odd (suc n) ≡ ¬ (odd n)

An **induction proof** looks as follows:

Theorem: P
Proof:
  By induction on $m : ℕ$:
    Base case:
      Proof for $P[m := 0]$
    Induction step:
      Proof for $P[m := suc\ m]$
        using **Induction hypothesis** P

Proving "Odd is not even"

Theorem "Odd is not even": odd n ≡ ¬ (even n)

Axiom "Zero is even": even 0 ▪▪▪▪▪▪ read this as: `even 0 ≡ true`
Axiom "Even successor": even (suc n) ≡ ¬ (even n)
Axiom "Zero is not odd": ¬ odd 0
Axiom "Odd successor": odd (suc n) ≡ ¬ (odd n)

Proof:
  By induction on `$n$ : ℕ`:
    Base case:
        odd 0
      ≡ ⟨ ? ⟩
        ¬ (even 0)
    Induction step:
        odd (suc n)
      ≡ ⟨ ? ⟩
        ¬ (odd n)
      ≡ ⟨ Induction hypothesis ⟩
        ¬ ¬ (even n)
      ≡ ⟨ ? ⟩
        ¬ even (suc n)

Natural Number Addition — Inductive Definition

- The set of all **natural numbers** is written ℕ.
- zero "0" is a natural number.
- If n is a natural number, then its successor "*suc n*" is a natural number, too.
- Nothing else is a natural number.
- Two natural numbers are only equal if constructed in the same way.

ℕ is an inductively-defined set.

Addition on ℕ can be defined as follows:

- The (infix) **addition operator** "+", when applied to two natural numbers, produces again a natural number
  $\_+\_ : ℕ \to ℕ \to ℕ$
- For every $q : ℕ$, we have:
  - $0 + q = q$
  - For every $n : ℕ$ we have: $(suc\ n) + q = suc\ (n + q)$

_+_ is an inductively-defined function.

Proving "Right-Identity of +"

Theorem "Right-identity of +": $m + 0 = m$
Proof:
  By induction on `$m$ : ℕ`:
    Base case:
        0 + 0
      = ⟨ "Definition of + for 0" ⟩
        0
    Induction step:
        suc m + 0
      = ⟨ "Definition of + for `suc`" ⟩
        suc (m + 0)
      = ⟨ Induction hypothesis ⟩
        suc m

An **induction proof** looks as follows:

Theorem: P
Proof:
  By induction on $m : ℕ$:
    Base case:
      Proof for $P[m := 0]$
    Induction step:
      Proof for $P[m := suc\ m]$
        using **Induction hypothesis** P

Proving "Right-Identity of +" — With Details

Theorem "Right-identity of +": $m + 0 = m$
Proof:
  By induction on `$m : \mathbb{N}$`:
    Base case `$0 + 0 = 0$`:
      $0 + 0$
    $= \langle$ "Definition of + for 0" $\rangle$
      0
    Induction step `$\mathsf{suc}\ m + 0 = \mathsf{suc}\ m$`:
      $\mathsf{suc}\ m + 0$
    $= \langle$ "Definition of + for `suc`" $\rangle$
      $\mathsf{suc}\ (m + 0)$
    $= \langle$ Induction hypothesis `$m + 0 = m$` $\rangle$
      $\mathsf{suc}\ m$

An **induction proof** looks as follows:

Theorem: P
Proof:
  By induction on $m : \mathbb{N}$:
    Base case:
      Proof for $P[m := 0]$
    Induction step:
      Proof for $P[m := suc\ m]$
        using **Induction hypothesis** P

Proving "Right-Identity of +" — Indentation!

```
Theorem "Right-identity of +": m + 0 = m
Proof:
␣␣By induction on `m : ℕ`:
␣␣␣␣Base case:
␣␣␣␣␣␣␣␣0 + 0
␣␣␣␣␣␣=⟨ "Definition of + for 0" ⟩
␣␣␣␣␣␣␣␣0
␣␣␣␣Induction step:
␣␣␣␣␣␣␣␣suc m + 0
␣␣␣␣␣␣=⟨ "Definition of + for `suc`" ⟩
␣␣␣␣␣␣␣␣suc (m + 0)
␣␣␣␣␣␣=⟨ Induction hypothesis ⟩
␣␣␣␣␣␣␣␣suc m
```

Press "Ctrl-Shift-v" to toggle "visible spaces".

Read Parse Error Messages!

≡ ⟨ Substitution ⟩
  — CalcCheck: Due to parse error in the expression below, this calculation step cannot be checked.
⟪ Parse error: "Cell 12" (line 19, column 16):
  unexpected "="
  expecting white space, "------", ",", or := «expressions»
  ⟫
⇒[y := z - y] ⟨ "Assignment" ⟩
  — CalcCheck: Found "Assignment"
  — CalcCheck: Due to parse error in the expression above, this calculation step cannot be checked.

```
18:    ≡⟨ Substitution ⟩
19:       (y = 42)[y = z - y]
20:    ⇒[ y := z - y ]   ⟨ "Assignment" ⟩
```

Submitting parse errors is unprofessional!

Carefully Check Indentation: Each Level ≥ 2 Spaces!

≡ ⟨ Substitution ⟩
  — CalcCheck: Due to parse error in the expression below, this calculation step cannot be checked
⟪ Parse error: "Cell 12" (line 18, column 25):
  unexpected """
  expecting white space, "------", or «expression»
  ⟫

```
16:    ≡⟨ Substitution ⟩
17:       (y = z - y)[y = z - y]
18:    ⇒[ y := z - y ]   ⟨ "Assignment" ⟩
19:       y = 42
```

Hint item where the parser expects an expression —

calculation operators need to be aligned
two spaces to the left of calculation expressions!

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-15

Part 2: A Look at the Outline

Academic Integrity (see also page 4) — **Course-Specific Notes**

Academic credentials you earn are rooted in principles of honesty and academic integrity.

In the context of COMPSCI 2LC3, in particular the following behaviours constitute academic dishonesty:

1. *Plagiarism*, i.e., **the submission of work that is not one's own** or for which other credit has been obtained.
2. **Collaboration where individual work is expected.**

   You have to produce your submissions for homework and assignment questions yourself, and without collaboration.

   For each assignment question there will normally be exercise questions similar to it — you **are allowed** to collaborate on these **exercise questions**. (The tutorials are typically not expected to cover all exercise questions.)

   - You are not allowed to copy & edit any portion of another student's work, nor from any websites, but you may use material from the course notes.
   - You are not allowed to give your solutions (or portions thereof) to another student.
   - You are not allowed to work on your homework or assignment with other students, nor with friends, parents, relatives, etc..
   - You are not allowed to post full or partial homework or assignment solutions on discussion boards or websites (e.g., github, FaceBook, etc..).
   - You are not allowed to solicit solutions to the problem on on-line forums or purchasing solutions from on-line sources.
   - You are not allowed to submit a combined solution with a classmate.
3. **Copying or using unauthorised aids in tests and examinations.**
4. **Accessing another students' Avenue or other relevant online account, or providing others access to your accounts.**
5. **Accessing or attempting to access midterm or exam material outside the individually assigned writing time and space.**
6. Meddling or attempting to meddle with online services used for course delivery.

Note: **If you cheat, you are cheating yourself.**

Later in the course, we intend to have individually-generated assignments and tests and so collaboration or cheating early on in the course will result in hardship during time-constrained midterms with individualised assignments where collaboration is no longer feasible and each person must use the allotted time to solve their individual problems.

You need to solve the Homeworks yourself!

- Assuming that you can pass this course without actually acquiring the expected reasoning skills is most likely unrealistic.
- You acquire the skills by doing Homeworks and Assignments yourself!
- If you provide your solutions to others,
  - that constitutes academic dishonesty as well!
- If you provide your solutions to others,
  - that actually reduces their chances to acquire the skills and pass the course!
- Large cluster of extremely similar submissions strongly suggest that large groups of students are not getting the expected learning:
  - I need to act!
- If homeworks were to be done with pen and paper, you would submit imperfect solutions without hesitation. . .

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-15

Part 3: Propositional Calculus: ∧ — Conjunction

The Conjunction Axiom: The "Golden Rule"

(3.35) **Axiom, Golden rule**: $p \wedge q \quad \equiv \quad p \equiv q \quad \equiv \quad p \vee q$

Can be used as:

- $p \wedge q \quad = \quad (p \equiv q \quad \equiv \quad p \vee q)$ — **Definition of ∧**
- $(p \equiv q) \quad = \quad (p \wedge q \quad \equiv \quad p \vee q)$
- . . .

Theorems:

(3.36) **Symmetry of ∧**: $p \wedge q \quad \equiv \quad q \wedge p$
(3.37) **Associativity of ∧**: $(p \wedge q) \wedge r \quad \equiv \quad p \wedge (q \wedge r)$
(3.38) **Idempotency of ∧**: $p \wedge p \quad \equiv \quad p$
(3.39) **Identity of ∧**: $p \wedge true \quad \equiv \quad p$
(3.40) **Zero of ∧**: $p \wedge false \quad \equiv \quad false$
(3.41) **Distributivity of ∧ over ∧**: $p \wedge (q \wedge r) \equiv (p \wedge q) \wedge (p \wedge r)$
(3.42) **Contradiction**: $p \wedge \neg p \quad \equiv \quad false$

Conjunction Theorems: Symmetry

(3.36) **Symmetry of ∧:** $(p \wedge q) \equiv (q \wedge p)$

Proving (3.36) **Symmetry of ∧:**

$$
\begin{array}{ll}
& p \wedge q \\
\equiv & \langle \text{(3.35) Definition of } \wedge \text{ (Golden rule)} \rangle \quad \text{— Unfold} \\
& p \equiv q \quad \equiv \quad p \vee q \\
\equiv & \langle \text{(3.2) Symmetry of } \equiv, \text{(3.24) Symmetry of } \vee \rangle \\
& q \equiv p \quad \equiv \quad q \vee p \\
\equiv & \langle \text{(3.35) Definition of } \wedge \text{ (Golden rule)} \rangle \quad \text{— Fold} \\
& q \wedge p
\end{array}
$$

Theorems Relating ∧ and ∨

(3.43) **Absorption**: $p \wedge (p \vee q) \equiv p$
$p \vee (p \wedge q) \equiv p$

(3.44) **Absorption**: $p \wedge (\neg p \vee q) \equiv p \wedge q$
$p \vee (\neg p \wedge q) \equiv p \vee q$

(3.45) **Distributivity of ∨ over ∧:** $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$

(3.46) **Distributivity of ∧ over ∨:** $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$

(3.47) **De Morgan**: $\neg(p \wedge q) \equiv \neg p \vee \neg q$
$\neg(p \vee q) \equiv \neg p \wedge \neg q$

Boolean Lattice Duality

A **Boolean-lattice expression** is
- either a variable,
- or *true* or *false*
- or an application of ¬_ to a Boolean-lattice expression
- or an application of _∧_ or _∨_ to two Boolean-lattice expressions.

The **dual** of a Boolean-lattice expressions is obtained by
- replacing *true* with *false* and vice versa,
- replacing _∧_ with _∨_ and vice versa.

The **dual** of a Boolean-lattice equation (equivalence) is the equation between the duals of the LHS and the RHS.

Metatheorem "Boolean lattice duality":
Every Boolean-lattice equation is valid iff its dual is valid.

Metatheorem "Boolean lattice duality":
Every Boolean-lattice equation is a theorem iff its dual is a theorem.

Theorems Relating ∧ and ≡

(3.48) **(3.48)** $p \wedge q \equiv p \wedge \neg q \equiv \neg p$

(3.49) Semi-distributivity of ∧ over ≡ $p \wedge (q \equiv r) \equiv p \wedge q \equiv p \wedge r \equiv p$

(3.50) Strong modus ponens for ≡ $p \wedge (q \equiv p) \equiv p \wedge q$

(3.51) **Replacement**: $(p \equiv q) \wedge (r \equiv p) \equiv (p \equiv q) \wedge (r \equiv q)$

Alternative Definitions of ≡ and ≢

(3.52) **Alternative definition of ≡:** $p \equiv q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$

(3.53) **Alternative definition of ≢:** $p \not\equiv q \equiv (\neg p \wedge q) \vee (p \wedge \neg q)$

Ladies or Tigers: First Case, Formalisation, Long S_2

In the first case, the following signs are on the doors of the rooms:

1	2
In this room there is a lady, and in the other room there is a tiger.	In one of these rooms there is a lady, and in one of these rooms there is a tiger.

We are told that one of the signs is true, and the other one is false.

$R1L$	:=	There is a lady in room 1	S_1	≡	$R1L \wedge R2T$
$R2T$	:=	There is a tiger in room 2	S_2	≡	$(R1L \vee \neg R2T) \wedge (\neg R1L \vee R2T)$

$S_1 \not\equiv S_2$

Ladies or Tigers: First Case, Long S_2, Solution

$R1L$	:=	There is a lady in room 1	S_1	≡	$R1L \wedge R2T$
$R2T$	:=	There is a tiger in room 2	S_2	≡	$(R1L \vee \neg R2T) \wedge (\neg R1L \vee R2T)$

$$
\begin{array}{ll}
& S_1 \not\equiv S_2 \\
= & \langle \text{Def. } S_1, S_2 \rangle \\
& (R1L \wedge R2T) \not\equiv ((R1L \vee \neg R2T) \wedge (\neg R1L \vee R2T)) \\
= & \langle \text{(3.14) } p \not\equiv q \equiv \neg p \equiv q, \text{(3.35) Golden Rule} \rangle \\
& \neg(R1L \wedge R2T) \equiv R1L \vee \neg R2T \equiv \neg R1L \vee R2T \equiv R1L \vee \neg R2T \vee \neg R1L \vee R2T \\
= & \langle \text{(3.28) Excluded Middle, (3.29) Zero of } \vee \rangle \\
& \neg(R1L \wedge R2T) \equiv R1L \vee \neg R2T \equiv \neg R1L \vee R2T \equiv \mathit{true} \\
= & \langle \text{(3.47) De Morgan, (3.3) Identity of } \equiv \rangle \\
& \neg R1L \vee \neg R2T \equiv R1L \vee \neg R2T \equiv \neg R1L \vee R2T \\
= & \langle \text{(3.32) } p \vee q \equiv p \vee \neg q \equiv p \rangle \\
& \neg R2T \equiv \neg R1L \vee R2T \\
= & \langle \text{(3.32) } p \vee q \equiv p \vee \neg q \equiv p \rangle \\
& \neg R2T \equiv \neg R1L \vee \neg R2T \equiv \neg R1L \\
= & \langle \text{(3.35) Golden Rule} \rangle \\
& \neg R1L \wedge \neg R2T \\
= & \langle R1T = \neg R1L \text{ and } R2L = \neg R2T \rangle \\
& R1T \wedge R2L
\end{array}
$$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-18

- **Introduction to Quantification (LADM ch. 8)**
- **Propositional Calculus: Implication ⇒**

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-18

Part 1: Introduction to Quantification (start LADM chapt. 8),

Quantification expansion

Counting Integral Points

How many integral points are in the triangle

$$
\begin{array}{ccc}
(0,n) & & \\
| & \diagdown & \\
(0,0) & — & (n,0)
\end{array}
$$

?

$$
\begin{array}{ll}
& \sum_{x=0}^{n} (n - x + 1) \\
= & \langle \text{Summing 1 values} \rangle \\
& \sum_{x=0}^{n} (\sum_{y=0}^{n-x} 1) \\
= & \langle \text{Switch to linear quantification notation} \rangle \\
& (\sum x \mid 0 \le x \le n \bullet (\sum y \mid 0 \le y \le n - x \bullet 1)) \\
= & \langle \text{Nesting} \rangle \\
& (\sum x, y \mid 0 \le x \le n \wedge 0 \le y \le n - x \bullet 1) \\
= & \langle \text{Isotonicity of } + \rangle \\
& (\sum x, y \mid 0 \le x \le n \wedge x \le x + y \le n \bullet 1) \\
= & \langle \text{Def. of } \Rightarrow \text{(3.60) with Transitivity of } \le \rangle \\
& (\sum x, y \mid 0 \le x \le x + y \le n \bullet 1) \\
= & \langle \text{Switching to } \mathbb{N}\text{, and 0 is the least natural number} \rangle \\
& (\sum x, y : \mathbb{N} \mid x + y \le n \bullet 1)
\end{array}
$$

Counting Integral Points

How many integral points are in the triangle

$$\begin{matrix} (0,n) & & \\ | & \searrow & \\ (0,0) & — & (n,0) \end{matrix}$$

?

$(\sum x, y : \mathbb{N} \mid x + y \leq n \bullet 1)$

How many integral points are in the circle of radius n around $(0,0)$?

$(\sum x, y : \mathbb{Z} \mid x \cdot x + y \cdot y \leq n \cdot n \bullet 1)$

Sum Quantification Examples

$(\sum k : \mathbb{N} \mid k < 5 \bullet k)$

- "The sum of all natural numbers less than five"

$(\sum k : \mathbb{N} \mid k < 5 \bullet k \cdot k)$

- "For all natural numbers k that are less than 5, adding up the value of $k \cdot k$"
- "The sum of all squares of natural numbers less than five"

$(\sum x, y : \mathbb{N} \mid x \cdot y = 120 \bullet 2 \cdot (x + y))$

- "For all natural numbers x and y with product 120, adding up the value of $2 \cdot (x + y)$"
- "The sum of the perimeters of all integral rectangles with area 120"

Product Quantification Examples

- "The factorial of n is the product of all positive integers up to n"

  factorial : $\mathbb{N} \to \mathbb{N}$

  factorial n = $(\prod k : \mathbb{N} \mid 0 < k \leq n \bullet k)$

- "The product of all odd natural numbers below 50."

  $(\prod n : \mathbb{N} \mid \neg(2 \mid n) \wedge n < 50 \bullet n)$

  $(\prod k : \mathbb{N} \mid 2 \cdot k + 1 < 50 \bullet 2 \cdot k + 1)$

  $(\prod k : \mathbb{N} \mid k < 25 \bullet 2 \cdot k + 1)$

Sum and Product Quantification

$(\sum x \mid R \bullet E)$

- "For all x satisfying R, summing up the value of E"
- "The sum of all E for x with R"

$(\sum x : T \bullet E)$

- "For all x of type T, summing up the value of E"
- "The sum of all E for x of type T"

$(\prod x \mid R \bullet E)$

- "The product of all E for x with R"

$(\prod x : T \bullet E)$

- "The product of all E for x of type T"

General Shape of Sum and Product Quantifications

$$(\sum x : t_1;\ y, z : t_2 \mid R \bullet E)$$

$$(\prod x : t_1;\ y, z : t_2 \mid R \bullet E)$$

- Any number of **variables** x, y, z can be quantified over
- The quantified variables may have **type annotations** (which act as **type declarations**)
- Expression $R : \mathbb{B}$ is the **range** of the quantification
- Expression E is the **body** of the quantification
- E will have a number type ($\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$)
- Both R and E may refer to the **quantified variables** x, y, z
- The type of the whole quantification expression is the type of E.

LADM/CalcCheck Quantification Notation

Conventional sum quantification notation: $\sum_{i=1}^{n} e \quad = \quad e[i := 1] + \ldots + e[i := n]$

The textbook uses a different, but systematic **linear** notation:

$$(\sum i \mid 1 \leq i \leq n : e) \qquad \text{or} \qquad (+ i \mid 1 \leq i \leq n : e)$$

We use a variant with a "spot" "•" instead of the colon ":" and only use "big" operators:

$(\sum i \mid 1 \leq i \leq n \bullet e)$ — `\sum` `\with` `\spot`

Reasons for using this kind of <u>**linear**</u> quantification notation:

- Clearly delimited introduction of **quantified variables** (**dummies**)
- **Arbitrary** Boolean expressions can define the **range**
  $(\sum i \mid 1 \leq i \leq 7 \wedge \mathit{even}\ i \bullet i) = 2 + 4 + 6$
- The notation extends easily to multiple quantified variables:
  $(\sum i, j : \mathbb{Z} \mid 1 \leq i < j \leq 4 \bullet i/j) = 1/2 + 1/3 + 1/4 + 2/3 + 2/4 + 3/4$

Meaning of Sum Quantification

Let i be a variable list, R a Boolean expression, and E an expression of a number type.

> The **meaning** of $(\sum i \mid R \bullet E)$ in state s is:
> - the sum of the meanings of E
> - in all those states that satisfy R
> - and are different from s at most in variables in i.

Examples:

- $(\sum i, j \mid i = j = i + 1 \bullet i \cdot j) \quad = \quad 0$
- $(\sum i, j \mid 0 < i < j < 4 \bullet i \cdot j) \quad = \quad 1 \cdot 2 + 1 \cdot 3 + 2 \cdot 3$
- $(\sum i, j \mid 1 \leq i \leq 2 \wedge 3 \leq j \leq 4 \bullet i \cdot j) \quad = \quad 1 \cdot 3 + 1 \cdot 4 + 2 \cdot 3 + 2 \cdot 4$
- In state $[(i,7), (j,11), (k,3)]$, we have:
  $(\sum i, j \mid 0 < i < j < k \bullet i \cdot j) \quad = \quad 1 \cdot 2$

Bound / Free Variable Occurrences

$(\sum i : \mathbb{N} \mid i < x \bullet i + 1) = 10$ — example expression

Is this true or false? In which states?

We have: $(\sum i : \mathbb{N} \mid i < x \bullet i + 1) = 10 \quad \equiv \quad x = 4$

The value of this example expression in a state depends only on x, not on i!

Renaming quantified variables <u>does not change the meaning</u>:

$$(\sum i : \mathbb{N} \mid i < x \bullet i + 1) \quad = \quad (\sum j : \mathbb{N} \mid j < x \bullet j + 1)$$

- **Occurrences** of quantified variables inside the quantified expression are **bound**
- Non-bound **variable occurences** are called **free**
- Variables of the same name may occur both free and bound in the same expression, e.g.: $3 \cdot i + (\sum i : \mathbb{N} \mid i < x \bullet 2 \cdot i)$
- The variable declarations after the quantification operator may be called **binding occurrences**.

Variable Binding is Everywhere! Including in Substitution!

Another example expression: $(x + 3 = 5 \cdot i)[i := 9]$

Is this true or false? In which states?

$(x + 3 = 5 \cdot i)[i := 9]$
$\equiv \langle$ Substitution, … $\rangle$
$x = 42$

The value of $(x + 3 = 5 \cdot i)[i := 9]$ in a state depends only on x, not on i!

Renaming substituted variables does not change the meaning:

$$(x + 3 = 5 \cdot i)[i := 9] \quad \equiv \quad (x + 3 = 5 \cdot j)[j := 9]$$

- **Occurrences** of substituted variables inside the target expression are **bound**
- The variable occurrences to the left of := in substitutions may be called **binding occurrences**.
- Non-bound **variable occurences** are called **free**.
  $i > 0 \wedge (x + 3 = 5 \cdot i)[i := 7 + i]$
- **Substitution does not bind to the right of := !**

Expanding Sum and Product Quantification

Sum quantification ($\sum$) is **"addition (+) of arbitrarily many terms"**:

$(\sum i \mid 5 \leq i < 9 \bullet i \cdot (i + 1))$
$= \langle$ Quantification expansion $\rangle$
$(i \cdot (i + 1))[i := 5] + (i \cdot (i + 1))[i := 6] + (i \cdot (i + 1))[i := 7] + (i \cdot (i + 1))[i := 8]$
$= \langle$ Substitution $\rangle$
$5 \cdot (5 + 1) + 6 \cdot (6 + 1) + 7 \cdot (7 + 1) + 8 \cdot (8 + 1)$

Product quantification ($\prod$) is **"multiplication ($\cdot$) of arbitrarily many factors"**:

$(\prod i \mid 0 \leq i < 3 \bullet 5 \cdot i + 1)$
$= \langle$ Quantification expansion $\rangle$
$(5 \cdot i + 1)[i := 0] \cdot (5 \cdot i + 1)[i := 1] \cdot (5 \cdot i + 1)[i := 2]$
$= \langle$ Substitution $\rangle$
$(5 \cdot 0 + 1) \cdot (5 \cdot 1 + 1) \cdot (5 \cdot 2 + 1)$

Quantification Examples

$(\sum i \mid 0 \le i < 4 \bullet i \cdot 8)$
$= \langle$ Quantification expansion, substitution $\rangle$
$0 \cdot 8 + 1 \cdot 8 + 2 \cdot 8 + 3 \cdot 8$

$(\prod i \mid 0 \le i < 3 \bullet i + (i+1))$
$= \langle$ Quantification expansion, substitution $\rangle$
$(0+1) \cdot (1+2) \cdot (2+3)$

$(\forall i \mid 1 \le i < 3 \bullet i \cdot d \ne 6)$
$= \langle$ Quantification expansion, substitution $\rangle$
$1 \cdot d \ne 6 \wedge 2 \cdot d \ne 6$

$(\exists i \mid 0 \le i < 6 \bullet b\ i = 0)$
$= \langle$ Quantification expansion, substitution $\rangle$
$b\ 0 = 0 \vee b\ 1 = 0 \vee b\ 2 = 0 \vee b\ 3 = 0 \vee b\ 4 = 0 \vee b\ 5 = 0$

General Quantification

It works not only for $+, \wedge, \vee \ldots$

Let a type T and an operator $\star : T \times T \to T$ be given.
If for an appropriate $u : T$ we have:

- **Symmetry:** $b \star c = c \star b$
- **Associativity:** $(b \star c) \star d = b \star (c \star d)$
- **Identity u:** $u \star b = b = b \star u$

we may use $\star$ as quantification operator:

$$(\star\, x : T_1, y : T_2 \mid R \bullet E)$$

- $R : \mathbb{B}$ is the **range** of the quantification
- $E : T$ is the **body** of the quantification
- E and R may refer to the **quantified variables** x and y
- The type of the whole quantification expression is T.

General Quantification: Instances

Let a type T and an operator $\star : T \times T \to T$ be given.
If for an appropriate $u : T$ we have:

- **Symmetry:** $b \star c = c \star b$
- **Associativity:** $(b \star c) \star d = b \star (c \star d)$
- **Identity u:** $u \star b = b = b \star u$

we may use $\star$ as quantification operator: $(\star\, x : T_1, y : T_2 \mid R \bullet E)$

- $\_\vee\_ : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$ is symmetric (3.24), associative (3.25), and has *false* as identity (3.30) — the "big operator" for $\vee$ is $\exists$":
  $(\exists k : \mathbb{N} \mid k > 0 \bullet k \cdot k < k + 1)$
- $\_\wedge\_ : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$ is symmetric (3.36), associative (3.27), and has *true* as identity (3.39) — the "big operator" for $\wedge$ is $\forall$":
  $(\forall k : \mathbb{N} \mid k > 2 \bullet prime\ k \Rightarrow \neg\, prime\ (k+1))$
- $\_+\_ : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ is symmetric (15.2), associative (15.1), and has 0 as identity (15.3) — the "big operator" for + is $\sum$":
  $(\sum n : \mathbb{Z} \mid 0 < n < 100 \wedge prime\ n \bullet n \cdot n)$

Meaning of General Quantification

Let a type T, and a symmetric and associative operator $\star : T \times T \to T$ with identity $u : T$ be given.
Further let x be a **variable list**, R a Boolean expression, and E an expression of type T.

> The **meaning** of $(\star\, x \mid R \bullet E)$ in state s is:
> - the nested application of $\star$ to the meanings of E
> - in all those states that satisfy R
> - and are different from s at most in variables in x,
>
> or u, if there are no such states.

Examples:

- $(\exists i, j \mid i = j = i + 1 \bullet i < j) \quad = \quad$ *false*
- $(\forall i, j \mid i = j = i + 1 \bullet i < j) \quad = \quad$ *true*
- $(\prod i, j \mid i = j = i + 1 \bullet i \cdot j) \quad = \quad 1$
- $(\exists i, j \mid 0 < i \le j < 3 \bullet i \ge j) \quad = \quad 1 \ge 1 \vee 1 \ge 2 \vee 2 \ge 2$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-18

Part 2: Propositional Calculus: Implication ⇒

Implication

(3.57) **Axiom, Definition of Implication, Definition of ⇒ from ∨:** $\boxed{p \Rightarrow q \equiv p \vee q \equiv q}$

(3.58) **Axiom, Consequence:** $\boxed{p \Leftarrow q \equiv q \Rightarrow p}$

Rewriting Implication:

(3.59) (Alternative) **Definition of Implication, Material implication:** $p \Rightarrow q \equiv \neg p \vee q$

(3.60) (Dual) **Definition of Implication, Definition of ⇒ from ∧:** $p \Rightarrow q \equiv p \wedge q \equiv p$

(3.61) **Contrapositive:** $p \Rightarrow q \equiv \neg q \Rightarrow \neg p$

All Propositional Axioms of the Equational Logic E

1. **(3.1) Axiom, Associativity of ≡**
2. **(3.2) Axiom, Symmetry of ≡**
3. **(3.3) Axiom, Identity of ≡**
4. **(3.8) Axiom, Definition of *false***
5. **(3.9) Axiom, Commutativity of ¬ with ≡**
6. **(3.10) Axiom, Definition of ≢**
7. **(3.24) Axiom, Symmetry of ∨**
8. **(3.25) Axiom, Associativity of ∨**
9. **(3.26) Axiom, Idempotency of ∨**
10. **(3.27) Axiom, Distributivity of ∨ over ≡**
11. **(3.28) Axiom, Excluded Middle**
12. **(3.35) Axiom, Golden rule**
13. **(3.57) Axiom, Definition of Implication**
14. **(3.58) Axiom, Definition of Consequence**

The "Golden Rule" and Implication

(3.35) **Axiom, Golden rule:** $\boxed{p \wedge q \equiv p \equiv q \equiv p \vee q}$

Can be used as:

- $p \wedge q = (p \equiv q \equiv p \vee q)$
- $(p \equiv q) = (p \wedge q \equiv p \vee q)$
- …
- $(p \wedge q \equiv p) \equiv (q \equiv p \vee q)$

(3.57) **Axiom, Definition of Implication:** $p \Rightarrow q \equiv p \vee q \equiv q$

(3.60) (Dual) **Definition of Implication:** $p \Rightarrow q \equiv p \wedge q \equiv p$

Some Implication Theorems

(3.62) $p \Rightarrow (q \equiv r) \equiv p \wedge q \equiv p \wedge r$

(3.63) **Distributivity of ⇒ over ≡:** $p \Rightarrow (q \equiv r) \equiv p \Rightarrow q \equiv p \Rightarrow r$

(3.64) **Self-distributivity of ⇒:** $p \Rightarrow (q \Rightarrow r) \equiv (p \Rightarrow q) \Rightarrow (p \Rightarrow r)$

(3.65) **Shunting:** $p \wedge q \Rightarrow r \equiv p \Rightarrow (q \Rightarrow r)$

How do start to prove the following? (For example, …)

(3.66) $p \wedge (p \Rightarrow q) \equiv p \wedge q \qquad \langle \ldots\ p \wedge q \equiv p \rangle$

(3.67) $p \wedge (q \Rightarrow p) \equiv p \qquad \langle \ldots\ p \wedge q \equiv p \rangle$

(3.68) $p \vee (p \Rightarrow q) \equiv true \qquad \langle \ldots\ \neg p \vee q \rangle$

(3.69) $p \vee (q \Rightarrow p) \equiv q \Rightarrow p \qquad \langle \ldots\ p \vee q \equiv q \rangle$

(3.70) $p \vee q \Rightarrow p \wedge q \equiv p \equiv q \qquad \langle \ldots$ Golden Rule $\ldots \rangle$

Additional Important Implication Theorems

(3.71) **Reflexivity of ⇒:** $p \Rightarrow p \equiv true$

(3.72) **Right-zero of ⇒:** $p \Rightarrow true \equiv true$

(3.73) **Left-identity of ⇒:** $true \Rightarrow p \equiv p$

(3.74) **Definition of ¬ from ⇒:** $p \Rightarrow false \equiv \neg p$

(3.15) Definition of ¬ from ≡: $\neg p \equiv p \equiv false$

(3.75) ***ex falso quodlibet*:** $false \Rightarrow p \equiv true$

(3.65) **Shunting:** $p \wedge q \Rightarrow r \equiv p \Rightarrow (q \Rightarrow r)$

(3.77) **Modus ponens:** $p \wedge (p \Rightarrow q) \Rightarrow q$

(3.78) **Case analysis:** $(p \Rightarrow r) \wedge (q \Rightarrow r) \equiv (p \vee q \Rightarrow r)$

(3.79) **Case analysis:** $(p \Rightarrow r) \wedge (\neg p \Rightarrow r) \equiv r$

Weakening/Strengthening Theorems

"$p \Rightarrow q$" can be read "p is stronger-than-or-equivalent-to q"

"$p \Rightarrow q$" can be read "p is at least as strong as q"

(3.76a) $p \quad \Rightarrow p \vee q$

(3.76b) $p \wedge q \quad \Rightarrow p$

(3.76c) $p \wedge q \quad \Rightarrow p \vee q$

(3.76d) $p \vee (q \wedge r) \quad \Rightarrow p \vee q$

(3.76e) $p \wedge q \quad \Rightarrow p \wedge (q \vee r)$

Implication as Order on Propositions

"$p \Rightarrow q$" can be read "p is stronger-than-or-equivalent-to q"

— similar to "$x \leq y$" as "x is less-or-equal y"
— similar to "$x \geq y$" as "x is greater-or-equal y"

"$p \Rightarrow q$" can be read "p is at least as strong as q"

— similar to "$x \leq y$" as "x is at most y"
— similar to "$x \geq y$" as "x is at least y"

(3.57) **Axiom, Definition of** $\Rightarrow$ from disjunction: $p \Rightarrow q \;\equiv\; p \vee q \;\equiv\; q$
— defines the order from maximum: $p \Rightarrow q \;\equiv\; ((p \vee q) \;=\; q)$
— analogous to: $x \leq y \;\equiv\; ((x \uparrow y) \;=\; y)$
— analogous to: $k \mid n \;\equiv\; ((lcm(k,n) \;=\; n)$

(3.60) (Dual) **Definition of** $\Rightarrow$ from conjunction: $p \Rightarrow q \;\equiv\; p \wedge q \equiv p$
— defines the order from minimum: $p \Rightarrow q \;\equiv\; ((p \wedge q) \;=\; p)$
— analogous to: $x \leq y \;\equiv\; ((x \downarrow y) \;=\; x)$
— analogous to: $k \mid n \;\equiv\; ((gcd(k,n) \;=\; k)$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-20

Implication as Order, Replacement, Monotonicity

Plan for Today

- **Continuing Propositional Calculus (LADM Chapter 3)**
  - Implication as order, order relations
  - Leibniz as axiom, and "Replacement" theorems
- Transitivity Calculations, Monotonicity
- (Coming up: LADM chapter 4, and then chapters 8 and 9.)

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-20

Part 1: Implication as Order, Order Relations

Recall: Weakening/Strengthening Theorems

"$p \Rightarrow q$" can be read "p is stronger-than-or-equivalent-to q"

"$p \Rightarrow q$" can be read "p is at least as strong as q"

(3.76a) $p \quad \Rightarrow p \vee q$

(3.76b) $p \wedge q \quad \Rightarrow p$

(3.76c) $p \wedge q \quad \Rightarrow p \vee q$

(3.76d) $p \vee (q \wedge r) \quad \Rightarrow p \vee q$

(3.76e) $p \wedge q \quad \Rightarrow p \wedge (q \vee r)$

Implication as Order on Propositions

"$p \Rightarrow q$" can be read "p is stronger-than-or-equivalent-to q"

— similar to "$x \leq y$" as "x is less-or-equal y"
— similar to "$x \geq y$" as "x is greater-or-equal y"

"$p \Rightarrow q$" can be read "p is at least as strong as q"

— similar to "$x \leq y$" as "x is at most y"
— similar to "$x \geq y$" as "x is at least y"

(3.57) **Axiom, Definition of** $\Rightarrow$ from disjunction: $p \Rightarrow q \;\equiv\; p \vee q \;\equiv\; q$
— defines the order from maximum: $p \Rightarrow q \;\equiv\; ((p \vee q) \;=\; q)$
— analogous to: $x \leq y \;\equiv\; ((x \uparrow y) \;=\; y)$
— analogous to: $k \mid n \;\equiv\; ((lcm(k,n) \;=\; n)$

(3.60) (Dual) **Definition of** $\Rightarrow$ from conjunction: $p \Rightarrow q \;\equiv\; p \wedge q \equiv p$
— defines the order from minimum: $p \Rightarrow q \;\equiv\; ((p \wedge q) \;=\; p)$
— analogous to: $x \leq y \;\equiv\; ((x \downarrow y) \;=\; x)$
— analogous to: $k \mid n \;\equiv\; ((gcd(k,n) \;=\; k)$

One View of Relations

- Let T_1 and T_2 be two types.
- A function of type $T_1 \to T_2 \to \mathbb{B}$ can be considered as *one view of* a **relation from T_1 to T_2**
  - We will see a different view of relations later …
  - … and **the** way to switch between these views.
  - With such a way of switching, the two views "are the same" in colloquial mathematics
  - Therefore we will occasionally just use the term "relation" also for functions of types $T_1 \to T_2 \to \mathbb{B}$
- A function of type $T \to T \to \mathbb{B}$ may then be called **a relation on T**.
- Some relations you are familar with:
  $\_=\_ : T \to T \to \mathbb{B}$
  $\_=\_ : \mathbb{Z} \to \mathbb{Z} \to \mathbb{B}$
  $\_=\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{B}$
  $\_\leq\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{B}$
  $\_\equiv\_ : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$
  $\_\Rightarrow\_ : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$

Order Relations

- Let T be a type.
- A relation $\_\leq\_$ on T is called:
  - **reflexive** iff $x \leq x$ is valid
  - **transitive** iff $x \leq y \;\wedge\; y \leq z \;\Rightarrow\; x \leq z$ is valid
  - **antisymmetric** iff $x \leq y \;\wedge\; y \leq x \;\Rightarrow\; x = y$ is valid
  - an **order** (or **ordering**) iff it is reflexive, transitive, and antisymmetric
- Orders you are familiar with:
  $\_=\_ : T \to T \to \mathbb{B}$
  $\_\leq\_ : \mathbb{Z} \to \mathbb{Z} \to \mathbb{B}$
  $\_\geq\_ : \mathbb{Z} \to \mathbb{Z} \to \mathbb{B}$
  $\_\leq\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{B}$
  $\_\geq\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{B}$
  $\_|\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{B}$
  $\_\equiv\_ : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$
  $\_\Rightarrow\_ : \mathbb{B} \to \mathbb{B} \to \mathbb{B}$
  $\_\subseteq\_ :$ **set** $T \to$ **set** $T \to \mathbb{B}$

Order Properties of Implication in LADM Chapter 3

(3.71) **Reflexivity of $\Rightarrow$:** $p \Rightarrow p$

(3.80b) **Reflexivity wrt. Equivalence:** $(p \equiv q) \Rightarrow (p \Rightarrow q)$

(3.80) **Mutual implication:** $(p \Rightarrow q) \wedge (q \Rightarrow p) \;\equiv\; p \equiv q$

(3.81) **Antisymmetry:** $(p \Rightarrow q) \wedge (q \Rightarrow p) \;\Rightarrow\; (p \equiv q)$

(3.82a) **Transitivity:** $(p \Rightarrow q) \wedge (q \Rightarrow r) \;\Rightarrow\; (p \Rightarrow r)$

(3.82b) **Transitivity:** $(p \equiv q) \wedge (q \Rightarrow r) \;\Rightarrow\; (p \Rightarrow r)$

(3.82c) **Transitivity:** $(p \Rightarrow q) \wedge (q \equiv r) \;\Rightarrow\; (p \Rightarrow r)$

Some Order-Related Concepts

An order $\_\leq\_$ on T may have (or may not have):

- a **least element** denoted b: A constant b such that $b \leq x$ is valid
  - $\_\leq\_ : \mathbb{Z} \to \mathbb{Z} \to \mathbb{B}$ has no least element
  - $\_\leq\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{B}$ has least element 0
  - $\_\geq\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{B}$ has no least element
  - $\_|\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{B}$ has least element 1
- a **greatest element** denoted t: A constant t such that $x \leq t$ is valid
  - $\_\leq\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{B}$ has no greatest element
  - $\_\geq\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{B}$ has greatest element 0
  - $\_|\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{B}$ has greatest element 0
- have **binary maxima**: An operation $\_\sqcup\_$ such that $x \sqcup y$ is the least element that is at least x and also at least y
- have **binary minima**: An operation $\_\sqcap\_$ such that $x \sqcap y$ is the greatest element that is at most x and also at most y

Monotonicity, Isotonicity, Antitonicity

- Let $\_\leq\_$ be an order on T
- Let $f : T \to T$ be a function on T
- Then f is called
  - **monotonic** iff $x \leq y \;\Rightarrow\; f\,x \leq f\,y$ is a theorem
  - **isotonic** iff $x \leq y \;\equiv\; f\,x \leq f\,y$ is a theorem
  - **antitonic** iff $x \leq y \;\Rightarrow\; f\,y \leq f\,x$ is a theorem
- Examples:
  - $suc\,\_ : \mathbb{N} \to \mathbb{N}$ is isotonic
  - $pred : \mathbb{N} \to \mathbb{N}$ is monotonic, but not isotonic
  - $\_+\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ is isotonic in the first argument:
    $x \leq y \;\equiv\; x + z \leq y + z$ is a theorem
  - $\_+\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ is isotonic in the second argument:
    $x \leq y \;\equiv\; z + x \leq z + y$ is a theorem
  - $\_-\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ is **monotonic in the first argument**:
    $x \leq y \;\Rightarrow\; x - z \leq y - z$ is a theorem
  - $\_-\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ is **antitonic in the second argument**:
    $x \leq y \;\Rightarrow\; z - y \leq z - x$ is a theorem

Monotonicity and Antitonicity Theorems for ⇒

(4.2) **Left-Monotonicity of ∨:** $(p \Rightarrow q) \;\Rightarrow\; (p \vee r \Rightarrow q \vee r)$

(4.3) **Left-Monotonicity of ∧:** $(p \Rightarrow q) \;\Rightarrow\; p \wedge r \;\Rightarrow\; q \wedge r$

— We'll be getting to LADM chapter 4 on Wednesday.

— But you can prove these already in the context of chapter 3!

Tutorials and Exercise Notebooks

- Doing the Homework (yourself) is **necessary** — **but not sufficient!**
- **The Exercise notebooks have content that you are expected to know as well!**
- Some of that content may be new to you. . . (e.g., Ex3.3, Ex3.4. . .)
- The tutorials will explain that content, and help you tackle related problems.
- Exercise 3.1 (Implication) builds on Ex2.5–2.7 (Equiv., Neg., Disjunction, Conjunction).

  Questions in this direction will be on Midterm 1.

  You are expected to know the theorems you will need to use, and to know also the names of these theorems.

  You will need practice using these theorems. **If you haven't started yet: Start now!**

  Best practice: **Produce different proofs for the theorems in Ex2.7 and Ex3.1.**

  Without that practice, Midterm 1 will probably be infeasible for you.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-20

Part 2: Leibniz as Axiom, Replacement Theorems

Leibniz's Rule as an Axiom

Recall the **inference rule** (scheme):

(1.5) **Leibniz:** $\dfrac{X = Y}{E[z := X] = E[z := Y]}$

Axiom scheme (E can be any expression, and z any variable):

(3.83) **Axiom, Leibniz:** $(e = f) \Rightarrow (E[z := e] = E[z := f])$

What is the difference?

- Given a theorem $X = Y$ and an expression E,
  the inference rule (1.5) **produces** a new theorem $E[z := X] = E[z := Y]$
- (3.83) **is** a theorem
- $((e = f) \Rightarrow (E[z := e] = E[z := f])) \quad = \quad true$

  Can be used **deep inside nested expressions**

  — making use of **local assumptions (that are typically not theorems)**

Leibniz's Rule as an Axiom — Examples

Recall the **inference rule** (scheme):

(1.5) **Leibniz:** $\dfrac{X = Y}{E[z := X] = E[z := Y]}$

Axiom scheme (E can be any expression, and z any variable):

(3.83) **Axiom, Leibniz:** $(e = f) \Rightarrow (E[z := e] = E[z := f])$

Examples

- $n = k + 1 \Rightarrow n \cdot (k - 1) = (k + 1) \cdot (k - 1)$
- $n = k + 1 \Rightarrow (z \cdot (k - 1))[z := n] = (z \cdot (k - 1))[z := k + 1]$
- $(n = k + 1 \Rightarrow n \cdot (k - 1) = k^2 - 1) = true$

  $\Rightarrow\quad (n > 5 \Rightarrow (n = k + 1 \Rightarrow n \cdot (k - 1) = k^2 - 1))$

  $= (n > 5 \Rightarrow true)$

Leibniz's Rule Axiom, and Further Replacement Rules

Axiom scheme (E can be any expression; $z, e, f : t$ can be of **any type** t):

(3.83) **Axiom, Leibniz:** $(e = f) \;\Rightarrow\; (E[z := e] = E[z := f])$

— Axiom (3.83) is rarely useful directly!

— Allmost all applications are via derived **"Replacement"** theorems

Replacement rules: (P can be any expression **of type** $\mathbb{B}$)

(3.84a) **"Replacement":** $(e = f) \wedge P[z := e] \;\equiv\; (e = f) \wedge P[z := f]$

(3.84b) **"Replacement":** $(e = f) \Rightarrow P[z := e] \;\equiv\; (e = f) \Rightarrow P[z := f]$

(3.84c) **"Replacement":** $q \wedge (e = f) \Rightarrow P[z := e] \;\equiv\; q \wedge (e = f) \Rightarrow P[z := f]$

Using a Replacement (LADM: "Substitution") Rule

Replacement rule: (P can be any expression **of type** $\mathbb{B}$)

(3.84a) **"Replacement":** $(e = f) \wedge P[z := e] \;\equiv\; (e = f) \wedge P[z := f]$

Textbook-style application:

$k = n + 1 \;\wedge\; k \cdot (n - 1) = n^2 - 1$

$= \langle$ (3.84a) **"Replacement"** $\rangle$

$k = n + 1 \;\wedge\; (n + 1) \cdot (n - 1) = n^2 - 1$

Not so fast! — CalcCheck cannot do second-order matching (yet):

$k = n + 1 \;\wedge\; k \cdot (n - 1) = n \cdot n - 1$

$= \langle$ Substitution $\rangle$

$k = n + 1 \;\wedge\; (z \cdot (n - 1) = n \cdot n - 1)[z := k]$

$= \langle$ (3.84a) **"Replacement"** $\rangle$

$k = n + 1 \;\wedge\; (z \cdot (n - 1) = n \cdot n - 1)[z := n + 1]$

$= \langle$ Substitution $\rangle$

$k = n + 1 \;\wedge\; (n + 1) \cdot (n - 1) = n \cdot n - 1$

Some Replacements

$((x > f\,5) \equiv (y < g\,7)) \;\wedge\; ((f\,x \leq g\,y) \equiv (x > f\,5))$

$\equiv \langle$ **?** $\rangle$

$((x > f\,5) \equiv (y < g\,7)) \;\wedge\; ((f\,x \leq g\,y) \equiv (y < g\,7))$

$((f\,5) = (g\,y)) \;\wedge\; ((f\,x \leq g\,y) \equiv x > (f\,5))$

$\equiv \langle$ **?** $\rangle$

$((f\,5) = (g\,y)) \;\wedge\; ((f\,x \leq g\,y) \equiv x > g\,y))$

$((x > f\,5) \equiv (y < g\,7)) \;\wedge\; ((f\,x \leq g\,y) \Rightarrow p(x - 1) \vee (x > f\,5))$

$\equiv \langle$ **?** $\rangle$

$((x > f\,5) \equiv (y < g\,7)) \;\wedge\; ((f\,x \leq g\,y) \Rightarrow p(x - 1) \vee (y < g\,7))$

Replacements 1 & 2

$((x >f\ 5) \equiv (y <g\ 7)) \quad\wedge\quad ((f\ x \le g\ y) \equiv (x >f\ 5))$

$\equiv$ ⟨ (3.51) **"Replacement"** $(p \equiv q) \wedge (r \equiv p) \equiv (p \equiv q) \wedge (r \equiv q)$ ⟩

$((x >f\ 5) \equiv (y <g\ 7)) \quad\wedge\quad ((f\ x \le g\ y) \equiv (y <g\ 7))$

$((f\ 5) = (g\ y)) \quad\wedge\quad ((f\ x \le g\ y) \equiv x > (f\ 5))$

$\equiv$ ⟨ Substitution ⟩

$((f\ 5) = (g\ y)) \quad\wedge\quad ((f\ x \le g\ y) \equiv x > z)[z := (f\ 5)]$

$\equiv$ ⟨ (3.84a) **"Replacement"** $(e = f) \wedge P[z := e] \equiv (e = f) \wedge P[z := f]$, Substitution ⟩

$((f\ 5) = (g\ y)) \quad\wedge\quad ((f\ x \le g\ y) \equiv x > g\ y))$

Replacement 3

$((x >f\ 5) \equiv (y <g\ 7)) \quad\wedge\quad ((f\ x \le g\ y) \Rightarrow p(x-1) \vee (x >f\ 5))$

$\equiv$ ⟨ Substitution ⟩

$((x >f\ 5) \equiv (y <g\ 7)) \wedge ((f\ x \le g\ y) \Rightarrow p(x-1) \vee z)[z := (x >f\ 5)]$

$\equiv$ ⟨ (3.84a) **"Replacement"** $(e = f) \wedge P[z := e] \equiv (e = f) \wedge P[z := f]$, **"Definition of $\equiv$"** $(p \equiv q) = (p = q)$, Substitution ⟩

$((x >f\ 5) \equiv (y <g\ 7)) \quad\wedge\quad ((f\ x \le g\ y) \Rightarrow p(x-1) \vee (y <g\ 7))$

In CalcCheck, $\equiv$ does not match $=$!

Explicit conversions using "Definition of $\equiv$" are necessary.

Replacing Variables by Boolean Constants

In each of the following, P can be any expression **of type $\mathbb{B}$**:

(3.85a) **Replace by *true*:** $p \Rightarrow P[z := p] \equiv p \Rightarrow P[z := true]$

(3.85b) $q \wedge p \Rightarrow P[z := p] \equiv q \wedge p \Rightarrow P[z := true]$

(3.86a) **Replace by *false*:** $P[z := p] \Rightarrow p \equiv P[z := false] \Rightarrow p$

(3.86b) $P[z := p] \Rightarrow p \vee q \equiv P[z := false] \Rightarrow p \vee q$

(3.87) **Replace by *true*:** $p \wedge P[z := p] \equiv p \wedge P[z := true]$

(3.88) **Replace by *false*:** $p \vee P[z := p] \equiv p \vee P[z := false]$

(3.89) **Shannon:** $P[z := p] \equiv (p \wedge P[z := true]) \vee (\neg p \wedge P[z := false])$

Note: Using Shannon on all propositional variables in sequence is equivalent to producing a truth table.

"Prove the following theorems (**without using Shannon or the proof method of case analysis by Shannon**), ..."

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-20

Part 3: Transitivity Calculations, Monotonicity

?

$7 \cdot 8$

$=$ ⟨ Evaluation ⟩

$(10-3) \cdot (12-4)$

$\le$ ⟨ Fact: $3 \le 4$ ⟩

$(10-4) \cdot (12-4)$

$\le$ ⟨ Fact: $4 \le 5$ ⟩

$(10-4) \cdot (12-5)$

$=$ ⟨ Evaluation ⟩

$6 \cdot 7$

$=$ ⟨ Evaluation ⟩

42

This proves: $7 \cdot 8 \le 42$

Recall: Calculational Proof Format

E_0

$=$ ⟨ Explanation of why $E_0 = E_1$ ⟩

E_1

$=$ ⟨ Explanation of why $E_1 = E_2$ — with comment ⟩

E_2

$=$ ⟨ Explanation of why $E_2 = E_3$ ⟩

E_3

Because the **calculational presentation** is **conjunctional**, this reads as:

$$E_0 = E_1 \quad\wedge\quad E_1 = E_2 \quad\wedge\quad E_2 = E_3$$

Because $=$ is **transitive**, this justifies:

$$E_0 = E_3$$

Extended Calculational Proof Format (1)

E_0

$\le$ ⟨ Explanation of why $E_0 \le E_1$ ⟩

E_1

$\le$ ⟨ Explanation of why $E_1 \le E_2$ — with comment ⟩

E_2

$\le$ ⟨ Explanation of why $E_2 \le E_3$ ⟩

E_3

Because the **calculational presentation** is **conjunctional**, this reads as:

$$E_0 \le E_1 \quad\wedge\quad E_1 \le E_2 \quad\wedge\quad E_2 \le E_3$$

Because $\le$ is **transitive**, this justifies:

$$E_0 \le E_3$$

Extended Calculational Proof Format (2)

E_0

$\le$ ⟨ Explanation of why $E_0 \le E_1$ ⟩

E_1

$=$ ⟨ Explanation of why $E_1 = E_2$ — with comment ⟩

E_2

$\le$ ⟨ Explanation of why $E_2 \le E_3$ ⟩

E_3

Because the **calculational presentation** is **conjunctional**, this reads as:

$$E_0 \le E_1 \quad\wedge\quad E_1 = E_2 \quad\wedge\quad E_2 \le E_3$$

Because $\le$ is **reflexive and transitive**, this justifies:

$$E_0 \le E_3$$

Extended Calculational Proof Format (3)

E_0

$\Rightarrow$ ⟨ Explanation of why $E_0 \Rightarrow E_1$ ⟩

E_1

$\equiv$ ⟨ Explanation of why $E_1 \equiv E_2$ — with comment ⟩

E_2

$\Rightarrow$ ⟨ Explanation of why $E_2 \Rightarrow E_3$ ⟩

E_3

Because the **calculational presentation** is **conjunctional**, this reads as:

$$(E_0 \Rightarrow E_1) \quad\wedge\quad (E_1 \equiv E_2) \quad\wedge\quad (E_2 \Rightarrow E_3)$$

Because $\Rightarrow$ is **reflexive and transitive**, this justifies:

$$E_0 \Rightarrow E_3$$

Extended Calculational Proof Format (4)

E_0

$\le$ ⟨ Explanation of why $E_0 \le E_1$ ⟩

E_1

$=$ ⟨ Explanation of why $E_1 = E_2$ — with comment ⟩

E_2

$<$ ⟨ Explanation of why $E_2 < E_3$ ⟩

E_3

Because the **calculational presentation** is **conjunctional**, this reads as:

$$E_0 \le E_1 \quad\wedge\quad E_1 = E_2 \quad\wedge\quad E_2 < E_3$$

Because $<$ is **transitive**, and because $\le$ is the reflexive closure of $<$, this justifies:

$$E_0 < E_3$$

Calculational Non-Proofs

E_0
$\leq$ ⟨ Explanation of why $E_0 \leq E_1$ ⟩
E_1
$=$ ⟨ Explanation of why $E_1 = E_2$ — with comment ⟩
E_2
$\geq$ ⟨ Explanation of why $E_2 \geq E_3$ ⟩
E_3

Because the **calculational presentation** is **conjunctional**, this reads as:

$$E_0 \leq E_1 \quad \wedge \quad E_1 = E_2 \quad \wedge \quad E_2 \geq E_3$$

This justifies nothing about the relation between E_0 and E_3 !

Leibniz is Special to Equality

How about the following?

$x - 3$
$\leq$ ⟨ Fact: $3 \leq 4$ ⟩
$x - 4$

Remember:

(1.5) **Leibniz:** $\dfrac{X = Y}{E[z := X] = E[z := Y]}$

Leibniz is available only for equality

Example Application of "Monotonicity of –"

- $\_-\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ is **monotone in the first argument**:
  $x \leq y \quad \Rightarrow \quad x - z \leq y - z$ is a theorem

---

Theorem "Monotonicity of –": $a \leq b \Rightarrow a - c \leq b - c$

```
Calculation:
    12 - n
  ≤⟨ "Monotonicity of -" with Fact `12 ≤ 20` ⟩
    20 - n
```

This step can be justified without "with" as follows:

```
Calculation:
    12 - n  ≤  20 - n
  ≡⟨ "Left-identity of ⇒" ⟩
    true  ⇒  (12 - n  ≤  20 - n)
  ≡⟨ Fact `12 ≤ 20` ⟩
    (12 ≤ 20)  ⇒  (12 - n  ≤  20 - n)
  — This is "Monotonicity of -"
```

Modus Pones via with₂

Modus ponens theorem: (3.77) **Modus ponens:** $p \wedge (p \Rightarrow q) \Rightarrow q$

Modus ponens inference rule: $\dfrac{P \Rightarrow Q \qquad P}{Q}$ ⇒-Elim $\qquad \dfrac{f : A \to B \qquad x : A}{(f\ x) : B}$ Fct. app.
("Implication elimination" rule)

Applying implication theorems:

A proof for $P \Rightarrow Q$ can be used as a recipe for turning a proof for P into a proof for Q.

Q_1
$\sqsubseteq$ ⟨ "Theorem 1" \`$P \Rightarrow (Q_1 \sqsubseteq Q_2)$\` with "Theorem 2" \`$P$\` ⟩
Q_2

Theorem "Monotonicity of –": $a \leq b \Rightarrow a - c \leq b - c$

```
Calculation:
    12 - n
  ≤⟨ "Monotonicity of -" with Fact `12 ≤ 20` ⟩
    20 - n
```

Example Application of "Antitonicity of –"

- $\_-\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ is **antitone in the second argument**:
  $x \leq y \quad \Rightarrow \quad z - y \leq z - x$ is a theorem

---

Theorem "Antitonicity of –": $b \leq c \quad \Rightarrow \quad a - c \leq a - b$

```
Calculation:
    m - 3
  ≤⟨ "Antitonicity of -" with Fact `2 ≤ 3` ⟩
    m - 2
```

Multiplication on ℕ is Monotonic...

Calculation:
42
$=$ ⟨ Evaluation ⟩
$6 \cdot 7$
$=$ ⟨ Evaluation ⟩
$(10 - 4) \cdot (12 - 5)$
$\leq$ ⟨ "Monotonicity of ·" with
"Antitonicity of –" with Fact \`$3 \leq 4$\`
⟩
$(10 - 3) \cdot (12 - 5)$
$\leq$ ⟨ "Monotonicity of ·" with
"Antitonicity of –" with Fact \`$4 \leq 5$\`
⟩
$(10 - 3) \cdot (12 - 4)$
$=$ ⟨ Evaluation ⟩
$7 \cdot 8$

with₂ Works Also With ≡ — Example Using "Isotonicity of +"

- $\_+\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$ is isotone in the first argument:
  $x \leq y \quad \equiv \quad x + z \leq y + z$ is a theorem

---

```
Calculation:
    2 + n
  ≤⟨ "Isotonicity of +" with Fact `2 ≤ 3` ⟩
    3 + n
```

This step can be justified without "with" as follows:

```
Calculation:
    2 + n ≤ 3 + n
  ≡⟨ "Identity of ≡" ⟩
    true  ≡  2 + n ≤ 3 + n
  ≡⟨ Fact `2 ≤ 3` ⟩
    2 ≤ 3  ≡  2 + n ≤ 3 + n
    — This is "Isotonicity of +"
```

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-22

LADM Chapter 4: "Relaxing the Proof Style" — New Proof Structures

Plan for Today

- **LADM Chapter 4: "Relaxing the Proof Style"** — **New Proof Structures**
  - Transitivity calculations with implication ⇒ or consequence ⇐
  - Proving implications: **Assuming** the antecedent
  - Proving **By cases**
  - **Using** theorems as proof methods
    - Proof by Contrapositive
    - Proof by Mutual Implication
- Coming up: LADM chapters 8 and 9.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-22

Part 1: Subproofs, Abbreviated Proofs for Implications

CalcCheck: Subproof Hint Items

You have used the following kinds of hint items:

- Theorem name references **"Identity of ≡"**
- Theorem number references **(3.32)**
- Certain key words and key phrases: Substitution, Evaluation, Induction hypothesis
- Fact `` `Expression` ``
- Composed hint items: "Identity of +" with `` `Substitution` ``
  "Monotonicity of +" with *HintItem*

A new kind of hint item:

> Subproof for `` `Expression` ``:
> Proof

For example, Fact `` `3 = 2 + 1` `` is really syntactic sugar for a subproof:

```
Calculation:
    3 · x
  =( Subproof for `3 = 2 + 1`:
       By evaluation
    )
    (2 + 1) · x
```

Abbreviated Proofs for Implications

This:

p
$\equiv$ ⟨ Why $p \equiv q$ ⟩
q
$\Rightarrow$ ⟨ Why $q \Rightarrow r$ ⟩
r

proves: $p \Rightarrow r$

Because:

$(p \equiv q) \wedge (q \Rightarrow r)$
$\Rightarrow$ ⟨ (3.82b) Transitivity of $\Rightarrow$ ⟩
$p \Rightarrow r$

This proof style will not be allowed in questions "belonging" to LADM Chapter 3!

(4.1) — Creating the Proof "Bottom-up"

Proving (4.1) $p \Rightarrow (q \Rightarrow p)$**:**

p
$\Rightarrow$ ⟨ (3.76a) Weakening $p \Rightarrow p \vee q$ ⟩ ...... **Rabbit!**
$\neg q \vee p$
$\equiv$ ⟨ (3.59) Definition of implication ⟩
$q \Rightarrow p$

We have: **Axiom (3.58) Consequence**: $p \Leftarrow q \quad \equiv \quad q \Rightarrow p$

This means that the $\Leftarrow$ relation is the **converse** of the $\Rightarrow$ relation.

Theorem: The converse of a transitive relation is transitive again, and the converse of an order is an order again.

CalcCheck supports **activation** of converse properties, enabling **reversed presentations following mathematical habits** of transitivity calculations such as the above.

— "... propositional logic following LADM chapters 3 and 4 ..."

(4.1) Implicitly Using "Consequence"

Proving (4.1) $p \Rightarrow (q \Rightarrow p)$**:**

$q \Rightarrow p$
$\equiv$ ⟨ (3.59) Definition of implication ⟩
$\neg q \vee p$
$\Leftarrow$ ⟨ (3.76a) Strenghtening — used as $p \vee q \Leftarrow p$ ⟩
p

In CalcCheck, if the **converse property** is not **activated**, then $\Leftarrow$ is a separate operator requiring explicit conversion:

```
Theorem (4.1): p ⇒ (q ⇒ p)
Proof:
    q ⇒ p
  ≡⟨ "Definition of ⇒" (3.59) ⟩
    ¬ q ∨ p
  ⇐⟨ "Strengthening" (3.76a), "Definition of ⇐" ⟩
    p
```

Recall: Weakening/Strengthening Theorems

(3.76a) $p \Rightarrow p \vee q$

(3.76b) $p \wedge q \Rightarrow p$

(3.76c) $p \wedge q \Rightarrow p \vee q$

(3.76d) $p \vee (q \wedge r) \Rightarrow p \vee q$

(3.76e) $p \wedge q \Rightarrow p \wedge (q \vee r)$

(4.2) Left-Monotonicity of ∨

$(p \Rightarrow q) \Rightarrow (p \vee r \Rightarrow q \vee r)$

$p \vee r \Rightarrow q \vee r$
$\equiv$ ⟨ (3.57) Definition of $\Rightarrow$ $p \Rightarrow q \equiv p \vee q \equiv q$ ⟩
$p \vee r \vee q \vee r \quad \equiv \quad q \vee r$
$\equiv$ ⟨ (3.26) Idempotency of $\vee$ ⟩
$p \vee q \vee r \quad \equiv \quad q \vee r$
$\equiv$ ⟨ (3.27) Distributivity of $\vee$ over $\equiv$ ⟩
$(p \vee q \quad \equiv \quad q) \vee r$
$\equiv$ ⟨ (3.57) Definition of $\Rightarrow$ $p \Rightarrow q \equiv p \vee q \equiv q$ ⟩
$(p \Rightarrow q) \vee r$
$\Leftarrow$ ⟨ (3.76a) Strengthening $p \Rightarrow p \vee q$ ⟩
$p \Rightarrow q$

(4.3) Left-Monotonicity of ∧

Proving (4.3) $(p \Rightarrow q) \Rightarrow p \wedge r \Rightarrow q \wedge r$**:**

$p \wedge r \Rightarrow q \wedge r$
$\equiv$ ⟨ (3.60) Definition of $\Rightarrow$ ⟩
$p \wedge r \wedge q \wedge r \equiv p \wedge r$
$\equiv$ ⟨ (3.38) Idempotency of $\wedge$ ⟩
$(p \wedge q) \wedge r \equiv p \wedge r$
$\equiv$ ⟨ (3.49) Semi-distributivity of $\wedge$ ⟩
$((p \wedge q) \equiv p) \wedge r \equiv r$
$\equiv$ ⟨ (3.60) Definition of $\Rightarrow$ ⟩
$(p \Rightarrow q) \wedge r \equiv r$
$\equiv$ ⟨ (3.60) Definition of $\Rightarrow$ ⟩
$r \Rightarrow (p \Rightarrow q)$
$\Leftarrow$ ⟨ (4.1) $p \Rightarrow (q \Rightarrow p)$ ⟩
$p \Rightarrow q$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-22

Part 2: Assuming the Antecedent

Proving Implications...

How to prove the following?

"=-Congruence of +": $b = c \quad \Rightarrow \quad a + b = a + c$

"We have been doing this via Leibniz (1.5)....."

- One of the "Replacement" theorems of the "Leibniz as Axiom" section can help.
- It may be nicer to turn this into a situation where the inference rule Leibniz (1.5) can be used again...

Assuming the Antecedent:

```
Lemma "=-Congruence of +":  b = c  ⇒  a + b = a + c
Proof:
  Assuming `b = c`:
      a + b
    =⟨ Assumption `b = c` ⟩
      a + c
```

Assuming the Antecedent

To prove an implication $p \Rightarrow q$
we can prove its conclusion q using p as **assumption**:

> **Assuming** `` `p` ``:
> *Proof of* q
> *possibly using:* Assumption `` `p` ``

Justification:

(4.4) **(Extended) Deduction Theorem:** Suppose adding $P_1, \ldots, P_n$ as axioms to propositional logic **E**, **with the free variables of the P_i considered to be constants**, allows Q to be proved.

Then $P_1 \wedge \ldots \wedge P_n \Rightarrow Q$ is a theorem.

That is:

Assumptions **cannot** be used with substitutions (with '$a, b := e, f$')
— just like induction hypotheses.

"Assuming the Antecedent" is not allowed in questions "belonging to" LADM chapt. 3!

Inference Rule for Proving Implications: ⇒-Introduction

One way to prove $P \Rightarrow Q$:

> **Assuming** \`$P$\`:
>
> > *Proof of* Q
> > *possibly using:* Assumption \`$P$\`

(And **Assuming** \`$P$\`: … can only prove theorems of shape $P \Rightarrow \cdots$.)

This directly corresponds to an application of the inference rule "⇒-Introduction" (which is missing in the Rosen book used in COMPSCI 1DM3):

$$\frac{\begin{matrix}\ulcorner P \urcorner \\ \vdots \\ Q\end{matrix}}{P \Rightarrow Q}\ \Rightarrow\text{-Intro} \qquad \frac{\begin{matrix}\ulcorner x : A \urcorner \\ \vdots \\ e : B\end{matrix}}{(\lambda x : A \bullet e) : A \to B}\ \lambda\text{-Abstraction}$$

Proving and Using Implication Theorems: `Assuming` and `with`$_2$

"Cancellation of ·": $z \neq 0 \Rightarrow (z \cdot x = z \cdot y \quad \equiv \quad x = y)$

Theorem "Non-zero multiplication": $a \neq 0 \Rightarrow (b \neq 0 \Rightarrow a \cdot b \neq 0)$
Proof:
  Assuming \`$a \neq 0$\`, \`$b \neq 0$\`:
    $a \cdot b \neq 0$
  ≡⟨ "Definition of ≠" ⟩
    $\neg (a \cdot b = 0)$
  ≡⟨ "Zero of ·" ⟩
    $\neg (a \cdot b = a \cdot 0)$
  ≡⟨ "Cancellation of ·" with Assumption \`$a \neq 0$\` ⟩
    $\neg (b = 0)$
  ≡⟨ "Definition of ≠", Assumption \`$b \neq 0$\` ⟩
    true

- *HintItem1* `with` *HintItem2* `and` *HintItem3*, *HintItem4* parses as (*HintItem1* `with` (*HintItem2* `and` *HintItem3*)), *HintItem4*

(4.3) Left-Monotonicity of ∧ (shorter proof, LADM-style)

(4.3) $(p \Rightarrow q) \Rightarrow ((p \wedge r) \Rightarrow (q \wedge r))$

PROOF:

Assume $p \Rightarrow q$ (which is equivalent to $p \wedge q \equiv p$)

  $p \wedge r$
≡ ⟨ Assumption $p \wedge q \equiv p$ ⟩
  $p \wedge q \wedge r$
⇒ ⟨ (3.76b) Weakening ⟩
  $q \wedge r$

How to do "which is equivalent to" in CALCCHECK?

- Transform before assuming
- or transform the assumption when using it
- or "Assuming … and using with …"

Transform Before Assuming — Proof for this:

Theorem (4.3) "Left-monotonicity of ∧" "Monotonicity of ∧":
  $(p \Rightarrow q) \Rightarrow ((p \wedge r) \Rightarrow (q \wedge r))$
Proof:
    $(p \Rightarrow q) \Rightarrow ((p \wedge r) \Rightarrow (q \wedge r))$
  ≡ ⟨ "Definition of ⇒ from ∧" ⟩
    $(p \wedge q \equiv p) \Rightarrow ((p \wedge r) \Rightarrow (q \wedge r))$
  Proof for this:
    Assuming \`$p \wedge q \equiv p$\`:
      $p \wedge r$
    ≡ ⟨ Assumption \`$p \wedge q \equiv p$\` ⟩
      $p \wedge q \wedge r$
    ⇒ ⟨ "Weakening" ⟩
      $q \wedge r$

Transform Assumption When Used — `with`$_3$

(4.3) $(p \Rightarrow q) \Rightarrow ((p \wedge r) \Rightarrow (q \wedge r))$

PROOF:

Assume $p \Rightarrow q$ (which is equivalent to $p \wedge q \equiv p$)

  $p \wedge r$
≡ ⟨ Assumption $p \wedge q \equiv p$ ⟩
  $p \wedge q \wedge r$
⇒ ⟨ (3.76b) Weakening ⟩
  $q \wedge r$

```
Theorem (4.3) "Left-monotonicity of ∧": (p ⇒ q) ⇒ (p ∧ r ⇒ q ∧ r)
Proof:
  Assuming `p ⇒ q`:
      p ∧ r
    ≡⟨ Assumption `p ⇒ q` with "Definition of ⇒" (3.60) ⟩
      p ∧ q ∧ r
    ⇒⟨ "Weakening" ⟩
      q ∧ r
```

Assuming … and using with …

(4.3) $(p \Rightarrow q) \Rightarrow ((p \wedge r) \Rightarrow (q \wedge r))$

PROOF:

Assume $p \Rightarrow q$ (which is equivalent to $p \wedge q \equiv p$)

  $p \wedge r$
≡ ⟨ Assumption $p \wedge q \equiv p$ ⟩
  $p \wedge q \wedge r$
⇒ ⟨ (3.76b) Weakening ⟩
  $q \wedge r$

```
Theorem (4.3) "Left-monotonicity of ∧" "Monotonicity of ∧":
    (p ⇒ q) ⇒ ((p ∧ r) ⇒ (q ∧ r))
Proof:
  Assuming `p ⇒ q` and using with "Definition of ⇒" (3.60):
      p ∧ r
    ≡⟨ Assumption `p ⇒ q` ⟩
      p ∧ q ∧ r
    ⇒⟨ "Weakening" (3.76b) ⟩
      q ∧ r
```

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-22

Part 3: Case Analysis and Other Proof Methods

LADM General Case Analysis

(4.6) $(p \vee q \vee r) \wedge (p \Rightarrow s) \wedge (q \Rightarrow s) \wedge (r \Rightarrow s) \Rightarrow s$

Proof pattern for general case analysis:

Prove: S
  By cases: P, Q, R
    (proof of $P \vee Q \vee R$ — omitted if obvious)
  Case P **:** (proof of $P \Rightarrow S$)
  Case Q **:** (proof of $Q \Rightarrow S$)
  Case R **:** (proof of $R \Rightarrow S$)

LADM Case Analysis Example: (4.2) $(p \Rightarrow q) \Rightarrow p \vee r \Rightarrow q \vee r$

Assume $p \Rightarrow q$
 Assume $p \vee r$
  Prove: $q \vee r$
  By Cases: p, r — $p \vee r$ holds by assumption
  Case p **:**
    p
  ⇒ ⟨ Assumption $p \Rightarrow q$ ⟩
    q
  ⇒ ⟨ Weakening (3.76a) ⟩
    $q \vee r$
  Case r **:**
    r
  ⇒ ⟨ Weakening (3.76a) ⟩
    $q \vee r$

Case Analysis Example (4.2) "Left-Monotonicity of ∨" in CALCCHECK

Theorem "Monotonicity of ∨": $(p \Rightarrow q) \Rightarrow (p \vee r) \Rightarrow (q \vee r)$
Proof:
  Assuming \`$p \Rightarrow q$\`, \`$p \vee r$\`:
    By cases: \`$p$\`, \`$r$\`
      Completeness: By assumption \`$p \vee r$\`
      Case \`$p$\`:
          p **— This is** assumption \`$p$\`
        ⇒ ⟨ Assumption \`$p \Rightarrow q$\` ⟩
          q
        ⇒ ⟨ "Weakening" ⟩
          $q \vee r$
      Case \`$r$\`:
          r **— This is** assumption \`$r$\`
        ⇒ ⟨ "Weakening" ⟩
          $q \vee r$

CalcCheck By cases with "Zero or successor of predecessor": $n = 0 \lor n = suc\ (pred\ n)$

```
Theorem "Right-identity of subtraction": m - 0 = m
Proof:
  By cases: `m = 0`, `m = suc (pred m)`
    Completeness: By "Zero or successor of predecessor"
    Case `m = 0`:
        m - 0 = m
      ≡⟨ Assumption `m = 0` ⟩
        0 - 0 = 0
      — This is "Subtraction from zero"
    Case `m = suc (pred m)`:
        m - 0
      =⟨ Assumption `m = suc (pred m)` ⟩
        (suc (pred m)) - 0
      =⟨ "Subtraction of zero from successor" ⟩
        suc (pred m)
      =⟨ Assumption `m = suc (pred m)` ⟩
        m
```

Case Analysis with Calculation for "Completeness:" …

```
By cases: `pos b`, `¬ pos b`
  Completeness:
      pos b ∨ ¬ pos b
    ≡⟨ "Excluded Middle" ⟩
      true
  Case `pos b`:
    By (15.31a) with Assumption `pos b`
```

- After "**Completeness:**" goes a proof for the disjunction of all cases listed after "**By cases:**"
- This can be any kind of proof.
- Inside the "**Case** 'p':" block, you may use "**Assumption** 'p'"

Proof by Contrapositive

(3.61) **Contrapositive:** $p \Rightarrow q \quad \equiv \quad \neg q \Rightarrow \neg p$

(4.12) **Proof method:** Prove $P \Rightarrow Q$ by proving its contrapositive $\neg Q \Rightarrow \neg P$

Proving $x + y \geq 2 \quad \Rightarrow \quad x \geq 1 \lor y \geq 1$:

$$\begin{array}{ll}
 & \neg(x \geq 1 \lor y \geq 1) \\
\equiv & \langle \text{De Morgan (3.47)} \rangle \\
 & \neg(x \geq 1) \land \neg(y \geq 1) \\
\equiv & \langle \text{Def. } \geq \text{ (15.39) with Trichotomy (15.44)} \rangle \\
 & x < 1 \land y < 1 \\
\Rightarrow & \langle \text{Monotonicity of + (15.42)} \rangle \\
 & x + y < 1 + 1 \\
\equiv & \langle \text{Def. 2} \rangle \\
 & x + y < 2 \\
\equiv & \langle \text{Def. } \geq \text{ (15.39) with Trichotomy (15.44)} \rangle \\
 & \neg(x + y \geq 2)
\end{array}$$

Proof by Contrapositive in CalcCheck — Using

```
Theorem "Example for use of Contrapositive": x + y ≥ 2 ⇒ x ≥ 1 ∨ y ≥ 1
Proof:
  Using "Contrapositive":
    Subproof for `¬ (x ≥ 1 ∨ y ≥ 1) ⇒ ¬ (x + y ≥ 2)`:
        ¬ (x ≥ 1 ∨ y ≥ 1)
      ≡⟨ "De Morgan" ⟩
        ¬ (x ≥ 1) ∧ ¬ (y ≥ 1)
      ≡⟨ "Complement of <" with (3.14) ⟩
        x < 1 ∧ y < 1
      ⇒⟨ "<-Monotonicity of +" ⟩
        x + y < 1 + 1
      ≡⟨ Evaluation ⟩
        x + y < 2
      ≡⟨ "Complement of <" with (3.14) ⟩
        ¬ (x + y ≥ 2)
```

- "**Using** *HintItem1*: *subproof1 subproof2*" is processed as "**By** *HintItem1* **with** *subproof1* **and** *subproof2*"
- If you get the subproof goals wrong, the `with` heuristic has no chance to succeed…

Proof by Mutual Implication — Using

(3.80) **Mutual implication:** $(p \Rightarrow q) \land (q \Rightarrow p) \ \equiv\ p \equiv q$

```
Theorem (15.44A) "Trichotomy — A":
    a < b  ≡  a = b  ≡  a > b
Proof:
  Using "Mutual implication":
    Subproof for `a = b  ⇒  (a < b  ≡  a > b)`:
      Assuming `a = b`:
          a < b
        ≡⟨ "Converse of <", Assumption `a = b` ⟩
          a > b
    Subproof for `(a < b  ≡  a > b) ⇒ a = b`:
        a < b  ≡  a > b
      ≡⟨ "Definition of <", "Definition of >" ⟩
        pos (b - a) ≡ pos (a - b)
      ≡⟨ (15.17), (15.19), "Subtraction" ⟩
        pos (b - a) ≡ pos (- (b - a))
      ⇒⟨ (15.33c) ⟩
        b - a = 0
      ≡⟨ "Cancellation of +" ⟩
        b - a + a = 0 + a
      ≡⟨ "Identity of +", "Subtraction", "Unary minus" ⟩
        a = b
```

Proof by Contradiction

(3.74) $p \Rightarrow false \quad \equiv \quad \neg p$

(4.9) **Proof by contradiction:** $\neg p \Rightarrow false \quad \equiv \quad p$

"This proof method is overused"

If you intuitively try to do a proof by contradiction:

- Formalise your proof
- This may already contain a direct proof!
- So check whether contradiction is still necessary
- …, or whether your proof can be transformed into one that does not use contradiction.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-25

Examples of Structured Proofs; General Quantification

Plan for Today

- Order on Integers via Positivity (LADM chapter 15, pp. 307–308)

  ⟹ Opportunities for structured proofs

- General quantification, LADM chapter 8

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-25

Part 1: Structured Proofs Example: Order on Integers via Positivity

LADM Theory of Integers — Positivity and Ordering

(15.30) **Axiom, Addition in** *pos*: $pos\ a \land pos\ b \Rightarrow pos\ (a + b)$

(15.31) **Axiom, Multiplication in** *pos*: $pos\ a \land pos\ b \Rightarrow pos\ (a \cdot b)$

(15.32) **Axiom:** $\neg\, pos\ 0$

(15.33) **Axiom:** $b \neq 0 \quad \Rightarrow \quad (pos\ b \equiv \neg pos\ (-b))$

(15.34) **Positivity of Squares:** $b \neq 0 \quad \Rightarrow \quad pos\ (b \cdot b)$

(15.35) $pos\ a \quad \Rightarrow \quad (pos\ b \quad \equiv \quad pos\ (a \cdot b))$

(15.36) **Axiom, Less:** $a < b \quad \equiv \quad pos\ (b - a)$

(15.37) **Axiom, Greater:** $a > b \quad \equiv \quad pos\ (a - b)$

(15.38) **Axiom, At most:** $a \leq b \quad \equiv \quad a < b \lor a = b$

(15.39) **Axiom, At least:** $a \geq b \quad \equiv \quad a > b \lor a = b$

(15.40) **Positive elements:** $pos\ b \quad \equiv \quad 0 < b$

LADM Theory of Integers — Ordering Properties

(15.41) **Transitivity:**
- (a) $a < b \;\wedge\; b < c \;\Rightarrow\; a < c$
- (b) $a \le b \;\wedge\; b < c \;\Rightarrow\; a < c$
- (c) $a < b \;\wedge\; b \le c \;\Rightarrow\; a < c$
- (d) $a \le b \;\wedge\; b \le c \;\Rightarrow\; a \le c$

(15.42) **Monotonicity of +:** $a < b \;\equiv\; a + d < b + d$

(15.43) **Monotonicity of ·:** $0 < d \;\Rightarrow\; (a < b \;\equiv\; a \cdot d < b \cdot d)$

(15.44) **Trichotomy:** $(a < b \;\equiv\; a = b \;\equiv\; a > b) \;\wedge\; \neg(a < b \;\wedge\; a = b \;\wedge\; a > b)$

(15.45) **Antisymmetry of ≤:** $a \le b \;\wedge\; a \ge b \;\equiv\; a = b$

(15.46) **Reflexivity of ≤:** $a \le a$

Structured Proof Example from LADM

Theorems for *pos*

(15.34) $b \ne 0 \Rightarrow pos(b \cdot b)$

We prove (15.34). For arbitrary nonzero b in D, we prove $pos(b \cdot b)$ by case analysis: either $pos.b$ or $\neg pos.b$ holds (see (15.33)).

Case $pos.b$. By axiom (15.31) with $a, b := b, b$, $pos(b \cdot b)$ holds.

Case $\neg pos.b \wedge b \ne 0$. We have the following.

$$
\begin{array}{ll}
 & pos(b \cdot b) \\
= & \langle (15.23), \text{ with } a, b := b, b \rangle \\
 & pos((-b) \cdot (-b)) \\
\Leftarrow & \langle \text{Multiplication } (15.31) \rangle \\
 & pos(-b) \wedge pos(-b) \\
= & \langle \text{Idempotency of } \wedge \; (3.38) \rangle \\
 & pos(-b) \\
= & \langle \text{Double negation (3.12) —note that } b \ne 0 \text{; (15.33)} \rangle \\
 & \neg pos.b \quad \text{—the case under consideration}
\end{array}
$$

The Same Proof in CalcCheck

Theorem (15.34) “Positivity of squares”: $b \ne 0 \Rightarrow$ pos $(b \cdot b)$
Proof:
  Assuming \`$b \ne 0$\`:
    By cases: \`pos $b$\`, \`¬ pos $b$\`
      Completeness: By “Excluded middle”
      Case \`pos $b$\`:
        By “Positivity under ·” (15.31) with assumption \`pos $b$\`
      Case \`¬ pos $b$\`:
          pos $(b \cdot b)$
        ≡ ⟨ (15.23) \`$- a \cdot - b = a \cdot b$\` ⟩
          pos $((-b) \cdot (-b))$
        ⇐ ⟨ “Positivity under ·” (15.31) ⟩
          pos $(-b)$ ∧ pos $(-b)$
        ≡ ⟨ “Idempotency of ∧”, “Double negation” ⟩
          ¬ ¬ pos $(-b)$
        ≡ ⟨ “Positivity under unary minus” (15.33) with assumption \`$b \ne 0$\` ⟩
          ¬ pos b **— This is** assumption \`¬ pos $b$\`

Case Analysis with Calculation for “Completeness:” …

By cases: \`pos b\`, \`¬ pos b\`
  Completeness:
      pos b ∨ ¬ pos b
    ≡⟨ “Excluded Middle” ⟩
      true
  Case \`pos b\`:
    By (15.31a) with Assumption \`pos b\`

- After “**Completeness:**” goes a proof for the disjunction of all cases listed after “**By cases:**”
- This can be any kind of proof.
- Inside the “**Case ‘*p*’:**” block, you may use “**Assumption ‘*p*’**”

Proof by Contrapositive in CalcCheck — `Using`

Theorem “Example for use of Contrapositive”: x + y ≥ 2 ⇒ x ≥ 1 ∨ y ≥ 1
Proof:
  Using “Contrapositive”:
    Subproof for \`¬ (x ≥ 1 ∨ y ≥ 1) ⇒ ¬ (x + y ≥2)\`:
        ¬ (x ≥ 1 ∨ y ≥ 1)
      ≡⟨ “De Morgan” ⟩
        ¬ (x ≥ 1) ∧ ¬ (y ≥ 1)
      ≡⟨ “Complement of <” with (3.14) ⟩
        x < 1 ∧ y < 1
      ⇒⟨ “<-Monotonicity of +” ⟩
        x + y < 1 + 1
      ≡⟨ Evaluation ⟩
        x + y < 2
      ≡⟨ “Complement of <” with (3.14) ⟩
        ¬ (x + y ≥ 2)

- “**Using** *HintItem1*: *subproof1 subproof2*” is processed as “**By** *HintItem1* **with** *subproof1* **and** *subproof2*”
- If you get the subproof goals wrong, the `with` heuristic has no chance to succeed…

Proof by Mutual Implication — `Using`

(3.80) **Mutual implication:** $(p \Rightarrow q) \wedge (q \Rightarrow p) \;\equiv\; p \equiv q$

Theorem “Cancellation of unary minus”: $- a = - b \;\equiv\; a = b$
Proof:
  Using “Mutual implication”:
    Subproof: ⋯⋯ Subproof goals determined by the enclosed proof can be omitted.
      Assuming \`$a = b$\`:
          $- a$
        = ⟨ Assumption \`$a = b$\` ⟩
          $- b$
    Subproof:
      Assuming \`$- a = - b$\`:
          a
        = ⟨ “Self-inverse of unary minus” ⟩
          $- - a$
        = ⟨ Assumption \`$- a = - b$\` ⟩
          $- - b$
        = ⟨ “Self-inverse of unary minus” ⟩
          b

The CalcCheck Language — Calculational Proofs on Steroids

- LADM emphasises use of axioms and theorems in calculations over other inference rules

Besides calculations, CalcCheck has the following proof structures:

- By *hint* — for discharging simple proof obligations,
- Assuming ‘*expression*’: — for assuming the antecedent,
- By cases: ‘$expression_1$’,…,‘$expression_n$’ — for proofs by case analysis
- By induction on ‘*var* : *type*’: — for proofs by induction
- Using *hint*: — for turning theorems into inference rules
- For any ‘*var* : *type*’: — corresponding to ∀-introduction

This does not sound that different from LADM —

— but in CalcCheck, these are actually used!

Proofs Structures Can Be Freely Combined…

Theorem (15.35) “Positivity under positive ·”: pos a ⇒ (pos b ≡ pos $(a \cdot b)$)
Proof:
  Assuming \`pos $a$\`:
    Using “Mutual implication”:
      Subproof for \`pos $b$ ⇒ pos $(a \cdot b)$\`:
          pos b ⇒ pos $(a \cdot b)$
        ⇐ ⟨ “Positivity under ·” ⟩
          pos a **— This is** Assumption \`pos $a$\`
      Subproof for \`pos $(a \cdot b)$ ⇒ pos $b$\`:
        Using “Contrapositive”:
          Subproof for \`¬ pos $b$ ⇒ ¬ pos $(a \cdot b)$\`:
            By cases: \`$b = 0$\`, \`$b \ne 0$\`
              Completeness: By “Definition of ≠”, “LEM”
              Case \`$b = 0$\`:
                  ¬ pos b ⇒ ¬ pos $(a \cdot b)$
                ≡ ⟨ Assumption \`$b = 0$\`, “Zero of ·” ⟩
                  ¬ pos 0 ⇒ ¬ pos 0 **— This is** “Reflexivity of ⇒”
              Case \`$b \ne 0$\`:
                  ¬ pos b
                ≡ ⟨ (15.33*b*) with Assumption \`$b \ne 0$\` ⟩

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-25

Part 2: General Quantification

Recall: Quantification Examples

$$
\begin{array}{ll}
 & (\textstyle\sum i \mid 0 \le i < 4 \bullet i \cdot 8) \\
= & \langle \text{Quantification expansion, substitution} \rangle \\
 & 0 \cdot 8 + 1 \cdot 8 + 2 \cdot 8 + 3 \cdot 8
\end{array}
$$

$$
\begin{array}{ll}
 & (\textstyle\prod i \mid 0 \le i < 3 \bullet i + (i + 1)) \\
= & \langle \text{Quantification expansion, substitution} \rangle \\
 & (0 + 1) \cdot (1 + 2) \cdot (2 + 3)
\end{array}
$$

$$
\begin{array}{ll}
 & (\forall i \mid 1 \le i < 3 \bullet i \cdot d \ne 6) \\
= & \langle \text{Quantification expansion, substitution} \rangle \\
 & 1 \cdot d \ne 6 \wedge 2 \cdot d \ne 6
\end{array}
$$

$$
\begin{array}{ll}
 & (\exists i \mid 0 \le i < 6 \bullet b\, i = 0) \\
= & \langle \text{Quantification expansion, substitution} \rangle \\
 & b\,0 = 0 \vee b\,1 = 0 \vee b\,2 = 0 \vee b\,3 = 0 \vee b\,4 = 0 \vee b\,5 = 0
\end{array}
$$

Recall: General Quantification

It works not only for $+, \wedge, \vee \ldots$

Let a type T and an operator $\star : T \times T \to T$ be given.
If for an appropriate $u : T$ we have:

- **Symmetry:** $b \star c = c \star b$
- **Associativity:** $(b \star c) \star d = b \star (c \star d)$
- **Identity** u**:** $u \star b = b = b \star u$

we may use $\star$ as quantification operator:

$$(\star\, x : T_1, y : T_2 \mid R \bullet E)$$

- $R : \mathbb{B}$ is the **range** of the quantification
- $E : T$ is the **body** of the quantification
- E and R may refer to the **quantified variables** x and y
- The type of the whole quantification expression is T.

Recall: General Quantification: Instances

Let a type T and an operator $\star : T \times T \to T$ be given.
If for an appropriate $u : T$ we have:

- **Symmetry:** $b \star c = c \star b$
- **Associativity:** $(b \star c) \star d = b \star (c \star d)$
- **Identity** u**:** $u \star b = b = b \star u$

we may use $\star$ as quantification operator: $(\star\, x : T_1, y : T_2 \mid R \bullet E)$

- $\_\vee\_ : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$ is symmetric (3.24), associative (3.25), and has *false* as identity (3.30) — the "big operator" for $\vee$ is $\exists$":
  $(\exists\, k : \mathbb{N} \mid k > 0 \bullet k \cdot k < k + 1)$
- $\_\wedge\_ : \mathbb{B} \times \mathbb{B} \to \mathbb{B}$ is symmetric (3.36), associative (3.27), and has *true* as identity (3.39) — the "big operator" for $\wedge$ is $\forall$":
  $(\forall\, k : \mathbb{N} \mid k > 2 \bullet \mathit{prime}\ k \Rightarrow \neg\, \mathit{prime}\ (k + 1))$
- $\_+\_ : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ is symmetric (15.2), associative (15.1), and has 0 as identity (15.3) — the "big operator" for $+$ is $\sum$":
  $(\sum\, n : \mathbb{Z} \mid 0 < n < 100 \wedge \mathit{prime}\ n \bullet n \cdot n)$

Recall: Meaning of General Quantification

Let a type T, and a symmetric and associative operator $\star : T \times T \to T$ with identity $u : T$ be given.

Further let x be a **variable list**, R a Boolean expression, and E an expression of type T.

> The **meaning** of $(\star\, x \mid R \bullet E)$ in state s is:
> - the nested application of $\star$ to the meanings of E
> - in all those states that satisfy R
> - and are different from s at most in variables in x,
>
> or u, if there are no such states.

LADM section 8.3 axiomatizes this semantics and makes it accessible to syntactic reasoning.

Trivial Range Axioms

(8.13) **Axiom, Empty Range** (where u is the identity of $\star$):

$$(\star\, x \mid \mathit{false} \bullet P) \quad = \quad u$$

$$(\forall\, x \mid \mathit{false} \bullet P) \quad = \quad \mathit{true}$$

$$(\exists\, x \mid \mathit{false} \bullet P) \quad = \quad \mathit{false}$$

$$(\textstyle\sum\, x \mid \mathit{false} \bullet P) \quad = \quad 0$$

$$(\textstyle\prod\, x \mid \mathit{false} \bullet P) \quad = \quad 1$$

(8.14) **Axiom, One-point Rule:** Provided $\neg \mathit{occurs}(`x\text{'}, `E\text{'})$,

$$(\star x \mid x = E \bullet P) \quad = \quad P[x := E]$$

Recall: Bound / Free Variable Occurrences

$(\sum\, i : \mathbb{N} \mid i < x \bullet i + 1) = 10$ — example expression

Is this true or false? In which states?

We have: $(\sum\, i : \mathbb{N} \mid i < x \bullet i + 1) = 10 \quad \equiv \quad x = 4$

The value of this example expression in a state depends only on x, not on i!

Renaming quantified variables <u>does not change the meaning</u>:

$$(\textstyle\sum\, i : \mathbb{N} \mid i < x \bullet i + 1) \quad = \quad (\textstyle\sum\, j : \mathbb{N} \mid j < x \bullet j + 1)$$

- **Occurrences** of quantified variables inside the quantified expression are **bound**
- Non-bound **variable occurences** are called **free**
- Variables of the same name may occur both free and bound in the same expression, e.g.: $3 \cdot i + (\sum\, i : \mathbb{N} \mid i < x \bullet 2 \cdot i)$
- The variable declarations after the quantification operator may be called **binding occurrences**.

The *occurs* Meta-Predicate

Definition: $\mathit{occurs}(`v\text{'}, `e\text{'})$ means that at least one variable in the list v of variables occurs **free** in at least one expression in expression list e.

$\mathit{occurs}(`i, n\text{'}, `(\sum\, i, n \mid 1 \le i \cdot n \le k \bullet n^i), (\sum\, n \mid 0 \le n < k \bullet n^i)\text{'})$ ✓

$\mathit{occurs}(`i\text{'}, `(i \cdot (5 + i))[i := k + 2]\text{'})$ ✗ — **Substitution is a variable binder, too!**

$\mathit{occurs}(`i\text{'}, `(i \cdot (5 + i))[i := i + 2]\text{'})$ ✓

The $\neg$*occurs* Proviso for the One-point Rule

(8.14) **Axiom, One-point Rule for $\sum$:** Provided $\neg \mathit{occurs}(`x\text{'}, `E\text{'})$,

$$(\textstyle\sum x \mid x = E \bullet P) \quad = \quad P[x := E]$$

(8.14) **Axiom, One-point Rule for $\prod$:** Provided $\neg \mathit{occurs}(`x\text{'}, `E\text{'})$,

$$(\textstyle\prod x \mid x = E \bullet P) \quad = \quad P[x := E]$$

Examples:

- $(\sum x \mid x = 1 \bullet x \cdot y) \quad = \quad 1 \cdot y$
- $(\prod x \mid x = y + 1 \bullet x \cdot x) \quad = \quad (y + 1) \cdot (y + 1)$
- $(\sum x \mid x = (\sum x \mid 1 \le x < 4 \bullet x) \bullet x \cdot y) \quad = \quad (\sum x \mid 1 \le x < 4 \bullet x) \cdot y \quad = \quad 6 \cdot y$

Counterexamples:

- $(\sum x \mid x = x + 1 \bullet x) \quad ? \quad x + 1$ — **"=" not valid!**
- $(\prod x \mid x = 2 \cdot x \bullet y + x) \quad ? \quad y + 2 \cdot x$ — **"=" not valid!**

The $\neg$*occurs* Proviso for the One-point Rule

(8.14) **Axiom, One-point Rule:** Provided $\neg \mathit{occurs}(`x\text{'}, `E\text{'})$,

$$(\star x \mid x = E \bullet P) \quad = \quad P[x := E]$$

$$(\forall x \mid x = E \bullet P) \quad \equiv \quad P[x := E]$$

$$(\exists x \mid x = E \bullet P) \quad \equiv \quad P[x := E]$$

Examples:

- $(\forall x \mid x = 1 \bullet x \cdot y = y) \quad \equiv \quad 1 \cdot y = y$
- $(\exists x \mid x = y + 1 \bullet x \cdot x > 42) \quad \equiv \quad (y + 1) \cdot (y + 1) > 42$

Counterexamples:

- $(\forall x \mid x = x + 1 \bullet x = 42) \quad ? \quad x + 1 = 42$ — **"≡" not valid!**
- $(\exists x \mid x = 2 \cdot x \bullet y + x = 42) \quad ? \quad y + 2 \cdot x = 42$ — **"≡" not valid!**

One-point Rule with Example Calculation

(8.14) **Axiom, One-point Rule:** Provided $\neg \mathit{occurs}(`x\text{'}, `E\text{'})$,

$$(\star\, x \mid x = E \bullet P) \quad = \quad P[x := E]$$

Example:

$(\sum\, i : \mathbb{N} \bullet 5 + 2 \cdot i < 7 \mid 5 + 7 \cdot i)$
$= \langle \ldots \rangle$
$(\sum\, i : \mathbb{N} \bullet i = 0 \mid 5 + 7 \cdot i)$
$= \langle$ One-point rule $\rangle$
$(5 + 7 \cdot i)[i := 0]$
$= \langle$ Substitution $\rangle$
$5 + 7 \cdot 0$

Automatic extraction of $\neg$*occurs* Provisos

(8.14) **Axiom, One-point Rule:** Provided $\neg \mathit{occurs}(`x\text{'}, `E\text{'})$,

$$(\forall\, x \mid x = E \bullet P) \quad \equiv \quad P[x := E]$$

$$(\exists\, x \mid x = E \bullet P) \quad \equiv \quad P[x := E]$$

Investigate the binders in scope at the metavariables P and E:

- P on the LHS occurs in scope of the binder $\forall\, x$
- P on the RHS occurs in scope of the binder $\_[x := \ldots]$

Therefore: Whether x occurs in P or not does not raise any problems.

- E on the LHS occurs in scope of the binder $\forall\, x$
- E on the RHS occurs in scope no binders

Therefore: An x that is free in E would be **bound** on the LHS, but **escape** into freedom on the RHS!

CalcCheck derives and checks $\neg$*occurs* **provisos automatically.**

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-27

Conditional Commands; General Quantification

Plan for Today

- More on **Command Correctness**: Chaining with ⇒; **Conditional Commands**

  ⟹ Another example of structured proofs

- **General Quantification (LADM chapter 8, ctd.)**

  ⟹ **Calculating with Quantifications**

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-27

Part 1: More Command Correctness

Recall: Partial Correctness for Pre-Postcond. Specs in Dynamic Logic Notation

- Program correctness statement in LADM (and much current use):

  $$\{ P \}\ C\ \{ Q \}$$

  This is called a "Hoare triple".

- **Partial Correctness Meaning:**

  If **command** C is started in a state in which the **precondition** P holds then it will terminate **only in states** in which the **postcondition** Q holds.

- **Dynamic logic** notation (used in CalcCheck):

  $$P \Rrightarrow[\ C\ ]\ Q$$

- **Assignment Axiom:**

  — Hoare triple: $\{ Q[x := E] \}\ x := E\ \{ Q \}$

  — **Dynamic logic** notation (used in CalcCheck): $Q[x := E] \Rrightarrow[\ x := E\ ]\ Q$

Transitivity Rules for Calculational Command Correctness Reasoning

Primitive inference rule "Sequence":

$$\vdash \frac{`P \Rrightarrow[\ C_1\ ]\ Q`, \quad `Q \Rrightarrow[\ C_2\ ]\ R`}{`P \Rrightarrow[\ C_1 ; C_2\ ]\ R`}$$

Strengthening the precondition:

$$\vdash \frac{`P_1 \Rightarrow P_2`, \quad `P_2 \Rrightarrow[\ C\ ]\ Q`}{`P_1 \Rrightarrow[\ C\ ]\ Q`}$$

Weakening the postcondition:

$$\vdash \frac{`P \Rrightarrow[\ C\ ]\ Q_1`, \quad `Q_1 \Rightarrow Q_2`}{`P \Rrightarrow[\ C\ ]\ Q_2`}$$

- Activated as transitivity rules
- Therefore used implicitly in calculations, e.g., proving $P \Rrightarrow[\ C_1 ; C_2\ ]\ R$ to the right

$$\begin{array}{ll} & P \\ \Rrightarrow[\ C_1\ ] & \langle \ldots \rangle \\ & Q \\ \Rightarrow & \langle \ldots \rangle \\ & Q' \\ \Rrightarrow[\ C_2\ ] & \langle \ldots \rangle \\ & R \end{array}$$

What Does this Program Fragment Do?

Let x and y be variables of type $\mathbb{Z}$.

```
x := x + y ;
y := x − y ;
x := x − y
```

How can you specify that?

Can you prove it?

Example execution:

$$\begin{array}{ll} & [\ (x, 5),\ (y, 6)\ ] \\ \leadsto & \langle \quad x := x + y \quad \rangle \\ & [\ (x, 11),\ (y, 6)\ ] \\ \leadsto & \langle \quad y := x - y \quad \rangle \\ & [\ (x, 11),\ (y, 5)\ ] \\ \leadsto & \langle \quad x := x - y \quad \rangle \\ & [\ (x, 6),\ (y, 5)\ ] \end{array}$$

Perhaps the values of x and y are swapped?

Specification Pattern "Auxiliary Variables"

Let x and y be variables of type $\mathbb{Z}$.

Specifying value swap:

$$\begin{array}{l} x = x_0 \wedge y = y_0 \\ \Rrightarrow[\\ \quad x := x + y ; \\ \quad y := x - y ; \\ \quad x := x - y \\ ] \\ x = y_0 \wedge y = x_0 \end{array}$$

You can prove that!

- Frequently, the postcondion needs to refer to values of the state variables "at the time of the precondition".
- With Hoare triples, the standard way to achieve this is the use of "auxiliary variables":
  - "auxiliary variables" (here: x_0 and y_0) do not occur in the program
  - they may occur in both precondition and postcondition
  - throughout the correctness proof, the "have the same values"
- Other formalisms "decorate" variable names:
  - Z: "Primed" postcondition variables:

    $$x' = y \quad \wedge \quad y' = x$$

  - ACSL: Referencing precondition variables as in the \old state:

    $$x \equiv \backslash old(y) \quad \wedge \quad y \equiv \backslash old(x)$$

Conditional Commands

- Pascal:

```
if condition then
  statement₁
else
  statement₂
```

- Ada:

```
if condition then
  statement₁
else
  statement₂
end if;
```

- C/Java:

```
if (condition)
  statement₁
else
  statement₂
```

- Python:

```
if condition:
  statement₁
else:
  statement₂
```

- sh:

```
if condition
then
  statement₁
else
  statement₂
fi
```

Conditional Rule

Primitive inference rule "Conditional":

$$\vdash \frac{`B \wedge P \Rrightarrow[\ C_1\ ]\ Q`, \quad `\neg B \wedge P \Rrightarrow[\ C_2\ ]\ Q`}{`P \Rrightarrow[\ \text{if } B \text{ then } C_1 \text{ else } C_2 \text{ fi}\ ]\ Q`}$$

```
Fact “Simple COND”:
   true ⇛[ if x = 1 then y := 42 else x := 1 fi ] x = 1
Proof:
    true
  ⇛[ if x = 1 then y := 42 else x := 1 fi ] ⟨ Subproof:
      Using “Conditional”:
        Subproof for `(true ∧ x = 1) ⇛[ y := 42 ] x = 1`:
            ?
        Subproof for `(true ∧ ¬ (x = 1)) ⇛[ x := 1 ] x = 1`:
            ?
    ⟩
    x = 1
```

```
Fact “Simple COND”:
   true ⇛[ if x = 1 then y := 42 else x := 1 fi ] x = 1
Proof:
    true
  ⇛[ if x = 1 then y := 42 else x := 1 fi ] ⟨ Subproof:
      Using “Conditional”:
        Subproof for `(true ∧ x = 1) ⇛[ y := 42 ] x = 1`:
            true ∧ x = 1
          ≡⟨ “Identity of ∧” ⟩
            x = 1
          ≡⟨ Substitution ⟩
            (x = 1)[y ≔ 42]
          ⇛[ y := 42 ] ⟨ “Assignment” ⟩
            x = 1
        Subproof for `(true ∧ ¬ (x = 1)) ⇛[ x := 1 ] x = 1`:
            true ∧ ¬ (x = 1)
          ⇒⟨ “Right-zero of ⇒” ⟩
            true
          ≡⟨ “Reflexivity of =” ⟩
            1 = 1
          ≡⟨ Substitution ⟩
            (x = 1)[x ≔ 1]
          ⇛[ x := 1 ]  ⟨ “Assignment” ⟩
            x = 1
    ⟩
    x = 1
```

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-27

Part 2: General Quantification

Bound / Free Variable Occurrences — The *occurs* Meta-Predicate

Renaming quantified variables does not change the meaning:

$$(\forall\, i \bullet x \cdot i = 0) \quad\equiv\quad (\forall\, j \bullet x \cdot j = 0)$$

- **Occurrences** of quantified variables inside the quantified expression are **bound**
- **Variable occurences** in an expression where they are not bound are **free**

$$i > 0 \lor (\forall\, i \mid 0 \le i \bullet x \cdot i = 0)$$

- The variable declarations after the quantification operator may be called **binding occurrences**.

Definition: $occurs(\text{'}v\text{'}, \text{'}e\text{'})$ means that at least one variable in the list v of variables occurs **free** in at least one expression in expression list e.

CalcCheck derives and checks $\neg occurs$ provisos automatically.

Textual Substitution Revisited

Let E and R be expressions and let x be a variable. **Original definition:**

> We write: $E[x := R]$ or E^x_R
> to denote an expression that is the same as E but with all occurrences of x replaced by (R).

This was for expressions E built from **constants, variables, operator applications** only!

In presence of **variable binders**, such as $\sum$, $\prod$, $\forall$, $\exists$ and substitution,

- only **free** occurrences of x can be replaced
- **and** we need to avoid **"capture of free variables"**:

(8.11) Provided $\neg occurs(\text{'}y\text{'}, \text{'}x, F\text{'})$,

$$(\star\, y \mid R \bullet P)[x := F] \quad=\quad (\star\, y \mid R[x := F] \bullet P[x := F])$$

LADM Chapter 8:
"$\star$ is a **metavariable** for operators $\_+\_$, $\_\cdot\_$, $\_\land\_$, $\_\lor\_$" (resp. $\sum$, $\prod$, $\forall$, $\exists$)

(8.11) is part of the Substitution keyword in CalcCheck.

Read LADM Chapter 8!

Substitution Examples

(8.11) Provided $\neg occurs(\text{'}y\text{'}, \text{'}x, F\text{'})$,

$$(\star\, y \mid R \bullet P)[x := F] \quad=\quad (\star\, y \mid R[x := F] \bullet P[x := F])$$

- $(\sum x \mid 1 \le x \le 2 \bullet y)[y := y + z]$
  $=$ ⟨ substitution ⟩
  $(\sum x \mid 1 \le x \le 2 \bullet y + z)$

- $(\sum x \mid 1 \le x \le 2 \bullet y)[y := y + x]$
  $=$ ⟨ (8.21) Variable renaming ⟩
  $(\sum z \mid 1 \le z \le 2 \bullet y)[y := y + x]$
  $=$ ⟨ substitution ⟩
  $(\sum z \mid 1 \le z \le 2 \bullet y + x)$

Substitution Examples (ctd.)

(8.11) Provided $\neg occurs(\text{'}y\text{'}, \text{'}x, F\text{'})$,

$$(\star\, y \mid R \bullet P)[x := F] \quad=\quad (\star\, y \mid R[x := F] \bullet P[x := F])$$

- $(\sum x \mid 1 \le x \le 2 \bullet y)[x := y + x]$
  $=$ ⟨ (8.21) Variable renaming ⟩
  $(\sum z \mid 1 \le z \le 2 \bullet y)[x := y + x]$
  $=$ ⟨ Substitution ⟩
  $(\sum z \mid 1 \le z \le 2 \bullet y)$
  $=$ ⟨ (8.21) Variable renaming ⟩
  $(\sum x \mid 1 \le x \le 2 \bullet y)$

(8.11f) Provided $\neg occurs(\text{'}x\text{'}, \text{'}E\text{'})$,

$$E[x := F] \quad=\quad E$$

Renaming of Bound Variables

(8.21) **Axiom, Dummy renaming** (α-conversion):
$(\star\, x \mid R \bullet P) \quad=\quad (\star\, y \mid R[x := y] \bullet P[x := y])$ provided $\neg occurs(\text{'}y\text{'}, \text{'}R, P\text{'})$.

$(\sum i \mid 0 \le i < k \bullet n^i)$
$=$ ⟨ Dummy renaming (8.21), $\neg occurs(\text{'}j\text{'}, \text{'}0 \le i < k,\ n^i\text{'})$ ⟩
$(\sum j \mid 0 \le j < k \bullet n^j)$

$(\sum i \mid 0 \le i < k \bullet n^i)$
? ⟨ Dummy renaming (8.21) ⟩ ×
$(\sum k \mid 0 \le k < k \bullet n^k)$ ▪▪▪▪▪▪ k **captured!**

Generally, use fresh variables for renaming to avoid variable capture!

In CalcCheck, renaming of bound variables is part of "Reflexivity of =", but can also be mentioned explicitly.

Leibniz Rules for Quantification

Try to use $x + x = 2 \cdot x$ and Leibniz (1.5) $\dfrac{X \;=\; Y}{E[z := X] \;=\; E[z := Y]}$ to obtain:

$$(\sum x \mid 0 \le x < 9 \bullet x + x) = (\sum x \mid 0 \le x < 9 \bullet 2 \cdot x)$$

- Choose E as: $(\sum x \mid 0 \le x < 9 \bullet z)$
- Perform substitution: $(\sum x \mid 0 \le x < 9 \bullet z)[z := x + x]$
  $(\sum y \mid 0 \le y < 9 \bullet x + x)$
- Not possible with (1.5)!
  — $E[z := X] = E[z := Y]$ **renames** x**!**

Special Leibniz rule for quantification:

$$\frac{P \;=\; Q}{(\star\, x \mid R \bullet E[z := P]) \;=\; (\star\, x \mid R \bullet E[z := Q])}$$

LADM Leibniz Rules for Quantification

Rewrite equalities in the **range** context of quantifications:

(8.12) **Leibniz** $\dfrac{P \;=\; Q}{(\star\, x \mid E[z := P] \bullet S) \;=\; (\star\, x \mid E[z := Q] \bullet S)}$

Rewrite equalities in the **body** context of quantifications:

(8.12) **Leibniz** $\dfrac{R \;\Rightarrow\; (P \;=\; Q)}{(\star\, x \mid R \bullet E[z := P]) \;=\; (\star\, x \mid R \bullet E[z := Q])}$

(These inference rules will also be used **implicitly**.)

Important: $P = Q$, repectively $R \Rightarrow (P = Q)$, needs to be a **theorem**!
These rules are **not** available for local **Assumptions**!
(Because x may occur in R, P, Q.)

The CalcCheck versions use **universally-quantified antecedents**.

Axiom "Leibniz for $\sum$ range": $(\forall\, x \bullet R_1 \equiv R_2) \Rightarrow (\sum x \mid R_1 \bullet E) = (\sum x \mid R_2 \bullet E)$
Axiom "Leibniz for $\sum$ body": $(\forall\, x \bullet R \Rightarrow E_1 = E_2) \Rightarrow (\sum x \mid R \bullet E_1) = (\sum x \mid R \bullet E_2)$

Formalise:

- The sum of the first n odd natural numbers is equal to n^2.

Formalise it in a way that makes it easy to prove!

```
Theorem “Odd-number sum”:
    (∑ i : ℕ ❙ i < n • suc i + i) = n · n
```

The sum of the first n odd natural numbers is equal to n^2

```
Theorem “Odd-number sum”:
    (∑ i : ℕ ❙ i < n • suc i + i) = n · n
Proof:
  By induction on `n : ℕ`:
    Base case:
        (∑ i : ℕ ❙ i < 0 • suc i + i)
      =⟨ ? ⟩



      =⟨ ? ⟩
        0 · 0
    Induction step:
        (∑ i : ℕ ❙ i < suc n • suc i + i)
      =⟨ ? ⟩



      =⟨ ? ⟩
        suc n · suc n
```

Empty Range Axioms

(8.13) **Axiom, Empty Range**:

$$(\sum x \mid false \bullet E) \quad = \quad 0$$

$$(\prod x \mid false \bullet E) \quad = \quad 1$$

The sum of the first n odd natural numbers is equal to n^2

```
Theorem "Odd-number sum":
    (∑ i : ℕ ❙ i < n • suc i + i) = n · n
Proof:
  By induction on `n : ℕ`:
    Base case:
        (∑ i : ℕ ❙ i < 0 • suc i + i)
      =( "Nothing is less than zero" )
        (∑ i : ℕ ❙ false • suc i + i)
      =( "Empty range for ∑")
        0
      =( "Definition of · for 0" )
        0 · 0
    Induction step:
        (∑ i : ℕ ❙ i < suc n • suc i + i)
      =( "Split off term at top", Substitution )
        (∑ i : ℕ ❙ i < n • suc i + i) + (suc n + n)
      =( Induction hypothesis )
        suc n + n + n · n
      =( "Definition of · for `suc`" )
        suc n + n · suc n
      =( "Definition of · for `suc`" )
        suc n · suc n
```

Manipulating Ranges

(8.23) **Theorem Split off term**: For $n : \mathbb{N}$ and dummies $i : \mathbb{N}$,

$$(\star\, i \mid 0 \le i < n+1 \bullet P) \quad = \quad (\star\, i \mid 0 \le i < n \bullet P) \star P[i := n]$$

$$(\star\, i \mid 0 \le i < n+1 \bullet P) \quad = \quad P[i := 0] \star (\star\, i \mid 0 < i < n+1 \bullet P)$$

- Typical uses: Induction proofs, verification of loops
- Generalisation: $\mathbb{N} \longrightarrow \mathbb{Z}$, $\quad 0 \longrightarrow m : \mathbb{Z}$ (with $m \le n$)

The following work both with $m, n, i : \mathbb{N}$ and with $m, n, i : \mathbb{Z}$:

Theorem: Split off term from top:

$m \le n \quad \Rightarrow$
$$(\star\, i \mid m \le i < n+1 \bullet P) = (\star\, i \mid m \le i < n \bullet P) \star P[i := n]$$

Theorem: Split off term from bottom:

$m \le n \quad \Rightarrow$
$$(\star\, i \mid m \le i < n+1 \bullet P) = P[i := m] \star (\star\, i \mid m+1 \le i < n+1 \bullet P)$$

Manipulating Ranges

(8.23) **Theorem Split off term**: For $n : \mathbb{N}$ and dummies $i : \mathbb{N}$,

$$(\sum i \mid 0 \le i < n+1 \bullet P) \quad = \quad (\sum i \mid 0 \le i < n \bullet P) + P[i := n]$$

$$(\sum i \mid 0 \le i < n+1 \bullet P) \quad = \quad P[i := 0] + (\sum i \mid 0 < i < n+1 \bullet P)$$

- Typical uses: Induction proofs, verification of loops
- Generalisation: $\mathbb{N} \longrightarrow \mathbb{Z}$, $\quad 0 \longrightarrow m : \mathbb{Z}$ (with $m \le n$)

The following work both with $m, n, i : \mathbb{N}$ and with $m, n, i : \mathbb{Z}$:

Theorem: Split off term from top:

$m \le n \quad \Rightarrow$
$$(\sum i \mid m \le i < n+1 \bullet P) = (\sum i \mid m \le i < n \bullet P) + P[i := n]$$

Theorem: Split off term from bottom:

$m \le n \quad \Rightarrow$
$$(\sum i \mid m \le i < n+1 \bullet P) = P[i := m] + (\sum i \mid m+1 \le i < n+1 \bullet P)$$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-29

General Quantification 3, Predicate Logic 1

Plan for Today

- **General Quantification (LADM chapter 8) — last part**
- **Predicate Logic 1**:
  Axioms and Theorems about Universal and Existential Quantification
  (LADM chapter 9)

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-29

Part 1: General Quantification (ctd.)

Distributivity

(8.15) **Axiom,** (Quantification) **Distributivity:**

$$(\star\, x \mid R \bullet P) \star (\star\, x \mid R \bullet Q) = (\star\, x \mid R \bullet P \star Q),$$

provided each quantification is defined.

CalcCheck currently has no way to express or check this proviso —
— it remains in **your responsibility!**

$$(\sum i : \mathbb{N} \mid i < n \bullet f\ i) + (\sum i : \mathbb{N} \mid i < n \bullet g\ i)$$
$$= \langle \text{Quantification Distributivity (8.15)} \rangle$$
$$(\sum i : \mathbb{N} \mid i < n \bullet f\ i + g\ i)$$

Note: Some quantifications are not defined, e.g.: $(\sum n : \mathbb{N} \bullet n)$

Note that quantifications over $\wedge$ or $\vee$ are always defined:

$$(\forall x \mid R \bullet P \wedge Q) \quad = \quad (\forall x \mid R \bullet P) \wedge (\forall x \mid R \bullet Q)$$

$$(\exists x \mid R \bullet P \vee Q) \quad = \quad (\exists x \mid R \bullet P) \vee (\exists x \mid R \bullet Q)$$

Disjoint Range Split — LADM

(8.16) **Axiom, Range split:**

$$(\star\, x \mid R \vee S \bullet P) \quad = \quad (\star\, x \mid R \bullet P) \star (\star\, x \mid S \bullet P)$$

provided $R \wedge S = false$ and each quantification is defined.

$$(\Sigma x \mid R \vee S \bullet P) \quad = \quad (\Sigma x \mid R \bullet P) + (\Sigma x \mid S \bullet P)$$

provided $R \wedge S = false$ and each sum is defined.

$$(\forall x \mid R \vee S \bullet P) \quad = \quad (\forall x \mid R \bullet P) \wedge (\forall x \mid S \bullet P)$$

provided $R \wedge S = false$.

$$(\exists x \mid R \vee S \bullet P) \quad = \quad (\exists x \mid R \bullet P) \vee (\exists x \mid S \bullet P)$$

provided $R \wedge S = false$.

Disjoint Range Split for $\sum$ (LADM and CalcCheck)

(8.16) **Axiom, Range Split:** $\quad (\Sigma x \mid R \vee S \bullet P) = (\Sigma x \mid R \bullet P) + (\Sigma x \mid S \bullet P)$

provided $R \wedge S = false$ and each sum is defined.

CalcCheck currently cannot deal with "provided each sum is defined".
But once $\forall$ is available, $Q \wedge R = false$ does not need to be a proviso:

Theorem "Disjoint range split for $\sum$":
$\quad (\forall x \bullet R \wedge S \equiv \text{false}) \Rightarrow$
$\quad ((\sum x \mid R \vee S \bullet E) = (\sum x \mid R \bullet E) + (\sum x \mid S \bullet E))$

That is: Summing up over a large range can be done by adding the results of summing up two disjoint and complementary subranges.

$\Longrightarrow$ "**Divide and conquer**" algorithm design pattern

DIVIDE ET IMPERA
— Gaius Julius Caesar

Range Split "Axioms"

(8.16) **Axiom, Range split:**

$$(\star x \mid R \vee S \bullet P) \quad = \quad (\star x \mid R \bullet P) \star (\star x \mid S \bullet P)$$

provided $R \wedge S = \mathit{false}$ and each quantification is defined.

(8.17) **Axiom, Range Split:**

$$(\star x \mid R \vee S \bullet P) \star (\star x \mid R \wedge S \bullet P) \quad = \quad (\star x \mid R \bullet P) \star (\star x \mid S \bullet P)$$

provided each quantification is defined.

(8.18) Axiom, **Range Split** for idempotent $\star$:

$$(\star x \mid R \vee S \bullet P) \quad = \quad (\star x \mid R \bullet P) \star (\star x \mid S \bullet P)$$

provided each quantification is defined.

$$(\forall x \mid R \vee S \bullet P) \quad = \quad (\forall x \mid R \bullet P) \wedge (\forall x \mid S \bullet P)$$

$$(\exists x \mid R \vee S \bullet P) \quad = \quad (\exists x \mid R \bullet P) \vee (\exists x \mid S \bullet P)$$

Permutation of Bound Variables

Apparently not provable for general quantification from the quantification axioms in the textbook:

Dummy list permutation:

$$(\star x, y \mid R \bullet P) \quad = \quad (\star y, x \mid R \bullet P)$$

(without side conditions restricting variable occurrences!)

However, the following are easily provable from (8.19) **Interchange of dummies** — **Exercise:**

Dummy list permutation for $\forall$:

$$(\forall x, y \mid R \bullet P) \quad = \quad (\forall y, x \mid R \bullet P)$$

Dummy list permutation for $\exists$:

$$(\exists x, y \mid R \bullet P) \quad = \quad (\exists y, x \mid R \bullet P)$$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-09-29

Part 2: Predicate Logic 1

"Trading" Range Predicates with Body Predicates in $\forall$ and $\exists$

(9.2) Axiom, Trading: $\quad (\forall x \mid R \bullet P) \quad \equiv \quad (\forall x \bullet R \Rightarrow P)$

Trading Theorems for $\forall$:

(9.3a)	$(\forall x \mid R \bullet P) \quad \equiv \quad (\forall x \bullet \neg R \vee P)$
(9.3b)	$(\forall x \mid R \bullet P) \quad \equiv \quad (\forall x \bullet R \wedge P \equiv R)$
(9.3c)	$(\forall x \mid R \bullet P) \quad \equiv \quad (\forall x \bullet R \vee P \equiv P)$
(9.4a)	$(\forall x \mid Q \wedge R \bullet P) \quad \equiv \quad (\forall x \mid Q \bullet R \Rightarrow P)$
(9.4b)	$(\forall x \mid Q \wedge R \bullet P) \quad \equiv \quad (\forall x \mid Q \bullet \neg R \vee P)$
(9.4c)	$(\forall x \mid Q \wedge R \bullet P) \quad \equiv \quad (\forall x \mid Q \bullet R \wedge P \equiv R)$
(9.4d)	$(\forall x \mid Q \wedge R \bullet P) \quad \equiv \quad (\forall x \mid Q \bullet R \vee P \equiv P)$

(9.17) **Axiom, Generalised De Morgan:** $\quad (\exists x \mid R \bullet P) \quad \equiv \quad \neg(\forall x \mid R \bullet \neg P)$

(9.19) Trading for $\exists$: $\quad (\exists x \mid R \bullet P) \quad \equiv \quad (\exists x \bullet R \wedge P)$

(9.20) **Trading for $\exists$:** $\quad (\exists x \mid Q \wedge R \bullet P) \equiv (\exists x \mid Q \bullet R \wedge P)$

Using Instantiation for $\forall$

(9.13) **Instantiation:** $\quad (\forall x \bullet P) \quad \Rightarrow \quad P[x := E]$

A sharp version of Instantiation obtained via (3.60): $\quad (\forall x \bullet P) \equiv (\forall x \bullet P) \wedge P[x := E]$

Proving $(\forall x \bullet x + 1 > x) \quad \Rightarrow \quad y + 2 > y$:

$\quad (\forall x \bullet x + 1 > x)$

$=\ \langle$ **Instantiation (9.13) with (3.60)** $\rangle$

$\quad (\forall x \bullet x + 1 > x) \quad \wedge \quad y + 1 > y$

$\Rightarrow\ \langle$ **Left-Monotonicity of $\wedge$ (4.3) with Instantiation (9.13)** $\rangle$

$\quad (y + 1) + 1 > y + 1 \quad \wedge \quad y + 1 > y$

$\Rightarrow\ \langle$ Transitivity of $>$ (15.41) $\rangle$

$\quad y + 1 + 1 > y$

$=\ \langle 1 + 1 = 2 \rangle$

$\quad y + 2 > y$

Variable Binding Rearrangements

(8.19) **Axiom, Interchange of dummies:**

$$(\star x \mid R \bullet (\star y \mid S \bullet P)) \quad = \quad (\star y \mid S \bullet (\star x \mid R \bullet P))$$

provided $\neg occurs(\text{'}y\text{'}, \text{'}R\text{'})$ and $\neg occurs(\text{'}x\text{'}, \text{'}S\text{'})$, and each quantification is defined.

(8.20) **Axiom, Nesting:**

$$(\star x, y \mid R \wedge S \bullet P) \quad = \quad (\star x \mid R \bullet (\star y \mid S \bullet P))$$

provided $\neg occurs(\text{'}y\text{'}, \text{'}R\text{'})$.

(8.21) **Axiom, Dummy renaming** (α-conversion):

$$(\star x \mid R \bullet P) \quad = \quad (\star y \mid R[x := y] \bullet P[x := y])$$

provided $\neg occurs(\text{'}y\text{'}, \text{'}R, P\text{'})$.

Substitution* (8.11) *prevents capture of y by binders in R or P

Proving Split-off Term

We have:

(8.16) **Axiom, Range Split:**

$$(\Sigma x \mid R \vee S \bullet P) = (\Sigma x \mid R \bullet P) + (\Sigma x \mid S \bullet P)$$

provided $R \wedge S = \mathit{false}$ and each sum is defined.

How can you prove theorems like the following?

```
Theorem "Split off term" "Split off term at top":
    (∑ i : ℕ ❙ i < suc n • E) = (∑ i : ℕ ❙ i < n • E) + E[i = n]
```

- Use range split first —
  $\Longrightarrow$ need to transform the LHS range expression $i < suc\ n$ into an appropriate disjunction
  $\Longrightarrow$ the first disjunct should be the range expression $i < n$ from the RHS
- The second range will have one element
  $\Longrightarrow$ The second sum from the (8.16) RHS has range $i = n$
  $\Longrightarrow$ That second sum disappears via the **one-point rule**

Generalising De Morgan to Quantification

$\quad \neg(\exists i \mid 0 \le i < 4 \bullet P)$

$=\ \langle$ Expand quantification $\rangle$

$\quad \neg(P[i := 0] \vee P[i := 1] \vee P[i := 2] \vee P[i := 3])$

$=\ \langle$ (3.47) De Morgan $\rangle$

$\quad \neg P[i := 0] \wedge \neg P[i := 1] \wedge \neg P[i := 2] \wedge \neg P[i := 3]$

$=\ \langle$ Contract quantification $\rangle$

$\quad (\forall i \mid 0 \le i < 4 \bullet \neg P)$

(9.18b,c,a) **Generalised De Morgan**:

$$\neg(\exists x \mid R \bullet P) \quad \equiv \quad (\forall x \mid R \bullet \neg P)$$
$$(\exists x \mid R \bullet \neg P) \quad \equiv \quad \neg(\forall x \mid R \bullet P)$$
$$\neg(\exists x \mid R \bullet \neg P) \quad \equiv \quad (\forall x \mid R \bullet P)$$

(9.17) Axiom, **Generalised De Morgan**:

$$(\exists x \mid R \bullet P) \quad \equiv \quad \neg(\forall x \mid R \bullet \neg P)$$

Instantiation for $\forall$

$\quad P[x := E]$

$\equiv\ \langle$ (8.14) One-point rule $\rangle$

$\quad (\forall x \mid x = E \bullet P)$

$\Leftarrow\ \langle$ (9.10) Range weakening for $\forall$ $\rangle$

$\quad (\forall x \mid \mathit{true} \vee x = E \bullet P)$

$\equiv\ \langle$ (3.29) Zero of $\vee$ $\rangle$

$\quad (\forall x \mid \mathit{true} \bullet P)$

$\equiv\ \langle$ *true* range in quantification $\rangle$

$\quad (\forall x \bullet P)$

$$\frac{\forall x \bullet P}{P[x := E]}\ \forall\text{-Elim}$$

This proves: (9.13) **Instantiation:** $(\forall x \bullet P) \quad \Rightarrow \quad P[x := E]$

The one-point rule is **"sharper"** than Instantiation.

Using sharper rules often means fewer dead ends…

A sharp version obtained via (3.60):

$$(\forall x \bullet P) \quad \equiv \quad (\forall x \bullet P) \wedge P[x := E]$$

Recall: `with`$_2$

$\quad \neg(a \cdot b = a \cdot 0)$

$\equiv\langle$ "Cancellation of $\cdot$" with Assumption \`$a \neq 0$\` $\rangle$

$\quad \neg(b = 0)$

In a hint of shape "*HintItem1* `with` *HintItem2* `and` *HintItem3*":

- If *HintItem1* refers to a theorem of shape $p \Rightarrow q$,
- then *HintItem2* and *HintItem3* are used to prove p
- and q is used in the surrounding proof.

Here:

- *HintItem1* is "Cancellation of $\cdot$": $\quad z \neq 0 \Rightarrow (z \cdot x = z \cdot y \quad \equiv \quad x = y)$
- *HintItem2* is "Assumption $a \neq 0$"
- The surrounding proof uses: $\quad a \cdot b = a \cdot 0 \quad \equiv \quad b = 0$

Monotonicity `with` ...

$(\forall x \bullet x+1>x) \quad \wedge \quad y+1>y$

$\Rightarrow$ ⟨ **Left-Monotonicity of ∧ (4.3) with Instantiation (9.13)** ⟩

$(y+1)+1>y+1 \quad \wedge \quad y+1>y$

In a hint of shape "*HintItem1* `with` *HintItem2* `and` *HintItem3*":

- If *HintItem1* refers to a theorem of shape $p \Rightarrow q$,
- then *HintItem2* and *HintItem3* are used to prove p
- and q is used in the surrounding proof.

Here:

- *HintItem1* is "Left-Monotonicity of ∧": $(p \Rightarrow q) \Rightarrow ((p \wedge r) \Rightarrow (q \wedge r))$
- *HintItem2* is "Instantiation": $(\forall x \bullet x+1>x)$ $\Rightarrow (y+1)+1>y+1$
- The surrounding proof uses: $(\forall x \bullet x+1>x) \quad \wedge \quad y+1>y$ $\Rightarrow (y+1)+1>y+1 \quad \wedge \quad y+1>y$

`with`$_3$: Rewriting Theorems before Rewriting

ThmA `with` *ThmB*

- If *ThmB* gives rise to an equality/equivalence $L = R$: Rewrite *ThmA* with $L \mapsto R$
- E.g.: Assumption \`$p \Rightarrow q$\` with (3.60) \`$p \Rightarrow q \equiv p \wedge q \equiv q$\`

  The local theorem $p \Rightarrow q$ (resulting from the Assumption)

  rewrites via: $p \Rightarrow q \mapsto p \equiv p \wedge q$ (from (3.60))

  to: $p \equiv p \wedge q$

  which can be used for the rewrite: $p \mapsto p \wedge q$

Theorem (4.3) "Left-monotonicity of ∧": $(p \Rightarrow q) \Rightarrow ((p \wedge r) \Rightarrow (q \wedge r))$

Proof:

Assuming \`$p \Rightarrow q$\`:

$p \wedge r$

$\equiv$ ⟨ Assumption \`$p \Rightarrow q$\` with "Definition of ⇒ from ∧" ⟩

$p \wedge q \wedge r$

$\Rightarrow$ ⟨ "Weakening" ⟩

$q \wedge r$

Using Instantiation for ∀

(9.13) **Instantiation:** $(\forall x \bullet P) \Rightarrow P[x := E]$

A sharp version of Instantiation obtained via (3.60): $(\forall x \bullet P) \equiv (\forall x \bullet P) \wedge P[x := E]$

Theorem: $(\forall x : \mathbb{Z} \bullet x < x + 1) \Rightarrow y < y + 2$

Proof:

$(\forall x : \mathbb{Z} \bullet x < x + 1)$

$\equiv$ ⟨ "Instantiation" (9.13) with "Definition of ⇒ via ∧" (3.60) — explicit substitution needed! ⟩

$(\forall x : \mathbb{Z} \bullet x < x + 1) \wedge (x < x + 1)[x := y + 1]$

$\equiv$ ⟨ Substitution, Fact \`$1 + 1 = 2$\` ⟩

$(\forall x : \mathbb{Z} \bullet x < x + 1) \wedge y + 1 < y + 2$

$\Rightarrow$ ⟨ "Monotonicity of ∧" with "Instantiation" ⟩

$(x < x + 1)[x := y] \wedge y + 1 < y + 2$

$\equiv$ ⟨ Substitution ⟩

$y < y + 1 \wedge y + 1 < y + 2$

$\Rightarrow$ ⟨ "Transitivity of <" ⟩

$y < y + 2$

Theorems and Universal Quantification

(9.16) **Metatheorem**: P is a theorem iff $(\forall x \bullet P)$ is a theorem.

This is another justification for **implicit use of "Instantiation" (9.13)** $(\forall x \bullet P) \Rightarrow P[x := E]$:

Theorem: $(\forall x : \mathbb{Z} \bullet x < x + 1) \Rightarrow y < y + 2$

Proof:

Assuming (1) \`$\forall x : \mathbb{Z} \bullet x < x + 1$\`:

y

$<$ ⟨ Assumption (1) — implicit instantiation with `E := y` ⟩

$y + 1$

$<$ ⟨ Assumption (1) — implicit instantiation with `E := y + 1` ⟩

$y + 1 + 1$

$=$ ⟨ Fact \`$1 + 1 = 2$\` ⟩

$y + 2$

Implicit Universal Quantification in Theorems 1

(9.16) **Metatheorem**: P is a theorem iff $(\forall x \bullet P)$ is a theorem.

(If proving "$x+1>x$" is considered to *really mean* proving "$\forall x \bullet x+1>x$", then the x in "$x+1>x$" is called *implicitly universally quantified*.)

Proof method: To prove $(\forall x \bullet P)$, we prove P for arbitrary x.

That is really a prose version of the following **inference rule:**

$$\frac{P}{\forall x \bullet P} \ \forall\text{-Intro} \quad \text{(prov. } x \text{ not free in assumptions)}$$

In CalcCheck:

- Proving $(\forall v : \mathbb{N} \bullet P)$: **For any** '$v : \mathbb{N}$': *Proof for P* (Non-local assumptions with free v are not usable.)

Using "For any" for "Proof by Generalisation"

In CalcCheck:

- Proving $(\forall v : \mathbb{N} \bullet P)$: **For any** '$v : \mathbb{N}$': *Proof for P*

Proving $\forall x : \mathbb{N} \bullet x < x + 1$**:**

For any \`$x : \mathbb{N}$\`:

$x < x + 1$

$\equiv$ ⟨ Identity of + ⟩

$x + 0 < x + 1$

$\equiv$ ⟨ Cancellation of + ⟩

$0 < 1$

$\equiv$ ⟨ Fact \`$1 = suc\ 0$\` ⟩

$0 < suc\ 0$

$\equiv$ ⟨ Zero is less than successor ⟩

true

Implicit Universal Quantification in Theorems 2

(9.16) **Metatheorem**: P is a theorem iff $(\forall x \bullet P)$ is a theorem.

LADM Proof method: To prove $(\forall x \mid R \bullet P)$, we prove P for arbitrary x in range R.

That is:

- Assume R to prove P (and assume nothing else that mentions x)
- This proves $R \Rightarrow P$
- Then, by (9.16), $(\forall x \bullet R \Rightarrow P)$ is a theorem.
- With (9.2) Trading for ∀, this is transformed into $(\forall x \mid R \bullet P)$.

In CalcCheck:

- Proving $(\forall v : \mathbb{N} \bullet P)$: **For any** '$v : \mathbb{N}$': *Proof for P*
- Proving $(\forall v : \mathbb{N} \mid R \bullet P)$: **For any** '$v : \mathbb{N}$' **satisfying** '$R$': *Proof for P using* **Assumption** 'R'

Using "For any ... satisfying" for "Proof by Generalisation"

In CalcCheck:

- Proving $(\forall v : \mathbb{N} \mid R \bullet P)$: **For any** '$v : \mathbb{N}$' **satisfying** '$R$': *Proof for P using* **Assumption** 'R'

Proving $\forall x : \mathbb{N} \mid x < 2 \bullet x < 3$**:**

For any \`$x : \mathbb{N}$\` satisfying \`$x < 2$\`:

x

$<$ ⟨ Assumption \`$x < 2$\` ⟩

2

$<$ ⟨ Fact \`$2 < 3$\` ⟩

3

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-02

Predicate Logic (2)

Warm-Up

- What does "assuming the antecedent" mean?
- Give the rule for quantification nesting.
- State the one-point rule and the empty range axiom.
- State the quantification distributivity axiom.
- Give the rule for disjoint range split.
- Give the rule for substitution into quantification.
- State the basic trading laws for ∀ and ∃.
- State the theorem of instantiation for ∀.

Plan for Today

- **Predicate Logic 2**: Selected Important Properties of Universal and Existential Quantifications (LADM chapter 9)

Coming up:

- Types (see also LADM section 8.1) and Sets (LADM chapter 11)

Combined Quantification Examples

- "There is a least integer."
- "There exists an integer b such that every integer n is at least b".
- "There exists an integer b such that for every integer n, we have $b \le n$".

$(\exists\, b : \mathbb{Z} \bullet (\forall\, n : \mathbb{Z} \bullet b \le n))$

- "π can be enclosed within rational bounds that are less than any ε apart"
- "For every positive real number ε, there are rational numbers r and s with $r < s < r + \varepsilon$, such that $r < \pi < s$"

$(\forall\, \varepsilon : \mathbb{R} \mid 0 < \varepsilon$
$\quad \bullet (\exists\, r, s : \mathbb{Q} \mid r < s < r + \varepsilon \bullet r < \pi < s))$

Proof Patterns Corresponding to the Elimination and Introduction Rules for ∀

$\dfrac{\forall\, x \bullet P}{P[x := E]}$ ∀-Elim $\qquad \dfrac{P}{\forall\, x \bullet P}$ ∀-Intro (prov. x not free in assumptions)

(9.13) **Instantiation:** $(\forall\, x \bullet P) \quad \Rightarrow \quad P[x := E]$

$\quad y + 2$
$<$ ⟨ Assumption \`$\forall\, x : \mathbb{Z} \bullet x < x + 1$\` — implicit instantiation w. $E := y + 2$ ⟩
$\quad y + 2 + 1$

$\quad (\forall\, x : \mathbb{Z} \bullet x < x + 1)$
$\equiv$ ⟨ "Instantiation" (9.13) with "Definition of ⇒ via ∧" (3.60) — explicit substitution needed! ⟩
$\quad (\forall\, x : \mathbb{Z} \bullet x < x + 1) \wedge (x < x + 1)[x := y + 1]$

- Proving $(\forall\, v : \mathbb{N} \bullet P)$: **For any \`$v : \mathbb{N}$\`:** *Proof for P* (Non-local assumptions with free v are not usable.)
- Proving $(\forall\, v : \mathbb{N} \mid R \bullet P)$: **For any \`$v : \mathbb{N}$\` satisfying \`$R$\`:** *Proof for P using* **Assumption** \`$R$\`

∃-Introduction

Recall: (9.13) **Instantiation:** $(\forall\, x \bullet P) \quad \Rightarrow \quad P[x := E]$

Dual: (9.28) **∃-Introduction:** $P[x := E] \quad \Rightarrow \quad (\exists\, x \bullet P)$

An expression E with $P[x := E]$ is called a "**witness**" of $(\exists\, x \bullet P)$.

Proving an existential quantification via ∃-Introduction requires "**exhibiting a witness**".

Inference rule:

$\dfrac{P[x := E]}{\exists\, x \bullet P}$ ∃-Intro $\qquad \dfrac{\forall\, x \bullet P}{P[x := E]}$ ∀-Elim

Using ∃-Introduction for "Proof by Example"

(9.28) **∃-Introduction:** $P[x := E] \quad \Rightarrow \quad (\exists\, x \bullet P)$

An expression E with $P[x := E]$ is called a "**witness**" of $(\exists\, x \bullet P)$.

Proving an existential quantification via ∃-Introduction requires "**exhibiting a witness**".

$\quad (\exists\, x : \mathbb{N} \bullet x \cdot x < x + x)$
$\Leftarrow$ ⟨ ∃-Introduction ⟩
$\quad (x \cdot x < x + x)[x := 1]$
$\equiv$ ⟨ Substitution ⟩
$\quad 1 \cdot 1 < 1 + 1$
$\equiv$ ⟨ Evaluation ⟩
$\quad$ *true*

Using ∃-Introduction for "Proof by Counter-Example"

(9.28) **∃-Introduction:** $P[x := E] \quad \Rightarrow \quad (\exists\, x \bullet P)$

$\quad \neg(\forall\, x : \mathbb{N} \bullet x + x < x \cdot x)$
$\equiv$ ⟨ Generalised De Morgan ⟩
$\quad (\exists\, x : \mathbb{N} \bullet \neg(x + x < x \cdot x))$
$\Leftarrow$ ⟨ ∃-Introduction ⟩
$\quad (\neg(x + x < x \cdot x))[x := 2]$
$\equiv$ ⟨ Substitution ⟩
$\quad \neg(2 + 2 < 2 \cdot 2)$
$\equiv$ ⟨ Fact \`$2 + 2 < 2 \cdot 2 \equiv false$\` ⟩
$\quad \neg$*false*
$\equiv$ ⟨ Negation of *false* ⟩
$\quad$ *true*

Witnesses

(9.30v) **Metatheorem Witness**: If $\neg occurs(`x', `Q')$, then:

$(\exists\, x \mid R \bullet P) \Rightarrow Q$ is a theorem iff $(R \wedge P) \Rightarrow Q$ is a theorem

Theorem "Witness": $(\exists\, x \mid R \bullet P) \Rightarrow Q \quad \equiv \quad (\forall\, x \bullet R \wedge P \Rightarrow Q)$ prov. $\neg occurs(`x', `Q')$
Proof:

$\quad (\exists\, x \mid R \bullet P) \Rightarrow Q$
$=$ ⟨ (9.19) Trading for ∃ ⟩
$\quad (\exists\, x \bullet R \wedge P) \Rightarrow Q$
$=$ ⟨ (3.59) $p \Rightarrow q \equiv \neg p \vee q$, (9.18b) Gen. De Morgan ⟩
$\quad (\forall\, x \bullet \neg(R \wedge P)) \vee Q$
$=$ ⟨ (9.5) Distributivity of ∨ over ∀ — $\neg occurs(`x', `Q')$ ⟩
$\quad (\forall\, x \bullet \neg(R \wedge P) \vee Q)$
$=$ ⟨ (3.59) $p \Rightarrow q \equiv \neg p \vee q$ ⟩
$\quad (\forall\, x \bullet R \wedge P \Rightarrow Q)$

The last line is, by Metatheorem (9.16), a theorem iff $(R \wedge P) \Rightarrow Q$ is.

LADM Theory of Integers — Axioms and Some Theorems

(15.1) **Axiom, Associativity:** $(a + b) + c = a + (b + c)$
$\quad (a \cdot b) \cdot c = a \cdot (b \cdot c)$

(15.2) **Axiom, Symmetry:** $a + b = b + a$
$\quad a \cdot b = b \cdot a$

(15.3) **Axiom, Additive identity:** $0 + a = a$

(15.4) **Axiom, Multiplicative identity:** $1 \cdot a = a$

(15.5) **Axiom, Distributivity:** $a \cdot (b + c) = a \cdot b + a \cdot c$

(15.6) **Axiom, Additive Inverse:** $(\exists x \bullet x + a = 0)$

(15.7) **Axiom, Cancellation of ·:** $c \ne 0 \Rightarrow (c \cdot a = c \cdot b \equiv a = b)$

(15.8) **Cancellation of +:** $a + b = a + c \quad \equiv \quad b = c$

(15.10b) **Unique mult. identity:** $a \ne 0 \Rightarrow (a \cdot z = a \equiv z = 1)$

(15.12) **Unique additive inverse:** $x + a = 0 \wedge y + a = 0 \Rightarrow x = y$

"Witness":
$\quad (\exists\, x \mid R \bullet P) \Rightarrow Q$
$\equiv \quad (\forall\, x \bullet R \wedge P \Rightarrow Q)$
$\quad$ prov. $\neg occurs(`x', `Q$

(15.6) **Additive Inverse:**
$\quad (\exists\, x \bullet x + a = 0)$

(15.8) **Cancellation of** +:
$\quad a + b = a + c \quad \equiv \quad b = c$

```
Theorem (15.8) “Cancellation of +”: a + b = a + c  ≡  b = c
Proof:
  Using “Mutual implication”:
    Subproof for `b = c  ⇒  a + b = a + c`:
      Assuming `b = c`:
          a + b
        =⟨ Assumption `b = c` ⟩
          a + c
    Subproof for `a + b = a + c  ⇒  b = c`:
          a + b = a + c  ⇒  b = c
        ≡⟨ “Left-identity of ⇒”, “Additive inverse” with `a ≔ a` ⟩
          (∃ x : ℤ • x + a = 0)  ⇒  a + b = a + c  ⇒  b = c
        ≡⟨ “Witness”, “Trading for ∀” ⟩
          ∀ x : ℤ ❙ x + a = 0  •  a + b = a + c  ⇒  b = c
      Proof for this:
        For any `x : ℤ` satisfying `x + a = 0`:
          Assuming `a + b = a + c`:
              b
            =⟨ “Identity of +” ⟩
              0 + b
            =⟨ Assumption `x + a = 0` ⟩
              x + a + b
            =⟨ Assumption `a + b = a + c` ⟩
              x + a + c
            =⟨ Assumption `x + a = 0` ⟩
              0 + c
            =⟨ “Identity of +” ⟩
              c
```

(15.6) Additive Inverse
$\quad (\exists\, x \bullet x + a = 0)$

$$\frac{(\exists x \bullet P) \qquad \begin{matrix} \ulcorner P \urcorner \\ \vdots \\ R \end{matrix}}{R}$$ ∃-Elim (prov. x not free in R, assumptions)

```
Theorem (15.8) “Cancellation of +”: a + b = a + c  ≡  b = c
Proof:
  Using “Mutual implication”:
    Subproof for `b = c  ⇒  a + b = a + c`:
      Assuming `b = c`:
          a + b
        =⟨ Assumption `b = c` ⟩
          a + c
    Subproof for `a + b = a + c  ⇒  b = c`:
          a + b = a + c  ⇒  b = c
        ≡⟨ “Left-identity of ⇒”, “Additive inverse” ⟩
          (∃ x : ℤ • x + a = 0)  ⇒  a + b = a + c  ⇒  b = c
      Proof for this:
        Assuming witness `x : ℤ` satisfying `x + a = 0`:
          Assuming `a + b = a + c`:
              b
            =⟨ “Identity of +” ⟩
              0 + b
            =⟨ Assumption `x + a = 0` ⟩
              x + a + b
            =⟨ Assumption `a + b = a + c` ⟩
              x + a + c
            =⟨ Assumption `x + a = 0` ⟩
              0 + c
            =⟨ “Identity of +” ⟩
              c
```

New Proof Strutures: Assuming witness

Assuming witness \`$x${ : *type*}$^?$\` **satisfying** \`$P$\` :

- introduces the bound variable ‘x’
- makes P available as assumption to the contained proof.
- This proves $(\exists x : type \bullet P) \Rightarrow R$
  if the contained proof proves R,

Assuming witness \`$x${: *type*}$^?$\` **satisfying** \`$P$\` **by** *hint* :

- introduces the bound variable ‘x’
- makes P available as assumption to the contained proof.
- *hint* needs to prove $(\exists x : type \bullet P)$
- This then proves R
  if the contained proof proves R
  (with the additional assumnption P)
- This can be understood as providing ∃-elimination:
  It uses *hint* to discharge the antecedent $(\exists x : type \bullet P)$
  and then has inferred proof goal R.

$$\frac{(\exists x \bullet P) \qquad \begin{matrix}\ulcorner P \urcorner \\ \vdots \\ R\end{matrix}}{R}\ \exists\text{-Elim}$$

(prov. x not free in R, assumptions)

(15.6) Additive Inverse
$(\exists x \bullet x + a = 0)$

$$\frac{(\exists x \bullet P) \qquad \begin{matrix}\ulcorner P \urcorner \\ \vdots \\ R\end{matrix}}{R}\ \exists\text{-Elim}$$

(prov. x not free in R, assumptions)

```
Theorem (15.8) “Cancellation of +”: a + b = a + c  ≡  b = c
Proof:
  Using “Mutual implication”:
    Subproof for `b = c  ⇒  a + b = a + c`:
      Assuming `b = c`:
          a + b
        =⟨ Assumption `b = c` ⟩
          a + c
    Subproof for `a + b = a + c  ⇒  b = c`:
      Assuming witness `x : ℤ` satisfying `x + a = 0`
          by “Additive inverse”:
        Assuming `a + b = a + c`:
            b
          =⟨ “Identity of +” ⟩
            0 + b
          =⟨ Assumption `x + a = 0` ⟩
            x + a + b
          =⟨ Assumption `a + b = a + c` ⟩
            x + a + c
          =⟨ Assumption `x + a = 0` ⟩
            0 + c
          =⟨ “Identity of +” ⟩
            c
```

Recall: Monotonicity With Respect To ⇒

Let $\_\le\_$ be an order on T, and let $f : T \to T$ be a function on T. Then f is called

- **monotonic** iff $x \le y \quad \Rightarrow \quad f\ x \le f\ y$,
- **antitonic** iff $x \le y \quad \Rightarrow \quad f\ y \le f\ x$.

(4.2) Left-Monotonicity of ∨: $(p \Rightarrow q) \Rightarrow (p \lor r \Rightarrow q \lor r)$

(4.3) Left-Monotonicity of ∧: $(p \Rightarrow q) \Rightarrow p \land r \Rightarrow q \land r$

Antitonicity of ¬: $(p \Rightarrow q) \Rightarrow \neg q \Rightarrow \neg p$

Left-Antitonicity of ⇒: $(p \Rightarrow q) \Rightarrow (q \Rightarrow r) \Rightarrow (p \Rightarrow r)$

Right-Monotonicity of ⇒: $(p \Rightarrow q) \Rightarrow (r \Rightarrow p) \Rightarrow (r \Rightarrow q)$

Guarded Right-Monotonicity of ⇒: $(r \Rightarrow (p \Rightarrow q)) \Rightarrow (r \Rightarrow p) \Rightarrow (r \Rightarrow q)$

Transitivity Laws are Monotonicity Laws

Notice: The following two “are” transitivity of ⇒:

- **Left-Antitonicity of ⇒:** $(p \Rightarrow q) \Rightarrow (q \Rightarrow r) \Rightarrow (p \Rightarrow r)$
- **Right-Monotonicity of ⇒:** $(p \Rightarrow q) \Rightarrow (r \Rightarrow p) \Rightarrow (r \Rightarrow q)$

This works also for other orders — with general monotonicity: Let

- $\_\le_1\_$ be an order on T_1, and $\_\le_2\_$ be an order on T_2,
- $f : T_1 \to T_2$ be a function from T_1 to T_2.

Then f is called

- **monotonic** iff $x \le_1 y \quad \Rightarrow \quad f\ x \le_2 f\ y$,
- **antitonic** iff $x \le_1 y \quad \Rightarrow \quad f\ y \le_2 f\ x$.

Transitivity of ≤ is antitonitcity of $(\_ \le r) : \mathbb{Z} \to \mathbb{B}$:

- **Left-Antitonicity of ≤:** $(p \le q) \Rightarrow (q \le r) \Rightarrow (p \le r)$
- **Right-Monotonicity of ≤:** $(p \le q) \Rightarrow (r \le p) \Rightarrow (r \le q)$

Weakening/Strengthening for ∀ and ∃ — “Cheap Antitonicity/Monotonicity”

(9.10) **Range** weakening/**strengthening for ∀:** $(\forall x \mid Q \lor R \bullet P) \Rightarrow (\forall x \mid Q \bullet P)$

(9.11) **Body weakening**/strengthening **for ∀:** $(\forall x \mid R \bullet P \land Q) \Rightarrow (\forall x \mid R \bullet P)$

(9.25) **Range weakening**/strengthening **for ∃:** $(\exists x \mid R \bullet P) \Rightarrow (\exists x \mid Q \lor R \bullet P)$

(9.26) **Body weakening**/strengthening **for ∃:** $(\exists x \mid R \bullet P) \Rightarrow (\exists x \mid R \bullet P \lor Q)$

Recall:

(9.2) **Trading for ∀:** $(\forall x \mid R \bullet P) \equiv (\forall x \bullet R \Rightarrow P)$

(9.19) **Trading for ∃:** $(\exists x \mid R \bullet P) \equiv (\exists x \bullet R \land P)$

Monotonicity for ∀

(9.12) **Monotonicity of ∀:**

$$(\forall x \mid R \bullet P_1 \Rightarrow P_2) \Rightarrow ((\forall x \mid R \bullet P_1) \Rightarrow (\forall x \mid R \bullet P_2))$$

Range-Antitonicity of ∀:

$$(\forall x \bullet R_2 \Rightarrow R_1) \Rightarrow ((\forall x \mid R_1 \bullet P) \Rightarrow (\forall x \mid R_2 \bullet P))$$

$(\forall x \bullet R_2 \Rightarrow R_1)$
⇒ ⟨ (9.12) with shunted (3.82a) Transitivity of ⇒ ⟩
$(\forall x \bullet (R_1 \Rightarrow P) \Rightarrow (R_2 \Rightarrow P))$
⇒ ⟨ (9.12) Monotonicity of ∀ ⟩
$(\forall x \bullet R_1 \Rightarrow P) \Rightarrow (\forall x \bullet R_2 \Rightarrow P)$
= ⟨ (9.2) Trading for ∀ ⟩
$(\forall x \mid R_1 \bullet P) \Rightarrow (\forall x \mid R_2 \bullet P)$

Monotonicity for ∃

(9.27) (Body) **Monotonicity of ∃:**

$$(\forall x \mid R \bullet P_1 \Rightarrow P_2) \Rightarrow ((\exists x \mid R \bullet P_1) \Rightarrow (\exists x \mid R \bullet P_2))$$

Range-Monotonicity of ∃:

$$(\forall x \bullet R_1 \Rightarrow R_2) \Rightarrow ((\exists x \mid R_1 \bullet P) \Rightarrow (\exists x \mid R_2 \bullet P))$$

Predicate Logic Laws You Really Need To Know Already Now

(8.13) **Empty Range**: $(\forall x \mid false \bullet P) = true$
$(\exists x \mid false \bullet P) = false$

(8.14) **One-point Rule:** Provided ¬*occurs*(‘x’, ‘E’), $(\forall x \mid x = E \bullet P) \equiv P[x := E]$
$(\exists x \mid x = E \bullet P) \equiv P[x := E]$

(9.17) **Generalised De Morgan**: $(\exists x \mid R \bullet P) \equiv \neg(\forall x \mid R \bullet \neg P)$

(9.2) **Trading for ∀**: $(\forall x \mid R \bullet P) \equiv (\forall x \bullet R \Rightarrow P)$

(9.4a) **Trading for ∀**: $(\forall x \mid Q \land R \bullet P) \equiv (\forall x \mid Q \bullet R \Rightarrow P)$

(9.19) **Trading for ∃**: $(\exists x \mid R \bullet P) \equiv (\exists x \bullet R \land P)$

(9.20) **Trading for ∃**: $(\exists x \mid Q \land R \bullet P) \equiv (\exists x \mid Q \bullet R \land P)$

(9.13) **Instantiation**: $(\forall x \bullet P) \Rightarrow P[x := E]$

(9.28) **∃-Introduction**: $P[x := E] \Rightarrow (\exists x \bullet P)$

…and correctly handle substitution, Leibniz, renaming of bound variables, monotonicity/antitonicity, For any …

Sentences: Predicate Logic Formulae without Free Variables

Definition: A sentence is a Boolean expression without free variables.

- Expressions without free variables are also called “closed”:
  A sentence is a closed Boolean expression.
- Recall: The value of an expression (in a state) only depends on its free variables.
- Therefore: **The value of a closed expression does not depend on the state.**
- That is, a closed Boolean expression, or sentence,
  - either always evaluates to *true*
  - or always evaluates to *false*
- In other words: A closed Boolean expression, or sentence,
  - is either valid
  - or a contradiction
- Also: For a closed Boolean expression, or sentence, φ
  - either φ is valid
  - or $\neg\varphi$ is valid
- This means: For a closed Boolean expression, or sentence, φ,
  only one of φ and $\neg\varphi$ can have a proof!

2018 Midterm 2

Prove one of the following two theorem statements — **only one is valid.** (Should be easy in less than ten steps.)

```
Theorem “M2-3A-1-yes”: (∃ x : ℤ • ∀ y : ℤ • (x - 2) · y + 1 = x - 1)
Theorem “M2-3A-1-no”: ¬ (∃ x : ℤ • ∀ y : ℤ • (x - 2) · y + 1 = x - 1)
```

- For a closed Boolean expression, or sentence, φ,
  only one of φ and $\neg\varphi$ can have a proof!

- “Practice with ∀ and ∃” starts with H12.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-04

Sequences, Types, Sets

Warm-Up

- What is an order?
- What does "assuming the antecedent" mean?
- Give the rule for quantification nesting.
- State the one-point rule and the empty range axiom.
- State the quantification distributivity axiom.
- Give the rule for disjoint range split.
- Give the rule for substitution into quantification.
- State the basic trading laws for ∀ and ∃.
- State the theorem of instantiation for ∀.
- State the ∃-introduction theorem.
- State monotonicity and antitonicity theorems for ∀ and ∃.
- What can you prove with "For any `$x : T$` satisfying `$R$`:"?

Plan for Today

- Sequences — a brief start (LADM chapter 13)
- Some remarks about Types (see also LADM section 8.1)
- "A Theory of Sets" (LADM chapter 11)

Coming up:

- Relations (see also LADM chapter 14)

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-04

Part 1: Sequences

Sequences

- We may write $[33, 22, 11]$ (Haskell notation) for the sequence that has
  - "33" as its first element,
  - "22" as its second element,
  - "11" as its third element, and
  - no further elements.

  (Notation "[...]" for sequences is not supported by CalcCheck. LADM writes "⟨...⟩".)
- Sequence matters: $[33, 22, 11]$ and $[11, 22, 33]$ are different!
- Multiplicity matters: $[33, 22, 11]$ and $[33, 22, 22, 11]$ are different!
- We consider the type *Seq A* of sequences with elements of type *A* as generated inductively by the following two constructors:

ϵ	: *Seq A*	`\eps`	empty sequence
$\_\triangleleft\_$	: $A \to Seq\ A \to Seq\ A$	`\cons`	"cons"

  ◃ associates to the right.
- Therefore: $[33, 22, 11] = 33 \triangleleft [22, 11] = 33 \triangleleft 22 \triangleleft [11] = 33 \triangleleft 22 \triangleleft 11 \triangleleft \epsilon$

Sequences — "cons" and "snoc"

- We consider the type *Seq A* of sequences with elements of type *A* as generated inductively by the following two constructors:

ϵ	: *Seq A*	`\eps`	empty sequence
$\_\triangleleft\_$	: $A \to Seq\ A \to Seq\ A$	`\cons`	"cons"

  ◃ associates to the right.
- Therefore: $[33, 22, 11] = 33 \triangleleft [22, 11] = 33 \triangleleft 22 \triangleleft [11] = 33 \triangleleft 22 \triangleleft 11 \triangleleft \epsilon$
- Appending single elements "at the end":

$\_\triangleright\_$	: $Seq\ A \to A \to Seq\ A$	`\snoc`	"snoc"

  ▹ associates to the left.
- (Con-)catenation:

$\_\frown\_$	: $Seq\ A \to Seq\ A \to Seq\ A$	`\catenate`

  ⌢ associates to the right.

Sequences — Induction Principle

- The set of all **sequences over type *A*** is written *Seq A*.
- The empty sequence "ϵ" is a sequence over type *A*.
- If x is an element of A and xs is a sequence over type A, then "$x \triangleleft xs$" (pronounced: "x cons xs") is a sequence over type A, too.
- Two sequences are equal **iff** they are constructed the same way from ϵ and ◃.

Induction principle for sequences:

- if $P(\epsilon)$ — If P holds for ϵ
- and if $P(xs)$ implies $P(x \triangleleft xs)$ for all $x : A$, — and whenever P holds for xs, it also holds for any $x \triangleleft xs$,
- then for all $xs : Seq\ A$ we have $P(xs)$. — then P holds for all sequences over A.

Sequences — Induction Proofs

Induction principle for sequences:

- if $P(\epsilon)$ — If P holds for ϵ
- and if $P(xs)$ implies $P(x \triangleleft xs)$ **for all** $x : A$, — and whenever P holds for xs, it also holds for any $x \triangleleft xs$,
- then for all $xs : Seq\ A$ we have $P(xs)$. — then P holds for all sequences over A.

An **induction proof** using this looks as follows:

Theorem: P
Proof:
By induction on $xs : Seq\ A$**:**
Base case:
Proof for $P[xs := \epsilon]$
Induction step:
Proof for $(\forall x : A \bullet P[xs := x \triangleleft xs])$
using **Induction hypothesis** P

Concatenation

```
Axiom (13.17) "Left-identity of ⌢"
              "Definition of ⌢ for ϵ":            ϵ ⌢ ys = ys
Axiom (13.18) "Mutual associativity of ◃ with ⌢"
              "Definition of ⌢ for ◃":            (x ◃ xs) ⌢ ys = x ◃ (xs ⌢ ys)
```

⟹ H13, Ex5.2

(Work through H13 before your tutorial!)

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-04

Part 2: Types

Types

A **type** **denotes a set of values** that

- can be associated with a variable
- an expression might evaluate to

Some basic types: $\mathbb{B}, \mathbb{Z}, \mathbb{N}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$

Some constructed types: $\mathit{Seq}\ \mathbb{N}$, $\mathbb{N} \to \mathbb{B}$, $\mathit{Seq}\ (\mathit{Seq}\ \mathbb{N}) \to \mathit{Seq}\ \mathbb{B}$, **set** $\mathbb{Z}$

"E : t" means: "Expression E is declared to have type t".

Examples:

- constants: $\mathit{true} : \mathbb{B}$, $\pi : \mathbb{R}$, $2 : \mathbb{Z}$, $2 : \mathbb{N}$
- variable declarations: $p : \mathbb{B}$, $k : \mathbb{N}$, $d : \mathbb{R}$
- type annotations in expressions:
  - $(x + y) \cdot x \quad \longrightarrow \quad (x : \mathbb{N} + y) \cdot x$
  - $(x + y) \cdot x \quad \longrightarrow \quad ((((x : \mathbb{N}) + (y : \mathbb{N})) : \mathbb{N}) \cdot (x : \mathbb{N})) : \mathbb{N}$

Function Types — LADM Version

- If the parameters of function f have types $t_1, \dots, t_n$
- and the result has type r,
- then f has type $t_1 \times \dots \times t_n \to r$

We write: $f : t_1 \times \dots \times t_n \to r$

Examples: $\neg\_ : \mathbb{B} \to \mathbb{B}$ $\quad \_+\_ : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ $\quad \_<\_ : \mathbb{Z} \times \mathbb{Z} \to \mathbb{B}$

Forming expressions using $\_<\_ : \mathbb{Z} \times \mathbb{Z} \to \mathbb{B}$:

- if expression a_1 has type $\mathbb{Z}$, and a_2 has type $\mathbb{Z}$
- then $a_1 < a_2$ is a (well-typed) expression
- and has type $\mathbb{B}$.

In general: For $f : t_1 \times \dots \times t_n \to r$,

- if expression a_1 has type t_1, and …, and a_n has type t_n
- then function application $f(a_1, \dots, a_n)$ is an expression
- and has type r.

Function Types — Mechanised Mathematics Version

- If the parameters of function f have types $t_1, \dots, t_n$
- and the result has type r,
- then f has type $t_1 \to \dots \to t_n \to r$

$\Rightarrow$ **We write:** $f : t_1 \to \dots \to t_n \to r$

(The function type constructor $\to$ **associates to the right!**)

Examples: $\neg : \mathbb{B} \to \mathbb{B}$ $\quad \_+\_ : \mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$ $\quad \_<\_ : \mathbb{Z} \to \mathbb{Z} \to \mathbb{B}$

Forming expressions using $\_<\_ : \mathbb{Z} \to \mathbb{Z} \to \mathbb{B}$:

$$\frac{a_1 : \mathbb{Z} \qquad a_2 : \mathbb{Z}}{(a_1 < a_2) : \mathbb{B}}$$

In general: For $f : A \to B$,

- if expression x has type A,
- then function application $f\ x$ is an expression
- and has type B.

$$\frac{f : A \to B \qquad x : A}{f\ x : B}$$

Well-typed Expressions?

$2 + k$ ✓ $\quad 42 - \mathit{true}$ ✗ $\quad \neg(3 \cdot x)$ ✗ $\quad (1/(x : \mathbb{R})) : \mathbb{R}$ ✓

Non-well-typed expressions make no sense!

Function Application — LADM Version

Consider function g defined by: (1.6) $\quad g(z) = 3 \cdot z + 6$

- Special **function application** syntax for argument that is identifier or constant:

$$g.z = 3 \cdot z + 6$$

LADM Table of Precedences

- $[x := e]$ (textual substitution) **(highest precedence)**
- . **(function application)**
- unary prefix operators $+, -, \neg, \#, \sim, \mathcal{P}$
- $**$
- $\cdot \quad / \quad \div \quad \mathit{mod} \quad \mathit{gcd}$
- $+ \quad - \quad \cup \quad \cap \quad \times \quad \circ \quad \bullet$
- $\downarrow \quad \uparrow$
- $\#$
- $\triangleleft \quad \triangleright \quad \hat{}$
- $= \quad \neq \quad < \quad > \quad \in \quad \subset \quad \subseteq \quad \supset \quad \supseteq \quad |$ (conjunctional)
- $\vee \quad \wedge$
- $\Rightarrow \quad \not\Rightarrow \quad \Leftarrow \quad \not\Leftarrow$
- $\equiv \quad \not\equiv$ **(lowest precedence)**

All non-associative binary infix operators **associate to the left**, **except** $**$, $\triangleleft$, $\Rightarrow$, $\to$, which **associate to the right**.

Function Application — Mechanised Mathematics Version

Consider function g defined by: (1.6) $\quad g\ z = 3 \cdot z + 6$

- **Function application** is denoted by **juxtaposition** ("putting side by side")
- **Lexical separation** for argument that is identifier or constant: **space required:**

  $$h\ z = g\ (g\ z)$$

  Superfluous parentheses (e.g., "$h(z) = g(g(z))$") are allowed, **ugly**, and bad style.
- Function application still has **higher precedence than other binary operators**.
- As non-associative binary infix operator, function application **associates to the left:** If $f : \mathbb{Z} \to (\mathbb{Z} \to \mathbb{Z})$, then $f\ 2\ 3 = (f\ 2)\ 3$, and $f\ 2 : \mathbb{Z} \to \mathbb{Z}$
- Typing rule for function application:

$$\frac{f : A \to B \qquad x : A}{f\ x : B}$$

COMPSCI 2LC3 Fall 2023 CalcCheck Default Table of Precedences

- (∞): $\_[\_ := \_\,]$ (textual substitution) **(highest precedence)**
- 140: unary postfix operators: $\_!$ $\quad \_\breve{}$ $\quad \_^*$ $\quad \_^+$ $\quad \_(\!|\_|\!)$
- 130: unary prefix operators: $+\_$ $\quad -\_$ $\quad \neg\_$ $\quad \#\_$ $\quad \sim\_$ $\quad \mathbb{P}\_$ $\quad \mathit{suc}\_$
- 120: $\_\_$ (**function application**), $\quad$ @
- 115: $**$
- 110: $\cdot \quad / \quad \div \quad \mathit{mod} \quad \mathit{gcd}$
- 105: $⨾ \quad / \quad \backslash$
- 100: $+ \quad - \quad \cup \quad \cap \quad \times \quad \circ \quad \oplus \quad ⇨ \quad \triangleleft \quad ⩤ \quad \triangleright \quad ⩥$
- 97: $\leftrightarrow$ (relation type)
- 95: $\to$ (function type)
- 90: $\downarrow \quad \uparrow$
- 70: $\#$
- 60: $\triangleleft \quad \triangleright \quad \frown$
- 50: $= \quad \neq \quad < \quad > \quad \in \quad \subset \quad \subseteq \quad \supset \quad \supseteq \quad | \quad \_(\_)\_$ **(conjunctional)**
- 40: $\vee \quad \wedge$
- 20: $\Rightarrow \quad \not\Rightarrow \quad \Leftarrow \quad \not\Leftarrow$
- 10: $\equiv \quad \not\equiv$
- 9: **:=** (assignment command, two characters)
- 5: ; (command sequencing)
- $(-\infty)$: $\circledast\_\,|\,\_\bullet\_$ (quantification notation, for $\circledast \in \{\forall, \exists, \bigcup, \bigcap, \sum, \prod, \dots\}$) **(lowest precedence)**

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-04

Part 3: Sets

LADM Chapter 11: A Theory of Sets

"A *set* is simply a collection of distinct (different) elements."

- 11.1 Set comprehension and membership
- 11.2 Operations on sets
- 11.3 Theorems concerning set operations (many! — mostly easy…)
- 11.4 Union and intersection of families of sets (quantification over $\cup$ and $\cap$)
- …

The Language of Set Theory — Overview

- The type **set** t of sets with elements of type t
- Set membership: For $e : t$ and $S : \mathbf{set}\ t$: $\quad e \in S$
- **Set comprehension:** $\{x : t \mid R \bullet E\}$ — following the pattern of quantification
- Set enumeration: $\{6, 7, 9\}$
- Set size: $\#\{6, 7, 9\} = 3$
- Set inclusion: $\subset, \subseteq, \supset, \supseteq$
- Set union and intersection: $\cup, \cap$
- Set difference: $S - T$
- Set complement: $\sim S$
- Power set (set of subsets): $\mathbb{P}\ S$
- Cartesian product (cross product, direct product) of sets: $S \times T$ (Section 14.1)

Set Membership versus Type Annotation

Let T be a **type**; let S be a **set**, that is, an expression of type **set** T, and let e be an expression ot type T, then

- $e \in S$ is an expression
- of type $\mathbb{B}$
- and denotes "e is **in** S"
  or "e is an **element of** S"

Because: $\_\in\_ : T \to \textbf{set}\ T \to \mathbb{B}$

Note:

- $e : T$ is nothing but the expression e, with type annotation T.
- If e has type T, then $e : T$ has the same value as e.

Cardinality of Finite Sets

(11.12) **Axiom, Size:** Provided $\neg occurs(\text{'}x\text{'}, \text{'}S\text{'})$,

$$\# S = (\Sigma x \mid x \in S \bullet 1)$$

This uses: $\#\_ : \textbf{set}\ t \to \mathbb{N}$

Note:

- $(\Sigma x \mid x \in S \bullet 1)$ is defined if and only if S is finite.
- $\#\{n : \mathbb{N} \mid true \bullet n\}$ **is undefined!**
- "$\#\mathbb{N}$" **is a type error!** — because $\mathbb{N} : Type$
- Types are not sets — like in Haskell:

```
Integer :: *
Data.Set.Set Integer :: *
```

The Axioms of Set Theory — Overview

(11.2) Provided $\neg occurs(\text{'}x\text{'}, \text{'}e_0, \ldots, e_{n-1}\text{'})$,

$$\{e_0, \ldots, e_{n-1}\} = \{x \mid x = e_0 \vee \cdots \vee x = e_{n-1} \bullet x\}$$

(11.3) **Axiom, Set membership:** Provided $\neg occurs(\text{'}x\text{'}, \text{'}F\text{'})$,

$$F \in \{x \mid R \bullet E\} \equiv (\exists x \mid R \bullet E = F)$$

(11.2f) **Empty Set:** $v \in \{\} \equiv false$

(11.4) **Axiom, Extensionality:** Provided $\neg occurs(\text{'}x\text{'}, \text{'}S, T\text{'})$,

$$S = T \equiv (\forall x \bullet x \in S \equiv x \in T)$$

(11.13T) **Axiom, Subset:** Provided $\neg occurs(\text{'}x\text{'}, \text{'}S, T\text{'})$,

$$S \subseteq T \equiv (\forall x \bullet x \in S \Rightarrow x \in T)$$

(11.14) **Axiom, Proper subset:** $S \subset T \equiv S \subseteq T \wedge S \neq T$
(11.20) **Axiom, Union:** $v \in S \cup T \equiv v \in S \vee v \in T$
(11.21) **Axiom, Intersection:** $v \in S \cap T \equiv v \in S \wedge v \in T$
(11.22) **Axiom, Set difference:** $v \in S - T \equiv v \in S \wedge v \notin T$
(11.23) **Axiom, Power set:** $v \in \mathbb{P} S \equiv v \subseteq S$

Set Comprehension

Set comprehension examples:

$$\{i : \mathbb{N} \mid i < 4 \bullet 2 \cdot i + 1\} = \{1, 3, 5, 7\}$$
$$\{x : \mathbb{Z} \mid 1 \leq x < 5 \bullet x \cdot x\} = \{1, 4, 9, 16\}$$
$$\{i : \mathbb{Z} \mid 5 \leq i < 8 \bullet i \triangleleft i \triangleleft \epsilon\} = \{(5 \triangleleft 5 \triangleleft \epsilon), (6 \triangleleft 6 \triangleleft \epsilon), (7 \triangleleft 7 \triangleleft \epsilon)\}$$

(11.1) **Set comprehension general shape:** $\{x : t \mid R \bullet E\}$

— This set comprehension **binds** variable x in R and E!

Evaluated in state s, this denotes the set containing the values of E evaluated in those states resulting from s by changing the binding of x to those values from type t that satisfy R.

Note: The braces "$\{\ldots\}$" are **only** used for set notation!

Abbreviation for special case: $\{x \mid R\} = \{x \mid R \bullet x\}$

(11.2) Provided $\neg occurs(\text{'}x\text{'}, \text{'}e_0, \ldots, e_{n-1}\text{'})$,

$$\{e_0, \ldots, e_{n-1}\} = \{x \mid x = e_0 \vee \cdots \vee x = e_{n-1} \bullet x\}$$

Note: This is covered by "Reflexivity of =" in CalcCheck.

Set Membership

(11.3) **Axiom, Set membership:** Provided $\neg occurs(\text{'}x\text{'}, \text{'}F\text{'})$,

$$F \in \{x \mid R \bullet E\} \equiv (\exists x \mid R \bullet E = F)$$

$F \in \{x \mid R\}$
= ⟨ Expanding abbreviation ⟩
$F \in \{x \mid R \bullet x\}$
= ⟨ (11.3) Axiom, Set membership — **provided $\neg occurs(\text{'}x\text{'}, \text{'}F\text{'})$** ⟩
$(\exists x \mid R \bullet x = F)$
= ⟨ (9.19) Trading for $\exists$ ⟩
$(\exists x \mid x = F \bullet R)$
= ⟨ (8.14) One-point rule — **provided $\neg occurs(\text{'}x\text{'}, \text{'}F\text{'})$** ⟩
$R[x := F]$

This proves: **Simple set compr. membership:** Prov. $\neg occurs(\text{'}x\text{'}, \text{'}F\text{'})$,

$$F \in \{x \mid R\} \equiv R[x := F]$$

Set Equality and Inclusion

(11.4) **Axiom, Extensionality:** Provided $\neg occurs(\text{'}x\text{'}, \text{'}S, T\text{'})$,

$$S = T \equiv (\forall x \bullet x \in S \equiv x \in T)$$

(11.13T) **Axiom, Subset:** Provided $\neg occurs(\text{'}x\text{'}, \text{'}S, T\text{'})$,

$$S \subseteq T \equiv (\forall x \bullet x \in S \Rightarrow x \in T)$$

(11.11b) **Metatheorem Extensionality:**
Let S and T be set expressions and v be a variable.
Then $S = T$ is a theorem iff $v \in S \equiv v \in T$ is a theorem. — Using "Set extensionality"

(11.13m) **Metatheorem Subset:**
Let S and T be set expressions and v be a variable. — Using "Set inclusion"
Then $S \subseteq T$ is a theorem iff $v \in S \Rightarrow v \in T$ is a theorem.

Extensionality (11.11b) and Subset (11.13m) will, **by LADM**, mostly be used as the following inference rules:

$$\frac{v \in S \equiv v \in T}{S = T} \qquad \frac{v \in S \Rightarrow v \in T}{S \subseteq T}$$

Using Set Extensionality — LADM-Style

Extensionality (11.11b) inference rule: $\dfrac{v \in S \equiv v \in T}{S = T}$

Ex. 8.2(a) Prove: $\{E, E\} = \{E\}$ for each expression E.

By extensionality (11.11b):

Proving $v \in \{E, E\} \equiv v \in \{E\}$**:**

$v \in \{E, E\}$
≡ ⟨ Set enumerations (11.2) ⟩
$v \in \{x \mid x = E \vee x = E\}$
≡ ⟨ Idempotency of $\vee$ (3.26) ⟩
$v \in \{x \mid x = E\}$
≡ ⟨ Set enumerations (11.2) ⟩
$v \in \{E\}$

Using Set Extensionality — More CalcCheck-Style

Axiom (11.4) "Set extensionality": $S = T \equiv (\forall x \bullet x \in S \equiv x \in T)$
— provided $\neg occurs(\text{'}x\text{'}, \text{'}S, T\text{'})$

Example (8.2a): $\{E, E\} = \{E\}$
Proof:
  Using "Set extensionality":
    Subproof for `$\forall v \bullet v \in \{E, E\} \equiv v \in \{E\}$`:
      For any `$v$`:
        $v \in \{E, E\}$
        ≡ ⟨ Set enumerations (11.2) ⟩
        $v \in \{x \mid x = E \vee x = E\}$
        ≡ ⟨ Idempotency of $\vee$ (3.26) ⟩
        $v \in \{x \mid x = E\}$
        ≡ ⟨ Set enumerations (11.2) ⟩
        $v \in \{E\}$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-06

Typed Set Theory, Introduction to Relations

Plan for Today

- **Continuing with LADM chapter 11: Set Theory — emphasizing types**
- **Starting with Relations (see also LADM chapter 14)**

Coming up (interleaved):

- **Explicit Induction Principles**
- **Induction** (LADM Chapter 12)
- **More Program Correctness** (LADM chapter 10, section 12.6)
- **Relations** (LADM Chapter 14)
- Sequences (LADM Chapter 13) will be further developed mainly in Exercises, Assignments, …

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-06

Part 0: Set Theory

The Axioms of Set Theory — Overview

(11.2) Provided $\neg occurs(`x', `e_0, \ldots, e_{n-1}')$,
$\{e_0, \ldots, e_{n-1}\} = \{x \mid x = e_0 \lor \cdots \lor x = e_{n-1} \bullet x\}$

(11.3) **Axiom, Set membership:** Provided $\neg occurs(`x', `F')$,
$F \in \{x \mid R \bullet E\} \quad \equiv \quad (\exists x \mid R \bullet E = F)$

(11.2f) **Empty Set:** $v \in \{\} \equiv \mathit{false}$

(11.4) **Axiom, Extensionality:** Provided $\neg occurs(`x', `S, T')$,
$S = T \quad \equiv \quad (\forall x \bullet x \in S \equiv x \in T)$

(11.13T) **Axiom, Subset:** Provided $\neg occurs(`x', `S, T')$,
$S \subseteq T \quad \equiv \quad (\forall x \bullet x \in S \Rightarrow x \in T)$

(11.14) **Axiom, Proper subset:** $S \subset T \quad \equiv \quad S \subseteq T \land S \neq T$
(11.20) **Axiom, Union:** $v \in S \cup T \quad \equiv \quad v \in S \lor v \in T$
(11.21) **Axiom, Intersection:** $v \in S \cap T \quad \equiv \quad v \in S \land v \in T$
(11.22) **Axiom, Set difference:** $v \in S - T \quad \equiv \quad v \in S \land v \notin T$
(11.23) **Axiom, Power set:** $v \in \mathbb{P} S \quad \equiv \quad v \subseteq S$

Set Equality and Inclusion

(11.4) **Axiom, Extensionality:** Provided $\neg occurs(`x', `S, T')$,
$S = T \quad \equiv \quad (\forall x \bullet x \in S \equiv x \in T)$

(11.13T) **Axiom, Subset:** Provided $\neg occurs(`x', `S, T')$,
$S \subseteq T \quad \equiv \quad (\forall x \bullet x \in S \Rightarrow x \in T)$

(11.11b) **Metatheorem Extensionality:**
Let S and T be set expressions and v be a variable.
Then $S = T$ is a theorem iff $v \in S \equiv v \in T$ is a theorem. — Using "Set extensionality"

(11.13m) **Metatheorem Subset:**
Let S and T be set expressions and v be a variable. — Using "Set inclusion"
Then $S \subseteq T$ is a theorem iff $v \in S \Rightarrow v \in T$ is a theorem.

Extensionality (11.11b) and Subset (11.13m) will, **by LADM**, mostly be used as the following inference rules:

$$\frac{v \in S \quad \equiv \quad v \in T}{S \quad = \quad T} \qquad \frac{v \in S \quad \Rightarrow \quad v \in T}{S \quad \subseteq \quad T}$$

LADM Set Equality via Equivalence

(11.4) **Axiom, Extensionality:** Provided $\neg occurs(`x', `S, T')$,
$S = T \quad \equiv \quad (\forall x \bullet x \in S \equiv x \in T)$

(11.9) **"Simple set comprehension equality":** $\{x \mid Q\} = \{x \mid R\} \equiv (\forall x \bullet Q \equiv R)$

(11.10) **Metatheorem set comprehension equality:**
$\{x \mid Q\} = \{x \mid R\}$ is valid iff $Q \equiv R$ is valid.

(11.11) **Methods for proving set equality $S = T$:**
(a) Use Leibniz directly
(b) Use axiom Extensionality (11.4) and prove $v \in S \equiv v \in T$
(c) Prove $Q \equiv R$ and conclude $\{x \mid Q\} = \{x \mid R\}$ via (11.9)/(11.10)

Note:
- In the informal setting, confusion about variable binding is easy!
- `Using "Set extensionality"` or `Using (11.9)` followed by `For any ...` make variable binding clear.

Using Set Extensionality — CalcCheck Example

Axiom (11.4) "Set extensionality": $S = T \quad \equiv \quad (\forall x \bullet x \in S \equiv x \in T)$
— provided $\neg occurs(`x', `S, T')$

Theorem (11.26) "Symmetry of $\cup$": $S \cup T = T \cup S$
Proof:
  Using "Set extensionality":
    Subproof for `∀ e • e ∈ S ∪ T ≡ e ∈ T ∪ S`:
      For any `e`:
        $e \in S \cup T$
      $\equiv \langle$ "Union" $\rangle$
        $e \in S \lor e \in T$
      $\equiv \langle$ "Symmetry of $\lor$" $\rangle$
        $e \in T \lor e \in S$
      $\equiv \langle$ "Union" $\rangle$
        $e \in T \cup S$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-06

Part 1: Typed Set Theory

Anything Wrong?

Let the set Q be defined by the following: $\quad \_\in\_, \_\notin\_ : A \to \mathbf{set}\ A \to \mathbb{B}$

(R) $Q = \{S \mid S \notin S\}$ **"The mother of all type errors"**

Then: **⟹ birth of type theory...**

$Q \in Q$
$\equiv \langle$ (R) $\rangle$
$Q \in \{S \mid S \notin S\}$
$\equiv \langle$ (11.3) Membership in set comprehension $\rangle$
$(\exists S \mid S \notin S \bullet Q = S)$
$\equiv \langle$ (9.19) Trading for $\exists$, (8.14) One-point rule $\rangle$
$Q \notin Q$
$\equiv \langle$ (11.0) Def. $\notin$ $\rangle$
$\neg(Q \in Q)$

With (3.15) $p \equiv \neg p \equiv \mathit{false}$, this proves:

(R′) false **— "Russell's paradox"**

"The Universe" in LADM

THE UNIVERSE

A theory of sets concerns sets constructed from some collection of elements. There is a theory of sets of integers, a theory of sets of characters, a theory of sets of sets of integers, and so forth. This collection of elements is called the *domain of discourse* or the *universe of values*; it is denoted by **U**. The universe can be thought of as the type of every set variable in the theory. For example, if the universe is $set(\mathbb{Z})$, then $v{:}set(\mathbb{Z})$.

When several set theories are being used at the same time, there is a different universe for each. The name **U** is then overloaded, and we have to distinguish which universe is intended in each case. This overloading is similar to using the constant 1 as a denotation of an integer, a real, the identity matrix, and even (in some texts, alas) the boolean *true*.

Overloading via type polymorphism: $\{\}, U : \mathbf{set}\ t$

$(\{\} : \mathbf{set}\ \mathbb{B}) = \{\} \qquad (U : \mathbf{set}\ \mathbb{B}) = \{\mathit{false}, \mathit{true}\}$
$(\{\} : \mathbf{set}\ \mathbb{N}) = \{\} \qquad (U : \mathbf{set}\ \mathbb{N}) = \{k : \mathbb{N} \mid \mathit{true}\}$

"The Universe" and Complement in LADM

the *domain of discourse* or the *universe of values*; it is denoted by **U**. The universe can be thought of as the type of every set variable in the theory. For example, if the universe is $set(\mathbb{Z})$, then $v{:}set(\mathbb{Z})$.

COMPLEMENT

The *complement* of S, written $\sim S$,[4] is the set of elements that are not in S (but are in the universe). In the Venn diagram in this paragraph, we have shown set S and universe **U**. The non-filled area represents $\sim S$.

(11.17) **Axiom, Complement:** $v \in \sim S \quad \equiv \quad v \in \mathbf{U} \land v \notin S$

For example, for $\mathbf{U} = \{0,1,2,3,4,5\}$, we have

$\sim\{3,5\} = \{0,1,2,4\}$,
$\sim\mathbf{U} = \emptyset$, $\quad \sim\emptyset = \mathbf{U}$.

We can easily prove

(11.18) $v \in \sim S \equiv v \notin S$ (for v in **U**).

"The" Universe

Frequently, a "domain of discourse" is assumed, that is, a set of "all objects under consideration".

This is often called a "**universe**". Special notation: U — `\universe`

Declaration: U : **set** t

Axiom "Universal set": $x \in U$ — remember: $\_\in\_ : t \to \mathbf{set}\ t \to \mathbb{B}$

Theorem: $(U : \mathbf{set}\ t) = \{x : t \bullet x\}$

Types are not sets! — $(U : \mathbf{set}\ t)$ is the set containing all values of type t.

We define a nicer notation: $\llcorner t \lrcorner = (U : \mathbf{set}\ t)$

"Definition of $\llcorner\_\lrcorner$": $\forall x : t \bullet x \in \llcorner t \lrcorner$

Example: $\llcorner \mathbb{B} \lrcorner = \{\mathit{false}, \mathit{true}\}$

Set Complement

(11.17) **Axiom, Complement:** $v \in {\sim}S \quad \equiv \quad v \in U \land v \notin S$

Complement can be expressed via difference: $\quad {\sim}S \quad = \quad U - S$

Complement ${\sim}$ **always implicitly depends on the universe U!**

Example: $\quad {\sim}\{true\} \quad = \quad \llcorner \mathbb{B} \lrcorner - \{true\} \quad = \quad \{false, true\} - \{true\} \quad = \quad \{false\}$

LADM: "We can easily prove
(11.18) $\quad v \in {\sim}S \quad \equiv \quad v \notin S \qquad$ (for v in U)."

Consider $\quad \mathbb{Z}_+ : \textbf{set}\ \mathbb{Z} \quad$ defined as $\quad \mathbb{Z}_+ = \{x : \mathbb{Z} \mid \mathsf{pos}\ x\}$:

- Let S be a subset of $\mathbb{Z}_+$. For example: $\quad S = \{2, 3, 7\}$
- Consider the complement ${\sim}S$
- Is $\quad -5 \in {\sim}S \quad$ true or false?

Power Set

(11.23) **Axiom, Power set:** $v \in \mathbb{P}\,S \quad \equiv \quad v \subseteq S$

Declaration: $\quad \mathbb{P}\_ \ : \ \textbf{set}\ t \to \textbf{set}\ (\textbf{set}\ t)$

— remember: $\quad \textbf{set} \ : \ Type \to Type$

$\mathbb{P}\,\{0,1\} = \{\{\}, \{0\}, \{1\}, \{0,1\}\}$

- For a type t, the **type of subsets of t** is $\quad \textbf{set}\ t$
- According to the textbook, **type annotations $v : t$**, in particular in variable declarations in quantifications and in set comprehensions, **may only use types t**.
- (The **specification notation Z** allows the use of sets in variable declarations — **this makes $\forall$ and $\exists$ rules more complicated.**)

  If you find a place where I **accidentally** still follow Z in writing "$\mathbb{P}\,t$" for a type t (instead of writing "$\textbf{set}\ t$" or "$\mathbb{P} \llcorner t \lrcorner$"), please point it out to me.

Calculate!

The size of a finite set S, that is, the number of its elements, is written $\#S$

- $\# \llcorner \mathbb{B} \lrcorner$
- $\#\{S : \textbf{set}\ \mathbb{B} \mid true \in S \bullet S\}$
- $\#\{T : \textbf{set}\ \mathbb{B} \mid \{\} \notin T \bullet T\}$
- $\#\{S : \textbf{set}\ \mathbb{N} \mid (\forall x : \mathbb{N} \mid x \in S \bullet x < n) \land \#S = k \bullet S\}$

- $\llcorner \mathbb{B} \lrcorner = \{false, true\}$
- $S \in \llcorner \textbf{set}\ \mathbb{B} \lrcorner \quad \equiv \quad S \subseteq \llcorner \mathbb{B} \lrcorner$
- $\llcorner \textbf{set}\ \mathbb{B} \lrcorner = \{\{\}, \{false\}, \{true\}, \{false, true\}\}$
- $T \in \llcorner \textbf{set set}\ \mathbb{B} \lrcorner \quad \equiv \quad T \subseteq \mathbb{P} \llcorner \mathbb{B} \lrcorner$

Metatheorem (11.25): Sets $\Longleftrightarrow$ Propositions

Let

- $P, Q, R, \ldots$ be set variables
- $p, q, r, \ldots$ be propositional variables
- E, F be expressions built from these set variables and $\cup, \cap, {\sim}, U, \{\}$.

Define the Boolean expressions E_p and F_p by replacing

$P, Q, R, \ldots$	with	$p, q, r, \ldots$	${\sim}$	with	$\neg$
$\cup$	with	$\lor$	U	with	$true$
$\cap$	with	$\land$	$\{\}$	with	$false$

Then:

- $E = F$ is valid iff $E_p \equiv F_p$ is valid.
- $E \subseteq F$ is valid iff $E_p \Rightarrow F_p$ is valid.
- $E = U$ is valid iff E_p is valid.

Metatheorem (11.25): Sets $\Longleftrightarrow$ Propositions — Examples

Let E, F be expressions built from set variables $P, Q, R, \ldots$ and $\cup, \cap, {\sim}, U, \{\}$.

Define the Boolean expressions E_p and F_p by replacing

$P, Q, R, \ldots$	with	$p, q, r, \ldots$	${\sim}$	with	$\neg$
$\cup$	with	$\lor$	U	with	$true$
$\cap$	with	$\land$	$\{\}$	with	$false$

Then:

- $E = F$ is valid iff $E_p \equiv F_p$ is valid.
- $E \subseteq F$ is valid iff $E_p \Rightarrow F_p$ is valid.
- $E = U$ is valid iff E_p is valid.

Free theorems!

$$P \cap (P \cup Q) \ = \ P$$
$$P \cap (Q \cup R) \ = \ (P \cap Q) \cup (P \cap R)$$
$$P \cup (Q \cap R) \ \subseteq \ P \cup Q$$
$$\vdots$$

Tuples and Tuple Types in CalcCheck

Tuples can have arbitrary "arity" at least 2.

Example: A triple with type: $\langle 2, true, \text{"}Hello\text{"} \rangle \ : \ \langle\!| \mathbb{Z}, \mathbb{B}, String |\!\rangle$

Example: A seven-tuple: $\quad \langle 3, true, 5 \triangleleft \epsilon, \langle 5, false \rangle, \text{"}Hello\text{"}, \{2, 8\}, \{42 \triangleleft \epsilon\} \rangle$

The type of this: $\quad \langle\!| \mathbb{Z}, \mathbb{B}, Seq\ \mathbb{Z}, \langle\!| \mathbb{Z}, \mathbb{B} |\!\rangle, String, \textbf{set}\ \mathbb{Z}, \textbf{set}\ (Seq\ \mathbb{Z}) |\!\rangle$

- Tuples are enclosed in $\quad \langle \ldots \rangle \quad$ as in LADM. (type "\<" and "\>")
- Tuple types are enclosed in $\quad \langle\!| \ldots |\!\rangle$. (type "\<!" and "\>!")
- Otherwise, tuples and tuple types "work" as in Haskell.
- In particular, there is no implicit nesting:

  $\langle\!| \langle\!| A, B |\!\rangle, C |\!\rangle$ and $\langle\!| A, B, C |\!\rangle$ and $\langle\!| A, \langle\!| B, C |\!\rangle |\!\rangle$ are three different types!

Pairs and Cartesian Products

If b and c are expressions, then $\langle b, c \rangle$ is their **2-tuple** or **ordered pair**

— "ordered" means that there is a **first** constituent (b) and a **second** constituent (c).

(14.2) **Axiom, Pair equality:** $\quad \langle b, c \rangle = \langle b', c' \rangle \quad \equiv \quad b = b' \land c = c'$

(14.3) **Axiom, Cross product:** $\quad S \times T \ = \ \{b, c \mid b \in S \land c \in T \bullet \langle b, c \rangle\}$

(14.4) **Membership:** $\quad \langle b, c \rangle \in S \times T \quad \equiv \quad b \in S \ \land \ c \in T$

Cartesian product of types: Two-tuple types: $\quad b : t_1 \ ; \ c : t_2 \quad$ iff $\quad \langle b, c \rangle : \langle\!| t_1, t_2 |\!\rangle$

Axiom, Pair projections:

$$fst \ : \ \langle\!| t_1, t_2 |\!\rangle \to t_1 \qquad fst\ \langle b, c \rangle = b$$
$$snd \ : \ \langle\!| t_1, t_2 |\!\rangle \to t_2 \qquad snd\ \langle b, c \rangle = c$$

Pair equality: For $p, q : \langle\!| t_1, t_2 |\!\rangle$,

$$p = q \quad \equiv \quad fst\ p = fst\ q \ \land \ snd\ p = snd\ q$$

Some Cross Product Theorems

(14.5) $\quad \langle x, y \rangle \in S \times T \quad \equiv \quad \langle y, x \rangle \in T \times S$

(14.6) $\quad S = \{\} \quad \Rightarrow \quad S \times T = T \times S = \{\}$

(14.7) $\quad S \times T = T \times S \quad \equiv \quad S = \{\} \lor T = \{\} \lor S = T$

(14.8) **Distributivity of $\times$ over $\cup$:**
$$S \times (T \cup U) \ = \ (S \times T) \cup (S \times U)$$
$$(S \cup T) \times U \ = \ (S \times U) \cup (T \times U)$$

(14.9) **Distributivity of $\times$ over $\cap$:**
$$S \times (T \cap U) \ = \ (S \times T) \cap (S \times U)$$
$$(S \cap T) \times U \ = \ (S \times U) \cap (T \times U)$$

(14.10) **Distributivity of $\times$ over $-$:**
$$S \times (T - U) \ = \ (S \times T) - (S \times U)$$
$$(S - T) \times U \ = \ (S \times U) - (T \times U)$$

(14.12) **Monotonicity:** $\quad S \subseteq S' \ \land \ T \subseteq T' \quad \Rightarrow \quad S \times T \subseteq S' \times T'$

Some Spice...

Converting between "different ways to take two arguments":

$$curry \quad : \quad (\langle\!| A, B |\!\rangle \to C) \to (A \to B \to C)$$
$$curry\ f\ x\ y \quad = \quad f\ \langle x, y \rangle$$

$$uncurry \quad : \quad (A \to B \to C) \to (\langle\!| A, B |\!\rangle \to C)$$
$$uncurry\ g\ \langle x, y \rangle \quad = \quad g\ x\ y$$

These functions correspond to the "Shunting" law:

(3.65) **Shunting:** $\quad p \land q \ \Rightarrow \ r \quad \equiv \quad p \Rightarrow (q \Rightarrow r)$

The "currying" concept is named for Haskell Brooks Curry (1900–1982), but goes back to Moses Ilyich Schönfinkel (1889–1942) and Gottlob Frege (1848–1925).

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-16

Relations in Set Theory

Plan for Today

- A Set Theory Exercise: Relative Pseudocomplement
- Correctness Variations: Ghost Variables
- Relations

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-16

Part 1: A Set Theory Exercise: Relative Pseudocomplement

Let c be defined by: $x \le c \quad \equiv \quad x \le 5$

What do you know about c? Why? (Prove it!)

Note: x is implicitly univerally quantified!

Proving $5 \le c$:

$5 \le c$
$\equiv$ ⟨ The given equivalence, with $x := 5$ ⟩
$5 \le 5$ — This is Reflexivity of $\le$

Proving $c \le 5$:

$c \le 5$
$\equiv$ ⟨ Given equivalence, with $x := c$ ⟩
$c \le c$ — This is Reflexivity of $\le$

With antisymmetry of $\le$ (that is, $a \le b \wedge b \le a \Rightarrow a = b$), we obtain $c = 5$ — An instance of:

(15.47) **Indirect equality**: $a = b \quad \equiv \quad (\forall z \bullet z \le a \quad \equiv \quad z \le b)$

Relative Pseudocomplement

Let $A, B :$ **set** t be two sets of the same type.

The **relative pseudocomplement** $A \Rightarrow B$ of A with respect to B is defined by:

$$X \subseteq (A \Rightarrow B) \quad \equiv \quad X \cap A \subseteq B$$

Calculate the **relative pseudocomplement** $A \Rightarrow B$ as a set expression not using $\Rightarrow$! That is:

Calculate $A \Rightarrow B \quad = \quad$ **?**

Using set extensionality, that is:

Calculate $x \in A \Rightarrow B \quad \equiv \quad x \in$ **?**

Characterisation of relative pseudocomplement of sets: $X \subseteq (A \Rightarrow B) \quad \equiv \quad X \cap A \subseteq B$

$x \in A \Rightarrow B$
$\equiv$ ⟨ $e \in S \equiv \{e\} \subseteq S$ — Exercise! ⟩
$\{x\} \subseteq A \Rightarrow B$
$\equiv$ ⟨ Def. $\Rightarrow$, with $X := \{x\}$ ⟩
$\{x\} \cap A \subseteq B$
$\equiv$ ⟨ (11.13) Subset ⟩
$(\forall y \mid y \in \{x\} \cap A \bullet y \in B)$
$\equiv$ ⟨ (11.21) Intersection ⟩
$(\forall y \mid y \in \{x\} \wedge y \in A \bullet y \in B)$
$\equiv$ ⟨ $y \in \{x\} \equiv y = x$ — Exercise! ⟩
$(\forall y \mid y = x \wedge y \in A \bullet y \in B)$
$\equiv$ ⟨ (9.4b) Trading for $\forall$, Def. $\notin$ ⟩
$(\forall y \mid y = x \bullet y \notin A \vee y \in B)$
$\equiv$ ⟨ (8.14) One-point rule ⟩
$x \notin A \vee x \in B$
$\equiv$ ⟨ (11.17) Set complement, (11.20) Union ⟩
$x \in {\sim} A \cup B$

Theorem: $A \Rightarrow B \quad = \quad {\sim} A \cup B$

Characterisation of relative pseudocomplement of sets: $X \subseteq A \Rightarrow B \quad \equiv \quad X \cap A \subseteq B$

Theorem "Pseudocomplement via $\cup$": $A \Rightarrow B \quad = \quad {\sim} A \cup B$

Calculation:

$x \in A \Rightarrow B$
$\equiv$ ⟨ Pseudocomplement via $\cup$ ⟩
$x \in {\sim} A \cup B$
$\equiv$ ⟨ (11.20) Union, (11.17) Set complement ⟩
$\neg(x \in A) \vee x \in B$
$\equiv$ ⟨ (3.59) Material implication ⟩
$x \in A \Rightarrow x \in B$

Corollary "Membership in pseudocomplement":
$x \in A \Rightarrow B \quad \equiv \quad x \in A \Rightarrow x \in B$

Easy to see: On sets, relative pseudocomplement wrt. $\{\}$ is complement:
$A \Rightarrow \{\} \quad = \quad {\sim} A$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-16

Part 2: Correctness Variations: Ghost Variables

Goal of Assignment 1.3: Correctness of a Program Containing a `while`-Loop

Theorem "Correctness of `elem`":

```
   true
⇒[ xs := xs₀ ;
   b := false ;
   while xs ≠ ϵ do
      if head xs = x
      then b := true
      else skip
      fi ;
      xs := tail xs
   od
 ]
 (b ≡ x ∈ xs₀)   ⁞⁞⁞ Parentheses!
```

Proof:

```
  true
⇒[ xs := xs₀ ;
   b := false
 ]    ⟨ "Initialisation for `elem`" ⟩
(∃ us • (us ⌢ xs = xs₀) ∧ (b ≡ x ∈ us))
⇒[ while xs ≠ ϵ do
      if head xs = x
      then b := true
      else skip
      fi ;
      xs := tail xs
   od
 ]    ⟨ "While" with "Invariant for `elem`" ⟩
¬ (xs ≠ ϵ) ∧ (∃ us • (us ⌢ xs = xs₀) ∧ (b ≡ x ∈ us))
⇒     ⟨ "Postcondition for `elem`" ⟩
(b ≡ x ∈ xs₀)
```

Invariant involves quantifier: Good for practice with quantifier reasoning…

Easier to Prove than Assignment 1.3: With Ghost Variable — Ex6.1

Theorem "Correctness of `elem`":

```
   true
⇒[ xs := xs₀ ;
     us := ϵ ;        ⁞⁞⁞ Ghost variable: Does not influence program flow or result
     b := false ;
     ⁞⁞⁞ Invariant: (us ⌢ xs = xs₀) ∧ (b ≡ x ∈ us)
     while xs ≠ ϵ do
        if head xs = x then b := true else skip fi ;
        us := us ▹ head xs ;          ⁞⁞⁞ Ghost assignment
        xs := tail xs
     od
 ]
 (b ≡ x ∈ xs₀)        ⁞⁞⁞ Parentheses needed because of precedences!
```

"Ghost variables" can make proofs easier: They can be used to keep track of values that are important for **understanding** the logic of the program.

With language support for "ghost variables", they are compiled away, to avoid run-time cost.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-16

Part 3: Introduction to Relations

Predicates and Tuple Types — Relations are Tuple Sets — Think Database Tables!

$\_called\_ \;:\; P \to P \to \mathbb{B}$

$(uncurry\ \_called\_) : \langle P, P \rangle \to \mathbb{B}$ is the **characteristic function** of the set

$R_{called} \;:\; \textbf{set}\ \langle P, P \rangle$

$R_{called} \;=\; \{p, q : P \mid p\ called\ q \bullet \langle p, q \rangle\}$

R_{called} is a **(binary) relation**.

$D \;:\; P \to City \to City \to \mathbb{B}$

$D\ p\ a\ b \;\equiv\;$ p drove from a to b

$R_D \;:\; \textbf{set}\ \langle P, City, City \rangle$

$R_D \;=\; \{p : P; a, b : City \mid D\ p\ a\ b \bullet \langle p, a, b \rangle\}$

R_D is a **(ternary) relation**.

Relations are Everywhere in Specification and Reasoning in CS

- Operations are easily defined and understood via set theory
- These operations satisfy many algebraic properties
- **Formalisation using relation-algebraic operations needs no quantifiers**
- **Similar to** how matrix operations do away with quantifications and indexed variables a_{ij} in **linear algebra**
- Like linear algebra, **relation algebra**
  - raises the level of abstraction
  - makes reasoning easier by reducing necessity for quantification
- Starting with lots of quantification over elements, while **proving properties via set theory**.
- Moving towards **abstract relation algebra** (avoiding any mention of and quantification over elements)

Relations

- LADM: A **relation** on $B_1 \times \cdots \times B_n$ is a subset of $B_1 \times \cdots \times B_n$ — where $B_1, \ldots, B_n$ are sets
- CalcCheck: Normally: A **relation** on $\langle t_1, \ldots, t_n \rangle$ is a subset of $\llcorner \langle t_1, \ldots, t_n \rangle \lrcorner$, that is, an item of type $\textbf{set}\ \langle t_1, \ldots, t_n \rangle$ — where $t_1, \ldots, t_n$ are types
- A relation on the tuple (Cartesian product) type $\langle t_1, \ldots, t_n \rangle$ is an **n-ary relation**.
  "Tables" in relational databases are n-ary relations.
- A relation on the pair (Cartesian product) type $\langle t_1, t_2 \rangle$ is a **binary relation**.
- The **type** of binary relations on $\langle t_1, t_2 \rangle$ is written $t_1 \leftrightarrow t_2$, with

  $t_1 \leftrightarrow t_2 \;=\; \textbf{set}\ \langle t_1, t_2 \rangle$ — `\rel`
- The **set** of binary relations on $B \times C$ is written $B \leftrightarrow\!\!\!\!\leftrightarrow C$, with

  $B \leftrightarrow\!\!\!\!\leftrightarrow C \;=\; \mathbb{P}\,(B \times C)$ — `\Rel`

Binary Relation Types Contain Subsets of Cartesian Products

- The **type** of binary relations between types t_1 and t_2:
  $t_1 \leftrightarrow t_2 \;=\; \textbf{set}\ \langle t_1, t_2 \rangle$ — `\rel`
- The **set** of binary relations between sets B and C:
  $B \leftrightarrow\!\!\!\!\leftrightarrow C \;=\; \mathbb{P}\,(B \times C)$ — `\Rel`

Note that for a type t, the universal set

$U : \textbf{set}\ t$

is the set of all members of t.

Or, $(U : \textbf{set}\ t)$ is "type t as a set".

We **abbreviate**: $\llcorner\ t\ \lrcorner := (U : \textbf{set}\ t)$, (`\llcorner`…`\lrcorner`) and have:

$S \in \llcorner\ \textbf{set}\ t\ \lrcorner \;\equiv\; S \subseteq \llcorner\ t\ \lrcorner$

Consider $R : t_1 \leftrightarrow t_2$ and $x : t_1$ and $y : t_2$.

$R \in \llcorner\ t_1 \leftrightarrow t_2\ \lrcorner$
$\equiv$ ⟨ Def. $\leftrightarrow$ ⟩
$R \in \llcorner\ \textbf{set}\ \langle t_1, t_2 \rangle\ \lrcorner$
$\equiv$ ⟨ Membership in $\llcorner\ \textbf{set}\ \_\ \lrcorner$ ⟩
$R \subseteq \llcorner\ \langle t_1, t_2 \rangle\ \lrcorner$
$\equiv$ ⟨ Def. **set**, Def. $\times$, Def. $\llcorner\ \lrcorner$ ⟩
$R \subseteq \llcorner\ t_1\ \lrcorner \times \llcorner\ t_2\ \lrcorner$
$\equiv$ ⟨ Def. $\mathbb{P}$, Def. $\leftrightarrow\!\!\!\!\leftrightarrow$ ⟩
$R \in \llcorner\ t_1\ \lrcorner \leftrightarrow\!\!\!\!\leftrightarrow \llcorner\ t_2\ \lrcorner$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-18

with, **Relations in Set Theory**

Plan for Today

- with$_2$ and with$_3$
- Relations
  - Relationship notation and reasoning
  - Set operations as relation operations
  - Set-theoretic definition of relational operations: Converse, composition

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-18

Part 1: with$_2$ and with$_3$

`with` — Overview

CalcCheck currently knows three kinds of "`with`":

- "with$_1$": For explicit substitutions: **"Identity of +"** with '$x := 2$'
- *ThmA* `with` *ThmB* and *ThmB$_2$* …
  - "with$_2$": If *ThmA* gives rise to an implication $A_1 \Rightarrow A_2 \Rightarrow \ldots (L = R)$:
    Perform **conditional rewriting**, rigidly applying $L\sigma \mapsto R\sigma$
    if using *ThmB* and *ThmB$_2$* … to prove $A_1\sigma, A_2\sigma, \ldots$ succeeds

  Using hi_1:
  sp_1
  sp_2

  is essentially syntactic sugar for: By hi_1 with sp_1 and sp_2
- "with$_3$": *ThmA* `with` *ThmB*
  - If *ThmB* gives rise to an equality/equivalence $L = R$:
    Rewrite *ThmA* with $L \mapsto R$ to *ThmA*′,
    and use *ThmA*′ for rewriting the goal.

`with`$_2$: Conditional Rewriting

ThmA `with` *ThmB* and *ThmB$_2$* …

- If *ThmA* gives rise to an implication $A_1 \Rightarrow A_2 \Rightarrow \ldots (L = R)$, where $FVar(L) = FVar(A_1 \Rightarrow A_2 \Rightarrow \ldots (L = R))$:
  - Find substitution σ such that $L\sigma$ matches goal
  - Resolve $A_1\sigma, A_2\sigma, \ldots$ using *ThmB* and *ThmB$_2$* …
  - Rewrite goal applying $L\sigma \mapsto R\sigma$ rigidly.
- E.g.: "Cancellation of ·" with Assumption '$m + n \neq 0$'

  when trying to prove $(m + n) \cdot (n + 2) = (m + n) \cdot 5 \cdot k$:
  - "Cancellation of ·" is: $c \neq 0 \Rightarrow (c \cdot a = c \cdot b \equiv a = b)$
  - We try to use: $c \cdot a = c \cdot b \mapsto a = b$, so L is $c \cdot a = c \cdot b$
  - Matching L against goal produces $\sigma = [a, b, c := (n + 2), (5 \cdot k), (m + n)]$
  - $(c \neq 0)\sigma$ is $(m + n) \neq 0$ and can be proven by "Assumption '$m + n \neq 0$'"
  - The goal is rewritten to $(a = b)\sigma$, that is, $(n + 2) = 5 \cdot k$.

Limitations of Conditional Rewriting Implementation of `with`$_2$

- If *ThmA* gives rise to an implication $A_1 \Rightarrow A_2 \Rightarrow \ldots (L = R)$:
  - Find substitution σ such that $L\sigma$ matches goal
  - Resolve $A_1\sigma, A_2\sigma, \ldots$ using *ThmB* and *ThmB$_2$* …
  - Rewrite goal applying $L\sigma \mapsto R\sigma$ rigidly.

  ThmA `with` *ThmB* and *ThmB$_2$* …
- E.g.: "Transitivity of ⊆" with Assumptions \`$Q \cap S \subseteq Q$\` and \`$Q \subseteq R$\` when trying to prove \`$Q \cap S \subseteq R$\`
  - "Transitivity of ⊆" is: $Q \subseteq R \Rightarrow R \subseteq S \Rightarrow Q \subseteq S$
  - For application, a **fresh renaming** is used: $q \subseteq r \Rightarrow r \subseteq s \Rightarrow q \subseteq s$
  - We try to use: $q \subseteq s \mapsto true$, so L is: $q \subseteq s$
  - Matching L against goal produces $\sigma = [q, s := Q \cap S, R]$
  - $(q \subseteq r)\sigma$ is $(Q \cap S \subseteq r)$, and $(r \subseteq s)\sigma$ is $r \subseteq R$
    — **which cannot be proven** by "Assumption '$Q \cap S \subseteq Q$'"
    resp. by "Assumption '$Q \subseteq R$'"
  - *Narrowing* or *unification* would be needed for such cases
    — **not yet implemented**
  - Adding an explicit substitution should help:

"Transitivity of ⊆" with \`$R := Q$\` and assumption \`$Q \cap S \subseteq Q$\` and assumption \`$Q \subseteq R$\`

`with`$_3$: Rewriting Theorems before Rewriting

ThmA `with` *ThmB*

- If *ThmB* gives rise to an equality/equivalence $L = R$: Rewrite *ThmA* with $L \mapsto R$
- E.g.: Assumption `$p \Rightarrow q$` with (3.60) `$p \Rightarrow q \equiv p \wedge q \equiv q$`

The local theorem $p \Rightarrow q$ (resulting from the Assumption)
rewrites via: $p \Rightarrow q \mapsto p \equiv p \wedge q$ (from (3.60))
to: $p \equiv p \wedge q$
which can be used for the rewrite: $p \mapsto p \wedge q$

Theorem (4.3) "Left-monotonicity of $\wedge$": $(p \Rightarrow q) \Rightarrow ((p \wedge r) \Rightarrow (q \wedge r))$
Proof:
Assuming `$p \Rightarrow q$`:
$p \wedge r$
$\equiv$ ⟨ Assumption `$p \Rightarrow q$` with "Definition of $\Rightarrow$ via $\wedge$" ⟩
$p \wedge q \wedge r$
$\Rightarrow$ ⟨ "Weakening" ⟩
$q \wedge r$

`with`$_3$: Rewriting Theorems before Rewriting

ThmA `with` *ThmB*

- If *ThmB* gives rise to an equality/equivalence $L = R$: Rewrite *ThmA* with $L \mapsto R$
- E.g.: "Instantiation" with (3.60)

"Instantiation" `$(\forall x \bullet P) \Rightarrow P[x := E]$` rewrites via (3.60) `$q \Rightarrow r \mapsto q \equiv q \wedge r$`
to: $(\forall x \bullet P) \equiv (\forall x \bullet P) \wedge P[x := E]$
which can be used as: $(\forall x \bullet P) \mapsto (\forall x \bullet P) \wedge P[x := E]$

H11:
$(\forall x : \mathbb{Z} \bullet 5 < f\ x)$
$\equiv$ ⟨ "Instantiation" with "Definition of $\Rightarrow$ via $\wedge$" (3.60) ⟩ ▪▪▪▪▪▪ with$_3$
$(\forall x : \mathbb{Z} \bullet 5 < f\ x) \wedge (5 < f\ x)[x := 9]$
$\Rightarrow$ ⟨ "Monotonicity of $\wedge$" with "Instantiation" ⟩ ▪▪▪▪▪▪ with$_2$
$(5 < f\ x)[x := 8] \wedge (5 < f\ x)[x := 9]$

How can you simplify if you know $P_1 \Rightarrow P_2$?

$\vdots$
$\equiv$ ⟨ ... ⟩
$\ldots \vee P_1 \vee P_2 \vee \ldots$
$\equiv$ ⟨ **?** ⟩
?

$\vdots$
$\equiv$ ⟨ ... ⟩
$\ldots \wedge P_1 \wedge P_2 \wedge \ldots$
$\equiv$ ⟨ **?** ⟩
?

$\vdots$
$\equiv$ ⟨ ... ⟩
$\ldots \vee P_1 \vee P_2 \vee \ldots$
$\equiv$ ⟨ "Reason for $P_1 \Rightarrow P_2$" with "Def. of $\Rightarrow$ via $\vee$" ⟩
$\ldots \vee P_2 \vee \ldots$

$\vdots$
$\equiv$ ⟨ ... ⟩
$\ldots \wedge P_1 \wedge P_2 \wedge \ldots$
$\equiv$ ⟨ "Reason for $P_1 \Rightarrow P_2$" with "Def. of $\Rightarrow$ via $\wedge$" ⟩
$\ldots \wedge P_1 \wedge \ldots$

How can you simplify if you know $S_1 \subseteq S_2$?

$\vdots$
$=$ ⟨ ... ⟩
$\ldots \cup S_1 \cup S_2 \cup \ldots$
$=$ ⟨ **?** ⟩
?

$\vdots$
$=$ ⟨ ... ⟩
$\ldots \cap S_1 \cap S_2 \cap \ldots$
$=$ ⟨ **?** ⟩
?

$\longrightarrow$ Set Theory:

- "Set inclusion via $\cup$" $S \subseteq T \equiv S \cup T = T$
- "Set inclusion via $\cap$" $S \subseteq T \equiv S \cap T = S$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-18

Part 2: Introduction to Relations (ctd.)

What is a Relation?

A **relation**
is a subset
of a Cartesian product.

What is a Binary Relation?

A **binary relation**
is a set of pairs.

(Graphs), Simple Graphs

A **graph** consists of:

- a set of "nodes" or "vertices"
- a set of "edges" or "arrows"
- "incidence" information specifying how edges connect nodes

— *more details another day.*

A **simple graph** consists of:

- a set of "nodes", and
- a set of "edges", which **are** pairs of nodes.

(A simple graph has no "parallel edges".)

Formally: A **simple graph** $\langle N, E \rangle$ is a pair consisting of

- a set N, the elements of which are called "nodes", and
- a relation E with $E \in N \leftrightarrow N$, the element pairs of which are called "edges".

Simple Graphs

A **simple graph** consists of:

- a set of "nodes", and
- a set of "edges", which **are** pairs of nodes.

(A simple graph has no "parallel edges".)

Formally: A **simple graph** $\langle N, E \rangle$ is a pair consisting of

- a set N, the elements of which are called "nodes", and
- a relation E with $E \in N \leftrightarrow N$, the element pairs of which are called "edges".

Even more formally: A **simple graph** $\langle N, E \rangle$ is a pair consisting of

- a set N, and
- a relation E with $E \in N \leftrightarrow N$.

Given a simple graph $\langle N, E \rangle$, the elements of N are called "nodes" and the elements of E are called "edges".

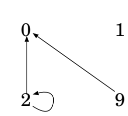
Simple Graphs: Example

Formally: A **simple graph** $\langle N, E \rangle$ is a pair consisting of

- a set N, the elements of which are called "nodes", and
- a relation E with $E \in N \leftrightarrow N$, the element pairs of which are called "edges".

Example: $G_1 = \langle \{2, 0, 1, 9\}, \{\langle 2, 0 \rangle, \langle 9, 0 \rangle, \langle 2, 2 \rangle\} \rangle$

Graphs are normally visualised via **graph drawings**:



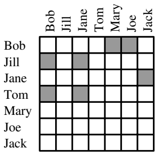
Simple graphs are essentially just relations!

Reasoning with relations is reasoning about graphs!

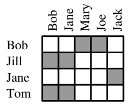
Visualising Binary Relations

$⌞ Person ⌟ = \{Bob, Jill, Jane, Tom, Mary, Joe, Jack\}$

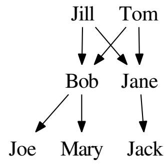
$parentOf = \{\langle Jill, Bob\rangle, \langle Jill, Jane\rangle, \langle Tom, Bob\rangle, \langle Tom, Jane\rangle,$
$\langle Bob, Mary\rangle, \langle Bob, Joe\rangle, \langle Jane, Jack\rangle\}$



$parentOf : Person \leftrightarrow Person$



$parentOf \in (parents \leftrightarrow children)$



$parents = \mathrm{Dom}\ parentOf = \{Bob, Jill, Jane, Tom\}$
$children = \mathrm{Ran}\ parentOf = \{Bob, Jane, Mary, Joe, Jack\}$

Expressing relationship: $\langle Jill, Bob\rangle \in parentOf \quad\equiv\quad Jill ⦗ parentOf ⦘ Bob$

Notation for Relationship

Notations for "x is related via R with y":

- explicit membership notation: $\langle x, y\rangle \in R$
- ambiguous traditional infix notation: $x\,R\,y$
- CALCCHECK: $x ⦗ R ⦘ y$

Type "\ ((... \ ))" for these "tortoise shell bracket" Unicode codepoints

The operator $\_⦗\_⦘\_ : t_1 \to (t_1 \leftrightarrow t_2) \to t_2 \to \mathbb{B}$

- is conjunctional:
  $(1 = x ⦗ R ⦘ y < 5) \quad\equiv\quad (1 = x) \wedge (x ⦗ R ⦘ y) \wedge (y < 5)$
- and calculational:

x
$⦗ R ⦘ \quad \langle$ Reason why $x ⦗ R ⦘ y$ $\rangle$
y

Experimental Key Bindings

— US keyboard only! Firefox only?

- `Alt-=` for $\equiv$ in addition to `\==`
- `Alt-<` for $\langle$ in addition to `\<`
- `Alt->` for $\rangle$ in addition to `\>`
- `Alt-(` for ⦗ in addition to `\((`
- `Alt-)` for ⦘ in addition to `\))`

Set Operations Used as Operations on Binary Relations

Relation union: $\langle u, v\rangle \in (R \cup S) \equiv \langle u, v\rangle \in R \vee \langle u, v\rangle \in S$
$u ⦗ R \cup S ⦘ v \equiv u ⦗ R ⦘ v \vee u ⦗ S ⦘ v$

Relation intersection: $u ⦗ R \cap S ⦘ v \equiv u ⦗ R ⦘ v \wedge u ⦗ S ⦘ v$

Relation difference: $u ⦗ R - S ⦘ v \equiv u ⦗ R ⦘ v \wedge \neg(u ⦗ S ⦘ v)$

Relation complement: $u ⦗ \sim R ⦘ v \equiv \neg(u ⦗ R ⦘ v)$

Relation extensionality: $R = S \equiv (\forall x \bullet \forall y \bullet x ⦗ R ⦘ y \equiv x ⦗ S ⦘ y)$
$R = S \equiv (\forall x, y \bullet x ⦗ R ⦘ y \equiv x ⦗ S ⦘ y)$

Relation inclusion: $R \subseteq S \equiv (\forall x \bullet \forall y \bullet x ⦗ R ⦘ y \Rightarrow x ⦗ S ⦘ y)$
$R \subseteq S \equiv (\forall x \bullet \forall y \mid x ⦗ R ⦘ y \bullet x ⦗ S ⦘ y)$
$R \subseteq S \equiv (\forall x, y \bullet x ⦗ R ⦘ y \Rightarrow x ⦗ S ⦘ y)$
$R \subseteq S \equiv (\forall x, y \mid x ⦗ R ⦘ y \bullet x ⦗ S ⦘ y)$

Empty and Universal Binary Relations

- The **empty relation** on $\langle t_1, t_2\rangle$ is $\{\} : t_1 \leftrightarrow t_2$
  $x ⦗ \{\} ⦘ y \equiv false$
  $\langle x, y\rangle \in \{\} \equiv false$
- The **universal relation** on $\langle t_1, t_2\rangle$ is $⌞\langle t_1, t_2\rangle⌟ : t_1 \leftrightarrow t_2$ or $U : t_1 \leftrightarrow t_2$
  $x ⦗ ⌞\langle t_1, t_2\rangle⌟ ⦘ y \equiv true$ $\qquad x ⦗ U ⦘ y \equiv true$
  $\langle x, y\rangle \in ⌞\langle t_1, t_2\rangle⌟ \equiv true$ $\qquad \langle x, y\rangle \in U \equiv true$
- The **universal relation on $B \times C$** is $B \times C$
  $x ⦗ B \times C ⦘ y \equiv x \in B \wedge y \in C$
  (14.4) $\langle x, y\rangle \in B \times C \equiv x \in B \wedge y \in C$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-20

Relations in Set Theory

Plan for Today

- Relations
  - Set-theoretic definition of relational operations: Converse, composition

Relation-Algebraic Operations: Operations on Relations

- Set operations $\sim, \cup, \cap, -, \Rightarrow$ are all available.
- If $R : B \leftrightarrow C$, $\qquad B \xrightarrow{R} C$
  then its **converse** $R^{\smile} : C \leftrightarrow B$
  (in the textbook called "inverse" and written: R^{-1})
  stands for "going R backwards": $\quad c ⦗ R^{\smile} ⦘ b \equiv b ⦗ R ⦘ c$
- If $R : B \leftrightarrow C$ and $S : C \leftrightarrow D$, $\qquad B \xrightarrow{R} C \xrightarrow{S} D$
  then their **composition** $R ⨾ S$
  (in the textbook written: $R \circ S$)
  is a relation in $B \leftrightarrow D$, and stands for
  "going first a step via R, and then a step via S":
  $b ⦗ R ⨾ S ⦘ d \equiv (\exists c : C \bullet b ⦗ R ⦘ c ⦗ S ⦘ d)$

The resulting **relation algebra**

- allows concise formalisations **without quantifications**,
- enables simple calculational proofs.

Proving Self-inverse of Converse: $(R^{\smile})^{\smile} = R$

$(R^{\smile})^{\smile} = R$
$\equiv$ ⟨ Relation extensionality ⟩
$\forall x, y \bullet x ⦗ (R^{\smile})^{\smile} ⦘ y \equiv x ⦗ R ⦘ y$
$\equiv$ ⟨ ... ⟩
$true$

Using "Relation extensionality":
Subproof for `$\forall x, y \bullet x ⦗ (R^{\smile})^{\smile} ⦘ y \equiv x ⦗ R ⦘ y$`:
For any x, y:
$x ⦗ (R^{\smile})^{\smile} ⦘ y$
$\equiv$ ⟨ Converse ⟩
$y ⦗ R^{\smile} ⦘ x$
$\equiv$ ⟨ Converse ⟩
$x ⦗ R ⦘ y$

Proving Isotonicity of Converse

Proving $R \subseteq S \equiv R^{\smile} \subseteq S^{\smile}$:

$R^{\smile} \subseteq S^{\smile}$
$\equiv$ ⟨ Relation inclusion ⟩
$\forall y, x \mid y ⦗ R^{\smile} ⦘ x \bullet y ⦗ S^{\smile} ⦘ x$
$\equiv$ ⟨ Converse, dummy permutation ⟩
$\forall x, y \mid x ⦗ R ⦘ y \bullet x ⦗ S ⦘ y$
$\equiv$ ⟨ Relation inclusion ⟩
$R \subseteq S$

Operations on Relations: Composition $B \xrightarrow{R} C \xrightarrow{S} D$

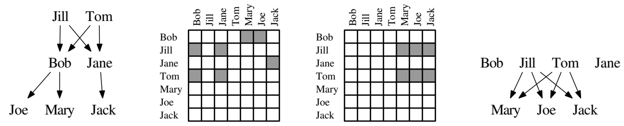
If $R : B \leftrightarrow C$ and $S : C \leftrightarrow D$, then their **composition** $R ⨾ S : B \leftrightarrow D$ is defined by:

(14.20) $\quad b \,⦇ R ⨾ S ⦈\, d \;\equiv\; (\exists c : C \bullet b \,⦇ R ⦈\, c \,⦇ S ⦈\, d)$ (for $b : B, d : D$)

(14.20) $\quad b \,⦇ R ⨾ S ⦈\, d \;\equiv\; (\exists c : C \bullet b \,⦇ R ⦈\, c \land c \,⦇ S ⦈\, d)$ (for $b : B, d : D$)

$parentOf = \{\langle Jill, Bob\rangle, \langle Jill, Jane\rangle, \langle Tom, Bob\rangle, \langle Tom, Jane\rangle, \langle Bob, Mary\rangle, \langle Bob, Joe\rangle, \langle Jane, Jack\rangle\}$

$grandparentOf = parentOf ⨾ parentOf$
$= \{\langle Jill, Mary\rangle, \langle Jill, Joe\rangle, \langle Jill, Jack\rangle \langle Tom, Mary\rangle, \langle Tom, Joe\rangle, \langle Tom, Jack\rangle\}$



Sub-identity and Identity Relations

- The **(sub-)identity relation** on $B : \mathbf{set}\ t$ is $\mathsf{id}\ B : t \leftrightarrow t$

$\mathsf{id}\ B = \{x : t \mid x \in B \bullet \langle x, x\rangle\}$:

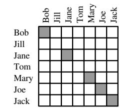
$x \,⦇ \mathsf{id}\ B ⦈\, y \;\equiv\; x = y \in B$

$\langle x, y\rangle \in \mathsf{id}\ B \;\equiv\; x = y \land y \in B$

— LADM writes ι_B

— Writing "id B" follows the Z notation

id *children* =

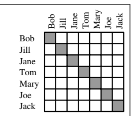


- The **identity relation** on $t : Type$ is $\mathbb{I} : t \leftrightarrow t$ with $\mathbb{I} = \mathsf{id}\ U$

$x \,⦇ \mathbb{I} ⦈\, y \;\equiv\; x = y$

$\langle x, y\rangle \in \mathbb{I} \;\equiv\; x = y$

$(\mathbb{I} : Person \leftrightarrow Person)$ =



- **The "id" and "$\mathbb{I}$" notations are different from some previous years!**

Domain and Range of Binary Relations

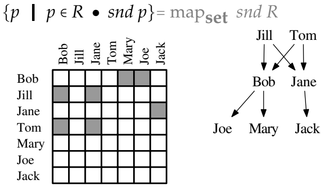
For $R : t_1 \leftrightarrow t_2$, we define $Dom\ R : \mathbf{set}\ t_1$ and $Ran\ R : \mathbf{set}\ t_2$ as follows:

(14.16) $Dom\ R = \{x : t_1 \mid (\exists y : t_2 \bullet x \,⦇ R ⦈\, y)\} = \{p \mid p \in R \bullet fst\ p\} = \mathrm{map}_{\mathbf{set}}\ fst\ R$

(14.17) $Ran\ R = \{y : t_2 \mid (\exists x : t_1 \bullet x \,⦇ R ⦈\, y)\} = \{p \mid p \in R \bullet snd\ p\} = \mathrm{map}_{\mathbf{set}}\ snd\ R$

"Membership in `Dom`":
$x \in Dom\ R \equiv (\exists y : t_2 \bullet x \,⦇ R ⦈\, y)$

"Membership in `Ran`":
$y \in Ran\ R \equiv (\exists x : t_1 \bullet x \,⦇ R ⦈\, y)$



$parents = Dom\ parentOf = \{Bob, Jill, Jane, Tom\}$

$children = Ran\ parentOf = \{Bob, Jane, Mary, Joe, Jack\}$

Formalise Without Quantifiers!

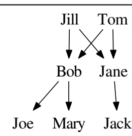
P = type of persons
$C : P \leftrightarrow P$
$p \,⦇ C ⦈\, q \equiv p$ called q

Remember: For $R : t_1 \leftrightarrow t_2$:
"Membership in `Dom`":
$x \in Dom\ R \equiv (\exists y : t_2 \bullet x \,⦇ R ⦈\, y)$

"Membership in `Ran`":
$y \in Ran\ R \equiv (\exists x : t_1 \bullet x \,⦇ R ⦈\, y)$

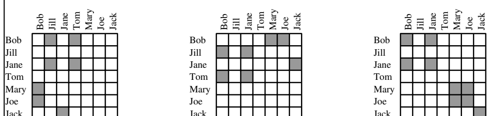
1. Helen called somebody.
   $Helen \in Dom\ C \quad \equiv \quad (\exists y : P \bullet Helen \,⦇ C ⦈\, y)$
2. For everybody, there is somebody they haven't called.
   $Dom\ (\sim C) = \llcorner P \lrcorner$
   $Dom\ (\sim C) = U$

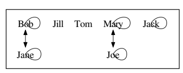
Combining Several Operations



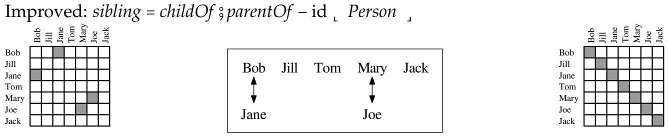
How to define siblings?

- First attempt: $childOf ⨾ parentOf$, with $childOf = parentOf^{\smile}$





- Improved: $sibling = childOf ⨾ parentOf - \mathsf{id} \llcorner Person \lrcorner$



Properties of Converse $B \xrightarrow{R} C$

If $R : B \leftrightarrow C$, then its **converse** $R^{\smile} : C \leftrightarrow B$ is defined by:

(14.18) $\quad \langle c, b\rangle \in R^{\smile} \;\equiv\; \langle b, c\rangle \in R$ (for $b : B$ and $c : C$)

(14.18) $\quad c \,⦇ R^{\smile} ⦈\, b \;\equiv\; b \,⦇ R ⦈\, c$ (for $b : B$ and $c : C$)

(14.19) **Properties of Converse:** Let $R, S : B \leftrightarrow C$ be relations.

(a) $Dom\ (R^{\smile}) = Ran\ R$
(b) $Ran\ (R^{\smile}) = Dom\ R$
(c) If $R \in S \leftrightarrow T$, then $R^{\smile} \in T \leftrightarrow S$
(d) $(R^{\smile})^{\smile} = R$
(e) $R \subseteq S \;\equiv\; R^{\smile} \subseteq S^{\smile}$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-20

Part 2: Relation-Algebraic Formalisation Examples

P = type of persons
$C : P \leftrightarrow P$ — "called"
$B : P \leftrightarrow P$ — "brother of"
$Aos : P$
$Jun : P$

Convert into English (via predicate logic):

$Aos \,⦇ C ⦈\, Jun$
$Aos \,⦇ C ⨾ B ⦈\, Jun$
$Aos \,⦇ \sim (C ⨾ \sim B) ⦈\, Jun$
$Aos \,⦇ \sim (\sim C ⨾ B) ⦈\, Jun$
$Aos \,⦇ \sim ((C \cap \sim (B ⨾ C^{\smile})) ⨾ \sim B) ⦈\, Jun$
$(B ⨾ (\{Jun\} \times \llcorner P \lrcorner)) \cap (C ⨾ C^{\smile}) \subseteq \mathsf{id} \llcorner P \lrcorner$

Translating between Relation Algebra and Predicate Logic

$R = S$	$\equiv$	$(\forall x, y \bullet x \,⦇ R ⦈\, y \equiv x \,⦇ S ⦈\, y)$
$R \subseteq S$	$\equiv$	$(\forall x, y \bullet x \,⦇ R ⦈\, y \Rightarrow x \,⦇ S ⦈\, y)$
$u \,⦇ \{\} ⦈\, v$	$\equiv$	$false$
$u \,⦇ U ⦈\, v$	$\equiv$	$true$
$u \,⦇ A \times B ⦈\, v$	$\equiv$	$u \in A \land v \in B$
$u \,⦇ \sim S ⦈\, v$	$\equiv$	$\neg(u \,⦇ S ⦈\, v)$
$u \,⦇ S \cup T ⦈\, v$	$\equiv$	$u \,⦇ S ⦈\, v \lor u \,⦇ T ⦈\, v$
$u \,⦇ S \cap T ⦈\, v$	$\equiv$	$u \,⦇ S ⦈\, v \land u \,⦇ T ⦈\, v$
$u \,⦇ S - T ⦈\, v$	$\equiv$	$u \,⦇ S ⦈\, v \land \neg(u \,⦇ T ⦈\, v)$
$u \,⦇ S \Rightarrow T ⦈\, v$	$\equiv$	$u \,⦇ S ⦈\, v \Rightarrow (u \,⦇ T ⦈\, v)$
$u \,⦇ \mathbb{I} ⦈\, v$	$\equiv$	$u = v$
$u \,⦇ \mathsf{id}\ A ⦈\, v$	$\equiv$	$u = v \in A$
$u \,⦇ R^{\smile} ⦈\, v$	$\equiv$	$v \,⦇ R ⦈\, u$
$u \,⦇ R ⨾ S ⦈\, v$	$\equiv$	$(\exists x \bullet u \,⦇ R ⦈\, x \,⦇ S ⦈\, v)$

P = type of persons
$C : P \leftrightarrow P$ — "called"
$B : P \leftrightarrow P$ — "brother of"
$Aos : P$
$Jun : P$

Convert into English (via predicate logic):

$Aos \,⦇ C ⨾ B ⦈\, Jun$
$\equiv \langle$ (14.20) Relation composition $\rangle$
$(\exists b \bullet Aos \,⦇ C ⦈\, b \,⦇ B ⦈\, Jun)$

"Aos called some brother of Jun."
"Aos called a brother of Jun."

$Aos ⦇ {\sim}(C \fatsemi {\sim}B) ⦈ Jun$
$\equiv$ ⟨ (11.17r) Relation complement ⟩
$\neg(Aos ⦇ C \fatsemi {\sim}B ⦈ Jun)$
$\equiv$ ⟨ (14.20) Relation composition ⟩
$\neg(\exists\, p \bullet Aos ⦇ C ⦈ p ⦇ {\sim}B ⦈ Jun)$
$\equiv$ ⟨ (11.17r) Relation complement ⟩
$\neg(\exists\, p \bullet Aos ⦇ C ⦈ p \wedge \neg(p ⦇ B ⦈ Jun))$
$\equiv$ ⟨ (9.18b) Generalised De Morgan ⟩
$(\forall\, p \bullet \neg(Aos ⦇ C ⦈ p \wedge \neg(p ⦇ B ⦈ Jun)))$
$\equiv$ ⟨ (3.47) De Morgan, (3.12) Double negation ⟩
$(\forall\, p \bullet \neg(Aos ⦇ C ⦈ p) \vee p ⦇ B ⦈ Jun)$
$\equiv$ ⟨ (9.3a) Trading for $\forall$ ⟩
$(\forall\, p \mid Aos ⦇ C ⦈ p \bullet p ⦇ B ⦈ Jun)$

"Everybody Aos called is a brother of Jun."
"Aos called only brothers of Jun."

Formalise Without Quantifiers! (2)

P	:=	type of persons
C	:	$P \leftrightarrow P$
$p ⦇ C ⦈ q$	:≡	p called q

1. Helen called somebody who called her.
2. For arbitrary people x, z, if x called z, then there is sombody whom x called, and who was called by somebody who also called z.
3. For arbitrary people x, y, z, if x called y, and y was called by somebody who also called z, then x called z.
4. Obama called everybody directly, or indirectly via at most two intermediaries.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-23

Relations in Set Theory

Plan for Today

- Relations
  - Some properties of relation composition, e.g., $\fatsemi$ is monotonic
  - Some properties of relations, e.g., "R is transitive", "E is an order"

  Moving towards relation-algebraic formalisations and reasoning

Translating between Relation Algebra and Predicate Logic

$R = S$	$\equiv$	$(\forall\, x, y \bullet x ⦇ R ⦈ y \equiv x ⦇ S ⦈ y)$
$R \subseteq S$	$\equiv$	$(\forall\, x, y \bullet x ⦇ R ⦈ y \Rightarrow x ⦇ S ⦈ y)$
$u ⦇ \{\} ⦈ v$	$\equiv$	$false$
$u ⦇ U ⦈ v$	$\equiv$	$true$
$u ⦇ A \times B ⦈ v$	$\equiv$	$u \in A \wedge v \in B$
$u ⦇ {\sim}S ⦈ v$	$\equiv$	$\neg(u ⦇ S ⦈ v)$
$u ⦇ S \cup T ⦈ v$	$\equiv$	$u ⦇ S ⦈ v \vee u ⦇ T ⦈ v$
$u ⦇ S \cap T ⦈ v$	$\equiv$	$u ⦇ S ⦈ v \wedge u ⦇ T ⦈ v$
$u ⦇ S - T ⦈ v$	$\equiv$	$u ⦇ S ⦈ v \wedge \neg(u ⦇ T ⦈ v)$
$u ⦇ S \Rightarrow T ⦈ v$	$\equiv$	$u ⦇ S ⦈ v \Rightarrow (u ⦇ T ⦈ v)$
$u ⦇ \mathbb{I} ⦈ v$	$\equiv$	$u = v$
$u ⦇ \mathrm{id}\ A ⦈ v$	$\equiv$	$u = v \in A$
$u ⦇ R^{\smile} ⦈ v$	$\equiv$	$v ⦇ R ⦈ u$
$u ⦇ R \fatsemi S ⦈ v$	$\equiv$	$(\exists\, x \bullet u ⦇ R ⦈ x ⦇ S ⦈ v)$

P	=	type of persons	
C	:	$P \leftrightarrow P$	— "called"
B	:	$P \leftrightarrow P$	— "brother of"
Aos	:	P	
Jun	:	P	

Convert into English (via predicate logic):

$Aos ⦇ C ⦈ Jun$

$Aos ⦇ C \fatsemi B ⦈ Jun$

$Aos ⦇ {\sim}(C \fatsemi {\sim}B) ⦈ Jun$

$Aos ⦇ {\sim}({\sim}C \fatsemi B) ⦈ Jun$

$Aos ⦇ {\sim}((C \cap {\sim}(B \fatsemi C^{\smile})) \fatsemi {\sim}B) ⦈ Jun$

$(B \fatsemi (\{Jun\} \times U)) \cap (C \fatsemi C^{\smile}) \quad \subseteq \quad \mathbb{I}$

$Aos ⦇ {\sim}((C \cap {\sim}(B \fatsemi C^{\smile})) \fatsemi {\sim}B) ⦈ Jun$
$\equiv$ ⟨ Relation complement ⟩
$\neg(Aos ⦇ (C \cap {\sim}(B \fatsemi C^{\smile})) \fatsemi {\sim}B ⦈ Jun)$
$\equiv$ ⟨ Relation composition ⟩
$\neg(\exists\, p \bullet Aos ⦇ C \cap {\sim}(B \fatsemi C^{\smile}) ⦈ p ⦇ {\sim}B ⦈ Jun)$
$\equiv$ ⟨ Relation intersection ⟩
$\neg(\exists\, p \bullet Aos ⦇ C ⦈ p \wedge Aos ⦇ {\sim}(B \fatsemi C^{\smile}) ⦈ p \wedge p ⦇ {\sim}B ⦈ Jun)$
$\equiv$ ⟨ Relation complement ⟩
$\neg(\exists\, p \bullet Aos ⦇ C ⦈ p \wedge \neg(Aos ⦇ B \fatsemi C^{\smile} ⦈ p) \wedge \neg(p ⦇ B ⦈ Jun))$
$\equiv$ ⟨ Relation composition ⟩
$\neg(\exists\, p \bullet Aos ⦇ C ⦈ p \wedge \neg(\exists q \bullet Aos ⦇ B ⦈ q ⦇ C^{\smile} ⦈ p) \wedge \neg(p ⦇ B ⦈ Jun))$
$\equiv$ ⟨ (9.18b) Generalised De Morgan ⟩
...

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-20

Part 2: Some Properties of Relation Composition

First Simple Properties of Composition

If $R : B \leftrightarrow C$ and $S : C \leftrightarrow D$, then their **composition** $R \fatsemi S : B \leftrightarrow D$ is defined by:

(14.20) $b ⦇ R \fatsemi S ⦈ d \equiv (\exists c : C \bullet b ⦇ R ⦈ c \wedge c ⦇ S ⦈ d)$ (for $b : B, d : D$)

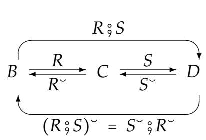
(14.22) **Associativity of $\fatsemi$:** $Q \fatsemi (R \fatsemi S) = (Q \fatsemi R) \fatsemi S$

Left- and Right-identities of $\fatsemi$: If $R \in X \leftrightarrow Y$, then: $\mathrm{id}\ X \fatsemi R = R = R \fatsemi \mathrm{id}\ Y$

We defined: $\mathbb{I} = \mathrm{id}\ U$ with: **Relationship via $\mathbb{I}$:** $x ⦇ \mathbb{I} ⦈ y \equiv x = y$

$\mathbb{I}$ is "the" identity of composition: **Identity of $\fatsemi$:** $\mathbb{I} \fatsemi R = R = R \fatsemi \mathbb{I}$

Contravariance: $(R \fatsemi S)^{\smile} = S^{\smile} \fatsemi R^{\smile}$

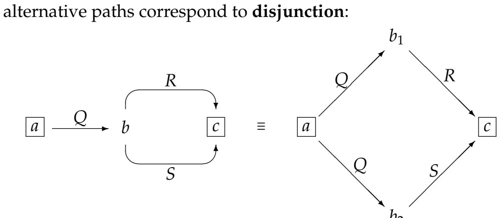


Distributivity of Relation Composition over Union

Composition distributes over **union** from both sides:

(14.23) $Q \fatsemi (R \cup S) = Q \fatsemi R \cup Q \fatsemi S$
$(P \cup Q) \fatsemi R = P \fatsemi R \cup Q \fatsemi R$

In **control flow** diagrams (NFA) — boxed variables are free; others existentially quantified; alternative paths correspond to **disjunction**:



$(\exists\, b \bullet a ⦇ Q ⦈ b ⦇ R \cup S ⦈ c) \equiv$
$(\exists\, b_1, b_2 \bullet a ⦇ Q ⦈ b_1 ⦇ R ⦈ c \vee a ⦇ Q ⦈ b_2 ⦇ S ⦈ c)$

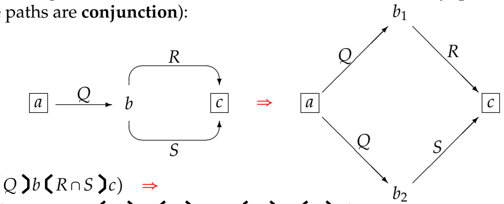
Sub-Distributivity of Composition over Intersection

Composition **sub**-distributes over **intersection** from both sides:

(14.24) $Q ⨾ (R \cap S) \quad \subseteq \quad Q ⨾ R \cap Q ⨾ S$

$(P \cap Q) ⨾ R \quad \subseteq \quad P ⨾ R \cap Q ⨾ R$

In **constraint** diagrams (boxed variables are free; others existentially quantified; alternative paths are **conjunction**):



$(\exists b \bullet a (\!|Q|\!) b (\!|R \cap S|\!) c) \quad \Rightarrow$

$(\exists b_1, b_2 \bullet a (\!|Q|\!) b_1 (\!|R|\!) c \wedge a (\!|Q|\!) b_2 (\!|S|\!) c)$

Counterexample for $\Leftarrow$:

$Q :=$ neighbour of $\quad R :=$ brother of $\quad S :=$ parent of

Monotonicity of Relation Composition

Relation composition is monotonic in both arguments:

$Q \subseteq R \quad \Rightarrow \quad Q ⨾ S \subseteq R ⨾ S$

$Q \subseteq R \quad \Rightarrow \quad P ⨾ Q \subseteq P ⨾ R$

We could prove this via **"Relation inclusion"** *and* **"For any"**, *but we don't need to:*

Assume $Q \subseteq R$, which by "Definition of $\subseteq$ via $\cup$" is equivalent to $Q \cup R = R$:

Proving $Q ⨾ S \subseteq R ⨾ S$:

$R ⨾ S$

$=$ ⟨ Assumption $Q \cup R = R$ ⟩

$(Q \cup R) ⨾ S$

$=$ ⟨ (14.23) Distributivity of ⨾ over $\cup$ ⟩

$Q ⨾ S \cup R ⨾ S$

$\supseteq$ ⟨ (11.31) Strengthening $S \subseteq S \cup T$ ⟩

$Q ⨾ S$

`with`$_3$: Rewriting Theorems before Rewriting

ThmA `with` *ThmB*

- If *ThmB* gives rise to an equality/equivalence $L = R$: Rewrite *ThmA* with $L \mapsto R$
- E.g.: Assumption \`$Q \subseteq R$\` with "Relation inclusion":

  $Q \subseteq R$ rewrites via $Q \subseteq R \mapsto \forall x \bullet \forall y \bullet x (\!|Q|\!) y \Rightarrow x (\!|R|\!) y$

  to: $\forall x \bullet \forall y \bullet x (\!|Q|\!) y \Rightarrow x (\!|R|\!) y$

  which can be instantiated to: to: $a (\!|Q|\!) b \Rightarrow a (\!|R|\!) b$

```
  ∃ b • a ( Q ) b ∧ b ( S ) c
⇒⟨ "Body monotonicity of ∃" with "Monotonicity of ∧"
    with assumption `Q ⊆ R` with "Relation inclusion" ⟩
  ∃ b • a ( R ) b ∧ b ( S ) c
```

`with`$_2$ and `with`$_3$: Example

```
  ∃ b • a ( Q ) b ∧ b ( S ) c
⇒⟨ "Body monotonicity of ∃" with "Monotonicity of ∧"
    with assumption `Q ⊆ R` with "Relation inclusion" ⟩
  ∃ b • a ( R ) b ∧ b ( S ) c
```

- assumption '$Q \subseteq R$' gives you $Q \subseteq R$
- assumption '$Q \subseteq R$' with "Relation inclusion" gives you via `with`$_3$: $\forall x \bullet \forall y \bullet x (\!|Q|\!) y \Rightarrow x (\!|R|\!) y$

  and then via implicit "Instantiation" triggered by the next `with`$_2$: $a (\!|Q|\!) b \Rightarrow a (\!|R|\!) b$
- "Monotonicity of $\wedge$" with assumption '$Q \subseteq R$' with "Relation inclusion"

  gives you via `with`$_2$: $a (\!|Q|\!) b \wedge b (\!|S|\!) c \Rightarrow a (\!|R|\!) b \wedge b (\!|S|\!) c$
- "Body monotonicity of $\exists$" with "Monotonicity of $\wedge$" with assumption '$Q \subseteq R$' with "Relation inclusion"

  gives you via `with`$_2$: $(\exists b \bullet a (\!|Q|\!) b \wedge b (\!|S|\!) c) \Rightarrow (\exists b \bullet a (\!|R|\!) b \wedge b (\!|S|\!) c)$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-25

Properties of Relations

Plan for Today

- Some properties of relations, e.g., "R is univalent", "F is bijective"
- Symbols following the Z Notation: Function Set Arrows, Domain- and Range-Restrictions

Moving towards relation-algebraic formalisations and reasoning

Properties of Homogeneous Relations (ctd.)

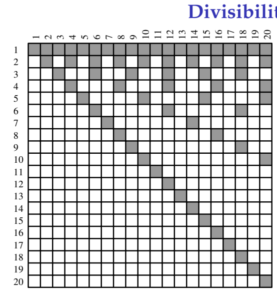
reflexive	$\mathbb{I} \subseteq R$	$(\forall b : B \bullet b (\!|R|\!) b)$
irreflexive	$\mathbb{I} \cap R = \{\}$	$(\forall b : B \bullet \neg(b (\!|R|\!) b))$
symmetric	$R^{\smile} = R$	$(\forall b, c : B \bullet b (\!|R|\!) c \equiv c (\!|R|\!) b)$
antisymmetric	$R \cap R^{\smile} \subseteq \mathbb{I}$	$(\forall b, c \bullet b (\!|R|\!) c \wedge c (\!|R|\!) b \Rightarrow b = c)$
asymmetric	$R \cap R^{\smile} = \{\}$	$(\forall b, c : B \bullet b (\!|R|\!) c \Rightarrow \neg(c (\!|R|\!) b))$
transitive	$R ⨾ R \subseteq R$	$(\forall b, c, d \bullet b (\!|R|\!) c \wedge c (\!|R|\!) d \Rightarrow b (\!|R|\!) d)$

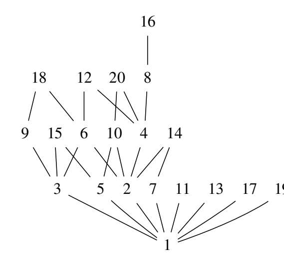
R is an **equivalence (relation) on** B iff it is reflexive, transitive, and symmetric.

R is a **(partial) order on** B iff it is reflexive, transitive, and antisymmetric.
(E.g., $\leq, \geq, \subseteq, \supseteq, |$)

R is a **strict-order on** B iff it is irreflexive, transitive, and asymmetric.
(E.g., $<, >, \subset, \supset$)

Divisibility Order with Hasse Diagram

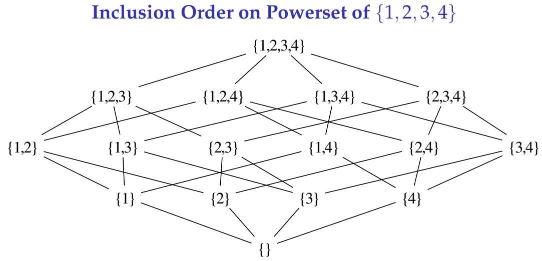




Hasse diagram for an **order**:

- Edge direction is **upwards** — **antisymmetric**
- Loops not drawn — **reflexive**
- Transitive edges not drawn — **transitive**

Inclusion Order on Powerset of $\{1, 2, 3, 4\}$



Hasse diagram for an **order**:

- Edge direction is **upwards** — **antisymmetric**
- Loops not drawn — **reflexive**
- Transitive edges not drawn — **transitive**

Properties of Heterogeneous Relations

A relation $R : B \leftrightarrow C$ is called:

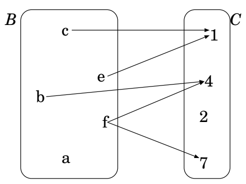
univalent determinate	$R^{\smile} ⨾ R \subseteq \mathbb{I}$	$\forall b, c_1, c_2 \bullet b (\!|R|\!) c_1 \wedge b (\!|R|\!) c_2 \Rightarrow c_1 = c_2$
total	$Dom\ R = U$ $Dom\ R = \llcorner B \lrcorner$ $\mathbb{I} \subseteq R ⨾ R^{\smile}$	$\forall b : B \bullet (\exists c : C \bullet b (\!|R|\!) c)$
injective	$R ⨾ R^{\smile} \subseteq \mathbb{I}$	$\forall b_1, b_2, c \bullet b_1 (\!|R|\!) c \wedge b_2 (\!|R|\!) c \Rightarrow b_1 = b_2$
surjective	$Ran\ R = U$ $Ran\ R = \llcorner C \lrcorner$ $\mathbb{I} \subseteq R^{\smile} ⨾ R$	$\forall c : C \bullet (\exists b : B \bullet b (\!|R|\!) c)$
a **mapping**	iff it is univalent and total	
bijective	iff it is injective and surjective	

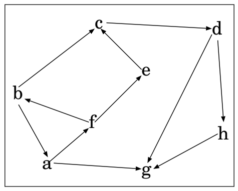
Univalent relations are also called **(partial) functions**.

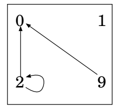
Mappings are also called **total functions**.

Properties of Heterogeneous Relations — Examples 1

univalent	$R^{\smile} ⨾ R \subseteq \mathbb{I}$	$\forall b, c_1, c_2 \bullet b \,(R)\, c_1 \wedge b \,(R)\, c_2 \Rightarrow c_1 = c_2$
total	$Dom\ R = U$ $\mathbb{I} \subseteq R ⨾ R^{\smile}$	$\forall b : B \bullet (\exists c : C \bullet b \,(R)\, c)$
a **mapping**	iff it is univalent and total	

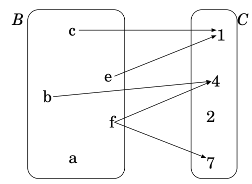


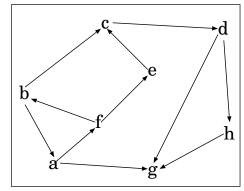


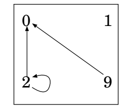


Properties of Heterogeneous Relations — Examples 2

injective	$R ⨾ R^{\smile} \subseteq \mathbb{I}$	$\forall b_1, b_2, c \bullet b_1 \,(R)\, c \wedge b_2 \,(R)\, c \Rightarrow b_1 = b_2$
surjective	$Ran\ R = U$ $\mathbb{I} \subseteq R^{\smile} ⨾ R$	$\forall c : C \bullet (\exists b : B \bullet b \,(R)\, c)$
bijective	iff it is injective and surjective	







Function Types versus Sets of Univalent Relations

A relation $R : B \leftrightarrow C$ is called:

univalent	$R^{\smile} ⨾ R \subseteq \mathbb{I}$	$\forall b, c_1, c_2 \bullet b \,(R)\, c_1 \wedge b \,(R)\, c_2 \Rightarrow c_1 = c_2$
total	$Dom\ R = U$	$\forall b : B \bullet (\exists c : C \bullet b \,(R)\, c)$
a **mapping**	iff it is univalent and total	

Univalent relations are also called **(partial) functions**.

Mappings are also called **total functions**.

— **These are of different type that functions of function type $B \rightarrow C$!**

The distinction corresponds to the way in which elements of the **Haskell** datatype *Data.Map.Map a b* are distinct from Haskell functions of type $a \rightarrow b$.

- A (set-theoretic) relation $R : B \leftrightarrow C$ **is** a set of pairs — "data"
- A function $f : B \rightarrow C$ is a different kind of entity — in Haskell, "computation"

  If $b : B$, then $f\ b$ is **never undefined**.

  (But may be **unspecified**, such as *head* ϵ in A1.3.)

Properties of Heterogeneous Relations — Notes

univalent	$R^{\smile} ⨾ R \subseteq \mathbb{I}$	$\forall b, c_1, c_2 \bullet b \,(R)\, c_1 \wedge b \,(R)\, c_2 \Rightarrow c_1 = c_2$
surjective	$\mathbb{I} \subseteq R^{\smile} ⨾ R$	$\forall c : C \bullet (\exists b : B \bullet b \,(R)\, c)$
total	$\mathbb{I} \subseteq R ⨾ R^{\smile}$	$\forall b : B \bullet (\exists c : C \bullet b \,(R)\, c)$
injective	$R ⨾ R^{\smile} \subseteq \mathbb{I}$	$\forall b_1, b_2, c \bullet b_1 \,(R)\, c \wedge b_2 \,(R)\, c \Rightarrow b_1 = b_2$

All these properties are defined for arbitrary relations! (Not only for functions!)

- R is univalent and surjective
  iff $R^{\smile} ⨾ R = \mathbb{I}$
  iff $R^{\smile}$ is a left-inverse of R
- R is total and injective
  iff $R ⨾ R^{\smile} = \mathbb{I}$
  iff $R^{\smile}$ is a right-inverse of R

It is convenient to have abbreviations, for example:

f is a partial function from X to Y:	$f \in X \nrightarrow Y$	
f is an injective mapping from X to Y:	$f \in X \rightarrowtail Y$	⟶ Z arrows!
f is a partial surjection from X to Y:	$f \in X \twoheadrightarrow Y$ (partial)	

The Z Specification Notation

- Mathematical notation intended for software specification

  Used for requirements contracts with customers who would be given a two-page "Z Reference Card"
- Very influential in Formal Methods; ISO-standardised
- Two parts:
  - Z **is** a typed set theory in first-order predicate logic

    — very close to the logic and set theory you are using in CalcCheck

    — except that in Z:
    - types **are** maximal sets
    - sets can be used in variable declarations: $\forall x : S \mid \ldots \bullet \ldots$,

      — which makes quantifier reasoning harder.
    - functions **are** univalent relations

    (CalcCheck and Haskell are type theories with embedded typed set theories.)
  - "Schemas" modelling of states and state transitions
- Avenue ⟶ Resources ⟶ Links ⟶ Z Specification Notation

Function Sets — Z Definition and Description [Spivey 1992]

In Z, $X \leftrightarrow Y = \mathbb{P}(X \times Y)$, and $x \mapsto y = (x, y)$ is an abbreviation for pairs.

$\nrightarrow$	–	Partial functions
$\rightarrow$	–	Total functions
$\rightarrowtail$ (partial)	–	Partial injections
$\rightarrowtail$	–	Total injections
$\twoheadrightarrow$ (partial)	–	Partial surjections
$\twoheadrightarrow$	–	Total surjections
$\rightarrowtail\!\!\!\!\rightarrow$	–	Bijections

$X \nrightarrow Y == \{ f : X \leftrightarrow Y \mid (\forall x : X;\ y_1, y_2 : Y \bullet (x \mapsto y_1) \in f \wedge (x \mapsto y_2) \in f \Rightarrow y_1 = y_2) \}$

$X \rightarrow Y == \{ f : X \nrightarrow Y \mid \mathrm{dom}\, f = X \}$

$X \rightarrowtail\!\!\!\!\!\nrightarrow Y == \{ f : X \nrightarrow Y \mid (\forall x_1, x_2 : \mathrm{dom}\, f \bullet f(x_1) = f(x_2) \Rightarrow x_1 = x_2) \}$

$X \rightarrowtail Y == (X \rightarrowtail\!\!\!\!\!\nrightarrow Y) \cap (X \rightarrow Y)$

$X \twoheadrightarrow\!\!\!\!\!\nrightarrow Y == \{ f : X \nrightarrow Y \mid \mathrm{ran}\, f = Y \}$

$X \twoheadrightarrow Y == (X \twoheadrightarrow\!\!\!\!\!\nrightarrow Y) \cap (X \rightarrow Y)$

$X \rightarrowtail\!\!\!\!\twoheadrightarrow Y == (X \twoheadrightarrow Y) \cap (X \rightarrowtail Y)$

If X and Y are sets, $X \nrightarrow Y$ is the set of partial functions from X to Y. These are relations which relate each member x of X to at most one member of Y. This member of Y, if it exists, is written $f(x)$. The set $X \rightarrow Y$ is the set of total functions from X to Y. These are partial functions whose domain is the whole of X; they relate each member of X to exactly one member of Y.

Function Sets — Z Definition and Laws (1) [Spivey 1992]

In Z, $X \leftrightarrow Y = \mathbb{P}(X \times Y)$, and $x \mapsto y = (x, y)$ is an abbreviation for pairs, and $S \circ R = R ⨾ S$.

$X \nrightarrow Y == \{ f : X \leftrightarrow Y \mid (\forall x : X;\ y_1, y_2 : Y \bullet (x \mapsto y_1) \in f \wedge (x \mapsto y_2) \in f \Rightarrow y_1 = y_2) \}$

$X \rightarrow Y == \{ f : X \nrightarrow Y \mid \mathrm{dom}\, f = X \}$

$X \rightarrowtail\!\!\!\!\!\nrightarrow Y == \{ f : X \nrightarrow Y \mid (\forall x_1, x_2 : \mathrm{dom}\, f \bullet f(x_1) = f(x_2) \Rightarrow x_1 = x_2) \}$

$X \rightarrowtail Y == (X \rightarrowtail\!\!\!\!\!\nrightarrow Y) \cap (X \rightarrow Y)$

Laws:

$f \in X \nrightarrow Y \Leftrightarrow f \circ f^{\sim} = \mathrm{id}(\mathrm{ran}\, f)$

$f \in X \rightarrowtail\!\!\!\!\!\nrightarrow Y \Leftrightarrow f \in X \nrightarrow Y \wedge f^{\sim} \in Y \nrightarrow X$

$f \in X \rightarrowtail Y \Leftrightarrow f \in X \rightarrow Y \wedge f^{\sim} \in Y \nrightarrow X$

$f \in X \rightarrowtail\!\!\!\!\!\nrightarrow Y \Rightarrow f(\!|S \cap T|\!) = f(\!|S|\!) \cap f(\!|T|\!)$

Function Sets — Z Definition and Laws [Spivey 1992]

In Z, $X \leftrightarrow Y = \mathbb{P}(X \times Y)$, and $x \mapsto y = (x, y)$ is an abbreviation for pairs, and $S \circ R = R ⨾ S$.

$X \nrightarrow Y == \{ f : X \leftrightarrow Y \mid (\forall x : X;\ y_1, y_2 : Y \bullet (x \mapsto y_1) \in f \wedge (x \mapsto y_2) \in f \Rightarrow y_1 = y_2) \}$

$X \rightarrow Y == \{ f : X \nrightarrow Y \mid \mathrm{dom}\, f = X \}$

$X \twoheadrightarrow\!\!\!\!\!\nrightarrow Y == \{ f : X \nrightarrow Y \mid \mathrm{ran}\, f = Y \}$

$X \twoheadrightarrow Y == (X \twoheadrightarrow\!\!\!\!\!\nrightarrow Y) \cap (X \rightarrow Y)$

$X \rightarrowtail\!\!\!\!\twoheadrightarrow Y == (X \twoheadrightarrow Y) \cap (X \rightarrowtail Y)$

Laws:

$f \in X \rightarrowtail\!\!\!\!\twoheadrightarrow Y \Leftrightarrow f \in X \rightarrow Y \wedge f^{\sim} \in Y \rightarrow X$

$f \in X \twoheadrightarrow\!\!\!\!\!\nrightarrow Y \Rightarrow f \circ f^{\sim} = \mathrm{id}\ Y$

Z Function Sets in CalcCheck

For two sets $X : \textbf{set}\ t_1$ and $Y : \textbf{set}\ t_2$, we define the following **function sets**:

CalcCheck				Z
$f \in X \rightarrow Y$	`\tfun`	total function	$Dom\ f = X \wedge f^{\smile} ⨾ f \subseteq \mathrm{id}\ Y$	$f \in X \rightarrow Y$
$f \in X \nrightarrow Y$	`\pfun`	partial function	$Dom\ f \subseteq X \wedge f^{\smile} ⨾ f \subseteq \mathrm{id}\ Y$	$f \in X \nrightarrow Y$
$f \in X \rightarrowtail Y$	`\tinj`	total injection	$f ⨾ f^{\smile} = \mathrm{id}\ X \wedge f^{\smile} ⨾ f \subseteq \mathrm{id}\ Y$	$f \in X \rightarrowtail Y$
$f \in X \rightarrowtail\!\!\!\!\!\nrightarrow Y$	`\pinj`	partial injection	$f ⨾ f^{\smile} \subseteq \mathrm{id}\ X \wedge f^{\smile} ⨾ f \subseteq \mathrm{id}\ Y$	$f \in X \rightarrowtail\!\!\!\!\!\nrightarrow Y$
$f \in X \twoheadrightarrow Y$	`\tsurj`	total surjection	$Dom\ f = X \wedge f^{\smile} ⨾ f = \mathrm{id}\ Y$	$f \in X \twoheadrightarrow Y$
$f \in X \twoheadrightarrow\!\!\!\!\!\nrightarrow Y$	`\psurj`	partial surjection	$Dom\ f \subseteq X \wedge f^{\smile} ⨾ f = \mathrm{id}\ Y$	$f \in X \twoheadrightarrow\!\!\!\!\!\nrightarrow Y$
$f \in X \rightarrowtail\!\!\!\!\twoheadrightarrow Y$	`\tbij`	total bijection	$f ⨾ f^{\smile} = \mathrm{id}\ X \wedge f^{\smile} ⨾ f = \mathrm{id}\ Y$	$f \in X \rightarrowtail\!\!\!\!\twoheadrightarrow Y$
$f \in X \rightarrowtail\!\!\!\!\twoheadrightarrow\!\!\!\!\!\nrightarrow Y$	`\pbij`	partial bijection	$f ⨾ f^{\smile} \subseteq \mathrm{id}\ X \wedge f^{\smile} ⨾ f = \mathrm{id}\ Y$	

Counting ...

Let X and Y be finite sets with $\#\, X = x$ and $\#\, Y = y$:

- $\#\,(X \times Y) = ?$ — pairs
- $\#\,(X \leftrightarrow Y) = \#\,(\mathbb{P}\,(X \times Y)) = ?$ — relations
- $\#\,(X \rightarrow Y) = ?$ — total functions
- $\#\,(X \nrightarrow Y) = ?$ — partial functions
- $\#\,(X \rightarrowtail\!\!\!\!\twoheadrightarrow X) = ?$ — homogeneous total bijections
- $\#\,(X \rightarrowtail\!\!\!\!\twoheadrightarrow Y) = ?$ — total bijections
- $\#\,(X \rightarrowtail Y) = ?$ — total injections
- $\#\,(X \rightarrowtail\!\!\!\!\twoheadrightarrow\!\!\!\!\!\nrightarrow Y) = ?$ — partial bijections
- $\#\,(X \rightarrowtail\!\!\!\!\!\nrightarrow Y) = ?$ — partial injections
- $\#\,(X \twoheadrightarrow Y) = ?$ — total surjections
- $\#\,\{ S \mid S \subseteq Y \wedge \#\, S = x \} = ?$ — x-combinations of Y

More Z Symbols: Domain- and Range-Restriction and -Antirestriction

Given types t_1, t_2 : Type, sets A : **set** t_1 and B : **set** t_2, and relation $R : t_1 \leftrightarrow t_2$:

- **Domain restriction**: $A \lhd R \quad = \quad R \cap (A \times U)$
- **Domain antirestriction**: $A ⩤ R \quad = \quad R - (A \times U) \quad = \quad R \cap ({\sim} A \times U)$
- **Range restriction**: $R \rhd B \quad = \quad R \cap (U \times B)$
- **Range antirestriction**: $R ⩥ B \quad = \quad R - (U \times B) \quad = \quad R \cap (U \times {\sim} B)$

$$B \mathbin{;} (\{Jun\} \times U) \quad \cap \quad (C \mathbin{;} C^{\smile}) \quad \subseteq \quad \mathbb{I}$$

$\equiv$ ⟨ Domain- and range restriction properties ⟩

$$Dom(B \rhd \{Jun\}) \lhd (C \mathbin{;} C^{\smile}) \quad \subseteq \quad \mathbb{I}$$

Still no quantifiers, and no x, y of element type
— but not only relations, also sets!

(The abstract version of this is called **Peirce algebra**,
after Chales Sanders Peirce.)

Also in Z: Relational Image and Relation Overriding

Given types t_1, t_2 : Type, sets A : **set** t_1 and B : **set** t_2, and relations $R, S : t_1 \leftrightarrow t_2$:

- **Relational image**: $R (\!| A |\!) \quad = \quad Ran(A \lhd R)$

  "Relational image of set A under relation R

  Notation as "generalised function application"...

$$B \mathbin{;} (\{Jun\} \times U) \quad \cap \quad (C \mathbin{;} C^{\smile}) \quad \subseteq \quad \mathbb{I}$$

$\equiv$ ⟨ Domain- and range restriction properties ⟩

$$Dom(B \rhd \{Jun\}) \lhd (C \mathbin{;} C^{\smile}) \quad \subseteq \quad \mathbb{I}$$

$\equiv$ ⟨ Relational image ⟩

$$(B^{\smile} (\!| \{Jun\} |\!)) \lhd (C \mathbin{;} C^{\smile}) \quad \subseteq \quad \mathbb{I}$$

- **Relation overriding**: $R \oplus S \quad = \quad (Dom\ S ⩤ R) \cup S$

  "Updating R exactly where S relates with anything"

  In the relation $\quad C \oplus \{\langle Aos, Jun \rangle\}$, Aos called only Jun.

Predicate Logic Laws You Really Need To Know Now

(8.13) **Empty Range:** ...

(8.14) **One-point Rule:** Provided ..., ...

(8.15) **(Quantification) Distributivity:** ...

(8.16–18) **Range split:** ...

(9.17) **Generalised De Morgan**: ...

(9.2) **Trading for ∀:** ...

(9.19) **Trading for ∃:** ...

(9.13) **Instantiation:** ...

(9.28) **∃-Introduction**: ...

...and correctly handle substitution, Leibniz, bound variable rearrangements, monotonicity/antitonicity, For any ...

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-27

Quantifier Reasoning, Explicit Induction Principles

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-27

Part 1: Quantifier Reasoning Examples: Ex6.3

Ex6.3 — Domain of Union — Step 1

Theorem "Domain of union": Dom $(R \cup S)$ = Dom R ∪ Dom S
Proof:
  Using "Set extensionality":
    For any \`$x$\`:
      $x \in$ Dom $(R \cup S)$

    $\equiv$ ⟨ ? ⟩

      $x \in$ Dom R ∪ Dom S

Ex6.3 — Domain of Union — Step 2

Theorem "Domain of union": Dom $(R \cup S)$ = Dom R ∪ Dom S
Proof:
  Using "Set extensionality":
    For any \`$x$\`:
      $x \in$ Dom $(R \cup S)$
    $\equiv$ ⟨ "Membership in \`Dom\`" ⟩
      $\exists y \bullet x\ (\!|\ R \cup S\ |\!)\ y$
    $\equiv$ ⟨ "Relation union" ⟩
      $\exists y \bullet x\ (\!|\ R\ |\!)\ y \lor x\ (\!|\ S\ |\!)\ y$

    $\equiv$ ⟨ ? ⟩

      $(\exists y \bullet x\ (\!|\ R\ |\!)\ y) \lor (\exists y \bullet x\ (\!|\ S\ |\!)\ y)$
    $\equiv$ ⟨ "Membership in \`Dom\`" ⟩
      $x \in$ Dom R ∨ $x \in$ Dom S
    $\equiv$ ⟨ "Union" ⟩
      $x \in$ Dom R ∪ Dom S

Ex6.3 — Domain of Union — Step 3

Theorem "Domain of union": Dom $(R \cup S)$ = Dom R ∪ Dom S
Proof:
  Using "Set extensionality":
    For any \`$x$\`:
      $x \in$ Dom $(R \cup S)$
    $\equiv$ ⟨ "Membership in \`Dom\`" ⟩
      $\exists y \bullet x\ (\!|\ R \cup S\ |\!)\ y$
    $\equiv$ ⟨ "Relation union" ⟩
      $\exists y \bullet x\ (\!|\ R\ |\!)\ y \lor x\ (\!|\ S\ |\!)\ y$
    $\equiv$ ⟨ "Distributivity of ∃ over ∨" ⟩
      $(\exists y \bullet x\ (\!|\ R\ |\!)\ y) \lor (\exists y \bullet x\ (\!|\ S\ |\!)\ y)$
    $\equiv$ ⟨ "Membership in \`Dom\`" ⟩
      $x \in$ Dom R ∨ $x \in$ Dom S
    $\equiv$ ⟨ "Union" ⟩
      $x \in$ Dom R ∪ Dom S

Ex6.3 — Domain of ∩ — Step 1

Theorem "Domain of intersection": Dom $(R \cap S)$ ⊆ Dom R ∩ Dom S
Proof:
  Using "Set inclusion":
    For any \`$x$\`:
      $x \in$ Dom $(R \cap S)$
    $\equiv$ ⟨ "Membership in \`Dom\`" ⟩
      $\exists y \bullet x\ (\!|\ R \cap S\ |\!)\ y$
    $\equiv$ ⟨ "Relation intersection" ⟩
      $\exists y \bullet x\ (\!|\ R\ |\!)\ y \land x\ (\!|\ S\ |\!)\ y$

    $\Rightarrow$ ⟨ ? ⟩

      $(\exists y \bullet x\ (\!|\ R\ |\!)\ y) \land (\exists y \bullet x\ (\!|\ S\ |\!)\ y)$
    $\equiv$ ⟨ "Membership in \`Dom\`" ⟩
      $x \in$ Dom R ∧ $x \in$ Dom S
    $\equiv$ ⟨ "Intersection" ⟩
      $x \in$ Dom R ∩ Dom S

Ex6.3 — Domain of ∩ — Step 2

Theorem "Domain of intersection": Dom $(R \cap S)$ ⊆ Dom R ∩ Dom S
Proof:
  Using "Set inclusion":
    For any \`$x$\`:
      $x \in$ Dom $(R \cap S)$
    $\equiv$ ⟨ "Membership in \`Dom\`" ⟩
      $\exists y \bullet x\ (\!|\ R \cap S\ |\!)\ y$
    $\equiv$ ⟨ "Relation intersection" ⟩
      $\exists y \bullet x\ (\!|\ R\ |\!)\ y \land x\ (\!|\ S\ |\!)\ y$
    $\equiv$ ⟨ "Idempotency of ∧" ⟩
      $(\exists y \bullet x\ (\!|\ R\ |\!)\ y \land x\ (\!|\ S\ |\!)\ y) \land (\exists y \bullet x\ (\!|\ R\ |\!)\ y \land x\ (\!|\ S\ |\!)\ y)$

    $\Rightarrow$ ⟨ ? with "Weakening" ⟩

      $(\exists y \bullet x\ (\!|\ R\ |\!)\ y) \qquad \land \ (\exists y \bullet \qquad x\ (\!|\ S\ |\!)\ y)$
    $\equiv$ ⟨ "Membership in \`Dom\`" ⟩
      $x \in$ Dom R ∧ $x \in$ Dom S
    $\equiv$ ⟨ "Intersection" ⟩
      $x \in$ Dom R ∩ Dom S

Ex6.3 — Domain of ∩ — Step 3

Theorem "Domain of intersection": $\mathsf{Dom}\,(R \cap S) \subseteq \mathsf{Dom}\,R \cap \mathsf{Dom}\,S$
Proof:
  Using "Set inclusion":
    For any `$x$`:
      $x \in \mathsf{Dom}\,(R \cap S)$
    $\equiv\langle$ "Membership in `Dom`" $\rangle$
      $\exists y \bullet x \llparenthesis R \cap S \rrparenthesis y$
    $\equiv\langle$ "Relation intersection" $\rangle$
      $\exists y \bullet x \llparenthesis R \rrparenthesis y \land x \llparenthesis S \rrparenthesis y$
    $\equiv\langle$ "Idempotency of $\land$" $\rangle$
      $(\exists y \bullet x \llparenthesis R \rrparenthesis y \land x \llparenthesis S \rrparenthesis y) \land$
      $(\exists y \bullet x \llparenthesis R \rrparenthesis y \land x \llparenthesis S \rrparenthesis y)$
    $\Rightarrow\langle$ "Monotonicity of $\land$" with
      "Body monotonicity of $\exists$" with "Weakening" $\rangle$
      $(\exists y \bullet x \llparenthesis R \rrparenthesis y) \land (\exists y \bullet x \llparenthesis S \rrparenthesis y)$
    $\equiv\langle$ "Membership in `Dom`" $\rangle$
      $x \in \mathsf{Dom}\,R \land x \in \mathsf{Dom}\,S$
    $\equiv\langle$ "Intersection" $\rangle$
      $x \in \mathsf{Dom}\,R \cap \mathsf{Dom}\,S$

Ex6.3 — Domain of ∩ (B) — Step 1

Theorem "Domain of intersection": $\mathsf{Dom}\,(R \cap S) \subseteq \mathsf{Dom}\,R \cap \mathsf{Dom}\,S$
Proof:
  Using "Set inclusion":
    For any `$x$`:
      $x \in \mathsf{Dom}\,(R \cap S)$
    $\equiv\langle$ "Membership in `Dom`" $\rangle$
      $\exists y \bullet x \llparenthesis R \cap S \rrparenthesis y$
    $\equiv\langle$ "Relation intersection" $\rangle$
      $\exists y \bullet x \llparenthesis R \rrparenthesis y \land x \llparenthesis S \rrparenthesis y$

    $\Rightarrow\langle\ ?\ \rangle$

      $(\exists y \bullet x \llparenthesis R \rrparenthesis y) \land (\exists y \bullet x \llparenthesis S \rrparenthesis y)$
    $\equiv\langle$ "Membership in `Dom`" $\rangle$
      $x \in \mathsf{Dom}\,R \land x \in \mathsf{Dom}\,S$
    $\equiv\langle$ "Intersection" $\rangle$
      $x \in \mathsf{Dom}\,R \cap \mathsf{Dom}\,S$

> **Theorem** (9.21) "Distributivity of $\land$ over $\exists$":
> $P \land (\exists x \mid R \bullet Q) \equiv (\exists x \mid R \bullet P \land Q)$
> provided $\neg occurs(‘x’, ‘P’)$

Ex6.3 — Domain of ∩ (B) — Step 2

Theorem "Domain of intersection": $\mathsf{Dom}\,(R \cap S) \subseteq \mathsf{Dom}\,R \cap \mathsf{Dom}\,S$
Proof:
  Using "Set inclusion":
    For any `$x$`:
      $x \in \mathsf{Dom}\,(R \cap S)$
    $\equiv\langle$ "Membership in `Dom`" $\rangle$
      $\exists y \bullet x \llparenthesis R \cap S \rrparenthesis y$
    $\equiv\langle$ "Relation intersection" $\rangle$
      $\exists y \bullet x \llparenthesis R \rrparenthesis y \land x \llparenthesis S \rrparenthesis y$

    $\Rightarrow\langle\ ?\ \rangle$

      $\exists y \bullet x \llparenthesis R \rrparenthesis y \land (\exists y \bullet x \llparenthesis S \rrparenthesis y)$
    $\equiv\langle$ "Distributivity of $\land$ over $\exists$" $\rangle$
      $(\exists y \bullet x \llparenthesis R \rrparenthesis y) \land (\exists y \bullet x \llparenthesis S \rrparenthesis y)$
    $\equiv\langle$ "Membership in `Dom`" $\rangle$
      $x \in \mathsf{Dom}\,R \land x \in \mathsf{Dom}\,S$
    $\equiv\langle$ "Intersection" $\rangle$
      $x \in \mathsf{Dom}\,R \cap \mathsf{Dom}\,S$

> **Theorem** (9.21) "Distributivity of $\land$ over $\exists$":
> $P \land (\exists x \mid R \bullet Q) \equiv (\exists x \mid R \bullet P \land Q)$
> provided $\neg occurs(‘x’, ‘P’)$

Ex6.3 — Domain of ∩ (B) — Step 3

Theorem "Domain of intersection": $\mathsf{Dom}\,(R \cap S) \subseteq \mathsf{Dom}\,R \cap \mathsf{Dom}\,S$
Proof:
  Using "Set inclusion":
    For any `$x$`:
      $x \in \mathsf{Dom}\,(R \cap S)$
    $\equiv\langle$ "Membership in `Dom`" $\rangle$
      $\exists y \bullet x \llparenthesis R \cap S \rrparenthesis y$
    $\equiv\langle$ "Relation intersection" $\rangle$
      $\exists y \bullet x \llparenthesis R \rrparenthesis y \land x \llparenthesis S \rrparenthesis y$
    $\equiv\langle$ Substitution $\rangle$
      $\exists y \bullet x \llparenthesis R \rrparenthesis y \land (x \llparenthesis S \rrparenthesis y)[y := y]$
    $\Rightarrow\langle\ ?$ with "$\exists$-Introduction" $\rangle$
      $\exists y \bullet x \llparenthesis R \rrparenthesis y \land (\exists y \bullet x \llparenthesis S \rrparenthesis y)$
    $\equiv\langle$ "Distributivity of $\land$ over $\exists$" $\rangle$
      $(\exists y \bullet x \llparenthesis R \rrparenthesis y) \land (\exists y \bullet x \llparenthesis S \rrparenthesis y)$
    $\equiv\langle$ "Membership in `Dom`" $\rangle$
      $x \in \mathsf{Dom}\,R \land x \in \mathsf{Dom}\,S$
    $\equiv\langle$ "Intersection" $\rangle$
      $x \in \mathsf{Dom}\,R \cap \mathsf{Dom}\,S$

Ex6.3 — Domain of ∩ (B) — Step 4

Theorem "Domain of intersection": $\mathsf{Dom}\,(R \cap S) \subseteq \mathsf{Dom}\,R \cap \mathsf{Dom}\,S$
Proof:
  Using "Set inclusion":
    For any `$x$`:
      $x \in \mathsf{Dom}\,(R \cap S)$
    $\equiv\langle$ "Membership in `Dom`" $\rangle$
      $\exists y \bullet x \llparenthesis R \cap S \rrparenthesis y$
    $\equiv\langle$ "Relation intersection" $\rangle$
      $\exists y \bullet x \llparenthesis R \rrparenthesis y \land x \llparenthesis S \rrparenthesis y$
    $\equiv\langle$ Substitution $\rangle$
      $\exists y \bullet x \llparenthesis R \rrparenthesis y \land (x \llparenthesis S \rrparenthesis y)[y := y]$
    $\Rightarrow\langle$ "Body monotonicity of $\exists$" with "Monotonicity of $\land$" with "$\exists$-Introduction" $\rangle$
      $\exists y \bullet x \llparenthesis R \rrparenthesis y \land (\exists y \bullet x \llparenthesis S \rrparenthesis y)$
    $\equiv\langle$ "Distributivity of $\land$ over $\exists$" $\rangle$
      $(\exists y \bullet x \llparenthesis R \rrparenthesis y) \land (\exists y \bullet x \llparenthesis S \rrparenthesis y)$
    $\equiv\langle$ "Membership in `Dom`" $\rangle$
      $x \in \mathsf{Dom}\,R \land x \in \mathsf{Dom}\,S$
    $\equiv\langle$ "Intersection" $\rangle$
      $x \in \mathsf{Dom}\,R \cap \mathsf{Dom}\,S$

Distributivity over ∀

(9.5) **Axiom, Distributivity of $\lor$ over $\forall$:** If $\neg occurs(‘x’, ‘P’)$,
$$P \lor (\forall x \mid R \bullet Q) \equiv (\forall x \mid R \bullet P \lor Q)$$

(9.6) Provided $\neg occurs(‘x’, ‘P’)$,
$$(\forall x \mid R \bullet P) \equiv P \lor (\forall x \bullet \neg R)$$

(9.7) **Distributivity of $\land$ over $\forall$:** If $\neg occurs(‘x’, ‘P’)$,
$$\neg(\forall x \bullet \neg R) \Rightarrow (P \land (\forall x \mid R \bullet Q) \equiv (\forall x \mid R \bullet P \land Q))$$

(9.22.1) **Distributivity of $\land$ over $\forall$:** If $\neg occurs(‘x’, ‘P’)$,
$$(\exists x \bullet R) \Rightarrow (P \land (\forall x \mid R \bullet Q) \equiv (\forall x \mid R \bullet P \land Q))$$

(9.8) $(\forall x \mid R \bullet true) \equiv true$

(9.9) $(\forall x \mid R \bullet P \equiv Q) \Rightarrow ((\forall x \mid R \bullet P) \equiv (\forall x \mid R \bullet Q))$

Distributivity over ∃

(9.21) **Distributivity of $\land$ over $\exists$:** If $\neg occurs(‘x’, ‘P’)$,
$$P \land (\exists x \mid R \bullet Q) \equiv (\exists x \mid R \bullet P \land Q)$$

(9.22) Provided $\neg occurs(‘x’, ‘P’)$,
$$(\exists x \mid R \bullet P) \equiv P \land (\exists x \bullet R)$$

(9.23) **Distributivity of $\lor$ over $\exists$:** If $\neg occurs(‘x’, ‘P’)$,
$$(\exists x \bullet R) \Rightarrow ((\exists x \mid R \bullet P \lor Q) \equiv P \lor (\exists x \mid R \bullet Q))$$

(9.24) $(\exists x \mid R \bullet false) \equiv false$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-27

Part 2: Explicit Induction Principles

Natural Numbers Generated from 0 and *suc* — Explicit Induction Principle

Recall: Induction principle for the natural numbers:

- if $P(0)$ — If P holds for 0
- and if $P(m)$ implies $P(suc\ m)$, — and whenever P holds for m, it also holds for $suc\ m$,
- then for all $m : \mathbb{N}$ we have $P(m)$. — then P holds for all natural numbers.

As **inference rule**:

With variable $P : \mathbb{N} \to \mathbb{B}$:
$$\frac{P(0) \qquad \begin{array}{c}\ulcorner P(m) \urcorner \\ \vdots \\ P(suc\ m)\end{array}}{P(m)}$$

With $P : \mathbb{B}$ *as* ***metavariable*** *for an expression:*
$$\frac{P[m := 0] \qquad \begin{array}{c}\ulcorner P \urcorner \\ \vdots \\ P[m := suc\ m]\end{array}}{P}$$

As **axiom / theorem** — LADM p. 219: "weak induction":

Axiom "Induction over $\mathbb{N}$":
  $P[n := 0]$
  $\Rightarrow (\forall n : \mathbb{N} \mid P \bullet P[n := \mathsf{suc}\ n])$
  $\Rightarrow (\forall n : \mathbb{N} \bullet P)$

Proving "Right-identity of +" `Using` the Induction Principle (v0)

```
Axiom "Induction over ℕ":
   P[n = 0]
   ⇒ (∀ n : ℕ ❙ P • P[n = suc n])
   ⇒ (∀ n : ℕ • P)

Theorem "Right-identity of +": ∀ m : ℕ • m + 0 = m
Proof:
  Using "Induction over ℕ":
    Subproof for `(m + 0 = m)[m = 0]`:
      By substitution and "Definition of +"
    Subproof for `∀ m : ℕ ❙ m + 0 = m • (m + 0 = m)[m = suc m]`:
      For any `m : ℕ` satisfying `m + 0 = m`:
          (m + 0 = m)[m = suc m]
        =⟨ Substitution, "Definition of +" ⟩
          suc (m + 0) = suc m
        =⟨ Assumption `m + 0 = m`, "Reflexivity of =" ⟩
          true
```

(I never use this pattern with substitutions in the subproof goals.)

Proving “Right-identity of +” Using the Induction Principle (v1)

```
Axiom “Induction over ℕ”:
   P[n ≔ 0]
   ⇒ (∀ n : ℕ ❙ P • P[n ≔ suc n])
   ⇒ (∀ n : ℕ • P)

Theorem “Right-identity of +”: ∀ m : ℕ • m + 0 = m
Proof:
  Using “Induction over ℕ”:
    Subproof for `0 + 0 = 0`:
      By “Definition of +”
    Subproof for `∀ m : ℕ ❙ m + 0 = m • suc m + 0 = suc m`:
      For any `m : ℕ` satisfying `m + 0 = m`:
          suc m + 0
        =⟨ “Definition of +” ⟩
          suc (m + 0)
        =⟨ Assumption `m + 0 = m` ⟩
          suc m
```

Proving “Right-identity of +” Using the Induction Principle (v2)

```
Theorem “Right-identity of +”: ∀ m : ℕ • m + 0 = m
Proof:
  Using “Induction over ℕ”:
    Subproof:
        0 + 0
      =⟨ “Definition of +” ⟩
        0
    Subproof:
      For any `m : ℕ` satisfying “IndHyp” `m + 0 = m`:
          suc m + 0
        =⟨ “Definition of +” ⟩
          suc (m + 0)
        =⟨ Assumption “IndHyp” ⟩
          suc m
```

```
Axiom “Induction over ℕ”:
  P[n ≔ 0]
  ⇒ (∀ n : ℕ ❙ P • P[n ≔ suc n])
  ⇒ (∀ n : ℕ • P)
```

- (Subproof goals can be omitted where they are clear from the contained proof.)
- You need to understand (v0) and (v1) to be able to do (v2)!

“By induction on ...” versus Using Induction Principles

- Using induction principles directly is not much more verbose than “By induction on ...”
- “By induction on ...” only supports **very few** built-in induction principles
- Induction principles can be derived as theorems, or provided as axioms, and then can be used directly!

Sequences — Induction Principle

Induction principle for sequences:

- if $P(\epsilon)$ — If P holds for ϵ
- and if $P(xs)$ implies $P(x \triangleleft xs)$ for all $x : A$, — and whenever P holds for xs, it also holds for any $x \triangleleft xs$,
- then for all $xs : Seq\ A$ we have $P(xs)$. — then P holds for all sequences over A.

$$P[xs := \epsilon] \quad \Rightarrow \quad (\forall\ xs : Seq\ A \mid P \bullet (\forall\ x : A \bullet P[xs := x \triangleleft xs]) \quad \Rightarrow \quad (\forall\ xs : Seq\ A \bullet P)$$

```
Axiom “Induction over sequences”:
    P[xs ≔ ϵ]
    ⇒ (∀ xs : Seq A ❙ P • (∀ x : A • P[xs ≔ x ◃ xs]))
    ⇒ (∀ xs : Seq A • P)
```

$$P[m := 0] \quad \Rightarrow \quad (\forall\ m : \mathbb{N} \mid P \bullet P[m := suc\ m]) \quad \Rightarrow \quad (\forall\ m : \mathbb{N} \bullet P)$$

```
Axiom “Induction over ℕ”:
  P[n ≔ 0]
  ⇒ (∀ n : ℕ ❙ P • P[n ≔ suc n])
  ⇒ (∀ n : ℕ • P)
```

Recall: Tail is different — LADM Proof

Theorem (13.7) “Tail is different”: $\forall\ xs : Seq\ A \bullet \forall\ x : A \bullet x \triangleleft xs \neq xs$

Proof:

By induction on `$xs : Seq\ A$`:

Base case:

For any `$x : A$`:

$x \triangleleft \epsilon \neq \epsilon$

$\equiv \langle$ “Cons is not empty” $\rangle$

true

Induction step:

For any `$z : A$`, `$x : A$`:

$x \triangleleft z \triangleleft xs \neq z \triangleleft xs$

$\equiv \langle$ “Definition of $\neq$”, “Cancellation of $\triangleleft$” $\rangle$

$\neg (x = z \wedge z \triangleleft xs = xs)$

$\Leftarrow \langle$ “Consequence”, “De Morgan”, “Weakening”, “Definition of $\neq$” $\rangle$

$z \triangleleft xs \neq xs$

$\equiv \langle$ Induction hypothesis `$\forall\ x : A \bullet x \triangleleft xs \neq xs$` $\rangle$

true

(For explanations about using “By induction on `$xs : Seq\ A$`:” for proving “$\forall\ xs : Seq\ A \bullet P$”, see H13 and Ex5.2.)

Proving “Tail is different” Using the Induction Principle

Theorem “Induction over sequences”:
$P[xs := \epsilon] \Rightarrow (\forall\ xs : Seq\ A \mid P \bullet (\forall\ x : A \bullet P[xs := x \triangleleft xs])) \Rightarrow (\forall\ xs : Seq\ A \bullet P)$

Theorem (13.7) “Tail is different”: $\forall\ xs : Seq\ A \bullet \forall\ x : A \bullet x \triangleleft xs \neq xs$

Proof:

Using “Induction over sequences”:

Subproof for `$\forall\ x : A \bullet x \triangleleft \epsilon \neq \epsilon$`:

For any `$x : A$`:

By “Cons is not empty”

Subproof for `$\forall\ xs : Seq\ A \mid (\forall\ x : A \bullet x \triangleleft xs \neq xs) \bullet (\forall\ z : A \bullet (\forall\ x : A \bullet x \triangleleft z \triangleleft xs \neq z \triangleleft xs))$`:

For any `$xs : Seq\ A$` **satisfying** “Ind. Hyp.” `$(\forall\ x : A \bullet x \triangleleft xs \neq xs)$`:

For any `$z : A$`, `$x : A$`:

$x \triangleleft z \triangleleft xs \neq z \triangleleft xs$

$\equiv \langle$ “Definition of $\neq$”, “Injectivity of $\triangleleft$” $\rangle$

$\neg (x = z \wedge z \triangleleft xs = xs)$

$\Leftarrow \langle$ “De Morgan”, “Weakening”, “Definition of $\neq$” $\rangle$

$z \triangleleft xs \neq xs$

$\equiv \langle$ Assumption “Ind. Hyp.” $\rangle$

true

Proving “Tail is different” Using the Induction Principle — Less Verbose

Theorem “Induction over sequences”:

$P[xs := \epsilon]$

$\Rightarrow (\forall\ xs : Seq\ A \mid P \bullet (\forall\ x : A \bullet P[xs := x \triangleleft xs]))$

$\Rightarrow (\forall\ xs : Seq\ A \bullet P)$

Theorem (13.7) “Tail is different”: $\forall\ xs : Seq\ A \bullet \forall\ x : A \bullet x \triangleleft xs \neq xs$

Proof:

Using “Induction over sequences”:

Subproof for `$\forall\ x : A \bullet x \triangleleft \epsilon \neq \epsilon$`:

For any `$x : A$`:

By “Cons is not empty”

Subproof:

For any `$xs : Seq\ A$` **satisfying** “Ind. Hyp.” `$(\forall\ x : A \bullet x \triangleleft xs \neq xs)$`:

For any `$z : A$`, `$x : A$`:

$x \triangleleft z \triangleleft xs \neq z \triangleleft xs$

$\equiv \langle$ “Definition of $\neq$”, “Injectivity of $\triangleleft$” $\rangle$

$\neg (x = z \wedge z \triangleleft xs = xs)$

$\Leftarrow \langle$ “De Morgan”, “Weakening”, “Definition of $\neq$” $\rangle$

$z \triangleleft xs \neq xs$

$\equiv \langle$ Assumption “Ind. Hyp.” $\rangle$

true

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-27

Part 3: Residuals

Given: $x \leq z \quad \equiv \quad x \leq 5$

What do you know about z? Why? (Prove it!)

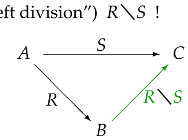
Given: $X \subseteq A \Rightarrow B \quad \equiv \quad X \cap A \subseteq B$

Calculate the **relative pseudocomplement** $A \Rightarrow B$!

Given, for $R : A \leftrightarrow B$ and $S : A \leftrightarrow C$: $\quad X \subseteq R \setminus S \quad \equiv \quad R \mathbin{;} X \subseteq S$

$R \setminus S$ is the largest solution $X : B \leftrightarrow C$ for $R \mathbin{;} X \subseteq S$.

Calculate the **right residual** (“left division”) $R \setminus S$!

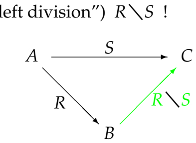


Same idea as for “⇒”:

Using extensionality, calculate $\quad b \,(R \setminus S)\, c \quad \equiv \quad b \,(\,?\,)\, c$

Given, for $R : A \leftrightarrow B$ and $S : A \leftrightarrow C$: $\quad X \subseteq R \setminus S \quad \equiv \quad R \mathbin{;} X \subseteq S$

Calculate the **right residual** (“left division”) $R \setminus S$!



$b \,(R \setminus S)\, c$

$= \langle$ Similar to the calculation for relative pseudocomplement $\rangle$

$(\forall\ a \mid a \,(R)\, b \bullet a \,(S)\, c)$

$= \langle$ Generalised De Morgan, Relation conversions — Ex. 6.3 (R1) $\rangle$

$b \,(\sim (R^{\smile} \mathbin{;} \sim S))\, c$

Therefore: $\quad R \setminus S = \sim (R^{\smile} \mathbin{;} \sim S)$

— monotonic in second argument; antitonic in first argument

Proving $b\,(R \backslash S)\,c \equiv (\forall a \mid a\,(R)\,b \bullet a\,(S)\,c)$:

$$
\begin{array}{ll}
 & b(R \backslash S)c \\
= & \langle\ e \in S \equiv \{e\} \subseteq S \text{ — Exercise! } \rangle \\
 & \{\langle b,c\rangle\} \subseteq (R \backslash S) \\
= & \langle\ \text{Def. } \backslash\text{: } X \subseteq R \backslash S \ \equiv\ R ; X \subseteq S\ \rangle \\
 & R ; \{\langle b,c\rangle\} \subseteq S \\
= & \langle\ \text{(11.13r) Relation inclusion}\ \rangle \\
 & (\forall a, c' \mid a\,(R ; \{\langle b,c\rangle\})\,c' \bullet a\,(S)\,c') \\
= & \langle\ \text{(14.20) Relation composition}\ \rangle \\
 & (\forall a, c' \mid (\exists b' \bullet a\,(R)\,b' \wedge b'\,(\{\langle b,c\rangle\})\,c') \bullet a\,(S)\,c') \\
= & \langle\ y \in \{x\} \equiv y = x \text{ — Exercise! } \rangle \\
 & (\forall a, c' \mid (\exists b' \bullet a\,(R)\,b' \wedge b' = b \wedge c = c') \bullet a\,(S)\,c') \\
= & \langle\ \text{(9.19) Trading for } \exists\ \rangle \\
 & (\forall a, c' \mid (\exists b' \mid b' = b \bullet a\,(R)\,b' \wedge c = c') \bullet a\,(S)\,c') \\
= & \langle\ \text{(8.14) One-point rule}\ \rangle \\
 & (\forall a, c' \mid a\,(R)\,b \wedge c = c' \bullet a\,(S)\,c') \\
= & \langle\ \text{(8.20) Quantifier nesting}\ \rangle \\
 & (\forall a \mid a\,(R)\,b \bullet (\forall c' \mid c = c' \bullet a\,(S)\,c')) \\
= & \langle\ \text{(1.3) Symmetry of } =\text{, (8.14) One-point rule}\ \rangle \\
 & (\forall a \mid a\,(R)\,b \bullet a\,(S)\,c)
\end{array}
$$

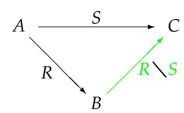
Right Residual: $X \subseteq R \backslash S \quad \equiv \quad R ; X \subseteq S$

Proving $R \backslash S = \sim(R^{\smile} ; \sim S)$:

$$
\begin{array}{ll}
 & b\,(R \backslash S)\,c \\
= & \langle\ \text{previous slide}\ \rangle \\
 & (\forall a \mid a\,(R)\,b \bullet a\,(S)\,c) \\
= & \langle\ \text{(9.18a) Generalised De Morgan}\ \rangle \\
 & \neg(\exists a \mid a\,(R)\,b \bullet \neg(a\,(S)\,c)) \\
= & \langle\ \text{(11.17r) Relation complement}\ \rangle \\
 & \neg(\exists a \mid a\,(R)\,b \bullet a\,(\sim S)\,c) \\
= & \langle\ \text{(9.19) Trading for } \exists\text{, (14.18) Converse}\ \rangle \\
 & \neg(\exists a \bullet b\,(R^{\smile})\,a \wedge a\,(\sim S)\,c) \\
= & \langle\ \text{(14.20) Relation composition}\ \rangle \\
 & \neg(b\,(R^{\smile} ; \sim S)\,c) \\
= & \langle\ \text{(11.17r) Relation complement}\ \rangle \\
 & b\,(\sim(R^{\smile} ; \sim S))\,c
\end{array}
$$

Given, for $R : A \leftrightarrow B$ and $S : A \leftrightarrow C$: $\qquad X \subseteq R \backslash S \quad \equiv \quad R ; X \subseteq S$

Calculate the **right residual** ("left division") $R \backslash S$! $\qquad$ ("R under S")



$$
\begin{array}{ll}
 & b\,(R \backslash S)\,c \\
= & \langle\ \text{Similar to the calculation for relative pseudocomplement}\ \rangle \\
 & (\forall a \mid a\,(R)\,b \bullet a\,(S)\,c) \\
= & \langle\ \text{Generalised De Morgan, Relation conversions — Ex. 6.3 (R1)}\ \rangle \\
 & b\,(\sim(R^{\smile} ; \sim S))\,c
\end{array}
$$

Therefore: $\quad R \backslash S = \sim(R^{\smile} ; \sim S)$

— monotonic in second argument; antitonic in first argument

Formalisations Using Residuals

"Aos called only brothers of Jun."

"Everybody called by Aos is a brother of Jun."

$$
\begin{array}{ll}
 & (\forall p \mid Aos\,(C)\,p \bullet p\,(B)\,Jun) \\
\equiv & \langle\ \text{(14.18) Relation converse}\ \rangle \\
 & (\forall p \mid p\,(C^{\smile})\,Aos \bullet p\,(B)\,Jun) \\
\equiv & \langle\ \text{Right residual}\ \rangle \\
 & Aos\,(C^{\smile} \backslash B)\,Jun
\end{array}
$$

Relationship via $\backslash$**:**

$$
\begin{array}{ll}
 & b\,(R \backslash S)\,c \\
\equiv & (\forall a \mid a\,(R)\,b \bullet a\,(S)\,c)
\end{array}
$$

"Aos called every brother of Jun."

"Every brother of Jun has been called by Aos."

$$
\begin{array}{ll}
 & (\forall p \mid p\,(B)\,Jun \bullet Aos\,(C)\,p) \\
\equiv & \langle\ \text{(14.18) Relation converse}\ \rangle \\
 & (\forall p \mid p\,(B)\,Jun \bullet p\,(C^{\smile})\,Aos) \\
\equiv & \langle\ \text{Right residual}\ \rangle \\
 & Jun\,(B \backslash C^{\smile})\,Aos
\end{array}
$$

Some Properties of Right Residuals

Characterisation of right residual: $\forall R : A \leftrightarrow B;\ S : A \leftrightarrow C \bullet \quad X \subseteq R \backslash S \quad \equiv \quad R ; X \subseteq S$

Two sub-cancellation properties follow easily:

$$
\begin{array}{rcl}
R ; (R \backslash S) & \subseteq & S \\
(Q \backslash R) ; (R \backslash S) & \subseteq & (Q \backslash S)
\end{array}
$$

Theorem "$\mathbb{I} \backslash$": $\mathbb{I} \backslash R = R$

Proof:

Using "Mutual inclusion":

Subproof:

$$
\begin{array}{ll}
 & \mathbb{I} \backslash R \\
= & \langle\ \text{"Identity of ;"}\ \rangle \\
 & \mathbb{I} ; (\mathbb{I} \backslash R) \\
\subseteq & \langle\ \text{"Cancellation of } \backslash\text{"}\ \rangle \\
 & R
\end{array}
$$

Subproof:

$$
\begin{array}{ll}
 & R \subseteq \mathbb{I} \backslash R \\
\equiv & \langle\ \text{"Characterisation of } \backslash\text{"}\ \rangle \\
 & \mathbb{I} ; R \subseteq R \\
\equiv & \langle\ \text{"Identity of ;", "Reflexivity of } \subseteq\text{"}\ \rangle \\
 & \text{true}
\end{array}
$$

Translating between Relation Algebra and Predicate Logic

$$
\begin{array}{rcl}
R = S & \equiv & (\forall x, y \bullet x\,(R)\,y \equiv x\,(S)\,y) \\
R \subseteq S & \equiv & (\forall x, y \bullet x\,(R)\,y \Rightarrow x\,(S)\,y) \\
u\,(\{\})\,v & \equiv & \mathit{false} \\
u\,(A \times B)\,v & \equiv & u \in A \wedge v \in B \\
u\,(\sim S)\,v & \equiv & \neg(u\,(S)\,v) \\
u\,(S \cup T)\,v & \equiv & u\,(S)\,v \vee u\,(T)\,v \\
u\,(S \cap T)\,v & \equiv & u\,(S)\,v \wedge u\,(T)\,v \\
u\,(S - T)\,v & \equiv & u\,(S)\,v \wedge \neg(u\,(T)\,v) \\
u\,(S \Rightarrow T)\,v & \equiv & u\,(S)\,v \Rightarrow u\,(T)\,v \\
u\,(\mathrm{id}\ A)\,v & \equiv & u = v \in A \\
u\,(\mathbb{I})\,v & \equiv & u = v \\
u\,(R^{\smile})\,v & \equiv & v\,(R)\,u \\
u\,(R ; S)\,v & \equiv & (\exists x \bullet u\,(R)\,x\,(S)\,v) \\
u\,(R \backslash S)\,v & \equiv & (\forall x \mid x\,(R)\,u \bullet x\,(S)\,v) \\
u\,(S / R)\,v & \equiv & (\forall x \mid v\,(R)\,x \bullet u\,(S)\,x)
\end{array}
$$

Translating between Relation Algebra and Predicate Logic

$$
\begin{array}{rcl}
R = S & \equiv & (\forall x, y \bullet x\,(R)\,y \equiv x\,(S)\,y) \\
R \subseteq S & \equiv & (\forall x, y \bullet x\,(R)\,y \Rightarrow x\,(S)\,y) \\
u\,(\{\})\,v & \equiv & \mathit{false} \\
u\,(A \times B)\,v & \equiv & u \in A \wedge v \in B \\
u\,(\sim S)\,v & \equiv & \neg(u\,(S)\,v) \\
u\,(S \cup T)\,v & \equiv & u\,(S)\,v \vee u\,(T)\,v \\
u\,(S \cap T)\,v & \equiv & u\,(S)\,v \wedge u\,(T)\,v \\
u\,(S - T)\,v & \equiv & u\,(S)\,v \wedge \neg(u\,(T)\,v) \\
u\,(S \Rightarrow T)\,v & \equiv & u\,(S)\,v \Rightarrow u\,(T)\,v \\
u\,(\mathrm{id}\ A)\,v & \equiv & u = v \in A \\
u\,(\mathbb{I})\,v & \equiv & u = v \\
u\,(R^{\smile})\,v & \equiv & v\,(R)\,u \\
u\,(R ; S)\,v & \equiv & (\exists x \mid u\,(R)\,x \bullet x\,(S)\,v) \\
u\,(R \backslash S)\,v & \equiv & (\forall x \mid x\,(R)\,u \bullet x\,(S)\,v) \\
u\,(S / R)\,v & \equiv & (\forall x \mid v\,(R)\,x \bullet u\,(S)\,x)
\end{array}
$$

Translating between Relation Algebra and Predicate Logic

$$
\begin{array}{rcl}
R = S & \equiv & (\forall x, y \bullet x\,(R)\,y \equiv x\,(S)\,y) \\
R \subseteq S & \equiv & (\forall x, y \bullet x\,(R)\,y \Rightarrow x\,(S)\,y) \\
u\,(\{\})\,v & \equiv & \mathit{false} \\
u\,(A \times B)\,v & \equiv & u \in A \wedge v \in B \\
u\,(\sim S)\,v & \equiv & \neg(u\,(S)\,v) \\
u\,(S \cup T)\,v & \equiv & u\,(S)\,v \vee u\,(T)\,v \\
u\,(S \cap T)\,v & \equiv & u\,(S)\,v \wedge u\,(T)\,v \\
u\,(S - T)\,v & \equiv & u\,(S)\,v \wedge \neg(u\,(T)\,v) \\
u\,(S \Rightarrow T)\,v & \equiv & u\,(S)\,v \Rightarrow u\,(T)\,v \\
u\,(\mathrm{id}\ A)\,v & \equiv & u = v \in A \\
u\,(\mathbb{I})\,v & \equiv & u = v \\
u\,(R^{\smile})\,v & \equiv & v\,(R)\,u \\
u\,(R ; S)\,v & \equiv & (\exists x \bullet u\,(R)\,x \wedge x\,(S)\,v) \\
u\,(R \backslash S)\,v & \equiv & (\forall x \bullet x\,(R)\,u \Rightarrow x\,(S)\,v) \\
u\,(S / R)\,v & \equiv & (\forall x \bullet v\,(R)\,x \Rightarrow u\,(S)\,x)
\end{array}
$$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-30

Bags, While, Quantification Calculations

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-30

Part 1: Bags/Multisets

"Multisets" or "Bags" — LADM Section 11.7

A **bag** (or **multiset**) is "like a set, but each element can occur any (finite) number of times".

Bag comprehension and enumeration: Written as for sets, but with delimiters $\Lbag$ and $\Rbag$.

Sets versus bags example:

$$\{ x : \mathbb{Z} \mid -2 \le x \le 2 \bullet x \cdot x \} \;=\; \{4, 1, 0\} \;=\; \{0, 1, 4\} \;=\; \{0, 0, 0, 1, 1, 4\}$$

$$\Lbag x : \mathbb{Z} \mid -2 \le x \le 2 \bullet x \cdot x \Rbag \;=\; \Lbag 4, 1, 0, 1, 4 \Rbag \;=\; \Lbag 0, 1, 1, 4, 4 \Rbag \;\neq\; \Lbag 0, 1, 4 \Rbag$$

The operator $\_\#\_ : t \to Bag\ t \to \mathbb{N}$ counts the number of occurrences of an element in a bag:

$$1 \# \Lbag 0, 0, 0, 1, 1, 4 \Rbag \;=\; 2$$

Bag extensionality and **bag inclusion** are defined via all occurrence counts:

$$B = C \;\equiv\; (\forall x \bullet x \# B = x \# C) \qquad B \subseteq C \;\equiv\; (\forall x \bullet x \# B \le x \# C)$$

Bag operations:

$$x \# (B \cup C) = (x \# B) + (x \# C)$$
$$x \# (B \cap C) = (x \# B) \downarrow (x \# C)$$
$$x \# (B - C) = (x \# B) - (x \# C)$$

Bag Product and Bag Reconstitution

Recall: A **bag** is "like a set, but each element can occur any (finite) number of times".

$$\Lbag x : \mathbb{Z} \mid -2 \le x \le 2 \bullet x \cdot x \Rbag = \Lbag 4, 1, 0, 1, 4 \Rbag = \Lbag 0, 1, 1, 4, 4 \Rbag \neq \Lbag 0, 1, 4 \Rbag$$

$\_\#\_ : t \to Bag\ t \to \mathbb{N}$ counts the number of occurrences: $1 \# \Lbag 0, 0, 0, 1, 1, 4 \Rbag = 2$

$\_\in\_ : t \to Bag\ t \to \mathbb{B}$ is membership, with $x \in B \equiv x \# B \neq 0$: $1 \in \Lbag 0, 0, 0, 1, 1, 4 \Rbag \equiv true$

Calculate: $\Lbag x \mid x \in \Lbag 0, 0, 0, 1, 1, 4 \Rbag \Rbag = ?$

Define $bagProd : Bag\ \mathbb{N} \to \mathbb{N}$ such that: $bagProd \Lbag e_1, e_2, \ldots, e_n \Rbag = e_1 \cdot e_2 \cdot \ldots \cdot e_n$

e.g., $bagProd \Lbag 2, 2, 3, 3, 5 \Rbag = 180$

- Easy with exponentiation $\_{**}\_$: $bagProd\ B = \prod\ ?$
- Without exponentiation: $?$

Related question: For sets, we have (11.5): $S = \{x \mid x \in S \bullet x\}$

What is the corresponding theorem for bags?

Bag reconstitution: $B = \Lbag\ ? \mid\ ? \bullet\ ? \Rbag$ ⟶ **Homework 16**

Pigeonhole Principle — LADM section 16.4

The pigeonhole principle is usually stated as follows.

(16.43) If more than n pigeons are placed in n holes, at least one hole will contain more than one pigeon.

Assume:

- $S : Bag\ \mathbb{R}$ is a bag of real numbers
- $av\ S$ is the average of the elements of S
- $max\ S$ is the maximum of the elements of S

Reformulating the pigeonhole principle: (16.44) $av\ S > 1 \;\Rightarrow\; max\ S > 1$

Generalising:

(16.45) Pigeonhole principle:

If $S : Bag\ \mathbb{R}$ is non-empty, then: $av\ S \le max\ S$

Stronger on integers:

(16.46) Pigeonhole principle:

If $S : Bag\ \mathbb{Z}$ is non-empty, then: $\lceil av\ S \rceil \le max\ S$

Generalised Pigeonhole Principle — Application

(16.46) Pigeonhole principle: If $S : Bag\ \mathbb{Z}$ is non-empty, then $\lceil av\ S \rceil \le max\ S$

(16.47) Example: In a room of eight people, at least two of them have birthdays on the same day of the week.

Proof: Let bag S contain, for each day of the week, the number of people in the room whose birthday is on that day. The number of people is 8 and the number of days is 7.

$$S = \Lbag d : Weekday \bullet \# \{ p \mid p\ inRoom\ r_0 \;\wedge\; p\ HasBirthdayOnA\ d \} \Rbag$$

Then:

$$\begin{aligned}
& max\ S \\
\ge\ & \langle \text{Pigeonhole principle (16.46) — } S \text{ contains integers} \rangle \\
& \lceil av\ S \rceil \\
=\ & \langle S \text{ has 7 values that sum to 8} \rangle \\
& \lceil 8/7 \rceil \\
=\ & \langle \text{Definition of ceiling} \rangle \\
& 2
\end{aligned}$$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-30

Part 2: The While Rule

The "While" Rule

The constituents of a while loop "while B do C od" are:

- The **loop condition** $B : \mathbb{B}$
- The **(loop) body** $C : Cmd$

The conventional **while rule** allows to infer only correctness statements for while loops that are in the shape of the conclusion of this inference rule, involving an **invariant** condition $Q : \mathbb{B}$:

```
      `B ∧ Q   ⇒[ C ]   Q`
⊢─────────────────────────────────────
   `Q   ⇒[ while B do C od ]   ¬ B ∧ Q`
```

This rule reads:

- If you can prove that execution of the loop body C starting in states satisfying the loop condition B **preserves** the invariant Q,
- then you have proof that the whole loop also preserves the invariant Q, and in addition establishes the negation of the loop condition.

The "While" Rule — Induction for Partial Correctness

```
      `B ∧ Q   ⇒[ C ]   Q`
⊢─────────────────────────────────────
   `Q   ⇒[ while B do C od ]   ¬ B ∧ Q`
```

The invariant will need to hold

- immediately before the loop starts,
- after each execution of the loop body,
- and therefore also after the loop ends.

The invariant will typically mention all variables that are changed by the loop, and explain how they are related.

In general, you have to identify an appropriate invariant yourself!

Well-written programs contain documentation of invariants for all loops.

Using the "While" Rule

Theorem "While-example":

```
Pre
⇒[ INIT ;
   while B
     do
       C
     od ;
   FINAL
 ]
Post
```

Proof:

```
   Pre        ┅┅┅ Precondition
⇒[ INIT ] ⟨ ? ⟩
   Q          ┅┅┅ Invariant
⇒[ while B do
       C
   od ]  ⟨ "While" with subproof:
             B ∧ Q   ┅┅┅ Loop condition and invariant
          ⇒[ C ] ⟨ ? ⟩
             Q       ┅┅┅ Invariant
         ⟩
   ¬ B ∧ Q   ┅┅┅ Negated loop condition, and invariant
⇒[ FINAL ] ⟨ ? ⟩
   Post      ┅┅┅ Postcondition
```

Goal of Assignment 1.3: Correctness of a Program Containing a `while`-Loop

Theorem "Correctness of `elem`":

```
   true
⇒[ xs := xs₀ ;
   b := false ;
   while xs ≠ ϵ do
       if head xs = x
       then b := true
       else skip
       fi ;
       xs := tail xs
   od
 ]
   (b ≡ x ∈ xs₀)   ┅┅┅ Parentheses!
```

Proof:

```
   true
⇒[ xs := xs₀ ;
   b := false
 ]     ⟨ "Initialisation for `elem`" ⟩
   (∃ us • (us ⌢ xs = xs₀) ∧ (b ≡ x ∈ us))
⇒[ while xs ≠ ϵ do
       if head xs = x
       then b := true
       else skip
       fi ;
       xs := tail xs
   od
 ]     ⟨ "While" with "Invariant for `elem`" ⟩
   ¬ (xs ≠ ϵ) ∧ (∃ us • (us ⌢ xs = xs₀) ∧ (b ≡ x ∈ us))
⇒      ⟨ "Postcondition for `elem`" ⟩
   (b ≡ x ∈ xs₀)
```

"Quantification is Somewhat Like Loops"

```
Theorem "Summing up":
     true
   ⇒[  s := 0 ;
       i := 0 ;
       while i ≠ n
         do
           s := s + f i ;
           i := i + 1
         od
     ]
     s = ∑ j : ℕ | j < n • f j
```

Invariant: $s = \sum j : \mathbb{N} \mid j < i \bullet f\ j$

— Generalised postcondition using the negated loop condition

(This is a frequent pattern.)

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-10-30

Part 3: More Quantification Calculations

(9.29) **Interchange of quantifications::** Provided $\neg occurs(\text{‘}y\text{’}, \text{‘}R\text{’}) \wedge \neg occurs(\text{‘}x\text{’}, \text{‘}Q\text{’})$,

$$(\exists x \mid R \bullet (\forall y \mid Q \bullet P)) \quad \Rightarrow \quad (\forall y \mid Q \bullet (\exists x \mid R \bullet P))$$

One direction only!

Understanding Interchange

Formalise: Every real number has an additive inverse.

$\quad true$

$=\ \langle$ Every real number does have an additive inverse $\rangle$

$\quad (\forall y : \mathbb{R} \bullet (\exists x : \mathbb{R} \bullet y + x = 0))$

$\Leftarrow\ \langle$ (9.29) Interchange of quantifications $\rangle$

$\quad (\exists x : \mathbb{R} \bullet (\forall y : \mathbb{R} \bullet y + x = 0))$

> This says: "There is a real number x which is an additive inverse for all real numbers".

$=\ \langle$ Different numbers have different additive inverses … $\rangle$

$\quad false$

Interchange — Proof

(9.29) **Interchange of quantifications::** Provided $\neg occurs(\text{‘}y\text{’}, \text{‘}R\text{’}) \wedge \neg occurs(\text{‘}x\text{’}, \text{‘}Q\text{’})$,

$$(\exists x \mid R \bullet (\forall y \mid Q \bullet P)) \quad \Rightarrow \quad (\forall y \mid Q \bullet (\exists x \mid R \bullet P))$$

Proof of simpler case ($R \equiv true$)**:**

$\quad (\exists x \bullet (\forall y \bullet P)) \quad \Rightarrow \quad (\forall y \bullet (\exists x \bullet P))$

$=\ \langle$ (3.57) Definition of $\Rightarrow$ $\rangle$

$\quad (\exists x \bullet (\forall y \bullet P)) \vee (\forall y \bullet (\exists x \bullet P)) \equiv (\forall y \bullet (\exists x \bullet P))$

$=\ \langle$ (9.5) Distributivity of $\vee$ over $\forall$ $\rangle$

$\quad (\forall y \bullet (\exists x \bullet (\forall y \bullet P)) \vee (\exists x \bullet P)) \equiv (\forall y \bullet (\exists x \bullet P))$

$=\ \langle$ (8.15) Distributivity of $\exists$ over $\vee$ $\rangle$

$\quad (\forall y \bullet (\exists x \bullet (\forall y \bullet P) \vee P)) \quad \equiv \quad (\forall y \bullet (\exists x \bullet P))$

$=\ \langle$ (9.13.1) Instantiation $(\forall y \bullet P) \Rightarrow P$, with (3.57): $(\forall y \bullet P) \vee P \equiv P$ $\rangle$

$\quad (\forall y \bullet (\exists x \bullet P)) \quad \equiv \quad (\forall y \bullet (\exists x \bullet P))$

$\quad$ — This is (3.5) Reflexivity of $\equiv$

Changing the Quantified Domain

$\quad (\sum i \mid 2 \leq i < 10 \bullet i^2)$

$=\ \langle$ (8.22) with `` `(_+_ 2)` `` *hasAnInverse* $\rangle$

$\quad (\sum k \mid 0 \leq k < 8 \bullet (k+2)^2)$

(8.22) **Change of dummy:** Provided f has an inverse and $\neg occurs(\text{‘}y\text{’}, \text{‘}R, P\text{’})$ (that is, "y is fresh"), then:

$$(\star x \mid R \bullet P) \quad = \quad (\star y \mid R[x := f\ y] \bullet P[x := f\ y])$$

Above: $f\ y\ =\ 2 + y$ and $f^{-1}\ x\ =\ x - 2$

A function f **has an inverse** f^{-1} iff $x = f\ y \quad \equiv \quad y = f^{-1}\ x$

Assume f has an inverse and $\neg occurs(\text{‘}y\text{’}, \text{‘}x, R, P\text{’})$

$\quad (\star y \mid R[x := f\ y] \bullet P[x := f\ y])$

$=\ \langle$ (8.14) One-point rule: $\neg occurs(\text{‘}x\text{’}, \text{‘}f\ y\text{’})$ $\rangle$

$\quad (\star y \mid R[x := f\ y] \bullet (\star x \mid x = f\ y \bullet P))$

$=\ \langle$ (8.20) Nesting: $\neg occurs(\text{‘}x\text{’}, \text{‘}R[x := f\ y]\text{’})$ $\rangle$

$\quad (\star x, y \mid R[x := f\ y] \wedge x = f\ y \bullet P)$

$=\ \langle$ (3.84a) Replacement $(e = f) \wedge E[z := e] \equiv (e = f) \wedge E[z := f]$ $\rangle$

$\quad (\star x, y \mid R[x := x] \wedge x = f\ y \bullet P)$

$=\ \langle$ $R[x := x] = R$; (8.20) Nesting: $\neg occurs(\text{‘}y\text{’}, \text{‘}R\text{’})$ $\rangle$

$\quad (\star x \mid R \bullet (\star y \mid x = f\ y \bullet P))$

$=\ \langle$ Inverse: $x = f\ y \quad \equiv \quad y = f^{-1}\ x$ $\rangle$

$\quad (\star x \mid R \bullet (\star y \mid y = f^{-1}\ x \bullet P))$

$=\ \langle$ (8.14) One-point rule: $\neg occurs(\text{‘}y\text{’}, \text{‘}f^{-1}\ x\text{’})$ $\rangle$

$\quad (\star x \mid R \bullet P[y := f^{-1}\ x])$

$=\ \langle$ Textual substitution, $\neg occurs(\text{‘}y\text{’}, \text{‘}P\text{’})$ $\rangle$

$\quad (\star x \mid R \bullet P)$

Changing the Quantified Domain — $occurs(\text{‘}y\text{’}, \text{‘}x\text{’})$

In the textbook:

(8.22) **Change of dummy:** Provided f has an inverse and $\neg occurs(\text{‘}y\text{’}, \text{‘}R, P\text{’})$,

$$(\star x \mid R \bullet P) \quad = \quad (\star y \mid R[x := f\ y] \bullet P[x := f\ y])$$

We might have that $occurs(\text{‘}y\text{’}, \text{‘}x\text{’})$.

(Note that x and y are metavariables for variables!)

Then x is the same variable as y, and $\neg occurs(\text{‘}x\text{’}, \text{‘}R, P\text{’})$.

Therefore $R[x := f\ y] = R$ and $P[x := f\ y] = P$.

So the theorem's consequence becomes trivial:

$$(\star x \mid R \bullet P) \quad = \quad (\star x \mid R \bullet P)$$

So (8.22) as stated in the textbook is valid, but the proof covers only the case $\neg occurs(\text{‘}y\text{’}, \text{‘}x\text{’})$.

Changing the Quantified Domain — Variants — see Ref. 5.1

Theorem (8.22) "Change of dummy in $\star$":

$\quad \forall f \bullet \forall g \bullet$

$\qquad (\forall x \bullet \forall y \bullet x = f\ y \ \equiv\ y = g\ x)$

$\qquad \Rightarrow (\ (\star x \mid R \bullet P)$

$\qquad\quad = (\star y \mid R[x := f\ y] \bullet P[x := f\ y]))$

Theorem (8.22.1) "Change of dummy in $\star$ — variant":

$\quad (\forall x \bullet \forall y \bullet x = f\ y \ \Rightarrow\ y = g\ x)$

$\quad \Rightarrow (\ (\star x \mid R \wedge x = f\ (g\ x) \bullet P)$

$\qquad = (\star y \mid R[x := f\ y] \bullet P[x := f\ y]))$

Theorem (8.22.3) "Change of restricted dummy in $\star$":

$\quad \forall f \bullet \forall g \bullet$

$\qquad (\forall x \mid R \bullet (\forall y \bullet x = f\ y \ \equiv\ y = g\ x))$

$\qquad \Rightarrow (\ (\star x \mid R \bullet P)$

$\qquad\quad = (\star y \mid R[x := f\ y] \bullet P[x := f\ y]))$

Modal Rules— Converse as Over-Approximation of Inverse

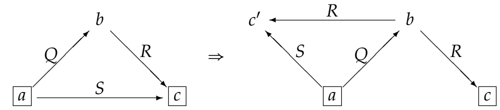
Modal rules: For $Q : \mathcal{A} \leftrightarrow \mathcal{B}$, $R : \mathcal{B} \leftrightarrow \mathcal{C}$, and $S : \mathcal{A} \leftrightarrow \mathcal{C}$:

$$Q \fatsemi R \cap S \ \subseteq\ Q \fatsemi (R \cap Q^{\smile} \fatsemi S)$$

$$Q \fatsemi R \cap S \ \subseteq\ (Q \cap S \fatsemi R^{\smile}) \fatsemi R$$

Useful to "**make information available locally**" (Q is replaced with $Q \cap S \fatsemi R^{\smile}$ for use in further proof steps.

In **constraint** diagrams (boxed variables are free; others existentially quantified; alternative paths are **conjunction**):



$(\exists b \bullet a \lgroup Q \rgroup b \lgroup R \rgroup c \wedge a \lgroup S \rgroup c) \quad \Rightarrow$

$\quad (\exists b, c' \bullet a \lgroup Q \rgroup b \lgroup R \rgroup c \wedge b \lgroup R \rgroup c' \wedge a \lgroup S \rgroup c')$

Proving a Modal Rule — Straight-forward Calculation

Theorem "Modal rule": $(Q \fatsemi R) \cap S \subseteq (Q \cap S \fatsemi R^{\smile}) \fatsemi R$

Proof:

Using "Relation inclusion":

Subproof for `` `∀ a • ∀ c • a ⦗ (Q ⨾ R) ∩ S ⦘ c ⇒ a ⦗ (Q ∩ S ⨾ R ˘) ⨾ R ⦘ c` ``:

For any `` `a` ``, `` `c` ``:

$\quad a \lgroup (Q \cap S \fatsemi R^{\smile}) \fatsemi R \rgroup c$

$\equiv \langle$ "Relation composition" $\rangle$

$\quad \exists b \bullet a \lgroup Q \cap S \fatsemi R^{\smile} \rgroup b \wedge b \lgroup R \rgroup c$

$\equiv \langle$ "Relation intersection", "Relation composition", "Relation converse" $\rangle$

$\quad \exists b \bullet a \lgroup Q \rgroup b \wedge (\exists c_2 \bullet a \lgroup S \rgroup c_2 \wedge b \lgroup R \rgroup c_2) \wedge b \lgroup R \rgroup c$

$\equiv \langle$ "Distributivity of $\wedge$ over $\exists$" $\rangle$

$\quad \exists b \bullet \exists c_2 \bullet a \lgroup Q \rgroup b \wedge a \lgroup S \rgroup c_2 \wedge b \lgroup R \rgroup c_2 \wedge b \lgroup R \rgroup c$

$\Leftarrow \langle\ ?\ \rangle$ ▪▪▪▪▪▪ This is the implication from the previous slide

$\quad \exists b_2 \bullet a \lgroup Q \rgroup b_2 \wedge b_2 \lgroup R \rgroup c \wedge a \lgroup S \rgroup c$

$\equiv \langle$ "Distributivity of $\wedge$ over $\exists$" $\rangle$

$\quad (\exists b_2 \bullet a \lgroup Q \rgroup b_2 \wedge b_2 \lgroup R \rgroup c) \wedge a \lgroup S \rgroup c$

$\equiv \langle$ "Relation intersection", "Relation composition" $\rangle$

$\quad a \lgroup (Q \fatsemi R) \cap S \rgroup c$

Proving a Modal Rule — Straight-forward Calculation (filled)

Theorem "Modal rule": $(Q \fatsemi R) \cap S \subseteq (Q \cap S \fatsemi R^{\smile}) \fatsemi R$

Proof:

Using "Relation inclusion":

Subproof for `` `∀ a • ∀ c • a ⦗ (Q ⨾ R) ∩ S ⦘ c ⇒ a ⦗ (Q ∩ S ⨾ R ˘) ⨾ R ⦘ c` ``:

For any `` `a` ``, `` `c` ``:

$\quad a \lgroup (Q \cap S \fatsemi R^{\smile}) \fatsemi R \rgroup c$

$\equiv \langle$ "Relation composition" $\rangle$

$\quad \exists b \bullet a \lgroup Q \cap S \fatsemi R^{\smile} \rgroup b \wedge b \lgroup R \rgroup c$

$\equiv \langle$ "Relation intersection", "Relation composition", "Relation converse" $\rangle$

$\quad \exists b \bullet a \lgroup Q \rgroup b \wedge (\exists c_2 \bullet a \lgroup S \rgroup c_2 \wedge b \lgroup R \rgroup c_2) \wedge b \lgroup R \rgroup c$

$\equiv \langle$ "Distributivity of $\wedge$ over $\exists$" $\rangle$

$\quad \exists b \bullet \exists c_2 \bullet a \lgroup Q \rgroup b \wedge a \lgroup S \rgroup c_2 \wedge b \lgroup R \rgroup c_2 \wedge b \lgroup R \rgroup c$

$\Leftarrow \langle$ "Body monotonicity of $\exists$" with "$\exists$-Introduction" $\rangle$

$\quad \exists b \bullet (a \lgroup Q \rgroup b \wedge a \lgroup S \rgroup c_2 \wedge b \lgroup R \rgroup c_2 \wedge b \lgroup R \rgroup c)[c_2 := c]$

$\equiv \langle$ Substitution, "Idempotency of $\wedge$" $\rangle$

$\quad \exists b_2 \bullet a \lgroup Q \rgroup b_2 \wedge b_2 \lgroup R \rgroup c \wedge a \lgroup S \rgroup c$

$\equiv \langle$ "Distributivity of $\wedge$ over $\exists$" $\rangle$

$\quad (\exists b_2 \bullet a \lgroup Q \rgroup b_2 \wedge b_2 \lgroup R \rgroup c) \wedge a \lgroup S \rgroup c$

$\equiv \langle$ "Relation intersection", "Relation composition" $\rangle$

$\quad a \lgroup (Q \fatsemi R) \cap S \rgroup c$

Proving a Modal Rule — Artificial `Assuming witness` Variant

Theorem “Modal rule”: $(Q \fatsemi R) \cap S \ \subseteq\ (Q \cap S \fatsemi R^{\smile}) \fatsemi R$
Proof:
  Using “Relation inclusion”:
    Subproof for `$\forall a \bullet \forall c \bullet a ⦗ (Q \fatsemi R) \cap S ⦘ c \Rightarrow a ⦗ (Q \cap S \fatsemi R^{\smile}) \fatsemi R ⦘ c$`:
      For any `$a$`, `$c$`:
        Assuming (1) `$a ⦗ (Q \fatsemi R) \cap S ⦘ c$`:
          Side proof for (2) `$\exists b_2 \bullet a ⦗ Q ⦘ b_2 \wedge b_2 ⦗ R ⦘ c \wedge a ⦗ S ⦘ c$`:
            $a ⦗ (Q \fatsemi R) \cap S ⦘ c$ — **This is** assumption (1)
            $\equiv$ ⟨ “Relation intersection”, “Relation composition” ⟩
            $(\exists b_2 \bullet a ⦗ Q ⦘ b_2 \wedge b_2 ⦗ R ⦘ c) \wedge a ⦗ S ⦘ c$
            $\equiv$ ⟨ “Distributivity of $\wedge$ over $\exists$” ⟩
            $\exists b_2 \bullet a ⦗ Q ⦘ b_2 \wedge b_2 ⦗ R ⦘ c \wedge a ⦗ S ⦘ c$
          Continuing:
            Assuming witness `$b_2$` **satisfying** (3) `$a ⦗ Q ⦘ b_2 \wedge b_2 ⦗ R ⦘ c \wedge a ⦗ S ⦘ c$` **by** local property (2):
              $a ⦗ (Q \cap S \fatsemi R^{\smile}) \fatsemi R ⦘ c$
              $\equiv$ ⟨ “Relation composition” ⟩
              $\exists b \bullet a ⦗ Q \cap S \fatsemi R^{\smile} ⦘ b \wedge b ⦗ R ⦘ c$
              $\Leftarrow$ ⟨ “$\exists$-Introduction” ⟩
              $(a ⦗ Q \cap S \fatsemi R^{\smile} ⦘ b \wedge b ⦗ R ⦘ c)[b := b_2]$
              $\equiv$ ⟨ Substitution, assumption (3), “Identity of $\wedge$” ⟩
              $a ⦗ Q \cap S \fatsemi R^{\smile} ⦘ b_2$
              $\equiv$ ⟨ “Relation intersection”, “Relation composition”, “Relation converse” ⟩
              $a ⦗ Q ⦘ b_2 \wedge \exists c_2 \bullet a ⦗ S ⦘ c_2 \wedge b_2 ⦗ R ⦘ c_2$
              $\equiv$ ⟨ Assumption (3), “Identity of $\wedge$” ⟩
              $\exists c_2 \bullet a ⦗ S ⦘ c_2 \wedge b_2 ⦗ R ⦘ c_2$
              $\Leftarrow$ ⟨ “$\exists$-Introduction” ⟩
              $(a ⦗ S ⦘ c_2 \wedge b_2 ⦗ R ⦘ c_2)[c_2 := c]$
              $\equiv$ ⟨ Substitution, assumption (3), “Identity of $\wedge$” ⟩
              true

Proof:
  Using “Relation inclusion”:
    Subproof for `$\forall a \bullet \forall c \bullet a ⦗ (Q \fatsemi R) \cap S ⦘ c \Rightarrow a ⦗ (Q \cap S \fatsemi R^{\smile}) \fatsemi R ⦘ c$`:
      For any `$a$`, `$c$`:
        Assuming (1) `$a ⦗ (Q \fatsemi R) \cap S ⦘ c$`:
          Assuming witness `$b_2$` **satisfying** (3) `$a ⦗ Q ⦘ b_2 \wedge b_2 ⦗ R ⦘ c \wedge a ⦗ S ⦘ c$`
              by “Distributivity of $\wedge$ over $\exists$” and “Relation intersection”
                and “Relation composition” and assumption (1):
            $a ⦗ (Q \cap S \fatsemi R^{\smile}) \fatsemi R ⦘ c$
            $\equiv$ ⟨ “Relation composition” ⟩
            $\exists b \bullet a ⦗ Q \cap S \fatsemi R^{\smile} ⦘ b \wedge b ⦗ R ⦘ c$
            $\Leftarrow$ ⟨ “$\exists$-Introduction” ⟩
            $(a ⦗ Q \cap S \fatsemi R^{\smile} ⦘ b \wedge b ⦗ R ⦘ c)[b := b_2]$
            $\equiv$ ⟨ Substitution, assumption (3), “Identity of $\wedge$” ⟩
            $a ⦗ Q \cap S \fatsemi R^{\smile} ⦘ b_2$
            $\equiv$ ⟨ “Relation intersection”, “Relation composition”, “Relation converse” ⟩
            $a ⦗ Q ⦘ b_2 \wedge \exists c_2 \bullet a ⦗ S ⦘ c_2 \wedge b_2 ⦗ R ⦘ c_2$
            $\equiv$ ⟨ Assumption (3), “Identity of $\wedge$” ⟩
            $\exists c_2 \bullet a ⦗ S ⦘ c_2 \wedge b_2 ⦗ R ⦘ c_2$
            $\Leftarrow$ ⟨ “$\exists$-Introduction” ⟩
            $(a ⦗ S ⦘ c_2 \wedge b_2 ⦗ R ⦘ c_2)[c_2 := c]$
            $\equiv$ ⟨ Substitution, assumption (3), “Identity of $\wedge$” ⟩
            true

Descending Chains in Numbers

Consider numbers with the usual strict-order $<$

and consider descending chains, like $17 > 12 > 9 > 8 > 3 > \ldots$

Are there infinite descending chains in

- $\mathbb{Z}$?
- $\mathbb{N}$?
- $\mathbb{R}$?
- $\mathbb{R}_+$?
- $\mathbb{Q}_+$?
- $\mathbb{C}$?

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-11-01

General Induction, Trees

Plan for Today

- General Induction (LADM section 12.4)
- **Tree Datastructures; Structural Induction**

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-11-01

Part 1: General Induction — LADM Section 12.4

Descending Chains in Numbers

Consider numbers with the usual strict-order $<$

and consider descending chains, like $17 > 12 > 9 > 8 > 3 > \ldots$

Are there infinite descending chains in

- $\mathbb{Z}$? — $0 > -1 > -2 > -3 > \ldots$
- $\mathbb{N}$? — **No**
- $\mathbb{R}$? — $0 > -1 > -2 > -3 > \ldots$
- $\mathbb{R}_+$? — $\pi^0 > \pi^{-1} > \pi^{-2} > \pi^{-3} > \ldots$
- $\mathbb{Q}_+$? — $1 > 1/2 > 1/3 > 1/4 > \ldots$
- $\mathbb{C}$? — no “default” order!

Relations $\prec$ with no infinite (descending) $\succ$-chains are **well-founded**.

Loops **terminate** iff they are “going down” some well-founded relation.

Idea Behind Induction — How Does It Work? — Informally

Proving $(\forall x : t \bullet P)$ by induction, **for an appropriate type t**:

- You are familiar with proving a base case and an induction step
- The base cases establish $P[x := S]$, for each S that are “simplest t”
- The induction steps work for $x : t$ for which we already know $P[x := x]$ and from that establish $P[x := C\,x]$ for elements $C\,x : t$ that “are slightly more complicated than x”.
- Since the construction principle(s) (“C”) used in the induction step is/are sufficiently powerful to construct all $x : t$, this justifies $(\forall x : t \bullet P)$.

Idea Behind Induction — How Does It Work? — Informally

Proving $(\forall x : t \bullet P)$ by induction, **for an appropriate type t**:

- You are familiar with proving a base case and an induction step
- The base cases establish $P[x := S]$, for each S that are “simplest t”
- The induction steps work for $x : t$ for which we already know $P[x := x]$ and from that establish $P[x := C\,x]$ for elements $C\,x : t$ that “are slightly more complicated than x”.
- Since the construction principle(s) (“C”) used in the induction step is/are sufficiently powerful to construct all $x : t$, this justifies $(\forall x : t \bullet P)$.

Looking at this from the other side:

- Each element $x : t$ is either a “simplest element” (“S”), or constructed via a construction principle (“C”) from “slightly simpler elements” y, that is, $x = C\,y$.
- In the first case, the base case gives you the proof for $P[x := S]$.
- In the second case, you obtain $P[x := Cy]$ via the induction step from a proof for $P[x := y]$, if you can find that.
- You can find that proof if repeated decomposition into S or C always terminates.

Idea Behind Induction — Reduction via Well-founded Relations

- Goal: prove $(\forall x : T \bullet P\,x)$ for some property $P : T \to \mathbb{B}$ (with $\neg occurs(‘x’, ‘P’)$)
- Situation: Elements of T are related via $\_\succ\_ : T \to T \to \mathbb{B}$ with “simpler” elements (constituents, predecessors, parts, …)
  “$y \prec x$” may read “y precedes x” or “y is an (immediate) constituent of x” or “y is simpler than x” or “y is below x”…
- If for every $x : T$ there is a proof that

  if $P\,y$ for all predecessors y of x, then $P\,x$,

  then for every $z : T$ with $\neg(P\,z)$:
  - there is a predecessor u of z with $\neg(P\,u)$
  - and so there is an infinite $\succ$-chain (of elements c with $\neg(P\,c)$) starting at z.

Theorem Mathematical induction over $\langle T, \prec \rangle$:
If there are no infinite $\succ$-chains in T, that is, **if $\prec$ is noetherian**, then:

$$(\forall x \bullet P\,x) \quad \equiv \quad (\forall x \bullet (\forall y \mid y \prec x \bullet P\,y) \Rightarrow P\,x)$$

“$\langle T, \precneq \rangle$ Admits Induction” (LADM Section 12.4)

Definition (12.19): $\langle T, \precneq \rangle$ **admits induction** iff the following principle of **mathematical induction over** $\langle T, \precneq \rangle$ holds for all properties $P : T \to \mathbb{B}$:

$$(\forall\, x \bullet P\, x) \quad \equiv \quad (\forall\, x \bullet (\forall\, y \mid y \precneq x \bullet P\, y) \Rightarrow P\, x)$$

Definition (12.21): $\langle T, \precneq \rangle$ is **well-founded** iff every non-empty subset of T has a minimal element wrt. $\precneq$, that is:

$$\forall\, S : \mathbf{set}\ T \quad \bullet \quad S \neq \{\} \quad \equiv \quad \exists\, x : T \bullet x \in S \land \forall\, y : T \mid y \precneq x \bullet y \notin S$$

Theorem (12.22): $\langle T, \precneq \rangle$ is well-founded iff it admits induction.

Definition (12.25′): $\langle T, \precneq \rangle$ is **noetherian** iff there are no infinite $\succneq$-chains in T.

Theorem (12.26): $\langle T, \precneq \rangle$ is well-founded iff it is noetherian.

Theorem Mathematical induction over $\langle T, \precneq \rangle$**:**
If there are no infinite $\succneq$-chains in T, that is, **if $\precneq$ is noetherian**, then:

$$(\forall\, x \bullet P\, x) \quad \equiv \quad (\forall\, x \bullet (\forall\, y \mid y \precneq x \bullet P\, y) \Rightarrow P\, x)$$

Mathematical Induction in $\mathbb{N}$

Consider $\_\precneq\_ : \mathbb{N} \to \mathbb{N} \to \mathbb{B}$ with $(x \precneq y) = (y \succneq x) = (y = suc\ x)$. $\_\precneq\_ = \ulcorner suc\_ \urcorner$

Mathematical induction over $(\mathbb{N}, \precneq)$**:**

$(\forall\, x : \mathbb{N} \bullet P\, x)$

$=$ ⟨ (12.19) Math. induction; Def. $\precneq$ ⟩

$(\forall\, x : \mathbb{N} \bullet (\forall\, y : \mathbb{N} \mid suc\ y = x \bullet P\, y) \Rightarrow P\, x)$

$=$ ⟨ Disjoint range split, with $true \equiv x = 0 \lor x > 0$ ⟩

$(\forall\, x : \mathbb{N} \mid x = 0 \bullet (\forall\, y : \mathbb{N} \mid suc\ y = x \bullet P\, y) \Rightarrow P\, x) \land$
$(\forall\, x : \mathbb{N} \mid x > 0 \bullet (\forall\, y : \mathbb{N} \mid suc\ y = x \bullet P\, y) \Rightarrow P\, x)$

$=$ ⟨ One-point rule; (8.22) Change of dummy ⟩

$((\forall\, y : \mathbb{N} \mid suc\ y = 0 \bullet P\, y) \Rightarrow P\, 0) \land$
$(\forall\, z : \mathbb{N} \bullet (\forall\, y : \mathbb{N} \mid suc\ y = suc\ z \bullet P\, y) \Rightarrow P\,(suc\ z))$

$=$ ⟨ (8.13) Empty range, with $suc\ y = 0 \equiv false$; Cancellation of suc, (8.14) One-point rule for $\forall$ ⟩

$P\, 0 \land (\forall\, z : \mathbb{N} \bullet P\, z \Rightarrow P\,(suc\ z))$

Mathematical Induction in $\mathbb{N}$ (ctd.)

Mathematical induction over $(\mathbb{N}, \ulcorner suc \urcorner)$**:**

$$(\forall\, x : \mathbb{N} \bullet P\, x) \quad \equiv \quad P\, 0 \land (\forall\, z : \mathbb{N} \bullet P\, z \Rightarrow P\,(suc\ z))$$

$$(\forall\, x : \mathbb{N} \bullet P\, x) \quad \equiv \quad P\, 0 \land (\forall\, z : \mathbb{N} \bullet P\, z \Rightarrow P\,(z + 1))$$

Absence of infinite **descending** $\ulcorner suc \urcorner$ chains is due to the **inductive definition of $\mathbb{N}$ with constructors 0 and** suc : “…and nothing else is a natural number.”

Mathematical induction over $(\mathbb{N}, <)$ **“Complete induction over $\mathbb{N}$”:**

$$(\forall\, x : \mathbb{N} \bullet P\, x) \equiv (\forall\, x : \mathbb{N} \bullet (\forall\, y : \mathbb{N} \mid y < x \bullet P\, y) \Rightarrow P\, x)$$

Complete induction gives you a **stronger induction hypothesis** for non-zero x — some proofs become easier.

Example for Complete Induction in $\mathbb{N}$

Mathematical induction over $(\mathbb{N}, <)$ **“Complete induction over $\mathbb{N}$”:**

$$(\forall\, x : \mathbb{N} \bullet P\, x) \equiv (\forall\, x : \mathbb{N} \bullet (\forall\, y : \mathbb{N} \mid y < x \bullet P\, y) \Rightarrow P\, x)$$

Theorem: Every natural number greater than 1 is a product of (one or more) prime numbers.
Formalisation: $\forall\, n : \mathbb{N} \bullet 1 < n \Rightarrow (\exists B : Bag\ \mathbb{N} \mid (\forall p \mid p \in B \bullet isPrime\ p) \bullet bagProd\ B = n)$
Proof:
- Using “Complete induction”:
  - **For any** `$n$`:
    - **Assuming** `$\forall\, m \mid m < n \bullet 1 < m \Rightarrow (\exists B : Bag\ \mathbb{N} \mid (\forall p \mid p \in B \bullet isPrime\ p) \bullet bagProd\ B = m)$`:
      - **Assuming** `$1 < n$`:
        - **By cases:** `$isPrime\ n$`, `$\neg(isPrime\ n)$`
          - **Completeness:** By “Excluded middle”
          - **Case** `$isPrime\ n$`:
            - …“∃-Introduction”: $B := ⦃n⦄$ …
          - **Case** `$\neg(isPrime\ n)$`:
            - …then $n = n_1 \cdot n_2$ with $n_1 < n > n_2$
            - …with witness: $bagProd\ B_1 = n_1$ and $bagProd\ B_2 = n_2$
            - …then $bagProd\ (B_1 \cup B_2) = n$

Mathematical Induction on Sequences

Cons induction: Mathematical induction over $(Seq\ A, \precneq)$ where

$$\precneq\ := \{x : A; xs, ys : Seq\ A \mid x \triangleleft xs = ys \bullet \langle xs, ys \rangle\}$$

$$(\forall\, xs : Seq\ A \bullet P\, xs) \quad \equiv \quad P\, \epsilon \land (\forall\, xs : Seq\ A \mid P\, xs \bullet (\forall\, x : A \bullet P(x \triangleleft xs)))$$

Snoc induction: Mathematical induction over $(Seq\ A, \precneq)$ where

$$\precneq\ := \{x : A; xs, ys : Seq\ A \mid xs \triangleright x = ys \bullet \langle xs, ys \rangle\}$$

$$(\forall\, xs : Seq\ A \bullet P\, xs) \quad \equiv \quad P\, \epsilon \land (\forall\, xs : Seq\ A \mid P\, xs \bullet (\forall\, x : A \bullet P(xs \triangleright x)))$$

Strict prefix induction: Mathematical induction over $(Seq\ A, \precneq)$ where

$$\precneq\ := \{us, xs, ys : Seq\ A \mid us \neq \epsilon \land xs \frown us = ys \bullet \langle xs, ys \rangle\}$$

$$(\forall\, xs : Seq\ A \bullet P\, xs) \quad \equiv \quad (\forall\, xs : Seq\ A \bullet (\forall\, ys : Seq\ A \mid ys \precneq xs \bullet P\, ys) \Rightarrow P\, xs)$$

Different induction hypotheses make certain proofs easier.

Structural Induction

Structural induction is mathematical induction over, *e.g.*,

- **finite sequences** with the strict suffix relation
- **expressions** with the direct constituent relation
- **propositional formulae** with the strict subformula relation
- **trees** with the appropriate strict subtree relation
- **proofs** with appropriate strict sub-proof relation
- **programs** with appropriate strict sub-program relation
- …

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

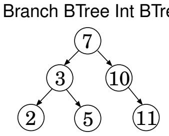
2023-11-01

Part 2: Inductive Datastructures: Trees

Inductively-defined Tree Data Structures

Binary (search) trees

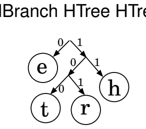
```
data BTree = EmptyB
  | Branch BTree Int BTree
```




```
bt1left = Branch
    (Branch EmptyB 2 EmptyB)
    3
    (Branch EmptyB 5 EmptyB)
bt1right = Branch
    EmptyB
    10
    (Branch EmptyB 11 EmptyB)
```

Huffman trees

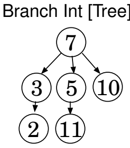
```
data HTree = Leaf Char
  | HBranch HTree HTree
```




```
hTree1 = HBranch (Leaf 'e')
    (HBranch
        (HBranch (Leaf 't') (Leaf 'r'))
        (Leaf 'h'))
decode hTree1 "100110" = "the"
```

Arbitrarily branching

```
data Tree
   = Branch Int [Tree]
```

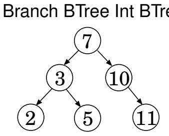



```
t1left = Branch 7
   [Branch 3 [Branch 2 []]
   ,Branch 5 [Branch 11 []]
   ,Branch 10 []
   ]
```

Binary Trees (Exercise 8.3)

Binary (search) trees

```
data BTree = EmptyB
  | Branch BTree Int BTree
```

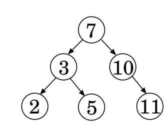



```
bt1left = Branch
  (Branch EmptyB 2 EmptyB)
  3
  (Branch EmptyB 5 EmptyB)
bt1right = Branch
  EmptyB
  10
  (Branch EmptyB 11 EmptyB)
```

```
Declaration:          ◬   : Tree A
Declaration:        _⊿◺_ : Tree A → A → Tree A → Tree A

Declaration: t1 : Tree ℕ
Axiom “Definition of `t1`”:
  t1 = ((◬ ⊿ 2 ◺ ◬) ⊿ 3 ◺ (◬ ⊿ 5 ◺ ◬))
        ⊿ 7 ◺
        (◬ ⊿ 10 ◺ (◬ ⊿ 11 ◺ ◬))

Fact “Alternative definition of `t1`”:
  t1 = (⌈ 2 ⌋ ⊿ 3 ◺ ⌈ 5 ⌋)
        ⊿ 7 ◺
        (◬ ⊿ 10 ◺ ⌈ 11 ⌋)
```

Binary Trees (Exercise 10.4)

```
Declaration:          ◬   : Tree A
Declaration:        _⊿◺_ : Tree A → A → Tree A → Tree A

Declaration: t1 : Tree ℕ
Axiom “Definition of `t1`”:
  t1 = ((◬ ⊿ 2 ◺ ◬) ⊿ 3 ◺ (◬ ⊿ 5 ◺ ◬))
       ⊿ 7 ◺
       (◬ ⊿ 10 ◺ (◬ ⊿ 11 ◺ ◬))

Fact “Alternative definition of `t1`”:
  t1 = (⌈ 2 ⌋ ⊿ 3 ◺ ⌈ 5 ⌋)
       ⊿ 7 ◺
       (◬ ⊿ 10 ◺ ⌈ 11 ⌋)

Axiom “Tree induction”:
      P[t = ◬]
  ∧  ( ∀ l, r : Tree A; x : A
      • P[t = l] ∧ P[t = r]  ⇒  P[t = l ⊿ x ◺ r]
     )
  ⇒  (∀ t : Tree A • P)
```



Using the Induction Principle for Binary Trees

```
Theorem “Self-inverse of tree mirror”: ∀ t : Tree A • (t ˘) ˘ = t
Proof:
  Using “Tree induction”:
    Subproof for `◬ ˘ ˘ = ◬`: By “Mirror”
    Subproof for `∀ l, r : Tree A; x : A
        • (l ˘) ˘ = l ∧ (r ˘) ˘ = r
        ⇒ (l ⊿ x ⊾ r)˘ ˘ = (l ⊿ x ⊾ r)`:
      For any `l, r, x`:
        Assuming “IHL” `(l ˘) ˘ = l`,
                 “IHR” `(r ˘) ˘ = r`:
            (l ⊿ x ⊾ r) ˘ ˘
          =⟨ “Mirror” ⟩
            (l ˘ ˘) ⊿ x ⊾ (r ˘ ˘)
          =⟨ Assumptions “IHL” and “IHR” ⟩
            l ⊿ x ⊾ r
```

```
Axiom “Tree induction”:
      P[t = ◬]
  ∧  ( ∀ l, r : Tree A; x : A
       • P[t = l] ∧ P[t = r]  ⇒  P[t = l ⊿ x ⊾ r]
     )
  ⇒  (∀ t : Tree A • P)
```

Recall: Induction — Reduction via Well-founded Relations

- Goal: prove $(\forall\, x : T \bullet P\, x)$ for some property $P : T \to \mathbb{B}$ (with $\neg occurs(‘x’, ‘P’)$)
- Situation: Elements of T are related via $\_\succsim\_ : T \to T \to \mathbb{B}$ with “simpler” elements (constituents, predecessors, parts, …)
  “$y \precsim x$” may read “y precedes x” or “y is an (immediate) constituent of x” or “y is simpler than x” or “y is below x”…
- If for every $x : T$ there is a proof that

  if $P\, y$ for all predecessors y of x, then $P\, x$,

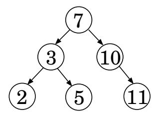
  then for every $z : T$ with $\neg(P\, z)$:
  - there is a predecessor u of z with $\neg(P\, u)$
  - and so there is an infinite $\succsim$-chain (of elements c with $\neg(P\, c)$) starting at z.

Theorem (12.19) **Mathematical induction over** $(T, \precsim)$**:**
If there are no infinite $\succsim$-chains in T, that is, **if $\precsim$ is well-founded**, then:

$$(\forall\, x \bullet P\, x) \quad \equiv \quad (\forall\, x \bullet (\forall\, y \mid y \precsim x \bullet P\, y) \Rightarrow P\, x)$$

Induction Principle for Binary Trees

```
Declaration:        ◬  : Tree A
Declaration:      _⊿_⊾_ : Tree A → A → Tree A → Tree A
```




```
Fact “Alternative definition of `t1`”:
  t1 = (⌜ 2 ⌟ ⊿ 3 ⊾ ⌜ 5 ⌟)
       ⊿ 7 ⊾
       (◬ ⊿ 10 ⊾ ⌜ 11 ⌟)

Declaration: _≾_ : Tree A → Tree A → 𝔹
Axiom “HTree ≾”:
    (t ≾ ◬              ≡  false)
 ∧  (t ≾ (l ⊿ x ⊾ r)  ≡  t = l  ∨  t = r)
```

Theorem (12.19) **Mathematical induction over** $(T, \precsim)$**, if $\precsim$ is well-founded**

$$(\forall\, x \bullet P\, x) \quad \equiv \quad (\forall\, x \bullet (\forall\, y \mid y \precsim x \bullet P\, y) \Rightarrow P\, x)$$

Equivalently:

```
Axiom “Tree induction”:
      P[t = ◬]
  ∧  ( ∀ l, r : Tree A; x : A
       • P[t = l] ∧ P[t = r]  ⇒  P[t = l ⊿ x ⊾ r]
     )
  ⇒  (∀ t : Tree A • P)
```

Trees are Everywhere!

- Search trees, dictionary datastructures — BinTree, balanced trees
- Huffman trees — used for compression encoding e.g. in JPEG
- Abstract Syntax Trees (ASTs) — central datastructures in compilers
  — *Recall:* For expressions, we write strings, but we think trees…
- …
- Every “data” in Haskell defines a (possibly degenerated) tree datastructure

In programming:

- **Trees are easy to deal with.**
- Graphs, even DAGs (directed acyclic graphs), can be tricky
  - — even with good APIs.
  - Choosing “the right” API is already hard!
  - The same holds for relations!

  — Because relations *are* graphs…

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-11-03

Change of Dummy in A1.3, Functions, λ

A1.3 — Direct Approach to “Invariant for ‘elem‘”

Theorem “Invariant for `elem`”:

$(\mathsf{xs} \neq \epsilon) \wedge (\exists\, \mathsf{us} \bullet \mathsf{us} \frown \mathsf{xs} = xs_0 \wedge (b \equiv x \in \mathsf{us}))$
$\Rightarrow [$ if head xs $= x$ then $b :=$ true else skip fi ; xs := tail xs $]$
$(\exists\, \mathsf{us} \bullet \mathsf{us} \frown \mathsf{xs} = xs_0 \wedge (b \equiv x \in \mathsf{us}))$

Proof:

$(\exists\, \mathsf{us} \bullet \mathsf{us} \frown \mathsf{xs} = xs_0 \wedge (b \equiv x \in \mathsf{us}))$
$[$ xs := tail xs $] \Leftarrow$ ⟨ “Assignment” with substitution ⟩
$(\exists\, \mathsf{us} \bullet \mathsf{us} \frown \text{tail xs} = xs_0 \wedge (b \equiv x \in \mathsf{us}))$
$[$ if head xs $= x$ then $b :=$ true else skip fi
$] \Leftarrow$ ⟨ **Subproof:**
  Using “Conditional”:
    Subproof:
      ? ⋯⋯ Long subproof
    Subproof:
      ? ⋯⋯ Long subproof with a lot of duplicated material
⟩
$(\mathsf{xs} \neq \epsilon) \wedge (\exists\, \mathsf{us} \bullet \mathsf{us} \frown \mathsf{xs} = xs_0 \wedge (b \equiv x \in \mathsf{us}))$

A1.3 — Direct Approach to “Invariant for ‘elem‘” — Looking More Closely

Theorem “Invariant for `elem`”:

$(\mathsf{xs} \neq \epsilon) \wedge (\exists\, \mathsf{us} \bullet \mathsf{us} \frown \mathsf{xs} = xs_0 \wedge (b \equiv x \in \mathsf{us}))$
$\Rightarrow [$ if head xs $= x$ then $b :=$ true else skip fi ; xs := tail xs $]$
$(\exists\, \mathsf{us} \bullet \mathsf{us} \frown \mathsf{xs} = xs_0 \wedge (b \equiv x \in \mathsf{us}))$

Proof:

$(\exists\, \mathsf{us} \bullet \mathsf{us} \frown \mathsf{xs} = xs_0 \wedge (b \equiv x \in \mathsf{us}))$
$[$ xs := tail xs $] \Leftarrow$ ⟨ “Assignment” with substitution ⟩
$(\exists\, \mathsf{us} \bullet \mathsf{us} \frown \text{tail xs} = xs_0 \wedge (b \equiv x \in \mathsf{us}))$
$[$ if head xs $= x$ then $b :=$ true else skip fi $] \Leftarrow$ ⟨ **Subproof:**
  Using “Conditional”:
    Subproof:
      ? ⋯⋯ Long subproof containing:
        ⋯⋯ $\Leftarrow$ ⟨“$\exists$ – *Introduction*”⟩
        ⋯⋯ $(us \frown tail\ xs = xs_0 \wedge \ldots)[us := us \triangleright head\ xs]$
    Subproof:
      ? ⋯⋯ Long subproof with a lot of duplicated material, in particular:
        ⋯⋯ $\Leftarrow$ ⟨“$\exists$ – *Introduction*”⟩
        ⋯⋯ $(us \frown tail\ xs = xs_0 \wedge \ldots)[us := us \triangleright head\ xs]$
⟩
$(\mathsf{xs} \neq \epsilon) \wedge (\exists\, \mathsf{us} \bullet \mathsf{us} \frown \mathsf{xs} = xs_0 \wedge (b \equiv x \in \mathsf{us}))$

Recall: Changing the Quantified Domain

$(\sum\, i \mid 2 \leq i < 10 \bullet i^2)$
$=$ ⟨ (8.22) “Change of dummy” with `$(\_+\_\ 2)$ *hasAnInverse*` ⟩
$(\sum\, k \mid 0 \leq k < 8 \bullet (k+2)^2)$

(8.22) **Change of dummy:** Provided f has an inverse and $\neg occurs(‘y’, ‘R, P’)$ (that is, “y is fresh”), then:

$$(\star\, x \mid R \bullet P) \quad = \quad (\star\, y \mid R[x := f\, y] \bullet P[x := f\, y])$$

Above: $f\, y = 2 + y$ and $f^{-1}\, x = x - 2$

A function f **has an inverse** f^{-1} iff $x = f\, y \equiv y = f^{-1}\, x$

Recall: Changing the Quantified Domain — Variants — see Ref. 5.1

Theorem (8.22) “Change of dummy in $\star$”:

$\forall f \bullet \forall g \bullet$
$(\forall\, x \bullet \forall\, y \bullet x = f\, y \equiv y = g\, x)$
$\Rightarrow ((\star\, x \mid R \bullet P)$
$= (\star\, y \mid R[x := f\, y] \bullet P[x := f\, y]))$

Theorem (8.22.1) “Change of dummy in $\star$ — variant”:

$(\forall\, x \bullet \forall\, y \bullet x = f\, y \Rightarrow y = g\, x)$
$\Rightarrow ((\star\, x \mid R \wedge x = f\,(g\, x) \bullet P)$
$= (\star\, y \mid R[x := f\, y] \bullet P[x := f\, y]))$

Theorem (8.22.3) “Change of restricted dummy in $\star$”:

$\forall f \bullet \forall g \bullet$
$(\forall\, x \mid R \bullet (\forall\, y \bullet x = f\, y \equiv y = g\, x))$
$\Rightarrow ((\star\, x \mid R \bullet P)$
$= (\star\, y \mid R[x := f\, y] \bullet P[x := f\, y]))$

Change of Dummy in A1.3 — (8.22)?

$(\exists\, \mathsf{us} \bullet \mathsf{us} \frown \text{tail xs} = xs_0 \wedge (b \equiv x \in \mathsf{us}))$
$\Leftarrow$ ⟨ ? ⟩
$(\exists\, \mathsf{us} \bullet \mathsf{us} \triangleright \text{head xs} \frown \text{tail xs} = xs_0 \wedge (b \equiv x \in \mathsf{us} \triangleright \text{head xs}))$

Trying to use the following to prove this:

Theorem (8.22) “Change of dummy in $\exists$”:
$(\forall\, x \bullet \forall\, y \bullet x = f\, y \equiv y = g\, x)$
$\Rightarrow ((\exists\, x \mid R \bullet P)$
$= (\exists\, y \mid R[x := f\, y] \bullet P[x := f\, y]))$

What are the functions involved?

Declaration: $f_1 : A \to \mathsf{Seq}\, A \to \mathsf{Seq}\, A$
Axiom “f_1”: $f_1\, x\, \mathsf{ys} = \mathsf{ys} \triangleright x$
Declaration: init : Seq $A \to$ Seq A
Axiom “init”: init $(\mathsf{xs} \triangleright y) = \mathsf{xs}$ ⋯⋯ like `tail`, only specified for non-empty sequences

For being able to use (8.22) “Change of dummy in $\exists$” with $f, g := f_1\,(head\ xs),\ init$, we would need: $(\forall xs \bullet \forall ys \bullet xs = f_1\, x\, ys \equiv ys = init\, xs)$
However, the $\Leftarrow$-part of the equivalence here is clearly not valid.

Change of Dummy in A1.3 — (8.22.1)?

$(\exists\, \mathsf{us} \bullet \mathsf{us} \frown \mathsf{tail\ xs} = xs_0 \wedge (b \equiv x \in \mathsf{us}))$
$\Leftarrow \langle\ ?\ \rangle$
$(\exists\, \mathsf{us} \bullet \mathsf{us} \triangleright \mathsf{head\ xs} \frown \mathsf{tail\ xs} = xs_0 \wedge (b \equiv x \in \mathsf{us} \triangleright \mathsf{head\ xs}))$

We do have the $\Rightarrow$-part of $(\forall xs \bullet \forall ys \bullet xs = f_1 x ys \equiv ys = init\, xs)$:

Lemma "f_1 to init": $\forall\, \mathsf{xs} \bullet \forall\, \mathsf{ys} \bullet \mathsf{xs} = f_1\, x\, \mathsf{ys} \Rightarrow \mathsf{ys} = \mathsf{init\ xs}$

For applying

Theorem (8.22.1) "Change of dummy in $\exists$ — variant":
$(\forall\, x \bullet \forall\, y \bullet x = f\, y \Rightarrow y = g\, x)$
$\Rightarrow (\ (\exists\, x \mid R \wedge x = f\,(g\, x) \bullet P)$
$= (\exists\, y \mid R[x := f\, y] \bullet P[x := f\, y]))$

, the range predicate of the LHS of the consequent needs to be in shape $R \wedge x = f\,(g\, x)$.

Since we only need a consequence calculation, not an equivalence, we can achieve this easily using "Range weakening for $\exists$".

Change of Dummy in A1.3 — (8.22.1)!

Theorem (8.22.1) "Change of dummy in $\exists$ — variant":
$(\forall\, x \bullet \forall\, y \bullet x = f\, y \Rightarrow y = g\, x)$
$\Rightarrow (\ (\exists\, x \mid R \wedge x = f\,(g\, x) \bullet P)$
$= (\exists\, y \mid R[x := f\, y] \bullet P[x := f\, y]))$

Declaration: $f_1 : A \to \mathsf{Seq}\ A \to \mathsf{Seq}\ A$
Axiom "f_1": $f_1\, x\, \mathsf{ys} = \mathsf{ys} \triangleright x$
Declaration: $\mathsf{init} : \mathsf{Seq}\ A \to \mathsf{Seq}\ A$
Axiom "init": $\mathsf{init}\ (\mathsf{xs} \triangleright y) = \mathsf{xs}$ ▪▪▪▪▪▪ like `tail`, only specified for non-empty sequences

Lemma "f_1 to init": $\forall\, \mathsf{xs} \bullet \forall\, \mathsf{ys} \bullet \mathsf{xs} = f_1\, x\, \mathsf{ys} \Rightarrow \mathsf{ys} = \mathsf{init\ xs}$

The fragment of the proof of "Invariant for \`*elem*\`" then becomes:

$\exists\, \mathsf{us} \bullet \mathsf{us} \frown \mathsf{tail\ xs} = xs_0 \wedge (b \equiv x \in \mathsf{us})$
$\Leftarrow \langle$ "Range weakening for $\exists$" $\rangle$
$\exists\, \mathsf{us} \mid \mathsf{true} \wedge \mathsf{us} = f_1\, (\mathsf{head\ xs})\, (\mathsf{init\ us}) \bullet \mathsf{us} \frown \mathsf{tail\ xs} = xs_0 \wedge (b \equiv x \in \mathsf{us})$
$\equiv \langle$ "Change of dummy in $\exists$ — variant" with "f_1 to init" $\rangle$
$\exists\, \mathsf{vs} \mid true[us := f_1\, (\mathsf{head\ xs})\, vs] \bullet (\mathsf{us} \frown \mathsf{tail\ xs} = xs_0 \wedge (b \equiv x \in \mathsf{us}))[us := f_1\, (\mathsf{head\ xs})\, vs]$
$\equiv \langle$ Substitution, "f_1" $\rangle$
$\exists\, \mathsf{us} \bullet \mathsf{us} \triangleright \mathsf{head\ xs} \frown \mathsf{tail\ xs} = xs_0 \wedge (b \equiv x \in \mathsf{us} \triangleright \mathsf{head\ xs})$

Look Again at the Functions

Declaration: $f_1 : A \to \mathsf{Seq}\ A \to \mathsf{Seq}\ A$
Axiom "f_1": $f_1\, x\, \mathsf{ys} = \mathsf{ys} \triangleright x$
Declaration: $\mathsf{init} : \mathsf{Seq}\ A \to \mathsf{Seq}\ A$
Axiom "init": $\mathsf{init}\ (\mathsf{xs} \triangleright y) = \mathsf{xs}$ ▪▪▪▪▪▪ like `tail`, only specified for non-empty sequences

We used the name "init" because we know it from Haskell.
Don't we know a name for f_1 as well? — *flip snoc* — flip _▹_
Same problem as for "init": We know "flip", but it is not imported in the current scope…
In doubt, reproduce known definitions and theorems:

Declaration: $\mathsf{flip} : (A \to B \to C) \to (B \to A \to C)$
Axiom "flip": $\mathsf{flip}\, f\, y\, x = f\, x\, y$

For the property we need here, the same proof:

Lemma "flip-snoc to init": $\forall\, \mathsf{xs} \bullet \forall\, \mathsf{ys} \bullet \mathsf{xs} = \mathsf{flip}\ \_\triangleright\_\ x\ \mathsf{ys} \Rightarrow \mathsf{ys} = \mathsf{init\ xs}$
Proof:
For any \`xs\`, \`ys\`:
Assuming (1) \`xs = flip _▹_ $x$ ys\`:
init xs
$= \langle$ Assumption (1) $\rangle$
init (flip _ ▹ _ x ys)

How to Prove that flip is Self-inverse?

Declaration: $\mathsf{flip} : (A \to B \to C) \to (B \to A \to C)$
Axiom "flip": $\mathsf{flip}\, f\, y\, x = f\, x\, y$

Theorem "Self-inverse \`flip\`": $\mathsf{flip}\ (\mathsf{flip}\, f) = f$
Proof:
$\mathsf{flip}\ (\mathsf{flip}\, f)\, x\, y$
$= \langle$ "flip" $\rangle$
$\mathsf{flip}\, f\, y\, x$
$= \langle$ "flip" $\rangle$
$f\, x\, y$

Not a proof!

The missing piece:

Theorem "Function extensionality": $f = g \equiv \forall\, x \bullet f\, x = g\, x$

Proving that flip is Self-inverse

Declaration: $\mathsf{flip} : (A \to B \to C) \to (B \to A \to C)$
Axiom "flip": $\mathsf{flip}\, f\, y\, x = f\, x\, y$

Theorem "Function extensionality": $f = g \equiv \forall\, x \bullet f\, x = g\, x$

Theorem "Self-inverse \`flip\`": $\mathsf{flip}\ (\mathsf{flip}\, f) = f$
Proof:
Using "Function extensionality":
Subproof for \`$\forall\, x \bullet \mathsf{flip}\ (\mathsf{flip}\, f)\, x = f\, x$\`:
For any \`$x$\`:
Using "Function extensionality":
For any \`$y$\`:
$\mathsf{flip}\ (\mathsf{flip}\, f)\, x\, y$
$= \langle$ "flip" $\rangle$
$\mathsf{flip}\, f\, y\, x$
$= \langle$ "flip" $\rangle$
$f\, x\, y$

More Conveniently Proving that flip is Self-inverse

Declaration: $\mathsf{flip} : (A \to B \to C) \to (B \to A \to C)$
Axiom "flip": $\mathsf{flip}\, f\, y\, x = f\, x\, y$

Theorem "Function extensionality": $f = g \equiv \forall\, x \bullet f\, x = g\, x$

Theorem "Function extensionality 2": $f = g \equiv \forall\, x, y \bullet f\, x\, y = g\, x\, y$
Proof:
By "Function extensionality", "Nesting for $\forall$"

Theorem "Self-inverse \`flip\`": $\mathsf{flip}\ (\mathsf{flip}\, f) = f$
Proof:
Using "Function extensionality 2":
For any \`$x$, $y$\`:
$\mathsf{flip}\ (\mathsf{flip}\, f)\, x\, y$
$= \langle$ "flip" $\rangle$
$\mathsf{flip}\, f\, y\, x$
$= \langle$ "flip" $\rangle$
$f\, x\, y$

Some "Prelude" Functions and Some of Their Properties

Declaration: $\mathsf{id} : A \to A$
Axiom "Identity function": $\mathsf{id}\, x = x$

Declaration: $\_\circ\_ : (B \to C) \to (A \to B) \to (A \to C)$
Axiom "Function composition": $(g \circ f)\, x = g\,(f\, x)$

Theorem "Associativity of $\circ$": $h \circ (g \circ f) = (h \circ g) \circ f$

Declaration: $\mathsf{curry} : (\langle\!| A, B |\!\rangle \to C) \to (A \to B \to C)$
Declaration: $\mathsf{uncurry} : (A \to B \to C) \to (\langle\!| A, B |\!\rangle \to C)$

Axiom "curry": $\mathsf{curry}\, g\, x\, y = g\, \langle x, y \rangle$
Axiom "uncurry": $\mathsf{uncurry}\, f\, \langle x, y \rangle = f\, x\, y$

Theorem "curry∘uncurry": $\mathsf{curry}\ (\mathsf{uncurry}\, f) = f$

Declaration: $\mathsf{swap} : \langle\!| A, B |\!\rangle \to \langle\!| B, A |\!\rangle$
Axiom "swap": $\mathsf{swap}\, \langle x, y \rangle = \langle y, x \rangle$

Theorem "flip∘curry": $\mathsf{flip}\ (\mathsf{curry}\, f) = \mathsf{curry}\ (f \circ \mathsf{swap})$

And If We Don't Want to Define flip?

Declaration: $\mathsf{flip} : (A \to B \to C) \to (B \to A \to C)$
Axiom "flip": $\mathsf{flip}\, f\, y\, x = f\, x\, y$

We can use **nameless functions** instead of *flip snoc*:

- In Haskell: $\backslash\ x\ ys \to ys ++ [x]$
- In CalcCheck: $\lambda\, x \bullet \lambda\, ys \bullet ys \triangleright x$
  - λ-abstractions follow the quantification notation pattern "as far as possible"
  - Module FunctionAbstraction provides in particular β-reduction
  - Module Quantification.GenQuant.Lambda provides those quantification properties that do carry over.

λ-Calculus

λ-abstraction creates nameless functions: If $E : B$, then $(\lambda\, x : A \bullet E) : A \to B$.

The following are usually introduced as left-to-right reduction rules:

Theorem "β-reduction": $(\lambda\, x \bullet E)\, a = E[x := a]$

Theorem "η-reduction": $(\lambda\, x \bullet F\, x) = F$ — provided $\neg occurs('x', 'F')$

In addition, "α-conversion" is capture-avoiding renaming of bound variables.
Function extensionality follows from η-reduction (and is actually equivalent):

Theorem "Function extensionality": $f = g \equiv \forall\, x \bullet f\, x = g\, x$
Proof:
Using "Mutual implication":
Subproof for \`$f = g \Rightarrow \forall\, x \bullet f\, x = g\, x$\`:
Assuming \`$f = g$\`:
For any \`$x$\`: **By** assumption \`$f = g$\`
Subproof:
Assuming (1) \`$\forall\, x \bullet f\, x = g\, x$\`:
f
$= \langle$ "η-reduction" $\rangle$
$\lambda\, x \bullet f\, x$
$= \langle$ Assumption (1) — implicitly using quantification Leibniz $\rangle$
$\lambda\, x \bullet g\, x$
$= \langle$ "η-reduction" $\rangle$
g

λ-Abstraction produces Functions, not Univalent Relations

λ-abstraction creates nameless **functions:** If, $E : B$ (and $R : \mathbb{B}$) with $x : A$, then:

- $(\lambda\, x : A \bullet E)$ is a **function** of function type $A \to B$
- $\{ x \bullet \langle x, E \rangle \} = \{ x, y \mid y = E \}$ is a **mapping** and an element of the set $\llcorner A \lrcorner \twoheadrightarrow \llcorner B \lrcorner$
- $(\lambda\, x : A \mid R \bullet E)$ is a **function** of function type $A \to B$
  For arguments $a : A$ for which $R[x := a]$ evaluates to *false*, the result is not specified.
- $\{ x \mid R \bullet \langle x, E \rangle \} = \{ x, y \mid R \wedge y = E \}$ is a **univalent relation** (partial function) and an element of the set $\llcorner A \lrcorner \nrightarrow \llcorner B \lrcorner$

  We have: $\forall\, a : A \mid \neg R[x := a] \bullet a \notin Dom\, \{ x \mid R \bullet \langle x, E \rangle \}$

  Example: For the **partial function** $Pred = \{ x, y \mid x = suc\, y \}$, we have $0 \notin Dom\ Pred$

Big-O

Does $O(n \cdot log\ n)$ talk about n? — Abuse of notation!

$O(n \cdot log\ n)$ talks about the function "$\lambda\, n \bullet n \cdot log\ n$"!

Declaration: $O : (\mathbb{R} \to \mathbb{R}) \to \mathsf{set}\,(\mathbb{R} \to \mathbb{R})$

Axiom "Definition of big O":

$$f \in O\, g \ \equiv\ \exists\, b \bullet \exists\, c \mid c > 0 \bullet \forall\, x \mid x > b \bullet \mathsf{abs}\,(f\ x) < c \cdot g\ x$$

Theorem: $(\lambda\, x \bullet 4 \cdot x + 7) \in O\,(\lambda\, x \bullet x)$

Proof:

$(\lambda\, x \bullet 4 \cdot x + 7) \in O\,(\lambda\, x \bullet x)$

$\equiv$ ⟨ "Definition of big O" ⟩

$\exists\, b \bullet \exists\, c \mid c > 0 \bullet \forall\, x \mid x > b \bullet \mathsf{abs}\,((\lambda\, x \bullet 4 \cdot x + 7)\, x) < c \cdot (\lambda\, x \bullet x)\, x$

$\equiv$ ⟨ "β-reduction", substitution ⟩

$\exists\, b \bullet \exists\, c \mid c > 0 \bullet \forall\, x \mid x > b \bullet \mathsf{abs}\,(4 \cdot x + 7) < c \cdot x$

$\Leftarrow$ ⟨ "$\exists$-Introduction" ⟩

$(\exists\, c \mid c > 0 \bullet \forall\, x \mid x > b \bullet \mathsf{abs}\,(4 \cdot x + 7) < c \cdot x)[b := 2]$

$\equiv$ ⟨ Substitution, "Trading for $\exists$" ⟩

$(\exists\, c \bullet c > 0 \wedge \forall\, x \mid x > 2 \bullet \mathsf{abs}\,(4 \cdot x + 7) < c \cdot x)$

$\Leftarrow$ ⟨ "$\exists$-Introduction" ⟩

$(c > 0 \wedge \forall\, x \mid x > 2 \bullet \mathsf{abs}\,(4 \cdot x + 7) < c \cdot x)[c := 8]$

$\equiv$ ⟨ Substitution, Fact `$8 > 0$`, "Identity of $\wedge$" ⟩

$(\forall\, x \mid x > 2 \bullet \mathsf{abs}\,(4 \cdot x + 7) < 8 \cdot x)$

Proof for this:

For any `$x$` **satisfying** `$2 < x$`:

Side proof for (1) `$4 \cdot x + 7 > 0$`:

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-11-06

Relation-Algebraic Calculational Proofs

Plan for Today

- Relation-algebraic calculational proofs — "abstract relation algebra"

Relation-algebraic proof …

- … will be the main topic of Exercises 9.*
- … will be on Midterm 2
- … is easier than quantifier reasoning

Recall: Translating between Relation Algebra and Predicate Logic

$R = S$	$\equiv$	$(\forall\, x, y \bullet x \llparenthesis R \rrparenthesis y \equiv x \llparenthesis S \rrparenthesis y)$
$R \subseteq S$	$\equiv$	$(\forall\, x, y \bullet x \llparenthesis R \rrparenthesis y \Rightarrow x \llparenthesis S \rrparenthesis y)$
$u \llparenthesis \{\} \rrparenthesis v$	$\equiv$	false
$u \llparenthesis A \times B \rrparenthesis v$	$\equiv$	$u \in A \wedge v \in B$
$u \llparenthesis {\sim} S \rrparenthesis v$	$\equiv$	$\neg(u \llparenthesis S \rrparenthesis v)$
$u \llparenthesis S \cup T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \vee u \llparenthesis T \rrparenthesis v$
$u \llparenthesis S \cap T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \wedge u \llparenthesis T \rrparenthesis v$
$u \llparenthesis S - T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \wedge \neg(u \llparenthesis T \rrparenthesis v)$
$u \llparenthesis S \Rightarrow T \rrparenthesis v$	$\equiv$	$u \llparenthesis S \rrparenthesis v \Rightarrow u \llparenthesis T \rrparenthesis v$
$u \llparenthesis \mathrm{id}\ A \rrparenthesis v$	$\equiv$	$u = v \in A$
$u \llparenthesis \mathbb{I} \rrparenthesis v$	$\equiv$	$u = v$
$u \llparenthesis R^\smile \rrparenthesis v$	$\equiv$	$v \llparenthesis R \rrparenthesis u$
$u \llparenthesis R \fatsemi S \rrparenthesis v$	$\equiv$	$(\exists\, x \bullet u \llparenthesis R \rrparenthesis x \wedge x \llparenthesis S \rrparenthesis v)$
$u \llparenthesis R \backslash S \rrparenthesis v$	$\equiv$	$(\forall\, x \bullet x \llparenthesis R \rrparenthesis u \Rightarrow x \llparenthesis S \rrparenthesis v)$
$u \llparenthesis S / R \rrparenthesis v$	$\equiv$	$(\forall\, x \bullet v \llparenthesis R \rrparenthesis x \Rightarrow u \llparenthesis S \rrparenthesis x)$

Using Extensionality/Inclusion and the Translation Table, you Proved:

Theorem "Self-inverse of $^\smile$": $R^{\smile\smile} = R$
Theorem "Converse of $\cap$": $(R \cap S)^\smile = R^\smile \cap S^\smile$
Theorem "Converse of $\fatsemi$": $(R \fatsemi S)^\smile = S^\smile \fatsemi R^\smile$
Theorem "Converse of $\mathbb{I}$": $\mathbb{I}^\smile = \mathbb{I}$
Theorem "Isotonicity of $^\smile$": $R \subseteq S \equiv R^\smile \subseteq S^\smile$
Theorem "Converse of $\cup$": $(R \cup S)^\smile = R^\smile \cup S^\smile$
Theorem "Distributivity of $\fatsemi$ over $\cup$": $Q \fatsemi (R \cup S) = Q \fatsemi R \cup Q \fatsemi S$
Theorem "Sub-distributivity of $\fatsemi$ over $\cap$": $Q \fatsemi (R \cap S) \subseteq Q \fatsemi R \cap Q \fatsemi S$
Theorem "Left-identity of $\fatsemi$" "Identity of $\fatsemi$": $\mathbb{I} \fatsemi R = R$
Theorem "Right-identity of $\fatsemi$" "Identity of $\fatsemi$": $R \fatsemi \mathbb{I} = R$
Theorem "Composition of reflexive relations": reflexive $R \Rightarrow$ reflexive $S \Rightarrow$ reflexive $(R \fatsemi S)$
Theorem "Converse of reflexive relations": reflexive $R \Rightarrow$ reflexive $(R^\smile)$
Theorem "Converse reflects reflectivity": reflexive $(R^\smile) \Rightarrow$ reflexive R
Theorem "Converse of transitive relations": transitive $R \Rightarrow$ transitive $(R^\smile)$
Theorem "Associativity of $\fatsemi$": $(Q \fatsemi R) \fatsemi S = Q \fatsemi (R \fatsemi S)$
Theorem "Distributivity of $\fatsemi$ over $\cup$": $(Q \cup R) \fatsemi S = Q \fatsemi S \cup R \fatsemi S$
Theorem "Sub-distributivity of $\fatsemi$ over $\cap$": $(Q \cap R) \fatsemi S \subseteq Q \fatsemi S \cap R \fatsemi S$
Theorem "Monotonicity of $\fatsemi$": $Q \subseteq R \Rightarrow Q \fatsemi S \subseteq R \fatsemi S$
Theorem "Converse of $\{\}$": $\{\}^\smile = \{\}$
Theorem "Co-difunctionality" "Hesitation": $R \subseteq R \fatsemi R^\smile \fatsemi R$
Theorem "Modal rule": $(Q \fatsemi R) \cap S \subseteq Q \fatsemi (R \cap Q^\smile \fatsemi S)$
Theorem "Dedekind rule": $(Q \fatsemi R) \cap S \subseteq (Q \cap S \fatsemi R^\smile) \fatsemi (R \cap Q^\smile \fatsemi S)$
Theorem "Schröder": $Q \fatsemi R \subseteq S \equiv {\sim} S \fatsemi R^\smile \subseteq {\sim} Q$

> All subexpressions have $\mathbb{B}$ or $\_{\leftrightarrow}\_$ types!
> Equations of relational expressions:
> **Relation algebra**

Relation Algebra

- For any two types B and C, on the type $B \leftrightarrow C$ of **relations between B and C** we have the ordering $\subseteq$ with:
  - binary minima $\_\cap\_$ and maxima $\_\cup\_$ (which are monotonic)
  - least relation $\{\}$ and largest ("universal") relation U (= ⌞B⌟ × ⌞C⌟)
  - complement operation $\sim\_$ such that $R \cap {\sim} R = \{\}$ and $R \cup {\sim} R = U$
  - relative pseudo-complement $R \Rightarrow S = {\sim} R \cup S$
- The composition operation $\_\fatsemi\_$
  - is defined on any two relations $R : B \leftrightarrow C_1$ and $S : C_2 \leftrightarrow D$ iff $C_1 = C_2$
  - is associative, monotonic, and has identities $\mathbb{I}$
  - distributes over union: $Q \fatsemi (R \cup S) = Q \fatsemi R \cup Q \fatsemi S$
- The converse operation $\_^\smile$
  - maps relation $R : B \leftrightarrow C$ to $R^\smile : C \leftrightarrow B$
  - is self-inverse ($R^{\smile\smile} = R$) and monotonic
  - is contravariant wrt. composition: $(R \fatsemi S)^\smile = S^\smile \fatsemi R^\smile$
- The Dedekind rule holds: $Q \fatsemi R \cap S \subseteq (Q \cap S \fatsemi R^\smile) \fatsemi (R \cap Q^\smile \fatsemi S)$
- The Schröder equivalences hold:
  $Q \fatsemi R \subseteq S \equiv Q^\smile \fatsemi {\sim} S \subseteq {\sim} R$ and $Q \fatsemi R \subseteq S \equiv {\sim} S \fatsemi R^\smile \subseteq {\sim} Q$
- $\fatsemi$ has left-residuals $S / R = {\sim}({\sim} S \fatsemi R^\smile)$ and right-residuals $Q \backslash S = {\sim}(Q^\smile \fatsemi {\sim} S)$

Recall: Monotonicity of Relation Composition

Relation composition is monotonic in both arguments:

$Q \subseteq R \Rightarrow Q \fatsemi S \subseteq R \fatsemi S$

$Q \subseteq R \Rightarrow P \fatsemi Q \subseteq P \fatsemi R$

We could prove this via **"Relation inclusion"** *and* **"For any"**, *but we don't need to:*

Assume $Q \subseteq R$, which by (11.45) is equivalent to $Q \cup R = R$:

Proving $Q \fatsemi S \subseteq R \fatsemi S$:

$R \fatsemi S$

$=$ ⟨ Assumption $Q \cup R = R$ ⟩

$(Q \cup R) \fatsemi S$

$=$ ⟨ (14.23) Distributivity of $\fatsemi$ over $\cup$ ⟩

$Q \fatsemi S \cup R \fatsemi S$

$\supseteq$ ⟨ (11.31) Strengthening $S \subseteq S \cup T$ ⟩

$Q \fatsemi S$

Recall: Relation-Algebraic Proof of Sub-Distributivity

Use set-algebraic properties and **Monotonicity of $\fatsemi$**: $Q \subseteq R \Rightarrow P \fatsemi Q \subseteq P \fatsemi R$

to prove: **Subdistributivity of $\fatsemi$ over $\cap$**: $Q \fatsemi (R \cap S) \subseteq (Q \fatsemi R) \cap (Q \fatsemi S)$

$Q \fatsemi (R \cap S)$

$=$ ⟨ Idempotence of $\cap$ (11.35) ⟩

$(Q \fatsemi (R \cap S)) \cap (Q \fatsemi (R \cap S))$

$\subseteq$ ⟨ **Mon. of $\cap$ with Mon. of $\fatsemi$ with** Weakening $X \cap Y \subseteq X$ ⟩

$(Q \fatsemi (R \cap S)) \cap (Q \fatsemi S)$

$\subseteq$ ⟨ **Mon. of $\cap$ with Mon. of $\fatsemi$ with** Weakening $X \cap Y \subseteq X$
— without two-sided monotonicity,
separate $\subseteq$-steps are needed in CalcCheck! ⟩

$(Q \fatsemi R) \cap (Q \fatsemi S)$

Recall: Properties of Homogeneous Relations

reflexive	$\mathbb{I} \subseteq R$	$(\forall\, b : B \bullet b \llparenthesis R \rrparenthesis b)$
irreflexive	$\mathbb{I} \cap R = \{\}$	$(\forall\, b : B \bullet \neg(b \llparenthesis R \rrparenthesis b))$
symmetric	$R^\smile = R$	$(\forall\, b, c : B \bullet b \llparenthesis R \rrparenthesis c \equiv c \llparenthesis R \rrparenthesis b)$
antisymmetric	$R \cap R^\smile \subseteq \mathbb{I}$	$(\forall\, b, c \bullet b \llparenthesis R \rrparenthesis c \wedge c \llparenthesis R \rrparenthesis b \Rightarrow b = c)$
asymmetric	$R \cap R^\smile = \{\}$	$(\forall\, b, c : B \bullet b \llparenthesis R \rrparenthesis c \Rightarrow \neg(c \llparenthesis R \rrparenthesis b))$
transitive	$R \fatsemi R \subseteq R$	$(\forall b, c, d \bullet b \llparenthesis R \rrparenthesis c \wedge c \llparenthesis R \rrparenthesis d \Rightarrow b \llparenthesis R \rrparenthesis d)$

R is an **equivalence (relation) on** B iff it is reflexive, transitive, and symmetric. (E.g., $=$, $\equiv$)

R is a **(partial) order on** B
iff it is reflexive, transitive, and antisymmetric.
(E.g., $\leq, \geq, \subseteq, \supseteq, \mid$)

R is a **strict-order on** B
iff it is irreflexive, transitive, and asymmetric.
(E.g., $<, >, \subset, \supset$)

Homogeneous Relation Properties are Preserved by Converse

reflexive	$\mathbb{I} \subseteq R$	$(\forall\, b : B \bullet b \llparenthesis R \rrparenthesis b)$
irreflexive	$\mathbb{I} \cap R = \{\}$	$(\forall\, b : B \bullet \neg(b \llparenthesis R \rrparenthesis b))$
symmetric	$R^\smile = R$	$(\forall\, b, c : B \bullet b \llparenthesis R \rrparenthesis c \equiv c \llparenthesis R \rrparenthesis b)$
antisymmetric	$R \cap R^\smile \subseteq \mathbb{I}$	$(\forall\, b, c \bullet b \llparenthesis R \rrparenthesis c \wedge c \llparenthesis R \rrparenthesis b \Rightarrow b = c)$
asymmetric	$R \cap R^\smile = \{\}$	$(\forall\, b, c : B \bullet b \llparenthesis R \rrparenthesis c \Rightarrow \neg(c \llparenthesis R \rrparenthesis b))$
transitive	$R \fatsemi R \subseteq R$	$(\forall b, c, d \bullet b \llparenthesis R \rrparenthesis c \wedge c \llparenthesis R \rrparenthesis d \Rightarrow b \llparenthesis R \rrparenthesis d)$
idempotent	$R \fatsemi R = R$	

Theorem: If $R : B \leftrightarrow B$ is reflexive/irreflexive/symmetric/antisymmetric/asymmetric/transitive/idempotent, then $R^\smile$ has that property, too.

Proof: Reflexivity:

$R^\smile$

$\supseteq$ ⟨ **Mon.** $^\smile$ **with** Reflexivity of R ⟩

$\mathbb{I}^\smile$

$=$ ⟨ Symmetry of $\mathbb{I}$ ⟩

$\mathbb{I}$

Transitivity:

$R^\smile \fatsemi R^\smile$

$=$ ⟨ Converse of $\fatsemi$ ⟩

$(R \fatsemi R)^\smile$

$\subseteq$ ⟨ **Mon.** $^\smile$ **with** Trans. of R ⟩

$R^\smile$

Reflexive and Transitive Implies Idempotent

reflexive	$\mathbb{I}$	$\subseteq$	R	$(\forall\, b : B \bullet b \llparenthesis R \rrparenthesis b)$
transitive	$R \fatsemi R$	$\subseteq$	R	$(\forall\, b, c, d \bullet b \llparenthesis R \rrparenthesis c \llparenthesis R \rrparenthesis d \Rightarrow b \llparenthesis R \rrparenthesis d)$
idempotent	$R \fatsemi R$	$=$	R	

Theorem: If $R : B \leftrightarrow B$ is reflexive and transitive, then it is also idempotent.

Reflexive and Transitive Implies Idempotent — Direct Approach

Theorem "Idempotency from reflexive and transitive":
reflexive R ⇒ transitive R ⇒ idempotent R

reflexive	$\mathbb{I}$	$\subseteq$	R
transitive	$R \fatsemi R$	$\subseteq$	R
idempotent	$R \fatsemi R$	$=$	R

Proof:
Assuming \`reflexive $R$\`, \`transitive $R$\`:

idempotent R
≡ ⟨ "Definition of idempotency" ⟩
$R \fatsemi R = R$
≡ ⟨ "Mutual inclusion" ⟩
$R \fatsemi R \subseteq R \ \wedge\ R \subseteq R \fatsemi R$
≡ ⟨ "Definition of transitivity", assumption \`transitive $R$\`, "Identity of ∧" ⟩
$R \subseteq R \fatsemi R$
≡ ⟨ "Identity of ⨾" ⟩
$R \fatsemi \mathbb{I} \subseteq R \fatsemi R$
⇐ ⟨ "Monotonicity of ⨾" ⟩
$\mathbb{I} \subseteq R$
≡ ⟨ Assumption \`reflexive $R$\` with "Definition of reflexivity" ⟩
true

Reflexive and Transitive Implies Idempotent — "and using with"

Theorem "Idempotency from reflexive and transitive":
reflexive R ⇒ transitive R ⇒ idempotent R

reflexive	$\mathbb{I}$	$\subseteq$	R
transitive	$R \fatsemi R$	$\subseteq$	R
idempotent	$R \fatsemi R$	$=$	R

Proof:
Assuming \`reflexive $R$\` **and using with** "Definition of reflexivity",
\`transitive $R$\` **and using with** "Definition of transitivity":

idempotent R
≡ ⟨ "Definition of idempotency" ⟩
$R \fatsemi R = R$
≡ ⟨ "Mutual inclusion" ⟩
$R \fatsemi R \subseteq R \ \wedge\ R \subseteq R \fatsemi R$
≡ ⟨ Assumption \`transitive $R$\`, "Identity of ∧" ⟩
$R \subseteq R \fatsemi R$
≡ ⟨ "Identity of ⨾" ⟩
$R \fatsemi \mathbb{I} \subseteq R \fatsemi R$
⇐ ⟨ "Monotonicity of ⨾" ⟩
$\mathbb{I} \subseteq R$
≡ ⟨ Assumption \`reflexive $R$\` ⟩
true

Reflexive and Transitive Implies Idempotent — Semi-formal

reflexive	$\mathbb{I}$	$\subseteq$	R	$(\forall\, b : B \bullet b \llparenthesis R \rrparenthesis b)$
transitive	$R \fatsemi R$	$\subseteq$	R	$(\forall\, b, c, d \bullet b \llparenthesis R \rrparenthesis c \llparenthesis R \rrparenthesis d \Rightarrow b \llparenthesis R \rrparenthesis d)$
idempotent	$R \fatsemi R$	$=$	R	

Theorem: If $R : B \leftrightarrow B$ is reflexive and transitive, then it is also idempotent.

Proof: By mutual inclusion and transitivity of R, we only need to show $R \subseteq R \fatsemi R$:

R
$=$ ⟨ Identity of ⨾ ⟩
$R \fatsemi \mathbb{I}$
$\subseteq$ ⟨ **Mon. ⨾ with** Reflexivity of R ⟩
$R \fatsemi R$

Reflexive and Transitive Implies Idempotent — Cyclic ⊆-chain Proving \`=\`

Theorem "Idempotency from reflexive and transitive":
reflexive R ⇒ transitive R ⇒ idempotent R

reflexive	$\mathbb{I}$	$\subseteq$	R
transitive	$R \fatsemi R$	$\subseteq$	R
idempotent	$R \fatsemi R$	$=$	R

Proof:
Assuming \`reflexive $R$\` **and using with** "Definition of reflexivity",
\`transitive $R$\` **and using with** "Definition of transitivity":
Using "Definition of idempotency":
Subproof for \`$R \fatsemi R = R$\`:

$R \fatsemi R$
⊆ ⟨ Assumption \`transitive $R$\` ⟩
R
= ⟨ "Identity of ⨾" ⟩
$R \fatsemi \mathbb{I}$
⊆ ⟨ "Monotonicity of ⨾" with assumption \`reflexive $R$\` ⟩
$R \fatsemi R$

Using cyclic ⊆-chains to prove equalities requires activation of antisymmetry of ⊆.

Most Homogeneous Relation Properties are Preserved by Intersection

reflexive	$\mathbb{I}$	$\subseteq$	R
irreflexive	$\mathbb{I} \cap R$	$=$	$\{\}$
transitive	$R \fatsemi R$	$\subseteq$	R
idempotent	$R \fatsemi R$	$=$	R

symmetric	$R^{\smile}$	$=$	R
antisymmetric	$R \cap R^{\smile}$	$\subseteq$	$\mathbb{I}$
asymmetric	$R \cap R^{\smile}$	$=$	$\{\}$

Theorem: If $R, S : B \leftrightarrow B$ are reflexive/irreflexive/symmetric/antisymmetric/asymmetric/transitive, then $R \cap S$ has that property, too.

Proof: Reflexivity:

$R \cap S$
⊇ ⟨ Mon. of ∩ with Refl. S ⟩
$R \cap \mathbb{I}$
⊇ ⟨ Mon. of ∩ with Refl. R ⟩
$\mathbb{I} \cap \mathbb{I}$
= ⟨ Idempotence of ∩ ⟩
$\mathbb{I}$

Transitivity:

$(R \cap S) \fatsemi (R \cap S)$
⊆ ⟨ Sub-distributivity of ⨾ over ∩ ⟩
$(R \fatsemi R) \cap (R \fatsemi S) \cap (S \fatsemi R) \cap (S \fatsemi S)$
⊆ ⟨ Weakening $X \cap Y \subseteq X$ ⟩
$(R \fatsemi R) \cap (S \fatsemi S)$
⊆ ⟨ Mon. ∩ with transitivity of R and S ⟩
$R \cap S$

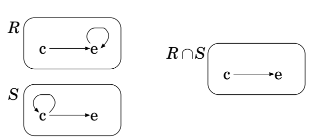
<u>Most</u> Homogeneous Relaton Properties are Preserved by Intersection

reflexive	$\mathbb{I}$	$\subseteq$	R
irreflexive	$\mathbb{I} \cap R$	$=$	$\{\}$
transitive	$R \fatsemi R$	$\subseteq$	R
idempotent	$R \fatsemi R$	$=$	R

symmetric	$R^{\smile}$	$=$	R
antisymmetric	$R \cap R^{\smile}$	$\subseteq$	$\mathbb{I}$
asymmetric	$R \cap R^{\smile}$	$=$	$\{\}$

Theorem: If $R, S : B \leftrightarrow B$ are reflexive/irreflexive/symmetric/antisymmetric/asymmetric/transitive, then $R \cap S$ has that property, too.

Counter-example for preservation of idempotence:



<u>Some</u> Homogeneous Relation Properties are Preserved by Union

reflexive	$\mathbb{I}$	$\subseteq$	R
irreflexive	$\mathbb{I} \cap R$	$=$	$\{\}$
transitive	$R \fatsemi R$	$\subseteq$	R
idempotent	$R \fatsemi R$	$=$	R

symmetric	$R^{\smile}$	$=$	R
antisymmetric	$R \cap R^{\smile}$	$\subseteq$	$\mathbb{I}$
asymmetric	$R \cap R^{\smile}$	$=$	$\{\}$

Theorem: If $R, S : B \leftrightarrow B$ are reflexive/irreflexive/symmetric, then $R \cup S$ has that property, too.

Proof:

Reflexivity:

$\mathbb{I}$
⊆ ⟨ Reflexivity of R ⟩
R
⊆ ⟨ Weakening $X \subseteq X \cup Y$ ⟩
$R \cup S$

Irreflexivity:

$\mathbb{I} \cap (R \cup S)$
= ⟨ Distributivity of ∩ over ∪ ⟩
$(\mathbb{I} \cap R) \cup (\mathbb{I} \cap S)$
= ⟨ Irreflexivity of R and S ⟩
$\{\} \cup \{\}$
= ⟨ Idempotence of ∪ ⟩
$\{\}$

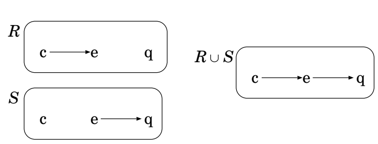
<u>Some</u> Homogeneous Relation Properties are Preserved by Union

reflexive	$\mathbb{I}$	$\subseteq$	R
irreflexive	$\mathbb{I} \cap R$	$=$	$\{\}$
transitive	$R \fatsemi R$	$\subseteq$	R
idempotent	$R \fatsemi R$	$=$	R

symmetric	$R^{\smile}$	$=$	R
antisymmetric	$R \cap R^{\smile}$	$\subseteq$	$\mathbb{I}$
asymmetric	$R \cap R^{\smile}$	$=$	$\{\}$

Theorem: If $R, S : B \leftrightarrow B$ are reflexive/irreflexive/symmetric, then $R \cup S$ has that property, too.

Counter-example for preservation of transitivity:



Weaker Formulation of Symmetry

reflexive	$\mathbb{I}$	$\subseteq$	R
irreflexive	$\mathbb{I} \cap R$	$=$	$\{\}$
transitive	$R \fatsemi R$	$\subseteq$	R
idempotent	$R \fatsemi R$	$=$	R

symmetric	$R^{\smile}$	$=$	R
antisymmetric	$R \cap R^{\smile}$	$\subseteq$	$\mathbb{I}$
asymmetric	$R \cap R^{\smile}$	$=$	$\{\}$

For proving symmetry of $R, S : B \leftrightarrow B$, it is sufficient to prove $R^{\smile} \subseteq R$.

In other words:

Theorem: If $R^{\smile} \subseteq R$, then $R^{\smile} = R$.

Proof: By mutual inclusion, we only need to show $R \subseteq R^{\smile}$:

R
$=$ ⟨ Self-inverse of converse ⟩
$(R^{\smile})^{\smile}$
$\subseteq$ ⟨ Mon. of $^{\smile}$ with Assumption $R^{\smile} \subseteq R$ ⟩
$R^{\smile}$

Symmetric and Transitive Implies Idempotent

symmetric	$R^\smile = R$	$(\forall\, b, c : B \bullet b \llparenthesis R \rrparenthesis c \equiv c \llparenthesis R \rrparenthesis b)$
transitive	$R \fatsemi R \subseteq R$	$(\forall b, c, d \bullet b \llparenthesis R \rrparenthesis c \llparenthesis R \rrparenthesis d \Rightarrow b \llparenthesis R \rrparenthesis d)$
idempotent	$R \fatsemi R = R$	

Theorem: A symmetric and transitive $R : B \leftrightarrow B$ is also idempotent.
Proof: By mutual inclusion and transitivity of R, we only need to show $R \subseteq R \fatsemi R$:

R
$=$ ⟨ Idempotence of $\cap$, Identity of $\fatsemi$ ⟩
$R \fatsemi \mathbb{I} \cap R$
$\subseteq$ ⟨ **Modal rule** $Q \fatsemi R \cap S \subseteq Q \fatsemi (R \cap Q^\smile \fatsemi S)$ ⟩
$R \fatsemi (\mathbb{I} \cap R^\smile \fatsemi R)$
$\subseteq$ ⟨ **Mon. $\fatsemi$ with** Weakening $X \cap Y \subseteq X$ ⟩
$R \fatsemi R^\smile \fatsemi R$
$=$ ⟨ Symmetry of R ⟩
$R \fatsemi R \fatsemi R$
$\subseteq$ ⟨ **Mon. $\fatsemi$ with** Transitivity of R ⟩
$R \fatsemi R$

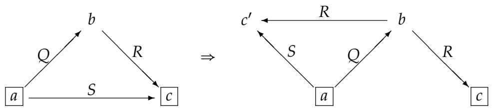
Symmetric and Transitive Implies Idempotent

symmetric	$R^\smile = R$	$(\forall\, b, c : B \bullet b \llparenthesis R \rrparenthesis c \equiv c \llparenthesis R \rrparenthesis b)$
transitive	$R \fatsemi R \subseteq R$	$(\forall b, c, d \bullet b \llparenthesis R \rrparenthesis c \llparenthesis R \rrparenthesis d \Rightarrow b \llparenthesis R \rrparenthesis d)$
idempotent	$R \fatsemi R = R$	

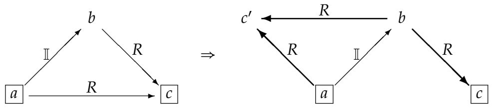
Theorem: A symmetric and transitive $R : B \leftrightarrow B$ is also idempotent.
Proof: By mutual inclusion and transitivity of R, we only need to show $R \subseteq R \fatsemi R$:

R
$=$ ⟨ Idempotence of $\cap$, Identity of $\fatsemi$ ⟩
$\mathbb{I} \fatsemi R \cap R$
$\subseteq$ ⟨ **Modal rule** $Q \fatsemi R \cap S \subseteq (Q \cap S \fatsemi R^\smile) \fatsemi R)$ ⟩
$(\mathbb{I} \cap R \fatsemi R^\smile) \fatsemi R$
$\subseteq$ ⟨ **Mon. $\fatsemi$ with** Weakening $X \cap Y \subseteq X$ ⟩
$R \fatsemi R^\smile \fatsemi R$
$=$ ⟨ Symmetry of R ⟩
$R \fatsemi R \fatsemi R$
$\subseteq$ ⟨ **Mon. $\fatsemi$ with** Transitivity of R ⟩
$R \fatsemi R$

Modal Rule for "Symmetric and Transitive Implies Idempotent"



$\mathbb{I} \fatsemi R \cap R$
$\subseteq$ ⟨ **Modal rule** $Q \fatsemi R \cap S \subseteq (Q \cap S \fatsemi R^\smile) \fatsemi R)$ ⟩
$(\mathbb{I} \cap R \fatsemi R^\smile) \fatsemi R$



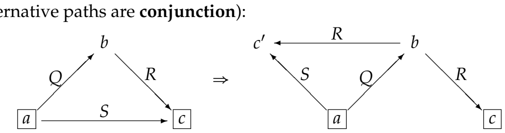
Modal Rules— Converse as Over-Approximation of Inverse

Modal rules: For $Q : \mathcal{A} \leftrightarrow \mathcal{B}$, $R : \mathcal{B} \leftrightarrow \mathcal{C}$, and $S : \mathcal{A} \leftrightarrow \mathcal{C}$:

$$Q \fatsemi R \cap S \subseteq Q \fatsemi (R \cap Q^\smile \fatsemi S)$$
$$Q \fatsemi R \cap S \subseteq (Q \cap S \fatsemi R^\smile) \fatsemi R$$

Useful to "**make information available locally**" (Q is replaced with $Q \cap S \fatsemi R^\smile$) for use in further proof steps.

In **constraint** diagrams (boxed variables are free; others existentially quantified; alternative paths are **conjunction**):



$(\exists b \bullet a \llparenthesis Q \rrparenthesis b \llparenthesis R \rrparenthesis c \wedge a \llparenthesis S \rrparenthesis c) \quad \Rightarrow$
$(\exists b, c' \bullet a \llparenthesis Q \rrparenthesis b \llparenthesis R \rrparenthesis c \wedge b \llparenthesis R \rrparenthesis c' \wedge a \llparenthesis S \rrparenthesis c')$

Modal Rules modulo Inclusion via Intersection

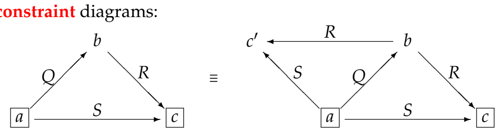
Modal rules: For $Q : \mathcal{A} \leftrightarrow \mathcal{B}$, $R : \mathcal{B} \leftrightarrow \mathcal{C}$, and $S : \mathcal{A} \leftrightarrow \mathcal{C}$:

$$Q \fatsemi R \cap S \subseteq Q \fatsemi (R \cap Q^\smile \fatsemi S)$$
$$Q \fatsemi R \cap S \subseteq (Q \cap S \fatsemi R^\smile) \fatsemi R$$

Equivalently, using $M \subseteq N \equiv M = M \cap N$ etc.:

$$Q \fatsemi R \cap S = Q \fatsemi (R \cap Q^\smile \fatsemi S) \cap S$$
$$Q \fatsemi R \cap S = (Q \cap S \fatsemi R^\smile) \fatsemi R \cap S$$

In **constraint** diagrams:



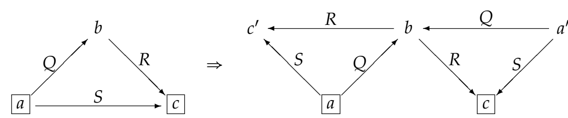
$(\exists b \bullet a \llparenthesis Q \rrparenthesis b \llparenthesis R \rrparenthesis c \wedge a \llparenthesis S \rrparenthesis c) \quad \equiv$
$\equiv \quad (\exists b, c' \bullet a \llparenthesis Q \rrparenthesis b \llparenthesis R \rrparenthesis c' \wedge a \llparenthesis S \rrparenthesis c' \wedge b \llparenthesis R \rrparenthesis c \wedge a \llparenthesis S \rrparenthesis c)$

Modal Rules and Dedekind Rule

Modal rules: For $Q : \mathcal{A} \leftrightarrow \mathcal{B}$, $R : \mathcal{B} \leftrightarrow \mathcal{C}$, and $S : \mathcal{A} \leftrightarrow \mathcal{C}$:

$$Q \fatsemi R \cap S \subseteq Q \fatsemi (R \cap Q^\smile \fatsemi S)$$
$$Q \fatsemi R \cap S \subseteq (Q \cap S \fatsemi R^\smile) \fatsemi R$$

Equivalent: **Dedekind Rule:** $\quad Q \fatsemi R \cap S \subseteq (Q \cap S \fatsemi R^\smile) \fatsemi (R \cap Q^\smile \fatsemi S)$



Dedekind Rule modulo Inclusion via Intersection

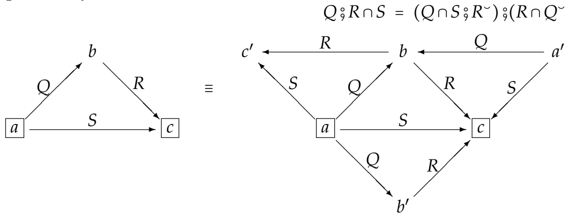
Modal rules: For $Q : \mathcal{A} \leftrightarrow \mathcal{B}$, $R : \mathcal{B} \leftrightarrow \mathcal{C}$, and $S : \mathcal{A} \leftrightarrow \mathcal{C}$:

$$Q \fatsemi R \cap S \subseteq Q \fatsemi (R \cap Q^\smile \fatsemi S)$$
$$Q \fatsemi R \cap S \subseteq (Q \cap S \fatsemi R^\smile) \fatsemi R$$

Equivalent: **Dedekind Rule:** $\quad Q \fatsemi R \cap S \subseteq (Q \cap S \fatsemi R^\smile) \fatsemi (R \cap Q^\smile \fatsemi S)$

Equivalently, via $M \subseteq N \equiv M = M \cap N$:

$$Q \fatsemi R \cap S = (Q \cap S \fatsemi R^\smile) \fatsemi (R \cap Q^\smile \fatsemi S) \cap (S \cap Q \fatsemi R)$$



Modal Rules and Dedekind Rule: Summary with Sharp Versions

For all $Q : \mathcal{A} \leftrightarrow \mathcal{B}$, $R : \mathcal{B} \leftrightarrow \mathcal{C}$, and $S : \mathcal{A} \leftrightarrow \mathcal{C}$:

Modal rules:
$$Q \fatsemi R \cap S \subseteq Q \fatsemi (R \cap Q^\smile \fatsemi S)$$
$$Q \fatsemi R \cap S \subseteq (Q \cap S \fatsemi R^\smile) \fatsemi R$$

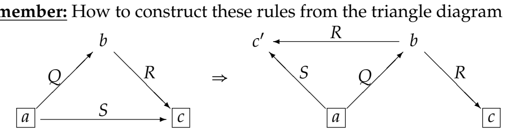
Modal rules (sharp versions):
$$Q \fatsemi R \cap S = Q \fatsemi (R \cap Q^\smile \fatsemi S) \cap S$$
$$Q \fatsemi R \cap S = (Q \cap S \fatsemi R^\smile) \fatsemi R \cap S$$

Dedekind: $\quad Q \fatsemi R \cap S \subseteq (Q \cap S \fatsemi R^\smile) \fatsemi (R \cap Q^\smile \fatsemi S)$

Dedekind (sharp version): $\quad Q \fatsemi R \cap S = (Q \cap S \fatsemi R^\smile) \fatsemi (R \cap Q^\smile \fatsemi S) \cap S$

Proofs: Exercise!

Remember: How to construct these rules from the triangle diagram set-up!



Symmetric and Transitive Implies Idempotent

symmetric	$R^\smile = R$	$(\forall\, b, c : B \bullet b \llparenthesis R \rrparenthesis c \equiv c \llparenthesis R \rrparenthesis b)$
transitive	$R \fatsemi R \subseteq R$	$(\forall b, c, d \bullet b \llparenthesis R \rrparenthesis c \llparenthesis R \rrparenthesis d \Rightarrow b \llparenthesis R \rrparenthesis d)$
idempotent	$R \fatsemi R = R$	

Theorem: A symmetric and transitive $R : B \leftrightarrow B$ is also idempotent.
Proof: By mutual inclusion and transitivity of R, we only need to show $R \subseteq R \fatsemi R$:

R
$=$ ⟨ Idempotence of $\cap$, Identity of $\fatsemi$ ⟩
$R \fatsemi \mathbb{I} \cap R$
$\subseteq$ ⟨ **Modal rule** $Q \fatsemi R \cap S \subseteq Q \fatsemi (R \cap Q^\smile \fatsemi S)$ ⟩
$R \fatsemi (\mathbb{I} \cap R^\smile \fatsemi R)$
$\subseteq$ ⟨ **Mon. $\fatsemi$ with** Weakening $X \cap Y \subseteq X$ ⟩
$R \fatsemi R^\smile \fatsemi R$
$=$ ⟨ Symmetry of R ⟩
$R \fatsemi R \fatsemi R$
$\subseteq$ ⟨ **Mon. $\fatsemi$ with** Transitivity of R ⟩
$R \fatsemi R$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-11-08

Continuing Relation-Algebraic Calculational Proofs

Recall: Relation Algebra

- For any two types B and C, on the type $B \leftrightarrow C$ of **relations between B and C** we have the ordering $\subseteq$ with:
  - binary minima $\_\cap\_$ and maxima $\_\cup\_$ (which are monotonic)
  - least relation $\{\}$ and largest ("universal") relation U (= ⌞ B ⌟ × ⌞ C ⌟)
  - complement operation $\sim\_$ such that $R \cap \sim R = \{\}$ and $R \cup \sim R = U$
  - relative pseudo-complement $R \Rightarrow S \; = \; \sim R \cup S$
- The composition operation $\_\fatsemi\_$
  - is defined on any two relations $R : B \leftrightarrow C_1$ and $S : C_2 \leftrightarrow D$ iff $C_1 = C_2$
  - is associative, monotonic, and has identities $\mathbb{I}$
  - distributes over union: $Q \fatsemi (R \cup S) = Q \fatsemi R \cup Q \fatsemi S$
- The converse operation $\_^{\smile}$
  - maps relation $R : B \leftrightarrow C$ to $R^{\smile} : C \leftrightarrow B$
  - is self-inverse ($R^{\smile\smile} = R$) and monotonic
  - is contravariant wrt. composition: $(R \fatsemi S)^{\smile} = S^{\smile} \fatsemi R^{\smile}$
- The Dedekind rule holds: $Q \fatsemi R \cap S \;\subseteq\; (Q \cap S \fatsemi R^{\smile}) \fatsemi (R \cap Q^{\smile} \fatsemi S)$
- The Schröder equivalences hold:

  $Q \fatsemi R \subseteq S \;\equiv\; Q^{\smile} \fatsemi \sim S \subseteq \sim R$ and $Q \fatsemi R \subseteq S \;\equiv\; \sim S \fatsemi R^{\smile} \subseteq \sim Q$
- $\fatsemi$ has left-residuals $S \,/\, R = \sim(\sim S \fatsemi R^{\smile})$ and right-residuals $Q \setminus S = \sim(Q^{\smile} \fatsemi \sim S)$

Recall: Properties of Homogeneous Relations

reflexive	$\mathbb{I} \subseteq R$	$(\forall\, b : B \bullet b \,(R)\, b)$
irreflexive	$\mathbb{I} \cap R = \{\}$	$(\forall\, b : B \bullet \neg(b \,(R)\, b))$
symmetric	$R^{\smile} = R$	$(\forall\, b, c : B \bullet b \,(R)\, c \equiv c \,(R)\, b)$
antisymmetric	$R \cap R^{\smile} \subseteq \mathbb{I}$	$(\forall\, b, c \bullet b \,(R)\, c \wedge c \,(R)\, b \Rightarrow b = c)$
asymmetric	$R \cap R^{\smile} = \{\}$	$(\forall\, b, c : B \bullet b \,(R)\, c \Rightarrow \neg(c \,(R)\, b))$
transitive	$R \fatsemi R \subseteq R$	$(\forall b, c, d \bullet b \,(R)\, c \wedge c \,(R)\, d \Rightarrow b \,(R)\, d)$

R is an **equivalence (relation) on** B iff it is reflexive, transitive, and symmetric. (E.g., $=$, $\equiv$)

R is a **(partial) order on** B
iff it is reflexive, transitive, and antisymmetric.
(E.g., $\leq$, $\geq$, $\subseteq$, $\supseteq$, $|$)

R is a **strict-order on** B
iff it is irreflexive, transitive, and asymmetric.
(E.g., $<$, $>$, $\subset$, $\supset$)

Recall: Properties of Heterogeneous Relations

A relation $R : B \leftrightarrow C$ is called:

univalent determinate	$R^{\smile} \fatsemi R \subseteq \mathbb{I}$	$\forall\, b, c_1, c_2 \bullet b \,(R)\, c_1 \wedge b \,(R)\, c_2 \Rightarrow c_1 = c_2$
total	$\mathrm{Dom}\, R = B$ $\mathbb{I} \subseteq R \fatsemi R^{\smile}$	$\forall\, b : B \bullet (\exists\, c : C \bullet b \,(R)\, c)$
injective	$R \fatsemi R^{\smile} \subseteq \mathbb{I}$	$\forall\, b_1, b_2, c \bullet b_1 \,(R)\, c \wedge b_2 \,(R)\, c \Rightarrow b_1 = b_2$
surjective	$\mathrm{Ran}\, R = C$ $\mathbb{I} \subseteq R^{\smile} \fatsemi R$	$\forall\, c : C \bullet (\exists\, b : B \bullet b \,(R)\, c)$
a **mapping**	iff it is univalent and total	
bijective	iff it is injective and surjective	

Univalent relations are also called **(partial) functions**.

Mappings are also called **total functions**.

For Univalent Relations, Sub-distributivity turns into Distributivity

If $F : A \leftrightarrow B$ is univalent, then $F \fatsemi (R \cap S) = (F \fatsemi R) \cap (F \fatsemi S)$

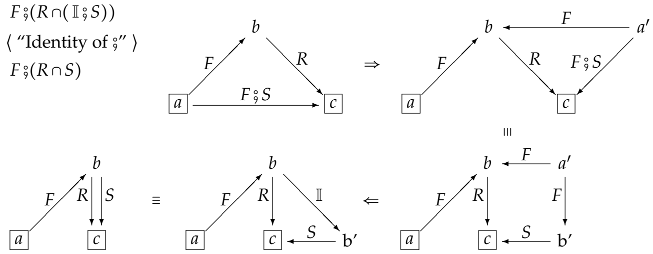
Proof: From sub-distributivity we have $\subseteq$; because of antisymmetry of $\subseteq$ (11.57) we only need to show $\supseteq$:

Assume that F is univalent, that is, $F^{\smile} \fatsemi F \subseteq \mathbb{I}$

$(F \fatsemi R) \cap (F \fatsemi S)$
$\subseteq$ ⟨ **"Modal rule"** $Q \fatsemi R \cap S \;\subseteq\; Q \fatsemi (R \cap Q^{\smile} \fatsemi S)$ ⟩
$F \fatsemi (R \cap (F^{\smile} \fatsemi F \fatsemi S))$
$\subseteq$ ⟨ **"Mon. of $\fatsemi$" with "Mon. of $\cap$" with "Mon. of $\fatsemi$" with** assumption '$F^{\smile} \fatsemi F \subseteq \mathbb{I}$' ⟩
$F \fatsemi (R \cap (\mathbb{I} \fatsemi S))$
$=$ ⟨ "Identity of $\fatsemi$" ⟩
$F \fatsemi (R \cap S)$

Composition with Univalent Distributes over Intersection: In Diagrams

$(F \fatsemi R) \cap (F \fatsemi S)$
$\subseteq$ ⟨ **"Modal rule"** $Q \fatsemi R \cap S \;\subseteq\; Q \fatsemi (R \cap Q^{\smile} \fatsemi S)$ ⟩
$F \fatsemi (R \cap (F^{\smile} \fatsemi F \fatsemi S))$
$\subseteq$ ⟨ **"Mon. of $\fatsemi$" with "Mon. of $\cap$" with "Mon. of $\fatsemi$" with** assumption '$F^{\smile} \fatsemi F \subseteq \mathbb{I}$' ⟩
$F \fatsemi (R \cap (\mathbb{I} \fatsemi S))$
$=$ ⟨ "Identity of $\fatsemi$" ⟩
$F \fatsemi (R \cap S)$



New Keywords: Monotonicity and Antitonicity

If $F : A \leftrightarrow B$ is univalent, then $F \fatsemi (R \cap S) = (F \fatsemi R) \cap (F \fatsemi S)$

Proof: From sub-distributivity we have $\subseteq$; because of antisymmetry of $\subseteq$ (11.57) we only need to show $\supseteq$:

Assume that F is univalent, that is, $F^{\smile} \fatsemi F \subseteq \mathbb{I}$

$(F \fatsemi R) \cap (F \fatsemi S)$
$\subseteq$ ⟨ **"Modal rule"** $Q \fatsemi R \cap S \;\subseteq\; Q \fatsemi (R \cap Q^{\smile} \fatsemi S)$ ⟩
$F \fatsemi (R \cap (F^{\smile} \fatsemi F \fatsemi S))$
$\subseteq$ ⟨ **Monotonicity with** assumption '$F^{\smile} \fatsemi F \subseteq \mathbb{I}$' ⟩
$F \fatsemi (R \cap (\mathbb{I} \fatsemi S))$
$=$ ⟨ "Identity of $\fatsemi$" ⟩
$F \fatsemi (R \cap S)$

Inverses are Defined from Composition and Identities

Definition: Let B and C be types, and $f : B \leftrightarrow C$ be a relation.
An **inverse of** f is a relation $g : C \leftrightarrow B$ such that $f \fatsemi g = \mathbb{I}$ and $g \fatsemi f = \mathbb{I}$.

Theorems:

- f has an inverse iff f is a **bijective mapping**.
- The inverse of a bijective mapping f is its converse $f^{\smile}$.

Note:
"Inverse" should always be defined this way, based on an associative composition with identities.
In such a context, if f has an inverse, it is also called an **isomorphism**.
(Ad-hoc "definitions of inverse" produce a moral proof obligation of the inverse properties.
Without these, one runs the risk of inducing strange theories...)

In particular: Converse of relations does **in general not** produce inverses.

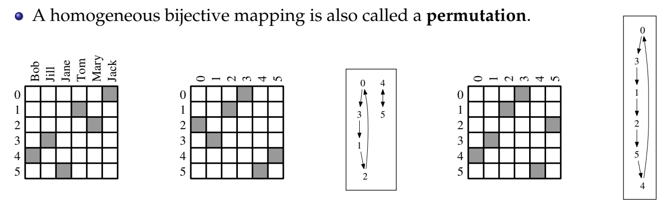
Inverses of Total Functions — Between Sets

We write "$f \in S_1 \twoheadrightarrow S_2$" for "$f$ is a mapping fron S_1 to S_2" — $\mathrm{Dom}\, f = S_1 \;\wedge\; f^{\smile} \fatsemi f \subseteq \mathrm{id}\; S_2$

(14.43) **Definition:** Let f with $f \in S_1 \twoheadrightarrow S_2$ be a **mapping** from S_1 to S_2.
An **inverse of** f is a mapping g from S_2 to S_1 such that $f \fatsemi g = \mathrm{id}\; S_1$ and $g \fatsemi f = \mathrm{id}\; S_2$.

Still:

- f has an inverse iff f is a bijective mapping.
- The inverse of a bijective mapping f is its converse $f^{\smile}$.
- A homogeneous bijective mapping is also called a **permutation**.

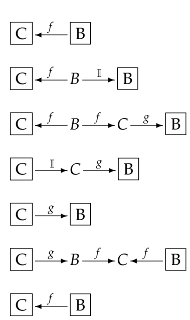


Inverses of Total Functions — Between Types

(14.43t) **Definition:** Let B and C be types, and $f : B \leftrightarrow C$ be a **mapping**.
An **inverse of** f is a mapping $g : C \leftrightarrow B$ such that $f \fatsemi g = \mathbb{I} = \mathrm{id}$ ⌞ B ⌟ and $g \fatsemi f = \mathbb{I} = \mathrm{id}$ ⌞ C ⌟.

Theorem: If g is an inverse of a mapping $f : B \to C$, then $g = f^{\smile}$.
Proof: (Using antisymmetry of $\subseteq$)

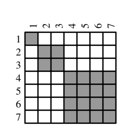
$f^{\smile}$
$=$ ⟨ Identity of $\fatsemi$ ⟩
$f^{\smile} \fatsemi \mathbb{I}$
$=$ ⟨ g is an inverse of f ⟩
$f^{\smile} \fatsemi f \fatsemi g$
$\subseteq$ ⟨ **Mon. of $\fatsemi$ with** f is univalent, that is, $f^{\smile} \fatsemi f \subseteq \mathbb{I}$ ⟩
$\mathbb{I} \fatsemi g$
$=$ ⟨ Identity of $\fatsemi$ ⟩
g
$\subseteq$ ⟨ Identity of $\fatsemi$, **Mon. of $\fatsemi$ with** f is total, that is, $\mathbb{I} \subseteq f \fatsemi f^{\smile}$ ⟩
$g \fatsemi f \fatsemi f^{\smile}$
$=$ ⟨ g is an inverse of f; Identity of $\fatsemi$ ⟩
$f^{\smile}$



Recall: Equivalence Relations

Recall: A (homogeneous) relation $R : B \leftrightarrow B$ is called:

reflexive	$\mathbb{I} \subseteq R$	$(\forall\, b : B \bullet b \,(R)\, b)$
symmetric	$R^{\smile} = R$	$(\forall\, b, c : B \bullet b \,(R)\, c \equiv c \,(R)\, b)$
transitive	$R \fatsemi R \subseteq R$	$(\forall b, c, d \bullet b \,(R)\, c \,(R)\, d \Rightarrow b \,(R)\, d)$
idempotent	$R \fatsemi R = R$	
equivalence	$\mathbb{I} \subseteq R = R \fatsemi R = R^{\smile}$	reflexive, transitive, symmetric



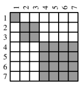
Equivalence Classes, Partitions

Definition (14.34): Let Ξ be an equivalence relation on B. Then $[b]_\Xi$, the **equivalence class of** b, is the subset of elements of B that are equivalent (under Ξ) to b:

$$x \in [b]_\Xi \quad \equiv \quad x \,(\Xi)\, b \qquad \text{Equivalently:} \qquad [b]_\Xi \;=\; \Xi\,(\!|\,\{b\}\,|\!)$$

Theorem: For an equivalence relation Ξ on B, the set $B|_\Xi = \{\, b : B \bullet \Xi\,(\!|\,\{b\}\,|\!)\,\}$ of equivalence classes of Ξ is a partition of $\llcorner\, B \,\lrcorner$.

$$\{\, \{1\},\ \{2,3\},\ \{4,5,6,7\} \,\}$$



Definition (11.76): If T : **set** t and S : **set** (**set** t), then:

S is a **partition of** T

$\equiv (\forall u, v \mid u \in S \wedge v \in S \wedge u \neq v \bullet u \cap v = \{\})$

$\wedge\ (\bigcup u \mid u \in S \bullet u) = T$

Theorem: There is a bijective mapping between equivalence relations on B and partitions of B.

The partition view can be useful for **implementing** equivalence relations.

Equivalence Quotients

For an equivalence relation Ξ on B, the set $B|_\Xi = \{\, b : B \bullet [b]_\Xi \,\}$ of equivalence classes of Ξ is also called **quotient of** B **via** Ξ.

The mapping $\chi = \{\, b \bullet \langle b, [b]_\Xi \rangle \,\}$ is the **quotient projection**.

χ satisfies:

- $\chi^\smile ⨾ \chi = \mathbb{I}$ — univalent and surjective
- $\chi ⨾ \chi^\smile = \Xi$ — therefore total, since Ξ is reflexive

The quotient together with the quotient projection is **determined uniquely up to isomorphism** by these two properties:

Let C be an "alternate quotient set candidate",
with $\gamma : B \leftrightarrow C$ satisfying $\gamma^\smile ⨾ \gamma = \mathbb{I}$ and $\gamma ⨾ \gamma^\smile = \Xi$.

Then $\varphi = \chi^\smile ⨾ \gamma$ is an isomorphism between $B|_\Xi$ and C:

- $\varphi ⨾ \varphi^\smile = \chi^\smile ⨾ \gamma ⨾ \gamma^\smile ⨾ \chi = \chi^\smile ⨾ \Xi ⨾ \chi = \chi^\smile ⨾ \chi ⨾ \chi^\smile ⨾ \chi = \mathbb{I} ⨾ \mathbb{I} = \mathbb{I}$ — total and injective
- $\varphi^\smile ⨾ \varphi = \gamma^\smile ⨾ \chi ⨾ \chi^\smile ⨾ \gamma = \gamma^\smile ⨾ \Xi ⨾ \gamma = \gamma^\smile ⨾ \gamma ⨾ \gamma^\smile ⨾ \gamma = \mathbb{I} ⨾ \mathbb{I} = \mathbb{I}$ — univalent and surjective

M1(A, B) Notes

- M1.1a) Only one induction needed for:
  Theorem "Minimum with addition": k ↓ (k + n) = k
  Theorem "Maximum with addition": k ↑ (k + n) = k + n
- M1.1b) Two inductions needed for:
  Theorem "At most via maximum": k ≤ n ⇒ k ↑ n = n
  Theorem "At most via minimum": k ≤ n ⇒ k ↓ n = k
- M1.1c) Three inductions needed, plus using M1.1b) in the right way— tricky!
  Congratulations to those who found checkable proofs for that, without proof checking!
- M1.2a) Familiarity with "∃-Introduction" is expected.
  Quantification has lowest precedence: $(\exists\, x \bullet E = F) = (\exists\, x \bullet (E = F))$
- M1.2b–d) Routine with correctness proofs is expected —
  we started these in Week 2 Homework 4.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-11-10

Reachability Concepts in (Simple) Graphs, Closures

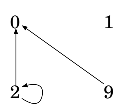
Recall: Simple Graphs

A **simple graph** (N, E) is a pair consisting of

- a set N, the elements of which are called "nodes", and
- a relation E with $E \in N \leftrightarrow N$, the element pairs of which are called "edges".

Example: $G_1 = (\{2,0,1,9\}, \{\langle 2,0\rangle, \langle 9,0\rangle, \langle 2,2\rangle\})$

Graphs are normally visualised via **graph drawings**:

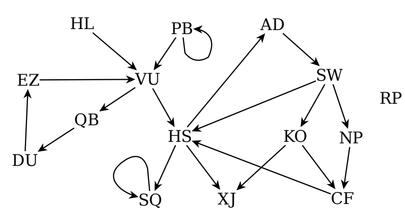


Simple graphs are exactly relations!

Reasoning with relations is reasoning about graphs!

Simple Reachability Statements in Graph $G = (V, E)$

- No edge ends at node s
  $s \notin Ran\ E$ or $s \in {\sim}(Ran\ E)$ — s is called a **source** of G
- No edge starts at node s
  $s \notin Dom\ E$ or $s \in {\sim}(Dom\ E)$ — s is called a **sink** of G
- Node n_2 is reachable from node n_1 via a three-edge path
  $n_1 \,(E ⨾ E ⨾ E)\, n_2$



Simple Reachability Statements in Graph $G_\mathbb{N} = (\llcorner\, \mathbb{N} \,\lrcorner, \ulcorner suc \urcorner)$

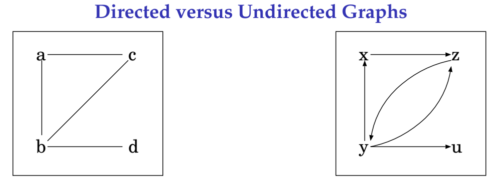
- No edge ends at node 0
  $0 \notin Ran\ \ulcorner suc \urcorner$ or $0 \in {\sim}(Ran\ \ulcorner suc \urcorner)$ — 0 is a **source** of $G_\mathbb{N}$

  0 is the only source of $G_\mathbb{N}$: ${\sim}(Ran\ \ulcorner suc \urcorner) = \{0\}$
- s is a sink iff no edge starts at node s
  $s \notin Dom\ \ulcorner suc \urcorner$ or $s \in {\sim}(Dom\ \ulcorner suc \urcorner)$

  $G_\mathbb{N}$ has no sinks: $Dom\ \ulcorner suc \urcorner = \llcorner\, \mathbb{N} \,\lrcorner$ or ${\sim}(Dom\ \ulcorner suc \urcorner) = \{\}$
- Node 5 is reachable from node 2 via a three-edge path:
  $2 \,(\ulcorner suc \urcorner ⨾ \ulcorner suc \urcorner ⨾ \ulcorner suc \urcorner)\, 5$

$$0 \longrightarrow 1 \longrightarrow 2 \longrightarrow 3 \longrightarrow 4 \longrightarrow 5 \longrightarrow 6 \longrightarrow 7 \longrightarrow \ldots$$

Directed versus Undirected Graphs



- Edges in simple undirected graphs can be considered as "unordered pairs" (two-element sets, or one-to-two-element sets)
- The **associated relation** of an undirected graph relates two nodes iff there is an edge between them
- **The associated relation of an undirected graph is always symmetric**
- In a **simple** graph, no two edges have the same source and the same target. (No "parallel edges".)
- Relations directly represent simple **directed** graphs.

Symmetric Closure

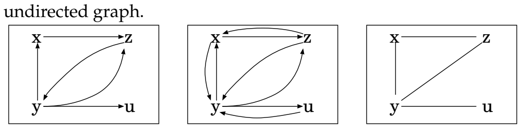
Relation $Q : B \leftrightarrow B$ is the **symmetric closure** of $R : B \leftrightarrow B$ iff Q is the smallest symmetric relation containing R,

or, equivalently, iff

- $R \subseteq Q$
- $Q = Q^\smile$
- $(\forall\, P : B \leftrightarrow B \mid R \subseteq P = P^\smile \bullet Q \subseteq P)$

Theorem: The symmetric closure of $R : B \leftrightarrow B$ is $R \cup R^\smile$.

Fact: If R represents a simple directed graph, then the symmetric closure of R is the associated relation of the corresponding simple undirected graph.



Reflexive Closure

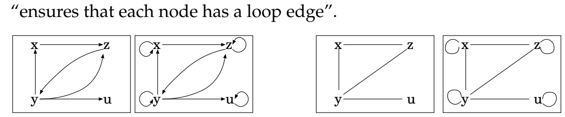
Relation $Q : B \leftrightarrow B$ is the **reflexive closure** of $R : B \leftrightarrow B$ iff Q is the smallest reflexive relation containing R,

or, equivalently, iff

- $R \subseteq Q$
- $\mathbb{I} \subseteq Q$
- $(\forall P : B \leftrightarrow B \mid R \subseteq P \wedge \mathbb{I} \subseteq P \bullet Q \subseteq P)$

Theorem: The reflexive closure of $R : B \leftrightarrow B$ is $R \cup \mathbb{I}$.

Fact: If R represents a graph, then the reflexive closure of R "ensures that each node has a loop edge".



Transitive Closure

Relation $Q : B \leftrightarrow B$ is the **transitive closure** of $R : B \leftrightarrow B$
iff Q is the smallest transitive relation containing R,

or, equivalently, iff

- $R \subseteq Q$
- $Q ⨾ Q \subseteq Q$
- $(\forall P : B \leftrightarrow B \mid R \subseteq P \wedge P ⨾ P \subseteq P \bullet Q \subseteq P)$

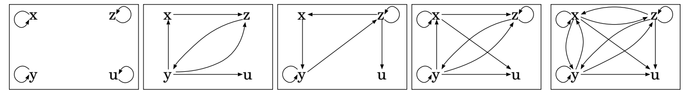
Definition: The transitive closure of $R : B \leftrightarrow B$ is written R^+.

Theorem: $R^+ = (\bigcap P \mid R \subseteq P \wedge P ⨾ P \subseteq P \bullet P)$.

Transitive Closure via Powers

Powers of a homogeneous relation $R : B \leftrightarrow B$:

- $R^0 = \mathbb{I}$
- $R^1 = R$
- $R^{n+1} = R^n ⨾ R$
- $R^2 = R ⨾ R$
- $R^3 = R ⨾ R ⨾ R$
- $R^4 = R ⨾ R ⨾ R ⨾ R$
- R^i is reachability via exactly i many R-steps



R^0 R^1 R^2 R^3 R^+

Theorem: $R^+ = (\bigcup i : \mathbb{N} \mid i > 0 \bullet R^i)$

This means:

- $R^+ = R \cup R^2 \cup R^3 \cup R^4 \cup \ldots$
- Transitive closure R^+ is reachability via at least one R-step

Reflexive Transitive Closure

$Q : B \leftrightarrow B$ is the **reflexive transitive closure** of $R : B \leftrightarrow B$
iff Q is the smallest reflexive transitive relation containing R,

or, equivalently, iff

- $R \subseteq Q$
- $\mathbb{I} \subseteq Q \wedge Q ⨾ Q \subseteq Q$
- $(\forall P : B \leftrightarrow B \mid R \subseteq P \wedge \mathbb{I} \subseteq P \wedge P ⨾ P \subseteq P \bullet Q \subseteq P)$

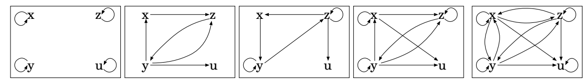
Definition: The reflexive transitive closure of R is written R^*.

Theorem: $R^* = (\bigcap P \mid R \subseteq P \wedge \mathbb{I} \subseteq P \wedge P ⨾ P \subseteq P \bullet P)$.

Theorem: $R^* = (\bigcup i : \mathbb{N} \bullet R^i)$

Transitive and Reflexive Transitive Closure via Powers

- R^i is reachability via exactly i many R-steps



R^0 R^1 R^2 R^3 R^*

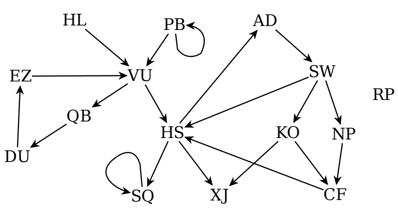
- $R^+ = (\bigcup i : \mathbb{N} \mid i > 0 \bullet R^i)$
- $R^+ = R \cup R^2 \cup R^3 \cup R^4 \cup \ldots$
- Transitive closure R^+ is reachability via at least one R-step

- $R^* = (\bigcup i : \mathbb{N} \bullet R^i)$
- $R^* = \mathbb{I} \cup R \cup R^2 \cup R^3 \cup R^4 \cup \ldots$
- Reflexive transitive closure R^* is reachability via any number of R-steps

- Variants of the **Warshall algorithm** calculate these closures in cubic time.

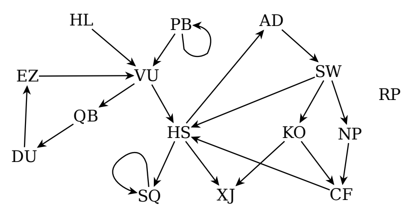
Reachability in graph $G = (V, E)$ — 1 (ctd.)

- No edge ends at node s
  $s \notin Ran\ E$ or $s \in \sim(Ran\ E)$ — s is called a **source** of G
- No edge starts at node s
  $s \notin Dom\ E$ or $s \in \sim(Dom\ E)$ — s is called a **sink** of G
- Node n_2 is reachable from node n_1 via a three-edge path
  $n_1 \,(\!| E^3 |\!)\, n_2$ or $n_1 \,(\!| E ⨾ E ⨾ E |\!)\, n_2$
- Node y is **reachable** from node x
  $x \,(\!| E^* |\!)\, y$ — **reachability**



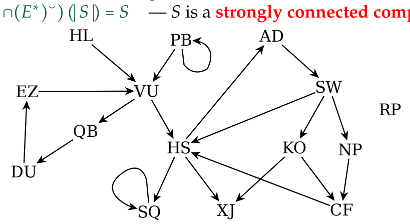
Reachability in graph $G = (V, E)$ — 2

- Node y is **reachable** from node x
  $x \,(\!| E^* |\!)\, y$ — **reachability**
- Every node is reachable from node r
  $\{r\} \times V \subseteq E^*$ or $E^* (\!| \{r\} |\!) = V$ — r is called a **root** of G
- Node y is **reachable via a non-empty path** from node x: $x \,(\!| E^+ |\!)\, y$
- Nodes x lies on a cycle: $x \,(\!| E^+ |\!)\, x$ or $x \,(\!| E^+ \cap \mathbb{I} |\!)\, x$ or $x \in Dom(E^+ \cap \mathbb{I})$



Reachability in graph $G = (V, E)$ — 3

- From every node, each node is reachable
  $V \times V \subseteq E^*$ — G is **strongly connected**
- From every node, each node is reachable by traversing edges in either direction
  $V \times V \subseteq (E \cup E^\smile)^*$ — G is **connected**
- Nodes n_1 and n_2 reachable from each other both ways
  $n_1 \,(\!| E^* \cap (E^*)^\smile |\!)\, n_2$ — n_1 and n_2 are **strongly connected**
- S is an equivalence class of strong connectedness between nodes
  $S \times S \subseteq E^* \wedge (E^* \cap (E^*)^\smile) (\!| S |\!) = S$ — S is a **strongly connected component (SCC)** of G

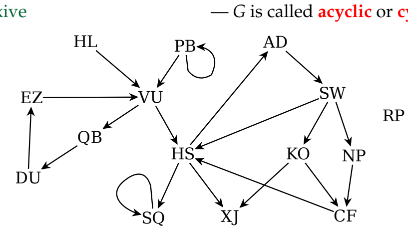


Reachability in graph $G = (V, E)$ — 4

- A node n is said to "lie on a cycle" if there is a non-empty path from n to n

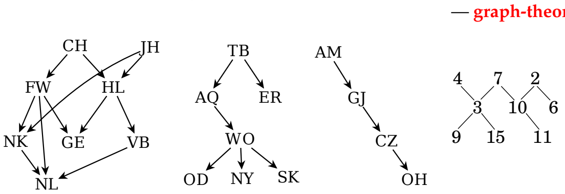
  $cycleNodes := Dom(E^+ \cap \mathbb{I})$

- No node lies on a cycle
  $Dom(E^+ \cap \mathbb{I}) = \{\}$
  $E^+ \cap \mathbb{I} = \{\}$
  E^+ is irreflexive — G is called **acyclic** or **cycle-free** or a **DAG**



Reachability in graph $G = (V, E)$ — 5 — DAGs

- No node lies on a cycle: $E^+ \cap \mathbb{I} = \{\}$ — G is a **directed acyclic graph**, or **DAG**
- Each node has at most one predecessor: $E ⨾ E^\smile \subseteq \mathbb{I}$ or E is injective
  — if G is also acyclic, then G is called a **(directed) forest**
- Every node is reachable from node r
  $\{r\} \times V \subseteq E^*$ — if G is also a forest, then G is called a **(directed) tree**, and r is its **root**
- For undirected graphs: A tree is a graph where for each pair of nodes there is exactly one path connecting them.
  — **graph-theoretic tree concept**



Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-11-10

Part 2: Closures Generalised

Recall: Reflexive Closure

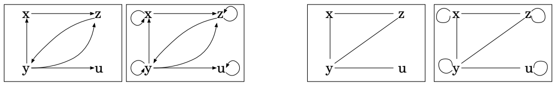
Relation $Q : B \leftrightarrow B$ is the **reflexive closure** of $R : B \leftrightarrow B$ iff Q is the smallest reflexive relation containing R,

or, equivalently, iff

- $R \subseteq Q$
- $\mathbb{I} \subseteq Q$
- $(\forall P : B \leftrightarrow B \mid R \subseteq P \wedge \mathbb{I} \subseteq P \bullet Q \subseteq P)$

Theorem: The reflexive closure of $R : B \leftrightarrow B$ is $R \cup \mathbb{I}$.

Fact: If R represents a graph, then the reflexive closure of R "ensures that each node has a loop edge".



Reflexive Closure Operator `reflClos` (in Ref9.4)

Axiom "Definition of `reflClos`": reflClos R = $R \cup \mathbb{I}$

Theorem "Closure properties of `reflClos`: Expanding":
$R \subseteq$ reflClos R
Proof:
?

Theorem "Closure properties of `reflClos`: Reflexivity":
reflexive (reflClos R)
Proof:
?

Theorem "Closure properties of `reflClos`: Minimality":
$R \subseteq S \wedge$ reflexive $S \Rightarrow$ reflClos $R \subseteq S$
Proof:
?

Relation $Q : B \leftrightarrow B$ is the **reflexive closure** of $R : B \leftrightarrow B$ iff Q is the smallest reflexive relation containing R, or, equivalently, iff

- $R \subseteq Q$
- $\mathbb{I} \subseteq Q$
- $(\forall P : B \leftrightarrow B \mid R \subseteq P \wedge \mathbb{I} \subseteq P \bullet Q \subseteq P)$

Closures

Let *pred* (for "predicate") be a property on relations, i.e., for some type B and C:

$pred \quad : \quad (B \leftrightarrow C) \to \mathbb{B}$

Relation $Q : B \leftrightarrow C$ is the ***pred*-closure** of $R : B \leftrightarrow C$ iff

- Q is the smallest relation
- that contains R
- and has property *pred*

or, equivalently, iff

- $R \subseteq Q$
- $pred\ Q$
- $(\forall P : B \leftrightarrow C \mid R \subseteq P \wedge pred\ P \bullet Q \subseteq P)$

Relation $Q : B \leftrightarrow B$ is the **reflexive closure** of $R : B \leftrightarrow B$ iff Q is the smallest reflexive relation containing R, or, equivalently, iff

- $R \subseteq Q$
- $\mathbb{I} \subseteq Q$
- $(\forall P : B \leftrightarrow B \mid R \subseteq P \wedge \mathbb{I} \subseteq P \bullet Q \subseteq P)$

(For some properties, closures are not defined, or not always defined.)

Formalising General Relation Closures

Let *pred* (for "predicate") be a property on relations, i.e.: $pred \ : \ (B \leftrightarrow C) \to \mathbb{B}$

Relation $Q : B \leftrightarrow C$ is the ***pred*-closure** of $R : B \leftrightarrow C$ iff

- Q is the smallest relation that contains R and has property *pred*,

or, equivalently, iff

- $R \subseteq Q$ and $pred\ Q$ and $(\forall P : B \leftrightarrow C \mid R \subseteq P \wedge pred\ P \bullet Q \subseteq P)$

General Relation Closures in Ref9.4:

Precedence 50 **for:** _is_closure – of_
Conjunctional: _is_closure – of_
Declaration: _is_closure – of_ :
$(A \leftrightarrow B) \to ((A \leftrightarrow B) \to \mathbb{B}) \to (A \leftrightarrow B) \to \mathbb{B}$

Axiom "Relation closure":
Q **is** *pred* closure-of R
$\equiv R \subseteq Q \wedge pred\ Q \wedge (\forall P \bullet R \subseteq P \wedge pred\ P \Rightarrow Q \subseteq P)$

Theorem "Well-definedness of `reflClos`":

Declaration: _is_closure – of_ :
$(A \leftrightarrow B) \to ((A \leftrightarrow B) \to \mathbb{B}) \to (A \leftrightarrow B) \to \mathbb{B}$

Axiom "Relation closure":
Q **is** *pred* closure-of R
$\equiv R \subseteq Q \wedge pred\ Q \wedge (\forall P \bullet R \subseteq P \wedge pred\ P \Rightarrow Q \subseteq P)$

Theorem "Well-definedness of `reflClos`":
reflClos R **is** reflexive closure-of R
Proof:
By "Relation closure"
with "Closure properties of `reflClos`: Expanding"
and "Closure properties of `reflClos`: Reflexivity"
and "Closure properties of `reflClos`: Minimality"

Theorem "Well-definedness of `reflClos`":

Declaration: _is_closure – of_ :
$(A \leftrightarrow B) \to ((A \leftrightarrow B) \to \mathbb{B}) \to (A \leftrightarrow B) \to \mathbb{B}$

Axiom "Relation closure":
Q **is** *pred* closure-of R
$\equiv R \subseteq Q \wedge pred\ Q \wedge (\forall P \bullet R \subseteq P \wedge pred\ P \Rightarrow Q \subseteq P)$

Theorem "Well-definedness of `reflClos`":
reflClos R **is** reflexive closure-of R
Proof:
Using "Relation closure":
Subproof for `$R \subseteq$ reflClos $R$`:
?
Subproof for `reflexive (reflClos $R$)`:
?
Subproof for `$\forall P \bullet R \subseteq P \wedge$ reflexive $P \Rightarrow$ reflClos $R \subseteq P$`:
For any `$P$`:
Assuming `$R \subseteq P$`, `reflexive $P$`:
?

Reachability

Let a directed graph $G = (V, E)$ with vertex/node set V and edge relation E (with $E \in V \leftrightarrow V$) be given.

Formalise via relation-algebraic expressions, and name the concepts:

- No edge ends at node s
- No edge starts at node s
- Node t is reachable from node s
- From every node, each node is reachable
- Each node in the vertex set S (with $S \in \mathbb{P}\ V$) is reachable from every node in S
- No node lies on a cycle
- Each node has at most one predecessor
- Every node is reachable from node r

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-11-13

Kleene Algebra, Arrays

Reminder: Limitations of Conditional Rewriting Implementation of `with`$_2$

- If *ThmA* gives rise to an implication $A_1 \Rightarrow A_2 \Rightarrow \ldots (L = R)$:
  - Find substitution σ such that $L\sigma$ matches goal
  - Resolve $A_1\sigma, A_2\sigma, \ldots$ using *ThmB* and $ThmB_2 \ldots$
  - Rewrite goal applying $L\sigma \mapsto R\sigma$ rigidly.

ThmA `with` *ThmB* and $ThmB_2 \ldots$

- E.g.: "Transitivity of $\subseteq$" with Assumptions `$Q \cap S \subseteq Q$` and `$Q \subseteq R$` when trying to prove `$Q \cap S \subseteq R$`
  - "Transitivity of $\subseteq$" is: $Q \subseteq R \Rightarrow R \subseteq S \Rightarrow Q \subseteq S$
  - For application, a **fresh renaming** is used: $q \subseteq r \Rightarrow r \subseteq s \Rightarrow q \subseteq s$
  - We try to use: $q \subseteq s \mapsto true$, so L is: $q \subseteq s$
  - Matching L against goal produces $\sigma = [q, s := Q \cap S, R]$
  - $(q \subseteq r)\sigma$ is $(Q \cap S \subseteq r)$, and $(r \subseteq s)\sigma$ is $r \subseteq R$
    — which cannot be proven by "Assumption '$Q \cap S \subseteq Q$'"
    resp. by "Assumption '$Q \subseteq R$'"
  - *Narrowing* or *unification* would be needed for such cases
    — not yet implemented
  - Adding an explicit substitution should help:
    "Transitivity of $\subseteq$" with `$R := Q$` and assumption `$Q \cap S \subseteq Q$` and assumption `$Q \subseteq R$`

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-11-13

Part 1: Kleene Algebra

Recall: Reflexive Transitive Closure

$Q : B \leftrightarrow B$ is the **reflexive transitive closure** of $R : B \leftrightarrow B$
iff Q is the smallest reflexive transitive relation containing R,

or, equivalently, iff

- $R \subseteq Q$
- $\mathbb{I} \subseteq Q \wedge Q \fatsemi Q \subseteq Q$
- $(\forall P : B \leftrightarrow B \mid R \subseteq P \wedge \mathbb{I} \subseteq P \wedge P \fatsemi P \subseteq P \bullet Q \subseteq P)$

Definition: The reflexive transitive closure of R is written R^*.

Theorem: $R^* = (\bigcap P \mid R \subseteq P \wedge \mathbb{I} \subseteq P \wedge P \fatsemi P \subseteq P \bullet P)$.

Theorem: $R^* = (\bigcup i : \mathbb{N} \bullet R^i)$

- R^i is reachability via exactly i many R-steps
- Reflexive transitive closure R^* is reachability via any number of R-steps
- Transitive closure $R^+ = (\bigcup i : \mathbb{N} \mid i > 0 \bullet R^i)$ is reachability via at least one R-step

Kleene Algebra

The transitive and reflexive-transitive closure operators satisfy many useful algebraic properties, e.g.:

- $(R^*)^{\smile} = (R^{\smile})^*$ $\qquad (R^+)^{\smile} = (R^{\smile})^+$
- $R^* = \mathbb{I} \cup R \cup R^* \fatsemi R^*$
- $(R \cup S)^* = (R^* \fatsemi S)^* \fatsemi R^*$
- $(R \cup S)^+ = R^+ \cup (R^* \fatsemi S)^+ \fatsemi R^*$
- $R^* \cup S^* \subseteq (R \cup S)^*$

On can prove such properties via reasoning about arbitrary unions $\bigcup$ of relation powers…

One can also derive these properties from a simple axiomatisations (Ex10.2, Ref10.1):

Axiom (KA.1) "Definition of *": $R^* = \mathbb{I} \cup R \cup R^* \fatsemi R^*$
Axiom (KA.2) "Left-induction for *": $R \fatsemi S \subseteq S \Rightarrow R^* \fatsemi S \subseteq S$
Axiom (KA.3) "Right-induction for *": $Q \fatsemi R \subseteq Q \Rightarrow Q \fatsemi R^* \subseteq Q$

Axiom (KA.4) "Definition of $^+$": $R^+ = R \fatsemi R^*$

Kleene Algebra — Example for Using the Induction Axioms

"Left-ind. *": $R \fatsemi S \subseteq S \Rightarrow R^* \fatsemi S \subseteq S$ "Right-ind. *": $Q \fatsemi R \subseteq Q \Rightarrow Q \fatsemi R^* \subseteq Q$

Theorem (KA.14) "Shuffle *": $R \fatsemi R^* = R^* \fatsemi R$
Proof:

$R \fatsemi R^*$
$\subseteq$ ⟨ "Identity of ⨾", "Monotonicity of ⨾" with "Reflexivity of *" ⟩
$R^* \fatsemi R \fatsemi R^*$
$\subseteq$ ⟨ "Right-induction for *" with \`$Q := R^* \fatsemi R$\` and subproof:
  $R^* \fatsemi R \fatsemi R$
  $\subseteq$ ⟨ Monotonicity with "* increases", "⨾-idempotency of *" ⟩
  $R^* \fatsemi R$
⟩
$R^* \fatsemi R$
$\subseteq$ ⟨ "Identity of ⨾", "Monotonicity of ⨾" with "Reflexivity of *" ⟩
$R^* \fatsemi R \fatsemi R^*$
$\subseteq$ ⟨ "Left-induction for *" with \`$S := R \fatsemi R^*$\` and subproof:
  $R \fatsemi R \fatsemi R^*$
  $\subseteq$ ⟨ Monotonicity with "* increases", "⨾-idempotency of *" ⟩
  $R \fatsemi R^*$
⟩
$R \fatsemi R^*$

Kleene Algebra — Not Only Relations: Formal Languages

Definition: A **word** over "alphabet" A is a sequence of elements of A.

Definition: A **formal language** over "alphabet" A is a set of words over A.

Interpret:

- $\mathbb{I}$ as the language containing only the empty word
- $\cup$ as language union
- ⨾ as **language concatenation**: $L_1 \fatsemi L_2 = \{ u, v \mid u \in L_1 \wedge v \in L_2 \bullet u \frown v \}$
- $\_^*$ as **language iteration**: $L^* = (\bigcup i : \mathbb{N} \bullet L^i)$

Then:

- Formal languages over A form a Kleene algebra.
- Regular languages over A form a Kleene algebra.

  (A regular language is generated by a regular grammar, and accepted by a finite automaton.)

Kleene Algebra — Not Only Relations: Control Flow Semantics

Definition: A **trace** is a sequence of commands,

Interpret:

- $\mathbb{I}$ as the singleton trace set containing the empty trace
- $\cup$ as trace set union
- ⨾ as trace set concatenation
- $\_^*$ as trace set iteration

Then:

- Kleene algebra can be used for reasoning about traces (possible executions) of imperative programs
- Kleene algebra provides semantics for control flow

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-11-13

Part 2: Programming with Arrays

⟹ Exercise 10.3

Modelling Arrays as Partial Functions

Precedence 100 **for:** $\_ \nrightarrow \_$
Associating to the right: $\_ \nrightarrow \_$
Declaration: $\_ \nrightarrow \_ : \mathsf{set}\ A \to \mathsf{set}\ B \to \mathsf{set}\ (A \leftrightarrow B)$ — type "\tfun" for $\nrightarrow$

Axiom "Definition of $\nrightarrow$":
$X \nrightarrow Y = \{ f \mid f^{\smile} \fatsemi f \subseteq \mathsf{id}\ Y \wedge \mathsf{Dom}\ f = X \}$

Useful for the domain of arrays:

Precedence 100 **for:** _.._
Non-associating: _.._
Declaration: $\_..\_ : \mathbb{N} \to \mathbb{N} \to \mathsf{set}\ \mathbb{N}$ ▪▪▪▪▪▪ type: \..
Axiom "Definition of ..": $m .. n = \{ i \mid m \le i \le n \}$
Theorem "Membership in ..": $i \in m .. n \equiv m \le i \le n$
Theorem "Membership in 0 ..": $i \in 0 .. n \equiv i \le n$

Array access: `a[i]` ⟹ $a\ @\ i$
Array update: `a[i] := E` ⟹ $a := a \oplus \{ \langle i, E \rangle \}$

Swapping Two Elements of an Array: Specification

$i \le k \ge j \wedge \mathsf{xs} = xs_0 \in (0 .. k) \nrightarrow \llcorner \mathbb{N} \lrcorner$
$\Rightarrow$[
  Swap
]
$\mathsf{xs} = xs_0 \oplus \{ \langle i, xs_0\ @\ j \rangle, \langle j, xs_0\ @\ i \rangle \}$

Swapping Two Elements of an Array: Implementation

```
z := xs[i] ;
xs[i] := xs[j] ;
xs[j] := z
```

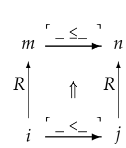
Theorem "Array swap":
$i \le k \ge j \wedge \mathsf{xs} = xs_0 \in (0 .. k) \nrightarrow \llcorner \mathbb{N} \lrcorner$
$\Rightarrow$[$z := \mathsf{xs}\ @\ i$;
  $\mathsf{xs} := \mathsf{xs} \oplus \{ \langle i, \mathsf{xs}\ @\ j \rangle \}$;
  $\mathsf{xs} := \mathsf{xs} \oplus \{ \langle j, z \rangle \}$
]
$\mathsf{xs} = xs_0 \oplus \{ \langle i, xs_0\ @\ j \rangle, \langle j, xs_0\ @\ i \rangle \}$

Sortedness

Declaration: $\mathsf{sorted} : (\mathbb{N} \leftrightarrow \mathbb{N}) \to \mathbb{B}$
Axiom "Definition of \`sorted\`":
$\mathsf{sorted}\ R \equiv R^{\smile} \fatsemi \ulcorner \_<\_ \urcorner \fatsemi R \subseteq \ulcorner \_\le\_ \urcorner$

Note: No assumption that R is univalent or contiguous!

Theorem "Sortedness":
$\mathsf{sorted}\ R \equiv \forall i \bullet \forall j \mid i < j \bullet \forall m \bullet \forall n \mid i \,(\, R \,)\, m \wedge j \,(\, R \,)\, n \bullet m \le n$



Specification of Sorting — First Attempt

$\mathsf{xs} \in (0\,..\,k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner$
$\Rightarrow[\ \mathsf{SORT}$
$]$
$\mathsf{xs} \in (0\,..\,k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner \quad \wedge \quad \mathsf{sorted\ xs}$

Theorem "Sorting 0":

$\mathsf{xs} \in (0\,..\,k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner$
$\Rightarrow[\ p := 0\,;$
$\quad \mathsf{while}\ p \neq k + 1\ \mathsf{do}$
$\qquad \mathsf{xs} := \mathsf{xs} \oplus \{\langle p, 42\rangle\}\,;$
$\qquad p := p + 1$
$\quad \mathsf{od}$
$]$
$\mathsf{xs} \in (0\,..\,k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner \ \wedge\ \mathsf{sorted\ xs}$

Proof:

$\mathsf{xs} \in (0\,..\,k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner$
$\Rightarrow\langle\ ?\ \rangle$
$\mathsf{xs} \in (0\,..\,k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner \ \wedge\ \mathsf{Ran}\,((0\,..\,0) \lhd \mathsf{xs}) = \{\mathsf{xs}\,@\,0\}$
$\Rightarrow[\ p := 0\ ]$ ⟨ "Assignment" with substitution ⟩
$\mathsf{xs} \in (0\,..\,k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner \ \wedge\ \mathsf{Ran}\,((0\,..\,p) \lhd \mathsf{xs}) = \{\mathsf{xs}\,@\,0\}$
$\Rightarrow[\ \mathsf{while}\ p \neq k + 1\ \mathsf{do}\ \mathsf{xs} := \mathsf{xs} \oplus \{\langle p, 42\rangle\}\,;\ p := p + 1\ \ \mathsf{od}$
$]$ ⟨ "While" with subproof:
?
⟩
$\neg(p \neq k + 1) \wedge \mathsf{xs} \in (0\,..\,k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner \ \wedge\ \mathsf{Ran}\,((0\,..\,p) \lhd \mathsf{xs}) = \{\mathsf{xs}\,@\,0\}$
$\Rightarrow\langle\ ?\ \rangle$
$\mathsf{xs} \in (0\,..\,k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner \ \wedge\ \mathsf{sorted\ xs}$

A Program Satisfying the Sorting Specification from the Previous Slide:

```
p := 0 ;
while p ≠ k + 1 do
    xs[p] := 42 ;
    p := p + 1
```

Bag-based Specification of Sorting

$xs_0 = \mathsf{xs} \in (0\,..\,k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner$
$\Rightarrow[\ \mathsf{SORT}$
$]$
$\mathsf{xs} \in (0\,..\,k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner \ \wedge\ \mathsf{sorted\ xs}$
$\quad \wedge\ \wr\, p \mid p \in \mathsf{xs} \bullet \mathsf{snd}\ p\, \wr = \wr\, p \mid p \in xs_0 \bullet \mathsf{snd}\ p\, \wr$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-11-15

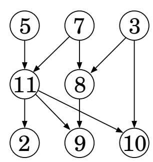
Topological Sort — LADM 14.4, pp. 287–291

Topological Sort — Introduction

A topological sort of a acyclic simple directed graph (V, B) is a linear order E containing B, that is, $E \cap E^{\smile} \subseteq \mathbb{I} \subseteq E \supseteq E \mathbin{;} E$ and $E \cup E^{\smile} = V \times V$ and $B \subseteq E$.

Since (V, B) is a DAG, B^* is an order: $B^* \cap B^{*\smile} \subseteq \mathbb{I} \subseteq B^* \supseteq B^* \mathbin{;} B^*$

E is normally presented as a sequence in *Seq* V that is sorted with repect to E and contains all elements of V.

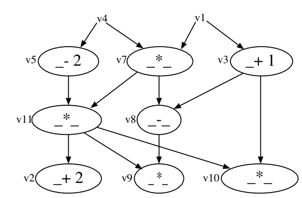


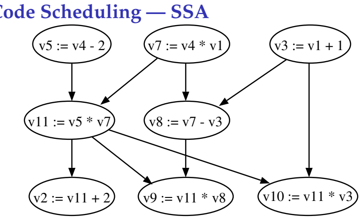
Example: The DAG above has, among others, the following topological sorts:

- [5, 7, 3, 11, 8, 2, 9, 10] — *visual left-to-right, top-to-bottom*
- [3, 5, 7, 8, 11, 2, 9, 10] — *smallest-numbered available vertex first*
- [5, 7, 3, 8, 11, 10, 9, 2] — *fewest edges first*
- [7, 5, 11, 3, 10, 8, 9, 2] — *largest-numbered available vertex first*
- [5, 7, 11, 2, 3, 8, 9, 10] — *attempting top-to-bottom, left-to-right*
- [3, 7, 8, 5, 11, 10, 2, 9] — *(arbitrary)*

$B = \{\langle 3,8\rangle, \langle 3,10\rangle, \langle 5,11\rangle, \langle 7,8\rangle, \langle 7,11\rangle, \langle 8,9\rangle, \langle 11,2\rangle, \langle 11,9\rangle, \langle 11,10\rangle\}$

Topological Sort — Code Scheduling — SSA





Static single assignment form: Each variable is assigned **once**, and assigned before use.

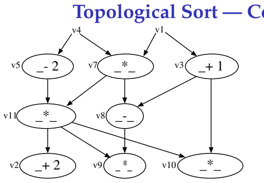
```
v5  := v4 - 2
v7  := v4 * v1
v3  := v1 + 1
v11 := v5 * v7
v8  := v7 - v3
v2  := v11 + 2
v9  := v11 * v8
v10 := v11 * v3
```

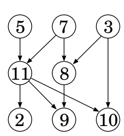
We can consider SSA as **encoding data-flow graphs**.

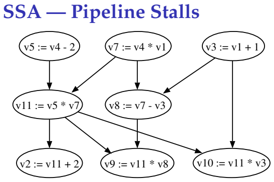
Each admissible re-ordering of an SSA sequence is a different topological sort of that graph.

It is frequently easier to think in terms of that graph than in terms of re-orderings!

Topological Sort — Code Scheduling — SSA — Pipeline Stalls







Static single assignment form: Each variable is assigned **once**, and assigned before use.

```
v7  := v4 * v1
v5  := v4 - 2
v11 := v5 * v7
v3  := v1 + 1
v10 := v11 * v3
v8  := v7 - v3
v9  := v11 * v8
v2  := v11 + 2
```

[7, <u>5, 11</u>, <u>3, 10</u>, <u>8, 9</u>, 2]

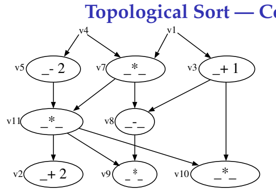
Let E be the topological sort of (V, B);
let $C = E - \mathbb{I}$ be the associated strict-order.

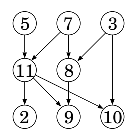
Depth-2 pipelining requires $B \subseteq C \mathbin{;} C$.
Depth-3 pipelining requires $B \subseteq C \mathbin{;} C \mathbin{;} C$.

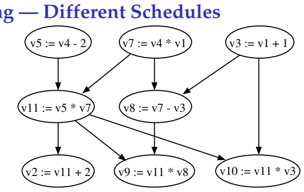
The "next-step" relation: $S = C - C \mathbin{;} C^+$

Depth-2 pipelining requires $B \cap S = \{\}$.
Depth-3 pipelining requires $B \cap (S \cup S \mathbin{;} S) = \{\}$.

Topological Sort — Code Scheduling — Different Schedules







Example: Most of the original example topological sorts induce pipeline stalls:

- [5, 7, 3, 11, 8, 2, 9, 10] — *visual left-to-right, top-to-bottom*
- [3, 5, 7, 8, <u>11, 2</u>, 9, 10] — *smallest-numbered available vertex first*
- [5, 7, <u>3, 8</u>, 11, 10, 9, 2] — *fewest edges first*
- [7, <u>5, 11</u>, <u>3, 10</u>, <u>8, 9</u>, 2] — *largest-numbered available vertex first*
- [5, <u>7, 11</u>, 2, <u>3, 8, 9</u>, 10] — *attempting top-to-bottom, left-to-right*
- [3, 7, 8, 5, <u>11, 10</u>, 2, 9] — *(arbitrary)*

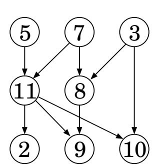
$B = \{\langle 3,8\rangle, \langle 3,10\rangle, \langle 5,11\rangle, \langle 7,8\rangle, \langle 7,11\rangle, \langle 8,9\rangle, \langle 11,2\rangle, \langle 11,9\rangle, \langle 11,10\rangle\}$

Topological Sort — Specification

A topological sort of a acyclic simple directed graph (V, B) is a linear order E containing B, that is, $E \cap E^{\smile} \subseteq \mathbb{I} \subseteq E \supseteq E \mathbin{;} E$ and $E \cup E^{\smile} = V \times V$ and $B \subseteq E$.

Since (V, B) is a DAG, B^* is an order: $B^* \cap B^{*\smile} \subseteq \mathbb{I} \subseteq B^* \supseteq B^* \mathbin{;} B^*$

E is normally presented as a sequence in *Seq* V that is sorted with repect to E and contains all elements of V.



Interface types: **var** vs : **set** T ...... Input: V
var s : *Seq* T ...... Output, representing E

C-style procedure declaration: *Seq* T topSort(**set** T vs)

Precondition: $vs = V$

Define: C is the expression "$\{u, v \mid u \text{ precedes } v \text{ in } s\}$" (of type $T \leftrightarrow T$)
E is the expression "$C \cup \mathbb{I}$" — both containing the free variable s

Real postcondition: $E \cap E^{\smile} \subseteq \mathbb{I} \subseteq E \supseteq E \mathbin{;} E \ \wedge\ E \cup E^{\smile} = V \times V \ \wedge\ B \subseteq E$.

One Formalisation of *_precedes_in_*

Precedence 50 **for:** *_precedes_in_*
Conjunctional: *_precedes_in_*
Declaration: *_precedes_in_* : $A \to A \to \mathsf{Seq}\ A \to \mathbb{B}$

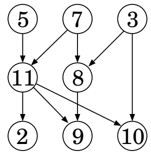
Axiom "Def. `_precedes_in_`": x precedes y in $\epsilon \ \equiv\ \mathsf{false}$
Axiom "Def. `_precedes_in_`": x precedes y in $(x \triangleleft \mathsf{zs}) \ \equiv\ y \in \mathsf{zs}$
Axiom "Def. `_precedes_in_`": $x \neq z \ \Rightarrow\ (x$ precedes y in $(z \triangleleft \mathsf{zs}) \ \equiv\ x$ precedes y in $\mathsf{zs})$

1 *precedes* 3 *in* $[1, 2]$ $\equiv$?

1 *precedes* 3 *in* $[3]$ $\equiv$?

1 *precedes* 3 *in* $[3, 1, 3]$ $\equiv$?

Topological Sort — Specification (ctd.)



A topological sort of a acyclic simple directed graph (V, B) is a linear order E containing B.

Since (V, B) is a DAG, B^* is an order: $B^* \cap B^{*\smile} \subseteq \mathbb{I} \subseteq B^* \supseteq B^* ; B^*$

E is normally presented as a sequence in $Seq\ V$ that is sorted with repect to E and contains all elements of V.

Interface types: **var** vs : **set** T ······ Input: V
var $s : Seq\ T$ ······ Output, representing E

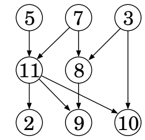
Precondition: $vs = V$

Define: C is the expression "$\{ u, v \mid u$ precedes v in $s \}$" (of type $T \leftrightarrow T$)
E is the expression "$C \cup \mathbb{I}$" — both containing the free variable s

Real postcondition: $E \cap E^{\smile} \subseteq \mathbb{I} \subseteq E \supseteq E ; E \wedge E \cup E^{\smile} = V \times V \wedge B \subseteq E$.

Representation-level postcondition: $(\forall u, v \mid u \,(B)\, v \bullet u$ precedes v in $s)$
$\wedge \{ v \mid v \in s \} = V$
$\wedge\ length\ s = \# V$

Topological Sort — Simple Algorithm



Given a DAG (V, B) (with V : **set** T),
calculate sequence s encoding a topological sort E.

var vs : **set** T; $s : Seq\ T$
vs := V ; — **not-yet-used vertices**
$\{ vs = V \}$ — **Precondition**
s := ϵ ; — **Initialising accumulator for result sequence**
$\{ (vs$ and $\{v \mid v \in s\}$ partition $V) \wedge length\ s + \# vs = \# V \wedge$
$(\forall u, v \mid v \in s \wedge u \,(B)\, v \bullet u$ precedes v in $s) \}$ — **Invariant**
while $vs \neq \{\}$ **do**
Choose a source u of the subgraph $(vs, B \cap (vs \times vs))$ induced by vs ;
vs, s := $vs - \{u\}, s \triangleright u$
od
$\{ (\forall u, v \mid u \,(B)\, v \bullet u$ precedes v in $s)$
$\wedge \{v \mid v \in s\} = V \wedge length\ s = \# V \}$ — **Postcondition**

The "Tableau" Presentation of the Previous Slide Closely Corresponds to Our Correctness Proof Presentation

Theorem "While-example":
Pre
⇒⟦ INIT ;
while B
do
C
od ;
FINAL
⟧
Post

Proof:
Pre ······ Precondition
⇒⟦ INIT ⟧ ⟨ ? ⟩
Q ······ Invariant
⇒⟦ while B do
C
od ⟧ ⟨ "While" with subproof:
$B \wedge Q$ ······ Loop condition and invariant
⇒⟦ C ⟧ ⟨ ? ⟩
Q ······ Invariant
⟩
$\neg B \wedge Q$ ······ Negated loop condition, and invariant
⇒⟦ FINAL ⟧ ⟨ ? ⟩
Post ······ Postcondition

Recall: The "While" Rule

The constituents of a while loop "while B do C od" are:

- The **loop condition** $B : \mathbb{B}$
- The **(loop) body** $C : Cmd$

The conventional **while rule** allows to infer only correctness statements for while loops that are in the shape of the conclusion of this inference rule, involving an **invariant** condition $Q : \mathbb{B}$:

```
   `B ∧ Q  ⇒⟦ C ⟧  Q`
⊢────────────────────────────────────
 `Q  ⇒⟦ while B do C od ⟧  ¬ B ∧ Q`
```

This rule reads:

- If you can prove that execution of the loop body C starting in states satisfying the loop condition B **preserves** the invariant Q,
- then you have proof that the whole loop also preserves the invariant Q, and in addition establishes the negation of the loop condition.

Recall: The "While" Rule — Induction for Partial Correctness

```
   `B ∧ Q  ⇒⟦ C ⟧  Q`
⊢────────────────────────────────────
 `Q  ⇒⟦ while B do C od ⟧  ¬ B ∧ Q`
```

The invariant will need to hold

- immediately before the loop starts,
- after each execution of the loop body,
- and therefore also after the loop ends.

The invariant will typically mention all variables that are changed by the loop, and explain how they are related.

Frequent pattern: Generalised postcondition using the negated loop condition

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-11-17

A2, Topological Sort

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-11-17

Part 1: A2: "Distributivity of ; with univalent over ∩" etc. . . .

For Univalent Relations . . . — LADM Hint, for M2-like Context

Theorem: If $F : A \leftrightarrow B$ is univalent, then $F ; (R \cap S) = (F ; R) \cap (F ; S)$

Hint: Assume determinacy; then show the equation using **relation extensionality**, and start from the RHS $\langle b, d \rangle \in (F ; R) \cap (F ; S)$. In the expansions of the two relation compositions here, introduce different bound variables.

For Univalent Relations . . . — LADM Hint, for M2-like Context

Theorem: If $F : A \leftrightarrow B$ is univalent, then $F ; (R \cap S) = (F ; R) \cap (F ; S)$

Hint: Assume determinacy; then show the equation using **relation extensionality**, and start from the RHS $\langle b, d \rangle \in (F ; R) \cap (F ; S)$. In the expansions of the two relation compositions here, introduce different bound variables.

Theorem "Distributivity of composition with univalent over ∩":
univalent F ⇒ $F ; (R \cap S) = F ; R \cap F ; S$
Proof:

For Univalent Relations . . . — LADM Hint, for M2-like Context

Theorem: If $F : A \leftrightarrow B$ is univalent, then $F ; (R \cap S) = (F ; R) \cap (F ; S)$

Hint: Assume determinacy; then show the equation using **relation extensionality**, and start from the RHS $\langle b, d \rangle \in (F ; R) \cap (F ; S)$. In the expansions of the two relation compositions here, introduce different bound variables.

Theorem "Distributivity of composition with univalent over ∩":
univalent F ⇒ $F ; (R \cap S) = F ; R \cap F ; S$
Proof:
Assuming `univalent F` **and using with** "Univalence":
Using "Relation extensionality":
For any `x`, `z`:
$x \,(F ; R \cap F ; S)\, z$
≡ ⟨ ? ⟩
$x \,(F ; (R \cap S))\, z$

Theorem "Distributivity of composition with univalent over ∩":
univalent $F \Rightarrow F \fatsemi (R \cap S) = F \fatsemi R \cap F \fatsemi S$
Proof:
Assuming \`univalent $F$\` **and using with** "Univalence":
Using "Relation extensionality":
For any \`$x$\`, \`$z$\`:
$x ⦗ F \fatsemi R \cap F \fatsemi S ⦘ z$
≡ ⟨ "Relation intersection", "Relation composition" ⟩
$(\exists y_1 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z) \land (\exists y_2 \bullet x ⦗ F ⦘ y_2 ⦗ S ⦘ z)$
≡ ⟨ ? ⟩
$\exists y \bullet x ⦗ F ⦘ y ⦗ R ⦘ z \land y ⦗ S ⦘ z$
≡ ⟨ "Relation intersection" ⟩
$\exists y \bullet x ⦗ F ⦘ y ⦗ R \cap S ⦘ z$
≡ ⟨ "Relation composition" ⟩
$x ⦗ F \fatsemi (R \cap S) ⦘ z$

Axiom "Univalence":
univalent R
$\equiv \forall b_1 \bullet \forall b_2 \bullet \forall a \bullet a ⦗ R ⦘ b_1 \land a ⦗ R ⦘ b_2 \Rightarrow b_1 = b_2$

Theorem "Distributivity of composition with univalent over ∩":
univalent $F \Rightarrow F \fatsemi (R \cap S) = F \fatsemi R \cap F \fatsemi S$
Proof:
Assuming \`univalent $F$\` **and using with** "Univalence":
Using "Relation extensionality":
For any \`$x$\`, \`$z$\`:
$x ⦗ F \fatsemi R \cap F \fatsemi S ⦘ z$
≡ ⟨ "Relation intersection", "Relation composition" ⟩
$(\exists y_1 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z) \land (\exists y_2 \bullet x ⦗ F ⦘ y_2 ⦗ S ⦘ z)$
≡ ⟨ "Distributivity of ∧ over ∃" ⟩
$\exists y_1 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land (\exists y_2 \bullet x ⦗ F ⦘ y_2 ⦗ S ⦘ z)$
≡ ⟨ "Distributivity of ∧ over ∃" ⟩
$\exists y_1 \bullet \exists y_2 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land x ⦗ F ⦘ y_2 ⦗ S ⦘ z$
≡ ⟨ ? ⟩
$\exists y \bullet x ⦗ F ⦘ y ⦗ R ⦘ z \land y ⦗ S ⦘ z$
≡ ⟨ "Relation intersection" ⟩
$\exists y \bullet x ⦗ F ⦘ y ⦗ R \cap S ⦘ z$
≡ ⟨ "Relation composition" ⟩
$x ⦗ F \fatsemi (R \cap S) ⦘ z$

Axiom "Univalence":
univalent R
$\equiv \forall b_1 \bullet \forall b_2 \bullet \forall a \bullet a ⦗ R ⦘ b_1 \land a ⦗ R ⦘ b_2 \Rightarrow b_1 = b_2$

Theorem "Distributivity of composition with univalent over ∩":
univalent $F \Rightarrow F \fatsemi (R \cap S) = F \fatsemi R \cap F \fatsemi S$
Proof:
Assuming \`univalent $F$\` **and using with** "Univalence":
Using "Relation extensionality":
For any \`$x$\`, \`$z$\`:
$x ⦗ F \fatsemi R \cap F \fatsemi S ⦘ z$
≡ ⟨ "Relation intersection", "Relation composition" ⟩
$(\exists y_1 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z) \land (\exists y_2 \bullet x ⦗ F ⦘ y_2 ⦗ S ⦘ z)$
≡ ⟨ "Distributivity of ∧ over ∃" ⟩
$\exists y_1 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land (\exists y_2 \bullet x ⦗ F ⦘ y_2 ⦗ S ⦘ z)$
≡ ⟨ "Distributivity of ∧ over ∃" ⟩
$\exists y_1 \bullet \exists y_2 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land x ⦗ F ⦘ y_2 ⦗ S ⦘ z$
≡ ⟨ ? ⟩
$\exists y_1 \bullet \exists y_2 \bullet y_2 = y_1 \land x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land x ⦗ F ⦘ y_2 ⦗ S ⦘ z$
≡ ⟨ ? ⟩
$\exists y \bullet x ⦗ F ⦘ y ⦗ R ⦘ z \land y ⦗ S ⦘ z$
≡ ⟨ "Relation intersection" ⟩
$\exists y \bullet x ⦗ F ⦘ y ⦗ R \cap S ⦘ z$
≡ ⟨ "Relation composition" ⟩
$x ⦗ F \fatsemi (R \cap S) ⦘ z$

Axiom "Univalence":
univalent R
$\equiv \forall b_1 \bullet \forall b_2 \bullet \forall a \bullet a ⦗ R ⦘ b_1 \land a ⦗ R ⦘ b_2 \Rightarrow b_1 = b_2$

Theorem "Distributivity of composition with univalent over ∩":
univalent $F \Rightarrow F \fatsemi (R \cap S) = F \fatsemi R \cap F \fatsemi S$
Proof:
Assuming \`univalent $F$\` **and using with** "Univalence":
Using "Relation extensionality":
For any \`$x$\`, \`$z$\`:
$x ⦗ F \fatsemi R \cap F \fatsemi S ⦘ z$
≡ ⟨ "Relation intersection", "Relation composition" ⟩
$(\exists y_1 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z) \land (\exists y_2 \bullet x ⦗ F ⦘ y_2 ⦗ S ⦘ z)$
≡ ⟨ "Distributivity of ∧ over ∃" ⟩
$\exists y_1 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land (\exists y_2 \bullet x ⦗ F ⦘ y_2 ⦗ S ⦘ z)$
≡ ⟨ "Distributivity of ∧ over ∃" ⟩
$\exists y_1 \bullet \exists y_2 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land x ⦗ F ⦘ y_2 ⦗ S ⦘ z$
≡ ⟨ ? ⟩
$\exists y_1 \bullet \exists y_2 \bullet y_2 = y_1 \land x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land x ⦗ F ⦘ y_2 ⦗ S ⦘ z$
≡ ⟨ "Trading for ∃", "One-point rule for ∃",
substitution, "Idempotency of ∧" ⟩
$\exists y \bullet x ⦗ F ⦘ y ⦗ R ⦘ z \land y ⦗ S ⦘ z$
≡ ⟨ "Relation intersection" ⟩
$\exists y \bullet x ⦗ F ⦘ y ⦗ R \cap S ⦘ z$
≡ ⟨ "Relation composition" ⟩
$x ⦗ F \fatsemi (R \cap S) ⦘ z$

Axiom "Univalence":
univalent R
$\equiv \forall b_1 \bullet \forall b_2 \bullet \forall a \bullet a ⦗ R ⦘ b_1 \land a ⦗ R ⦘ b_2 \Rightarrow b_1 = b_2$

Theorem "Distributivity of composition with univalent over ∩":
univalent $F \Rightarrow F \fatsemi (R \cap S) = F \fatsemi R \cap F \fatsemi S$
Proof:
Assuming \`univalent $F$\` **and using with** "Univalence":
Using "Relation extensionality":
For any \`$x$\`, \`$z$\`:
$x ⦗ F \fatsemi R \cap F \fatsemi S ⦘ z$
≡ ⟨ "Relation intersection", "Relation composition" ⟩
$(\exists y_1 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z) \land (\exists y_2 \bullet x ⦗ F ⦘ y_2 ⦗ S ⦘ z)$
≡ ⟨ "Distributivity of ∧ over ∃" ⟩
$\exists y_1 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land (\exists y_2 \bullet x ⦗ F ⦘ y_2 ⦗ S ⦘ z)$
≡ ⟨ "Distributivity of ∧ over ∃" ⟩
$\exists y_1 \bullet \exists y_2 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land x ⦗ F ⦘ y_2 ⦗ S ⦘ z$
≡ ⟨ ? ⟩
$\exists y_1 \bullet \exists y_2 \bullet (x ⦗ F ⦘ y_1 \land x ⦗ F ⦘ y_2 \Rightarrow y_2 = y_1)$
$\land\, x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land x ⦗ F ⦘ y_2 ⦗ S ⦘ z$
≡ ⟨ "Strong modus ponens" ⟩
$\exists y_1 \bullet \exists y_2 \bullet y_2 = y_1 \land x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land x ⦗ F ⦘ y_2 ⦗ S ⦘ z$
≡ ⟨ "Trading for ∃", "One-point rule for ∃",
substitution, "Idempotency of ∧" ⟩
$\exists y \bullet x ⦗ F ⦘ y ⦗ R ⦘ z \land y ⦗ S ⦘ z$
≡ ⟨ "Relation intersection" ⟩

Axiom "Univalence":
univalent R
$\equiv \forall b_1 \bullet \forall b_2 \bullet \forall a \bullet a ⦗ R ⦘ b_1 \land a ⦗ R ⦘ b_2 \Rightarrow b_1 = b_2$

Theorem "Distributivity of composition with univalent over ∩":
univalent $F \Rightarrow F \fatsemi (R \cap S) = F \fatsemi R \cap F \fatsemi S$
Proof:
Assuming \`univalent $F$\` **and using with** "Univalence":
Using "Relation extensionality":
For any \`$x$\`, \`$z$\`:
$x ⦗ F \fatsemi R \cap F \fatsemi S ⦘ z$
≡ ⟨ "Relation intersection", "Relation composition" ⟩
$(\exists y_1 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z) \land (\exists y_2 \bullet x ⦗ F ⦘ y_2 ⦗ S ⦘ z)$
≡ ⟨ "Distributivity of ∧ over ∃" ⟩
$\exists y_1 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land (\exists y_2 \bullet x ⦗ F ⦘ y_2 ⦗ S ⦘ z)$
≡ ⟨ "Distributivity of ∧ over ∃" ⟩
$\exists y_1 \bullet \exists y_2 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land x ⦗ F ⦘ y_2 ⦗ S ⦘ z$
≡ ⟨ Assumption \`univalent $F$\`, "Identity of ∧" ⟩
$\exists y_1 \bullet \exists y_2 \bullet (x ⦗ F ⦘ y_1 \land x ⦗ F ⦘ y_2 \Rightarrow y_2 = y_1)$
$\land\, x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land x ⦗ F ⦘ y_2 ⦗ S ⦘ z$
≡ ⟨ "Strong modus ponens" ⟩
$\exists y_1 \bullet \exists y_2 \bullet y_2 = y_1 \land x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land x ⦗ F ⦘ y_2 ⦗ S ⦘ z$
≡ ⟨ "Trading for ∃", "One-point rule for ∃",
substitution, "Idempotency of ∧" ⟩
$\exists y \bullet x ⦗ F ⦘ y ⦗ R ⦘ z \land y ⦗ S ⦘ z$
≡ ⟨ "Relation intersection" ⟩

Axiom "Univalence":
univalent R
$\equiv \forall b_1 \bullet \forall b_2 \bullet \forall a \bullet a ⦗ R ⦘ b_1 \land a ⦗ R ⦘ b_2 \Rightarrow b_1 = b_2$

Theorem "Distributivity of composition with univalent over ∩":
univalent $F \Rightarrow F \fatsemi (R \cap S) = F \fatsemi R \cap F \fatsemi S$
Proof:
Assuming \`univalent $F$\` **and using with** "Univalence":
Using "Relation extensionality":
For any \`$x$, $z$\`:
$x ⦗ F \fatsemi R \cap F \fatsemi S ⦘ z$
≡ ⟨ "Relation intersection", "Relation composition" ⟩
$(\exists y_1 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z) \land (\exists y_2 \bullet x ⦗ F ⦘ y_2 ⦗ S ⦘ z)$
≡ ⟨ "Distributivity of ∧ over ∃" ⟩
$\exists y_1 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land (\exists y_2 \bullet x ⦗ F ⦘ y_2 ⦗ S ⦘ z)$
≡ ⟨ "Distributivity of ∧ over ∃" ⟩
$\exists y_1 \bullet \exists y_2 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land x ⦗ F ⦘ y_2 ⦗ S ⦘ z$
≡ ⟨ ▪▪▪▪▪▪ Assumption `univalent F` with "Definition of ⇒ via ∧"
Subproof for \`$\forall y_1 \bullet \forall y_2 \bullet x ⦗ F ⦘ y_1 \land x ⦗ F ⦘ y_2 \equiv y_2 = y_1 \land x ⦗ F ⦘ y_1 \land x ⦗ F ⦘ y_2$\`
For any \`$y_1$\`, \`$y_2$\`:
Side proof for (1) \`$x ⦗ F ⦘ y_1 \land x ⦗ F ⦘ y_2 \Rightarrow y_2 = y_1$\`:
By Assumption \`univalent $F$\`
Continuing:
By local property (1) with "Definition of ⇒ via ∧"
⟩
$\exists y_1 \bullet \exists y_2 \bullet y_2 = y_1 \land x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land x ⦗ F ⦘ y_2 ⦗ S ⦘ z$
≡ ⟨ "Trading for ∃", "One-point rule for ∃"

Theorem "Distributivity of composition with univalent over ∩":
univalent $F \Rightarrow F \fatsemi (R \cap S) = F \fatsemi R \cap F \fatsemi S$
Proof:
Assuming \`univalent $F$\` **and using with** "Univalence":
Using "Relation extensionality":
For any \`$x$, $z$\`:
$x ⦗ F \fatsemi R \cap F \fatsemi S ⦘ z$
≡ ⟨ "Relation intersection", "Relation composition" ⟩
$(\exists y_1 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z) \land (\exists y_2 \bullet x ⦗ F ⦘ y_2 ⦗ S ⦘ z)$
≡ ⟨ "Distributivity of ∧ over ∃" ⟩
$\exists y_1 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land (\exists y_2 \bullet x ⦗ F ⦘ y_2 ⦗ S ⦘ z)$
≡ ⟨ "Distributivity of ∧ over ∃" ⟩
$\exists y_1 \bullet \exists y_2 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land x ⦗ F ⦘ y_2 ⦗ S ⦘ z$
≡ ⟨ ▪▪▪▪▪▪ Assumption `univalent F` with "Definition of ⇒ via ∧"
Subproof for \`$x ⦗ F ⦘ y_1 \land x ⦗ F ⦘ y_2 \equiv y_2 = y_1 \land x ⦗ F ⦘ y_1 \land x ⦗ F ⦘ y_2$\`:
▪▪▪▪▪▪ By Assumption `univalent F` with "Definition of ⇒ via ∧"
By "Definition of ⇒ via ∧" with Assumption \`univalent $F$\`
⟩
$\exists y_1 \bullet \exists y_2 \bullet y_2 = y_1 \land x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land x ⦗ F ⦘ y_2 ⦗ S ⦘ z$
≡ ⟨ "Trading for ∃", "One-point rule for ∃",
substitution, "Idempotency of ∧" ⟩
$\exists y \bullet x ⦗ F ⦘ y ⦗ R ⦘ z \land y ⦗ S ⦘ z$
≡ ⟨ "Relation intersection" ⟩

Theorem "Distributivity of composition with univalent over ∩":
univalent $F \Rightarrow F \fatsemi (R \cap S) = F \fatsemi R \cap F \fatsemi S$
Proof:
Assuming \`univalent $F$\` **and using with** "Univalence":
Using "Relation extensionality":
For any \`$x$, $z$\`:
$x ⦗ F \fatsemi R \cap F \fatsemi S ⦘ z$
≡ ⟨ "Relation intersection", "Relation composition" ⟩
$(\exists y_1 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z) \land (\exists y_2 \bullet x ⦗ F ⦘ y_2 ⦗ S ⦘ z)$
≡ ⟨ "Distributivity of ∧ over ∃" ⟩
$\exists y_1 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land (\exists y_2 \bullet x ⦗ F ⦘ y_2 ⦗ S ⦘ z)$
≡ ⟨ "Distributivity of ∧ over ∃" ⟩
$\exists y_1 \bullet \exists y_2 \bullet x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land x ⦗ F ⦘ y_2 ⦗ S ⦘ z$
≡ ⟨ "Definition of ⇒ via ∧" with Assumption \`univalent $F$\` ⟩
$\exists y_1 \bullet \exists y_2 \bullet y_2 = y_1 \land x ⦗ F ⦘ y_1 ⦗ R ⦘ z \land x ⦗ F ⦘ y_2 ⦗ S ⦘ z$
≡ ⟨ "Trading for ∃", "One-point rule for ∃", substitution, "Idempotency of ∧" ⟩
$\exists y \bullet x ⦗ F ⦘ y ⦗ R ⦘ z \land y ⦗ S ⦘ z$
≡ ⟨ "Relation intersection" ⟩
$\exists y \bullet x ⦗ F ⦘ y ⦗ R \cap S ⦘ z$
≡ ⟨ "Relation composition" ⟩
$x ⦗ F \fatsemi (R \cap S) ⦘ z$

Theorem "Partial-function application of ⨾":
univalent $f \land$ univalent $g \land x \in \mathsf{Dom}\,(f \fatsemi g) \Rightarrow (f \fatsemi g) @ x = g @ (f @ x)$
Proof: Assuming \`univalent $f$\`, \`univalent $g$\`, \`$x \in \mathsf{Dom}\,(f \fatsemi g)$\`:
Side proof for \`$x \in \mathsf{Dom}\,f$\`:
By assumption \`$x \in \mathsf{Dom}\,(f \fatsemi g)$\` with "Membership in domain of ⨾", "Weakening"
Side proof for \`$f @ x \in \mathsf{Dom}\,g$\`:
$x \in \mathsf{Dom}\,(f \fatsemi g)$ — **This is** an assumption
⇒ ⟨ "Membership in domain of ⨾", "Weakening" ⟩
$\exists y \mid x ⦗ f ⦘ y \bullet y \in \mathsf{Dom}\,g$
≡ ⟨ "Partial-function application" with assumption \`univalent $f$\` and local property \`$x \in \mathsf{Dom}\,f$\` ⟩
$\exists y \mid y = f @ x \bullet y \in \mathsf{Dom}\,g$
≡ ⟨ "One-point rule for ∃", substitution ⟩
$f @ x \in \mathsf{Dom}\,g$
Side proof for "U" \`univalent $(f \fatsemi g)$\`:
By "Univalence of composition" with assumptions \`univalent $f$\` and \`univalent $g$\`
Continuing:
$(f \fatsemi g) @ x = g @ (f @ x)$
≡ ⟨ "Partial-function application" with local property "U" and assumption \`$x \in \mathsf{Dom}\,(f \fatsemi g)$\` ⟩
$x ⦗ f \fatsemi g ⦘ g @ (f @ x)$
≡ ⟨ "Relation composition" ⟩
$\exists y \bullet x ⦗ f ⦘ y ⦗ g ⦘ g @ (f @ x)$
≡ ⟨ "Partial-function application" with assumption \`univalent $f$\`
and local property \`$x \in \mathsf{Dom}\,f$\`, "Trading for ∃" ⟩
$\exists y \mid y = f @ x \bullet y ⦗ g ⦘ g @ (f @ x)$
≡ ⟨ "One-point rule for ∃", substitution ⟩
$f @ x ⦗ g ⦘ g @ (f @ x)$
≡ ⟨ "Relationship with @" with assumption \`univalent $g$\` and local property \`$f @ x \in \mathsf{Dom}\,g$\` ⟩
true

Theorem "Injectivity and @":
  univalent f ∧ injective f ∧ $x_1 \in$ Dom f ∧ $x_2 \in$ Dom f ⇒ $(f @ x_1 = f @ x_2 \equiv x_1 = x_2)$
Proof:
  Assuming \`univalent $f$\`, \`injective $f$\` **and using with** "Injectivity",
    \`$x_1 \in$ Dom $f$\`, \`$x_2 \in$ Dom $f$\`:
   Using "Mutual implication":
    Subproof:
     Assuming \`$x_1 = x_2$\`:
       $f @ x_1$
      = ⟨ Assumption \`$x_1 = x_2$\` ⟩
       $f @ x_2$
    Subproof for \`$f @ x_1 = f @ x_2 \Rightarrow x_1 = x_2$\`:
     Side proof for \`$x_1$ ⦗ $f$ ⦘ $f @ x_1$\`:
      By "Relationship with @" with assumptions \`univalent $f$\` and \`$x_1 \in$ Dom $f$\`
     Continuing:
       $f @ x_1 = f @ x_2$
      ≡ ⟨ "Partial-function application" with assumptions \`univalent $f$\` and \`$x_2 \in$ Dom $f$\` ⟩
       x_2 ⦗ f ⦘ $f @ x_1$
      ≡ ⟨ "Identity of ∧", local property \`$x_1$ ⦗ $f$ ⦘ $f @ x_1$\` ⟩
       x_1 ⦗ f ⦘ $f @ x_1$ ∧ x_2 ⦗ f ⦘ $f @ x_1$
      ⇒ ⟨ Assumption \`injective $f$\` ⟩
       $x_1 = x_2$

Theorem "Injectivity and @":
  univalent f ∧ injective f ∧ $x_1 \in$ Dom f ∧ $x_2 \in$ Dom f ⇒ $(f @ x_1 = f @ x_2 \equiv x_1 = x_2)$
Proof:
  Assuming \`univalent $f$\`,
    \`injective $f$\` **and using with** "Injectivity",
    \`$x_1 \in$ Dom $f$\`, \`$x_2 \in$ Dom $f$\`:
   Using "Mutual implication":
    Subproof:
     Assuming \`$x_1 = x_2$\`:
       $f @ x_1$
      = ⟨ Assumption \`$x_1 = x_2$\` ⟩
       $f @ x_2$
    Subproof for \`$f @ x_1 = f @ x_2 \Rightarrow x_1 = x_2$\`:
      $x_1 = x_2$
     ⇐ ⟨ Assumption \`injective $f$\` ⟩
      x_1 ⦗ f ⦘ $f @ x_1$ ∧ x_2 ⦗ f ⦘ $f @ x_1$
     ≡ ⟨ "Relationship with @" with
       assumptions \`univalent $f$\` and \`$x_1 \in$ Dom $f$\`, "Identity of ∧" ⟩
      x_2 ⦗ f ⦘ $f @ x_1$
     ≡ ⟨ "Partial-function application" with assumptions \`univalent $f$\` and \`$x_2 \in$ Dom $f$\` ⟩
      $f @ x_1 = f @ x_2$

Theorem "Injectivity and @":
  univalent f ∧ injective f ∧ $x_1 \in$ Dom f ∧ $x_2 \in$ Dom f ⇒ $(f @ x_1 = f @ x_2 \equiv x_1 = x_2)$
Proof: ▪▪▪▪▪▪ Raymond Zhao
  Assuming \`univalent $f$\`, \`$x_1 \in$ Dom $f$\`, \`$x_2 \in$ Dom $f$\`:
   Assuming \`injective $f$\` **and using with** "Injectivity":
     $x_1 = x_2$
    ⇒ ⟨ "Leibniz" ⟩
     $(f @ z)[z := x_1] = (f @ z)[z := x_2]$
    ≡ ⟨ Substitution ⟩
     $f @ x_1 = f @ x_2$
    ≡ ⟨ "Partial-function application" with
      Assumption \`$x_2 \in$ Dom $f$\` and Assumption \`univalent $f$\` ⟩
     x_2 ⦗ f ⦘ $f @ x_1$
    ≡ ⟨ "Identity of ∧" ⟩
     true ∧ x_2 ⦗ f ⦘ $f @ x_1$
    ≡ ⟨ "Relationship with @" with
      Assumption \`univalent $f$\` and Assumption \`$x_1 \in$ Dom $f$\` ⟩
     x_1 ⦗ f ⦘ $f @ x_1$ ∧ x_2 ⦗ f ⦘ $f @ x_1$
    ⇒ ⟨ Assumption \`injective $f$\` ⟩
     $x_1 = x_2$

```
Theorem "Mirrored `decode2`":
  ∀ t : HTree A • ∀ bs : Seq 𝔹
  • decode2 t (map not bs) = map (second (map not)) (decode2 (t ˘) bs)
Proof:
  Using "HTree induction′":
    Subproof:
      For any `x : A`, `bs : Seq 𝔹`:
          map (second (map not)) (decode2 (⌜ x ⌟ ˘) bs)
        =⟨ "Mirror", "Definition of `decode2`" ⟩
          map (second (map not)) (just ⟨ x, bs ⟩)
        =⟨ "Maybe map", "Definition of `second`" ⟩
          just ⟨ x, map not bs ⟩
        =⟨ "Definition of `decode2`" ⟩
          decode2 ⌜ x ⌟ (map not bs)
    Subproof for `∀ l, r : HTree A
        • (∀ bs : Seq 𝔹 • decode2 l (map not bs) = map (second (map not)) (decode2 (l ˘) bs))
        ⇒ (∀ bs : Seq 𝔹 • decode2 r (map not bs) = map (second (map not)) (decode2 (r ˘) bs))
        ⇒ (∀ bs : Seq 𝔹 • decode2 (l ⩓ r) (map not bs) = map (second (map not)) (decode2 ((l ⩓ r) ˘) bs))`:
      For any `l, r : HTree A`:
        Assuming
            "IndHypL" `∀ bs : Seq 𝔹 • decode2 l (map not bs) = map (second (map not)) (decode2 (l ˘) bs)`,
            "IndHypR" `∀ bs : Seq 𝔹 • decode2 r (map not bs) = map (second (map not)) (decode2 (r ˘) bs)`:
          By induction on `bs : Seq 𝔹`:
            Base case:
                map (second (map not)) (decode2 ((l ⩓ r) ˘) ϵ)
              =⟨ "Mirror", "Definition of `decode2`", "Maybe map" ⟩
                nothing
              =⟨ "Definition of `decode2`", "Definition of `map` for ϵ" ⟩
                decode2 (l ⩓ r) (map not ϵ)
            Induction step:
              For any `b : 𝔹`:
                By cases: `b`, `b ≡ false`
                  Completeness: By "Definition of ¬ from ≡", "LEM"
                  Case `b`:
                      decode2 (l ⩓ r) (map not (b ◃ bs))
                    =⟨ Assumption `b`, "Definition of `map` for ◃", "Definition of `not`", "Definition of `false`" ⟩
                      decode2 (l ⩓ r) (false ◃ map not bs)
                    =⟨ "Definition of `decode2`" ⟩
                      decode2 l (map not bs)
                    =⟨ Assumption "IndHypL" ⟩
                      map (second (map not)) (decode2 (l ˘) bs)
                    =⟨ "Mirror", assumption `b`, "Definition of `decode2`" ⟩
                      map (second (map not)) (decode2 ((l ⩓ r) ˘) (b ◃ bs))
                  Case `b ≡ false`:
                      decode2 (l ⩓ r) (map not (b ◃ bs))
                    =⟨ Assumption `b ≡ false`, "Definition of `map` for ◃", "Definition of `not`", "Negation of `false`" ⟩
                      decode2 (l ⩓ r) (true ◃ map not bs)
                    =⟨ "Definition of `decode2`" ⟩
                      decode2 r (map not bs)
                    =⟨ Assumption "IndHypR" ⟩
                      map (second (map not)) (decode2 (r ˘) bs)
                    =⟨ "Mirror", assumption `b ≡ false`, "Definition of `decode2`" ⟩
                      map (second (map not)) (decode2 ((l ⩓ r) ˘) (b ◃ bs))
```

Logical Reasoning for Computer Science

COMPSCI 2LC3

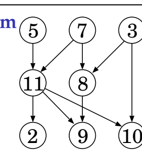
McMaster University, Fall 2023

Wolfram Kahl

2023-11-17

Part 2: Topological Sort

Recall: Topological Sort — Simple Algorithm

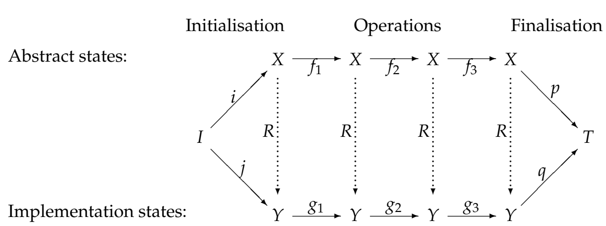


Given a DAG (V, B) (with V : **set** T),
calculate sequence s encoding a topological sort E.

var vs : **set** T; s : *Seq* T
$vs := V$ ⨾   — **not-yet-used vertices**
$\{ vs = V \}$ — **Precondition**
$s := \epsilon$ ⨾   — **Initialising accumulator for result sequence**
$\{ (vs$ and $\{v \mid v \in s\}$ partition $V) \wedge$ *length* $s + \# vs = \# V \wedge$
  $(\forall u, v \mid v \in s \wedge u$ ⦗ B ⦘ $v \bullet u$ precedes v in $s) \}$ — **Invariant**
while $vs \neq \{\}$ **do**
  Choose a source u of the subgraph $(vs, B \cap (vs \times vs))$ induced by vs ⨾
  $vs, s := vs - \{u\}, s \triangleright u$
od
$\{ (\forall u, v \mid u$ ⦗ B ⦘ $v \bullet u$ precedes v in $s)$
$\wedge \{v \mid v \in s\} = V \wedge$ *length* $s = \# V \}$ — **Postcondition**

How to "Choose a source u of the subgraph induced by vs" **efficiently?**

Data Refinement



Representation relation: $R : X \leftrightarrow Y$ — "**coupling invariant**" —
relates abstract states X with concrete implementation states Y:

- Compatible initialisation: $j \subseteq i ⨾ R$
- Operation simulation: $R ⨾ g_k \subseteq f_k ⨾ R$
- Compatible results: $R ⨾ q \subseteq p$

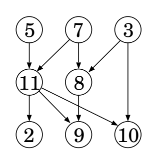
Topological Sort — Making Choosing Minimal Elements Easier

To store mappings $V \rightarrow\!\!\!\!\!\circ\; X$ in "**array ... of** X", "assume" $V = 0..k = \{i : \mathbb{N} \mid 0 \leq i \leq k\}$.

var *sources* : *Seq* $(0..k)$ — three new variables make vs superfluous
var *preCount* : **array** $0..k$ **of** ⌞ ℕ ⌟
var *postSet* : **array** $0..k$ **of** $\mathbb{P}(0..k)$ — read-only version of $B : V \leftrightarrow V$ as $V \rightarrow \mathbb{P}V$

Coupling invariant:
 $\{u \mid u \in sources\} = vs - (Ran\ B') \wedge$ — *sources* contains sources of $B' = B \cap (vs \times vs)$
 $(\forall v \mid v \in vs \bullet preCount[v] = \# (B'^{\smile} (\!| \{v\} |\!))) \wedge$
 $(\forall u \mid u \in vs \bullet postSet[u] = B' (\!| \{u\} |\!))$

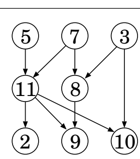
Initialisation:



for $v \in 0..k$ **do** $preCount[v] := \# (B^{\smile} (\!| \{v\} |\!))$ **od** ⨾
for $u \in 0..k$ **do** $postSet[u] := B (\!| \{u\} |\!)$ **od** ⨾
$sources := \epsilon$ ⨾
for $v \in 0..k$ **do if** $preCount[v] = 0$ **then** $sources := sources \triangleright v$ **fi od**

Topological Sort — Complete "Translated" LADM Algorithm

for $v \in 0..k$ **do** $preCount[v] := \# (B^{\smile} (\!| \{v\} |\!))$ **od** ⨾
for $u \in 0..k$ **do** $postSet[u] := B (\!| \{u\} |\!)$ **od** ⨾
$sources := \epsilon$ ⨾
for $v \in 0..k$ **do if** $preCount[v] = 0$ **then** $sources := sources \triangleright v$ **fi od**
ghost $vs := 0..k$ ⨾   — $B' = B \cap (vs \times vs)$
$s := \epsilon$
while $sources \neq \epsilon$ **do**   — **Coupling invariant:** $\{u \mid u \in sources\} = vs - (Ran\ B') \wedge$ $(\forall v \mid v \in vs \bullet preCount[v] = \# (B'^{\smile} (\!| \{v\} |\!)))$ $\wedge (\forall u \mid u \in vs \bullet postSet[u] = B' (\!| \{u\} |\!))$
  $u :=$ *head sources* ⨾
  $s := s \triangleright u$ ⨾
  $sources :=$ *tail sources* ⨾   — remove u from *sources*
  ghost $vs := vs - \{u\}$ ⨾
  for $v \in postSet[u]$ **do**
    $preCount[v] := preCount[v] - 1$ ⨾
    if $preCount[v] = 0$ **then** $sources := sources \triangleright v$ **fi**
  od
od



Topological Sort — Complete $O(\# B + \# V)$ Algorithm

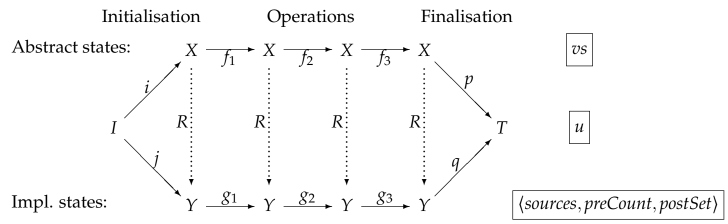
for $p \in B$ **do**
  $preCount[snd\ p] := preCount[snd\ p] + 1$
  $postSet[fst\ p] := postSet[fst\ p] \cup \{snd\ p\}$
od ⨾
$sources := \epsilon$ ⨾ **for** $v \in 0..k$ **do if** $preCount[v] = 0$ **then** $sources := sources \triangleright v$ **fi od**
ghost $vs := 0..k$ ⨾   — $B' = B \cap (vs \times vs)$
$s := \epsilon$
while $sources \neq \epsilon$ **do**   — **Coupling invariant:** $\{u \mid u \in sources\} = vs - (Ran\ B') \wedge$ $(\forall v \mid v \in vs \bullet preCount[v] = \# (B'^{\smile} (\!| \{v\} |\!)))$ $\wedge (\forall u \mid u \in vs \bullet postSet[u] = B' (\!| \{u\} |\!))$
  $u :=$ *head sources* ⨾
  $s := s \triangleright u$ ⨾
  $sources :=$ *tail sources* ⨾   — remove u from *sources*
  ghost $vs := vs - \{u\}$ ⨾
  for $v \in postSet[u]$ **do**
    $preCount[v] := preCount[v] - 1$ ⨾
    if $preCount[v] = 0$ **then** $sources := sources \triangleright v$ **fi**
  od
od

Topological Sort — Complete $O(\# B + \# V)$ Algorithm — Using Pair Iteration

```
for ⟨u, v⟩ ∈ B do
    preCount[v] := preCount[v] + 1
    postSet[u] := postSet[u] ∪ {v}
od ;
sources := ϵ ; for v ∈ 0 .. k do if preCount[v] = 0 then sources := sources ▹ v fi od
ghost vs := 0 .. k ;                                         — B′ = B ∩ (vs × vs)
s := ϵ
while sources ≠ ϵ do        — Coupling invariant:
    u := head sources ;
    s := s ▹ u ;
    sources := tail sources ;        — remove u from sources
    ghost vs := vs − {u} ;
    for v ∈ postSet[u] do
        preCount[v] := preCount[v] − 1 ;
        if preCount[v] = 0 then sources := sources ▹ v fi
    od
od
```

Coupling invariant:
$\{u \mid u \in sources\} = vs - (Ran\ B') \wedge$
$(\forall v \mid v \in vs \bullet preCount[v] = \#(B'^{\smile}(\!|\{v\}|\!)))$
$\wedge (\forall u \mid u \in vs \bullet postSet[u] = B'(\!|\{u\}|\!))$

Recapitulate: Data Refinement



Representation relation: $R : X \leftrightarrow Y$ — "**coupling invariant**" —
relates abstract states X with concrete implementation states Y:

- Compatible initialisation: $j \subseteq i ⨾ R$
- Operation simulation: $R ⨾ g_k \subseteq f_k ⨾ R$
- Compatible results: $R ⨾ q \subseteq p$

Topological Sort — Summary

- The "Simple Algorithm" can be proved correct wrt. a mathematical characterisation of "Choose a source u"
- As a "Finalisation" relation relating states with u-values, this is **not univalent.**
- Given the coupling invariant, "u := *head sources*" chooses a "compatible result".
- The **for**-loop updating the refined state implements "$vs := vs - \{u\}$" **by re-establishing the coupling invariant**
- **Separation of concerns** between
  - high-level algorithm correctness proof
  - data representation decisions for low-level efficiency implemented as **refinement**

  makes the whole proof is more modular, and easier to understand, and the development more maintainable and reusable.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-11-20

Relational Semantics of Simple Imperative Programs

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-11-20

Part 1: Ghosts for Complexity

Recall: Topological Sort — Complete $O(\# B + \# V)$ Algorithm (Pair Iteration)

```
for ⟨u, v⟩ ∈ B do
    preCount[v] := preCount[v] + 1
    postSet[u] := postSet[u] ∪ {v}
od ;
sources := ϵ ; for v ∈ 0 .. k do if preCount[v] = 0 then sources := sources ▹ v fi od
ghost vs := 0 .. k ;                                         — B′ = B ∩ (vs × vs)
s := ϵ
while sources ≠ ϵ do        — Coupling invariant:
    u := head sources ;
    s := s ▹ u ;
    sources := tail sources ;        — remove u from sources
    ghost vs := vs − {u} ;
    for v ∈ postSet[u] do
        preCount[v] := preCount[v] − 1 ;
        if preCount[v] = 0 then sources := sources ▹ v fi
    od
od
```

Coupling invariant:
$\{u \mid u \in sources\} = vs - (Ran\ B') \wedge$
$(\forall v \mid v \in vs \bullet preCount[v] = \#(B'^{\smile}(\!|\{v\}|\!)))$
$\wedge (\forall u \mid u \in vs \bullet postSet[u] = B'(\!|\{u\}|\!))$

Recall: Ghost Variables

If a language supports "ghost variables" then:

- ghost variables cannot occur in if-conditions, while-conditions, RHS of assignments, function call arguments.
- That is, values of ghost variables do not influence program flow or results.
- Compilers will normally suppress ghost variables and their assignments.

"Ghost variables" can make proofs easier: They can be used to keep track of values that are important for **understanding/documenting/proving** the logic of the program.

On the "topological sort" example of the previous slide, the ghost variables vs contains the state of the abstract version of the algorithm, so that the coupling invariant relating vs with the refined state ⟨ *sources*, *preCount*, *postSet* ⟩ can be verified before and after the loop body.

Ghost variables can also be used to "instrument" a program for proving complexity bounds — see the next slide.

Topological Sort — Complete $O(\# B + \# V)$-ghosted Algorithm

```
ghost int stepCount = 0 ;
for ⟨u, v⟩ ∈ B do
    preCount[v] := preCount[v] + 1 ; ghost stepCount++ ;
    postSet[u] := postSet[u] ∪ {v} ; ghost stepCount++
od ;
sources := ϵ ;
for v ∈ 0 .. k do ghost stepCount++ ; if preCount[v] = 0 then sources := sources ▹ v fi od
s := ϵ
while sources ≠ ϵ do
    u := head sources ; s := s ▹ u ; ghost stepCount++ ;
    sources := tail sources ;        — remove u from sources
    for v ∈ postSet[u] do
        preCount[v] := preCount[v] − 1 ; ghost stepCount++ ;
        if preCount[v] = 0 then sources := sources ▹ v fi
    od
od ;
ghost assert stepCount ≤ C1 · # B + C2 · # V        — complexity postcondition
```

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-11-20

Part 2: Relational Semantics

Formalising Partial Correctness — Syntax Types

So far, we have been using the **dynamic logic** notation:

$$P \Rightarrow [\, C \,]\, Q$$

with its **partial correctness meaning:**

> If command C is started in a state in which the **precondition** P holds
> then it will terminate **only** in a state in which the **postcondition** Q holds.

What are P, Q, C?

- P and Q are some kind of Boolean expressions — of type Expr𝔹
- C is a command — of type Cmd
- We also need expression e for assignment RHSs, "$x := e$" — of type ExprV

The Programming Language: Expressions and Commands

The types Cmd, ExprV, and Expr𝔹 are abstract syntax tree (AST) types

Declaration: $ExprV$, Expr𝔹 : Type
Declaration: Var' : Var → ExprV
Declaration: Int' : $\mathbb{Z}$ → ExprV
Declaration: _ +′_ : ExprV → ExprV → ExprV
Declaration: $true'$, $false'$: Expr𝔹
Declaration: ¬′_ : Expr𝔹 → Expr𝔹
Declaration: _ ∧′_ : Expr𝔹 → Expr𝔹 → Expr𝔹
Declaration: _ =′_ : ExprV → ExprV → Expr𝔹

Declaration: Cmd : Type
Declaration: _;_ : Cmd → Cmd → Cmd
Declaration: _ := _ : Var → ExprV → Cmd
Declaration: if_then_else_fi : Expr𝔹 → Cmd → Cmd → Cmd
Declaration: while_do_od : Expr𝔹 → Cmd → Cmd

Formalising Partial Correctness — Semantics Types

So far, we have been using the **dynamic logic** notation:

$$P \Rightarrow [\, C \,]\, Q$$

with its **partial correctness meaning:**

> If command C is started in a state in which the **precondition** P holds
> then it will terminate **only** in a state in which the **postcondition** Q holds.

What does "state" mean? "starts"? "holds"? "terminates"? …

- States assign variable to values
- here we simply model states as function — of type Var → Value
- "P holds in state s": semantics of Boolean expressions: sat : Expr𝔹 → set State
  ($s \in$ sat P iff "condition P is **sat**isfied in state s")
  (Alternatively, start from eval𝔹 : State𝔹 → Expr𝔹 → 𝔹 and define sat P = { s | eval𝔹 s P })

Types for Semantics of Expressions and Commands

What does "state" mean? "holds"? …

Imperative programs, such as Cmd, transform a State that assigns values to variables.

Declaration: Var : Type — variables
Declaration: Value : Type — storable values
Declaration: State : Type

Axiom "Definition of `State`": State = Var → Value

Declaration: eval : State → ExprV → Value — value expression semantics
Declaration: sat : Expr𝔹 → set State — Boolean expression semantics

Declaration: $\_ \oplus' \_ : (A \to B) \to ⦗ A, B ⦘ \to (A \to B)$ — state update
Axiom "Definition of function override":

$$(x = z \Rightarrow (f \oplus' \langle x, y \rangle)\, z = y)$$
$$\wedge\ (x \neq z \Rightarrow (f \oplus' \langle x, y \rangle)\, z = f\, z)$$

Semantics of Commands

What does "starts" mean? "terminates"? …

Program execution induces a **state transformation relation**.

Declaration: ⟦_⟧ : Cmd → (State ↔ State)

s_1 ⦗ ⟦ C ⟧ ⦘ s_2 iff "when started in state s_1, command C can terminate in state s_2".

Inductive definition of ⟦_⟧ over the structure of Cmd:

Axiom "Semantics of :=": ⟦ $x := e$ ⟧ = { s : State • ⟨ s , $s \oplus' \langle x$, eval s $e \rangle$ ⟩ }
Axiom "Semantics of ;": ⟦ C_1 ; C_2 ⟧ = ⟦ C_1 ⟧ ⨾ ⟦ C_2 ⟧
Axiom "Semantics of `if`":

⟦ if B then C_1 else C_2 fi ⟧ = (sat B ◁ ⟦ C_1 ⟧) ∪ (sat B ⩤ ⟦ C_2 ⟧)

Axiom "Semantics of `while`":

⟦ while B do C od ⟧ = (sat B ◁ ⟦ C ⟧)* ⩥ sat B

Formalising Partial Correctness

So far, we have been using the **dynamic logic** notation:

$$P \Rightarrow [\, C \,]\, Q$$

with its **partial correctness meaning:**

> If command C is started in a state in which the **precondition** P holds
> then it will terminate **only** in a state in which the **postcondition** Q holds.

Declaration: _ ⇒[_]_ : Expr𝔹 → Cmd → Expr𝔹 → 𝔹
Axiom "Partial Correctness":

(P ⇒[C] Q) ≡ ⟦ C ⟧ ⦇ sat P ⦈ ⊆ sat Q

Theorem "Partial Correctness":

(P ⇒[C] Q) ≡ ∀ s_1, s_2 • s_1 ∈ sat P ∧ s_1 ⦗ ⟦ C ⟧ ⦘ s_2 ⇒ s_2 ∈ sat Q

Soundness of the Inference Rules for Correctness

Since partial correctness statements (P ⇒[C] Q) are now defined via the relational semantics, we can prove **soundness** of the Hoare logic proof rules by deriving them, e.g.:

Derived inference rule "Sequence":

`P ⇒[ $C_1$ ] Q`, `Q ⇒[ $C_2$ ] R`
⊢ ――――――――――――――――
`P ⇒[ $C_1$ ; $C_2$ ] R`

Proof:

Assuming (C_1) `P ⇒[ $C_1$ ] Q` **and using with** "Partial correctness",
(C_2) `Q ⇒[ $C_2$ ] R` **and using with** "Partial correctness":

P ⇒[C_1 ; C_2] R
≡ ⟨ "Partial correctness" ⟩
⟦ C_1 ; C_2 ⟧ ⦇ sat P ⦈ ⊆ sat R
≡ ⟨ "Semantics of ;", "Relational image of ⨾" ⟩
⟦ C_2 ⟧ ⦇ ⟦ C_1 ⟧ ⦇ sat P ⦈ ⦈ ⊆ sat R
⇐ ⟨ Antitonicity with assumption (C_1) ⟩
⟦ C_2 ⟧ ⦇ sat Q ⦈ ⊆ sat R
≡ ⟨ Assumption (C_2) ⟩
true

Soundness of the Inference Rules for Correctness (ctd.)

Derived inference rule "Conditional":

`B ∧′ P ⇒[ $C_1$ ] Q`, `¬′ B ∧′ P ⇒[ $C_2$ ] Q`
⊢ ――――――――――――――――
`P ⇒[ if B then $C_1$ else $C_2$ fi ] Q`

Derived inference rule "While":

`B ∧′ Q ⇒[ C ] Q`
⊢ ――――――――――――――――
`Q ⇒[ while B do C od ] ¬′ B ∧′ Q`

"Operational Semantics", "Axiomatic Semantics"

For a command C : Cmd, we introduced it relational semantics ⟦ C ⟧ : $State \leftrightarrow State$.

This semantics only captures the **terminating behaviours** of C, in the shape of an "input-output relation".

This is also called **"big-step operational semantics"**, or **"natural semantics"**.

"Small-step operational semantics" maps C to a relation of type $State \leftrightarrow (State^* \cup State^\infty)$:

- Each start state s_0 is related to all possible execution sequences starting from s_0.
- All intermediate states (after each assignment) are recorded.
- Non-terminating behaviours give rise to infinite state sequences.
- Terminating behaviours give rise to finite sequences $s_0, \ldots, s_n$, with s_0 ⦗ ⟦ C ⟧ ⦘ s_n
  — this is either a proof obligation, or a way to define ⟦ C ⟧.

"Axiomatic semantics" is the set of correctness statements (P ⇒[C] Q) that can be derived about C in a inference system of the kind we have used.

As seen on the previous slides, such an inference system can (and should!) be justified against the operational semantics.

— More in COMPSCI 3MI3!

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-11-22

Total Correctness

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-11-22

Part 1: Relational Semantics: Partial Correctness

Bag-based Specification of Sorting

$xs_0 = \mathsf{xs} \in (0 \,..\, k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner$

$\Rightarrow[$ SORT

$]$

$\mathsf{xs} \in (0 \,..\, k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner \;\wedge\; \mathsf{sorted\ xs}$

$\quad \wedge\, \wr\, p \mid p \in \mathsf{xs} \bullet \mathsf{snd}\, p\, \wr = \wr\, p \mid p \in xs_0 \bullet \mathsf{snd}\, p\, \wr$

A Verified Sorting Algorithm

Theorem "Sorting 0′ ":

$xs_0 = \mathsf{xs} \in (0 \,..\, k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner$

$\Rightarrow[$ while true do

xs := xs ⊕ { ⟨ 0, 42 ⟩ }

od

$]$

$\mathsf{xs} \in (0 \,..\, k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner \;\wedge\; \mathsf{sorted\ xs}$

$\quad \wedge\, \wr\, p \mid p \in \mathsf{xs} \bullet \mathsf{snd}\, p\, \wr = \wr\, p \mid p \in xs_0 \bullet \mathsf{snd}\, p\, \wr$

```
while true do
   xs[0] := 42
```

Proof structure?

A Verified Sorting Algorithm

Theorem "Sorting 0′ ":

$xs_0 = \mathsf{xs} \in (0 \,..\, k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner$

$\Rightarrow[$ while true do

xs := xs ⊕ { ⟨ 0, 42 ⟩ }

od

$]$

$\mathsf{xs} \in (0 \,..\, k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner \;\wedge\; \mathsf{sorted\ xs}$

$\quad \wedge\, \wr\, p \mid p \in \mathsf{xs} \bullet \mathsf{snd}\, p\, \wr = \wr\, p \mid p \in xs_0 \bullet \mathsf{snd}\, p\, \wr$

```
while true do
   xs[0] := 42
```

Proof:

$xs_0 = \mathsf{xs} \in (0 \,..\, k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner$

$\Rightarrow$ ⟨ ? ⟩

?

$\Rightarrow[$ while true do xs := xs ⊕ { ⟨ 0, 42 ⟩ } od

$]$ ⟨ "While" with subproof:

?

$\Rightarrow[$ xs := xs ⊕ { ⟨ 0, 42 ⟩ } $]$

⟨ ? ⟩

?

⟩

?

$\Rightarrow$ ⟨ ? ⟩

$\mathsf{xs} \in (0 \,..\, k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner \;\wedge\; \mathsf{sorted\ xs}$

$\quad \wedge\, \wr\, p \mid p \in \mathsf{xs} \bullet \mathsf{snd}\, p\, \wr = \wr\, p \mid p \in xs_0 \bullet \mathsf{snd}\, p\, \wr$

Where do we flag the invariant?

A Verified Sorting Algorithm

Theorem "Sorting 0′ ":

$xs_0 = \mathsf{xs} \in (0 \,..\, k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner$

$\Rightarrow[$ while true do

xs := xs ⊕ { ⟨ 0, 42 ⟩ }

od

$]$

$\mathsf{xs} \in (0 \,..\, k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner \;\wedge\; \mathsf{sorted\ xs}$

$\quad \wedge\, \wr\, p \mid p \in \mathsf{xs} \bullet \mathsf{snd}\, p\, \wr = \wr\, p \mid p \in xs_0 \bullet \mathsf{snd}\, p\, \wr$

```
while true do
   xs[0] := 42
```

Proof:

$xs_0 = \mathsf{xs} \in (0 \,..\, k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner$

$\Rightarrow$ ⟨ ? ⟩

Q — **Invariant**

$\Rightarrow[$ while true do xs := xs ⊕ { ⟨ 0, 42 ⟩ } od

$]$ ⟨ "While" with subproof:

?

$\Rightarrow[$ xs := xs ⊕ { ⟨ 0, 42 ⟩ } $]$

⟨ ? ⟩

?

⟩

?

$\Rightarrow$ ⟨ ? ⟩

$\mathsf{xs} \in (0 \,..\, k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner \;\wedge\; \mathsf{sorted\ xs}$

$\quad \wedge\, \wr\, p \mid p \in \mathsf{xs} \bullet \mathsf{snd}\, p\, \wr = \wr\, p \mid p \in xs_0 \bullet \mathsf{snd}\, p\, \wr$

Which other conditions ere determined by the invariant?

A Verified Sorting Algorithm

Theorem "Sorting 0′ ":

$xs_0 = \mathsf{xs} \in (0 \,..\, k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner$

$\Rightarrow[$ while true do

xs := xs ⊕ { ⟨ 0, 42 ⟩ }

od

$]$

$\mathsf{xs} \in (0 \,..\, k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner \;\wedge\; \mathsf{sorted\ xs}$

$\quad \wedge\, \wr\, p \mid p \in \mathsf{xs} \bullet \mathsf{snd}\, p\, \wr = \wr\, p \mid p \in xs_0 \bullet \mathsf{snd}\, p\, \wr$

```
while true do
   xs[0] := 42
```

Proof:

$xs_0 = \mathsf{xs} \in (0 \,..\, k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner$

$\Rightarrow$ ⟨ ? ⟩

Q — Invariant

$\Rightarrow[$ while true do xs := xs ⊕ { ⟨ 0, 42 ⟩ } od

$]$ ⟨ "While" with subproof:

true ∧ Q

$\Rightarrow[$ xs := xs ⊕ { ⟨ 0, 42 ⟩ } $]$

⟨ ? ⟩

Q

⟩

¬ true ∧ Q

$\Rightarrow$ ⟨ ? ⟩

$\mathsf{xs} \in (0 \,..\, k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner \;\wedge\; \mathsf{sorted\ xs}$

$\quad \wedge\, \wr\, p \mid p \in \mathsf{xs} \bullet \mathsf{snd}\, p\, \wr = \wr\, p \mid p \in xs_0 \bullet \mathsf{snd}\, p\, \wr$

Can we already complete some proof obligations now, without even fixing the invariant?

A Verified Sorting Algorithm

Theorem "Sorting 0′ ":

$xs_0 = \mathsf{xs} \in (0 \,..\, k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner$

$\Rightarrow[$ while true do

xs := xs ⊕ { ⟨ 0, 42 ⟩ }

od

$]$

$\mathsf{xs} \in (0 \,..\, k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner \;\wedge\; \mathsf{sorted\ xs}$

$\quad \wedge\, \wr\, p \mid p \in \mathsf{xs} \bullet \mathsf{snd}\, p\, \wr = \wr\, p \mid p \in xs_0 \bullet \mathsf{snd}\, p\, \wr$

```
while true do
   xs[0] := 42
```

Proof:

$xs_0 = \mathsf{xs} \in (0 \,..\, k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner$

$\Rightarrow$ ⟨ ? ⟩

Q — Invariant

$\Rightarrow[$ while true do xs := xs ⊕ { ⟨ 0, 42 ⟩ } od

$]$ ⟨ "While" with subproof:

true ∧ Q

$\Rightarrow[$ xs := xs ⊕ { ⟨ 0, 42 ⟩ } $]$

⟨ ? ⟩

Q

⟩

¬ true ∧ Q

$\Rightarrow$ ⟨ "Definition of `false` ", "Zero of ∧", "ex falso quodlibet" ⟩

$\mathsf{xs} \in (0 \,..\, k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner \;\wedge\; \mathsf{sorted\ xs}$

$\quad \wedge\, \wr\, p \mid p \in \mathsf{xs} \bullet \mathsf{snd}\, p\, \wr = \wr\, p \mid p \in xs_0 \bullet \mathsf{snd}\, p\, \wr$

How can we choose the invariant to make the remaining proof obligations easy?

A Verified Sorting Algorithm

Theorem "Sorting 0′ ":

$xs_0 = \mathsf{xs} \in (0 \,..\, k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner$

$\Rightarrow[$ while true do

xs := xs ⊕ { ⟨ 0, 42 ⟩ }

od

$]$

$\mathsf{xs} \in (0 \,..\, k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner \;\wedge\; \mathsf{sorted\ xs}$

$\quad \wedge\, \wr\, p \mid p \in \mathsf{xs} \bullet \mathsf{snd}\, p\, \wr = \wr\, p \mid p \in xs_0 \bullet \mathsf{snd}\, p\, \wr$

```
while true do
   xs[0] := 42
```

Proof:

$xs_0 = \mathsf{xs} \in (0 \,..\, k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner$

$\Rightarrow$ ⟨ "Right-zero of ⇒" ⟩

true — Invariant

$\Rightarrow[$ while true do xs := xs ⊕ { ⟨ 0, 42 ⟩ } od

$]$ ⟨ "While" with subproof:

true ∧ true

$\Rightarrow[$ xs := xs ⊕ { ⟨ 0, 42 ⟩ } $]$

⟨ "Idempotency of ∧", "Assignment" with substitution ⟩

true

⟩

¬ true ∧ true

$\Rightarrow$ ⟨ "Contradiction", "ex falso quodlibet" ⟩

$\mathsf{xs} \in (0 \,..\, k) \mathrel{-\!\!\circ\!\!\rightarrow} \llcorner\mathbb{N}\lrcorner \;\wedge\; \mathsf{sorted\ xs}$

$\quad \wedge\, \wr\, p \mid p \in \mathsf{xs} \bullet \mathsf{snd}\, p\, \wr = \wr\, p \mid p \in xs_0 \bullet \mathsf{snd}\, p\, \wr$

This program has herewith been proven partially correct with respect to our sorting algorithm specification.

Partial Correctness: "Terminate Only in States Satisfying Postcondition"

Axiom "Partial Correctness": $(P \Rightarrow[\, C \,]\, Q) \quad \equiv \quad [\![C]\!]\, (\!|\, \mathsf{sat}\, P \,|\!) \subseteq \mathsf{sat}\, Q$

Axiom "Semantics of `while` ": $[\![\mathsf{while}\ B\ \mathsf{do}\ C\ \mathsf{od}]\!] = (\mathsf{sat}\, B \lhd [\![C]\!])^* \rhd \mathsf{sat}\, \overline{B}$

Theorem "Partial correctness of `while true` ": $P \Rightarrow[\, \mathsf{while}\ true'\ \mathsf{do}\ C\ \mathsf{od}\,]\, Q$

Proof:

$P \Rightarrow[\, \mathsf{while}\ true'\ \mathsf{do}\ C\ \mathsf{od}\,]\, Q$

≡ ⟨ "Partial correctness" ⟩

$[\![\mathsf{while}\ true'\ \mathsf{do}\ C\ \mathsf{od}]\!]\, (\!|\, \mathsf{sat}\, P \,|\!) \subseteq \mathsf{sat}\, Q$

≡ ⟨ "Semantics of `while` " ⟩

$((\mathsf{sat}\, true' \lhd [\![C]\!])^* \rhd \mathsf{sat}\, \overline{true'})\, (\!|\, \mathsf{sat}\, P \,|\!) \subseteq \mathsf{sat}\, Q$

≡ ⟨ "sat true′ " ⟩

$((U \lhd [\![C]\!])^* \rhd \overline{U})\, (\!|\, \mathsf{sat}\, P \,|\!) \subseteq \mathsf{sat}\, Q$

≡ ⟨ " $\rhd \overline{U}$ " ⟩

$\{\}\, (\!|\, \mathsf{sat}\, P \,|\!) \subseteq \mathsf{sat}\, Q$

≡ ⟨ "Relational image under {}" ⟩

$\{\} \subseteq \mathsf{sat}\, Q$ — **This is** "Empty set is least"

That is:

Any "while *true*" loop is partially correct with respect to any pre-post-condition specification.

Domain and Range Relation-algebraically

- In the abstract relation-algebraic setting, we are only dealing with **relation types** $A \leftrightarrow B$
- No set types, and therefore no direct way to express *Dom*, ◁ , (\| _ \|), etc.
- One candidate for "relations representing sets" are subidentities, $q \subseteq \mathbb{I}$
- In set theory, id$A$ is a relation that can just serve as a representation of set A
- id allows us to define ◁ :

  Theorem (14.237) "Domain restriction via ⨾": $A \lhd R = \mathrm{id}\, A \,⨾\, R$
- In the abstract relation-algebraic setting, the role of the operation

  $Dom : (A \leftrightarrow B) \to set\ A$

  is taken by the new operation

  $\mathsf{dom} : (A \leftrightarrow B) \to (A \leftrightarrow A)$

  $\mathsf{dom}\, R = R \,⨾\, R^{\smile} \cap \mathbb{I}$

  taking each relation R to the subidentity relation representing the set *Dom* R
- In set theory:

  $\mathsf{dom}\, R = \mathsf{id}\, (\mathsf{Dom}\, R)$

⟹ H18, H19

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-11-22

Part 2: Total Correctness

Precondition-Postcondition Specifications in Dynamic Logic Notation

- Program correctness statement in LADM (and much current use): "Hoare triple":
  $$\{ P \} \, C \, \{ Q \}$$
  Meaning (LADM ch. 10): "Total correctness":
  If command C is started in a state in which the **precondition** P holds
  then it will terminate in a state in which the **postcondition** Q holds.
- So far, we have been using the **dynamic logic** notation:
  $$P \Rrightarrow [\, C \,] \, Q$$
  with its **partial correctness meaning:**
  If command C is started in a state in which the **precondition** P holds
  then it will terminate **only in states** in which the **postcondition** Q holds.

Differences between partial and total correctness:
Commands that do not terminate properly:

- Commands that crash — evaluating undefined expressions
- Infinite loops

Undefined Behaviors in C

- Spatial memory safety violations — **int** $a[5]$; **int** $k = a[6]$;
- Temporal memory safety violations — **int** a; **int** $b = a + 1$;
- Integer overflow — $k = maxint + 2$; $m = minint - 3$;
- Strict aliasing violations
- Alignment violations
- Unsequenced modifications — *printf*("%d␣%d", a++, a++);
- Data races
- Loops that neither perform I/O nor terminate

Rules That Work for Both

Sequential composition:

```
Primitive inference rule “Sequence”:
    `P  ⇒[ C₁ ]  Q`,  `Q  ⇒[ C₂ ]  R`
  ⊢————————————————————————————————————
      `P  ⇒[ C₁ ; C₂ ]  R`
```

Strengthening the precondition:

```
     `P₁ ⇒ P₂`,    `P₂ ⇒[ C ] Q`
  ⊢————————————————————————————————
          `P₁ ⇒[ C ] Q`
```

Weakening the postcondition:

```
     `P ⇒[ C ] Q₁`,    `Q₁ ⇒ Q₂`
  ⊢————————————————————————————————
          `P ⇒[ C ] Q₂`
```

Total Correctness Rule for Assignment

Used so far: **Dynamic Logic Partial Correctness Assignment Axiom:**

$$Q[x := E] \quad \Rrightarrow [\, x \mathbin{\mathbf{:=}} E \,] \quad Q$$

Assignment ":=": Two characters; type ":="

LADM Total Correctness Assignment Axiom (10.1):

$$\{ \mathit{dom} \text{ ‘}E\text{’} \wedge Q[x := E] \} \quad x \mathbin{\mathbf{:=}} E \quad \{ Q \}$$

Substitution "≔": One Unicode character; type "\:="

For each *programming-language* expression E, the predicate
dom **‘***E***’**
is satisfied exactly in the states in which E is defined.
(*dom* is a ***meta-function*** taking expressions to Boolean conditions.)

Examples:

- $\mathit{dom} \text{ ‘}\mathit{sqrt}\ (x \,/\, y)\text{’} \quad \equiv \quad y \neq 0 \wedge x \,/\, y \geq 0$
- $\mathit{dom} \text{ ‘}a \mathbin{@} i\text{’} \quad \equiv \quad i \in \mathit{Dom}\ a$
- For *int*-variables i and j:
  $\mathit{dom} \text{ ‘}i + j\text{’} \quad \equiv \quad \mathit{minint} \leq x + y \leq \mathit{maxint}$

Conditional Rule

Each evaluation of an expression E needs to be guarded by a precondition *dom* **‘***E***’**:

$$\frac{\{ B \wedge P \} \quad C_1 \quad \{ Q \} \qquad\qquad \{ \neg B \wedge P \} \quad C_2 \quad \{ Q \}}{\{ \mathit{dom} \text{ ‘}B\text{’} \wedge P \} \quad \mathit{if}\ B\ \mathit{then}\ C_1\ \mathit{else}\ C_2\ \mathit{fi} \quad \{ Q \}}$$

"While" Rule

So far:

```
            `B ∧ Q  ⇒[ C ]  Q`
  ⊢————————————————————————————————————————
     `Q  ⇒[ while B do C od ]  ¬ B ∧ Q`
```

Now **two** additional ingredients:

- **Invariant:** $Q : \mathbb{B}$ — as before, ensuring functional correctness
- **Variant** (or "bound function"): $T : \mathbb{Z}$ — ensuring termination

$$\frac{\{ B \wedge Q \} \quad C \quad \{ Q \} \qquad \{ B \wedge Q \wedge T = t_0 \} \quad C \quad \{ T < t_0 \} \qquad B \wedge Q \Rightarrow T > 0}{\{ \mathit{dom} \text{ ‘}B\text{’} \wedge Q \} \quad \mathit{while}\ B\ \mathit{do}\ C\ \mathit{od} \quad \{ \neg B \wedge Q \}}$$

In each iteration:

- The invariant Q is preserved.
- The variant T decreases.

Termination: The relation $<$ on the subset $\{t : \mathbb{Z} \mid t > 0\}$ is well-founded.

"Merged" While Rule

Now **two** additional ingredients:

- **Invariant:** $Q : \mathbb{B}$ — as before, ensuring functional correctness
- **Variant** (or "bound function"): $T : \mathbb{Z}$ — ensuring termination

$$\frac{\{ B \wedge Q \wedge T = t_0 \} \quad C \quad \{ Q \wedge T < t_0 \} \qquad B \wedge Q \Rightarrow T > 0}{\{ \mathit{dom} \text{ ‘}B\text{’} \wedge Q \} \quad \mathit{while}\ B\ \mathit{do}\ C\ \mathit{od} \quad \{ \neg B \wedge Q \}} \text{ prov. } \neg \mathit{occurs}(\text{‘}t_0\text{’}, \text{‘}B, C, Q, T\text{’})$$

In each iteration:

- The invariant Q is preserved.
- The variant T decreases.

Recall: Total Correctness versus Partial Correctness

- Program correctness statement in LADM (and much current use): "Hoare triple":
  $$\{ P \} \, C \, \{ Q \}$$
  Meaning (LADM ch. 10): "Total correctness":
  If command C is started in a state in which the **precondition** P holds
  then it **will terminate** in a state in which the **postcondition** Q holds.
- So far, we have been using the **dynamic logic** notation:
  $$P \Rrightarrow [\, C \,] \, Q$$
  with its **partial correctness meaning:**
  If command C is started in a state in which the **precondition** P holds
  then it will terminate **only** in a state in which the **postcondition** Q holds.

Differences between partial and total correctness:
Commands that do not terminate properly:

- Commands that crash — evaluating undefined expressions
- Infinite loops

Relation-Algebraic Total and Partial Correctness

- Program correctness statement in LADM (and much current use): "Hoare triple":
  $$\{ P \} \, C \, \{ Q \}$$
  Meaning (LADM ch. 10): "Total correctness":
  If command C is started in a state in which the **precondition** P holds
  then it **will terminate** in a state in which the **postcondition** Q holds.

Axiom "Total Correctness":

$$(P \Rrightarrow [\langle C \rangle] \, Q) \quad \equiv \quad \mathsf{sat}\ P \subseteq \mathsf{Dom}\ [\![C]\!] \quad \wedge \quad [\![C]\!] \,(\!|\, \mathsf{sat}\ P \,|\!)\, \subseteq \mathsf{sat}\ Q$$

- So far, we have been using the **dynamic logic** notation:
  $$P \Rrightarrow [\, C \,] \, Q$$
  with its **partial correctness meaning:**
  If command C is started in a state in which the **precondition** P holds
  then it will terminate **only** in a state in which the **postcondition** Q holds.

Axiom "Partial Correctness":

$$(P \Rrightarrow [\, C \,] \, Q) \quad \equiv \quad [\![C]\!] \,(\!|\, \mathsf{sat}\ P \,|\!)\, \subseteq \mathsf{sat}\ Q$$

Total and Partial Correctness in Predicate Logic

- Program correctness statement in LADM (and much current use): "Hoare triple":
  $$\{ P \} \, C \, \{ Q \}$$
  Meaning (LADM ch. 10): "Total correctness":
  If command C is started in a state in which the **precondition** P holds
  then it **will terminate** in a state in which the **postcondition** Q holds.

Theorem "Total Correctness":

$$(P \Rrightarrow [\langle C \rangle] \, Q)$$
$$\equiv (\forall s_1 \mid s_1 \in \mathsf{sat}\ P \bullet \exists s_2 \mid s_1 \,(\, [\![C]\!] \,)\, s_2 \bullet s_2 \in \mathsf{sat}\ Q)$$
$$\wedge (\forall s_1, s_2 \bullet s_1 \in \mathsf{sat}\ P \wedge s_1 \,(\, [\![C]\!] \,)\, s_2 \Rightarrow s_2 \in \mathsf{sat}\ Q)$$

- So far, we have been using the **dynamic logic** notation:
  $$P \Rrightarrow [\, C \,] \, Q$$
  with its **partial correctness meaning:**
  If command C is started in a state in which the **precondition** P holds
  then it will terminate **only** in a state in which the **postcondition** Q holds.

Theorem "Partial Correctness":

$$(P \Rrightarrow [\, C \,] \, Q)$$
$$\equiv \forall s_1, s_2 \bullet s_1 \in \mathsf{sat}\ P \wedge s_1 \,(\, [\![C]\!] \,)\, s_2 \Rightarrow s_2 \in \mathsf{sat}\ Q$$

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-11-24

Temporal Logic: PLTL

Syntax and Semantics of Propositional Logic

- Given: A set $\mathcal{P}$ of **proposition symbols** $p, q, \ldots$
- A **propositional formula** $\varphi, \psi, \ldots$ is (an abstract syntax tree) generated by the following "grammar" (informal):
  $\varphi ::= T \mid F \mid p \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \Rightarrow \psi$
- A **state** is a function $\alpha : \mathcal{P} \to \mathbb{B}$
- The semantics of propositional formula φ is the function
  $[\![\varphi]\!] : (\mathcal{P} \to \mathbb{B}) \to \mathbb{B}$
  that maps each state α to a truth value, the "value of φ in α":
  $$[\![T]\!]\ \alpha = true$$
  $$[\![\neg\varphi]\!]\ \alpha = \neg([\![\varphi]\!]\ \alpha)$$
  $$[\![\varphi \wedge \psi]\!]\ \alpha = [\![\varphi]\!]\ \alpha \wedge [\![\psi]\!]\ \alpha$$
- α **satisfies** φ iff $[\![\varphi]\!]\ \alpha = true$; this is also written: $\alpha \models \varphi$
- φ **is valid** iff $(\forall \alpha \bullet [\![\varphi]\!]\ \alpha = true)$; this is also written: $\models \varphi$

Syntax and Semantics of Propositional Logic — Applications

- Define a (Haskell) datatype for propositional formule: *data PropForm p = ...*
- Write functions that takes each formula to its disjunctive/conjunctive normal form

```
toCNF, toDNF :: PropForm p → PropForm p
```

  Use CalcCheck to prove that your implementations are correct
- Define the semantics as an evaluation function

```
evalPropForm :: PropForm p → State p → Bool
```

- Define a representation of truth tables
- Write a truth table generation fucntion
- Write a validity checker using truth tables

```
validPropForm :: PropForm p → Bool
```

- Write a satisfiability checker using truth tables

```
satPropForm :: PropForm p → Maybe (State p)
```

- Look up the DPLL algorithm and write a more efficient satisfiability solver

Syntax and Semantics of Predicate Logic

- Given: A **vocabulary/signature** Σ consisting of
  - a countably infinite set of **variable symbols** $v, v_1, v_2, \ldots$
  - a countable set of **function symbols** $f, g, \ldots$ (with arity information)
  - a countable set of **predicate symbols** $p, q, \ldots$ (with arity information)
- A **term** t, t_1, t_2 is (an abstract syntax tree) generated by the following "grammar":
  $t ::= f(t_1, \ldots, t_n)$
- A **predicate-logic/first-order-logic formula** $\varphi, \psi, \ldots$ is (an abstract syntax tree) generated by the following "grammar":
  $\varphi ::= p(t_1, \ldots, t_n) \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \Rightarrow \psi \mid (\forall\ v \bullet \varphi) \mid (\exists\ v \bullet \varphi)$
- An **interpretation of** Σ **/** Σ**-structure** $\mathcal{A}$ consists of
  - a **domain** set D
  - a mapping of function symbols f to functions $f^{\mathcal{A}} : D^n \to D$
  - a mapping of predicate symbols p to functions $p^{\mathcal{A}} : D^n \to \mathbb{B}$
- A **variable assignment** for $\mathcal{A}$ is a function $\alpha : \mathcal{V} \to D$
- Semantics of terms: $[\![t]\!]_{\mathcal{A}} : (\mathcal{V} \to D) \to D$
- Semantics of formulae: $[\![\varphi]\!]_{\mathcal{A}} : (\mathcal{V} \to D) \to \mathbb{B}$; we write "$\mathcal{A}, \alpha \models \varphi$" for $[\![\varphi]\!]_{\mathcal{A}}\ \alpha = true$
- … $\longrightarrow$ **RSD chapters 3, 4**

Infinite Program Executions

- Even simple imperative programming languages have programs that do not terminate — **while** *true* **do** …
- Not all programs are expected to terminate:
  - Operating systems
  - Bank databases
  - Online shops
- Pre-postcondition specifications are useless for programs that are expected to not terminate!
- Different patterns of specification are used for such systems:
  - Each request will generate a response
  - The ledger is always balanced
  - Shipping commands are sent to the warehouse only after payment is confirmed
- Central concept: **Time**
- System behaviour: Different states at different time points
- Plausible abstraction: Discrete time, with time points taken from $\mathbb{N}$
- **Infinite state sequences**: Functions of type $\mathbb{N} \to$ State

How to Reason About Infinite state sequences?

- **Infinite state sequences**: Functions of type $\mathbb{N} \to$ State
- Specification example sketches in predicate logic:
  - $\forall\ t_0, rId, d_{in} \mid request(rId, d_{in}, t_0)$
    $\bullet\ \exists\ t_1, d_{out} \mid t_0 < t_1 \bullet response(rId, d_{out}, t_1)$
    $\wedge\ appropriate(d_{out}, d_{in})$
  - $\forall\ t \bullet (\sum\ a : Account \bullet balance\ a\ t) = 0$
  - …
- **Lots of quantification about time points!**
- **Quantification about time points follows relatively few patterns!**
- **Temporal logics** **"internalise"** these time point quantification patterns and allow to express them **without bound variables for time points**.

Syntax and Semantics of Propositional Linear-Time Temporal Logic (PLTL)

- Given: A set A of **atomic propositions** $p, q, \ldots$
- A **PLTL formula** $\varphi, \psi, \ldots$ is (an abstract syntax tree) generated by the following "grammar" (informal):
  $\varphi ::= T \mid F \mid p \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \Rightarrow \psi \mid F\ \varphi \mid G\ \varphi \mid X\ \varphi \mid \varphi\ U\ \psi$
- A **state** associates a truth value with each atom: State $= A \to \mathbb{B}$
- A **time line** α associates a state with each time point — for simplicity, we use $\mathbb{N}$ for time points:
  $\alpha : \mathbb{N} \to A \to \mathbb{B}$
- Given an LTL formula φ and a time line α, the semantics of φ in α, written "$[\![\varphi]\!]\ \alpha$", is a function that associates with each time point $t : \mathbb{N}$ the truth value "$[\![\varphi]\!]\ \alpha\ t$":

  Declaration: $[\![\_]\!]$: LTL $A \to (\mathbb{N} \to A \to \mathbb{B}) \to \mathbb{N} \to \mathbb{B}$

Syntax and Semantics of Propositional Linear-Time Temporal Logic (PLTL) 1

$[\![\varphi]\!]\ \alpha\ t = true$ iff LTL formula φ holds in time line $\alpha : \mathbb{N} \to A \to \mathbb{B}$ at time t:

Declaration: $[\![\_]\!]$: LTL $A \to (\mathbb{N} \to A \to \mathbb{B}) \to \mathbb{N} \to \mathbb{B}$

An atomic proposition p is true at time t iff the time line contains, at time t, a state in which p is true:

"Semantics of LTL atoms": $[\![\text{‘}\ p\ ]\!]\ \alpha\ t \equiv \alpha\ t\ p$

"Semantics of LTL ¬": $[\![\neg'\ \varphi]\!]\ \alpha\ t \equiv \neg [\![\varphi]\!]\ \alpha\ t$

"Semantics of LTL ∧": $[\![\varphi \wedge' \psi]\!]\ \alpha\ t \equiv [\![\varphi]\!]\ \alpha\ t \wedge [\![\psi]\!]\ \alpha\ t$

"Semantics of LTL ∨": $[\![\varphi \vee' \psi]\!]\ \alpha\ t \equiv [\![\varphi]\!]\ \alpha\ t \vee [\![\psi]\!]\ \alpha\ t$

"Semantics of LTL ⇒": $[\![\varphi \Rightarrow' \psi]\!]\ \alpha\ t \equiv [\![\varphi]\!]\ \alpha\ t \Rightarrow [\![\psi]\!]\ \alpha\ t$

α =

Time	p	q	r	s
0	√		√	
1	√	√		
2	√		√	
3		√		
4	√		√	
5	√	√		√
6, 16, 26, …	√		√	√
7, 17, 27, …	√	√		
8, 18, 28, …	√		√	
9, 19, 29, …	√	√	√	
10, 20, 30, …	√		√	
11, 21, 31, …	√	√		
12, 22, 32, …	√		√	
13, 23, 33, …	√	√		
14, 24, 34, …	√		√	
15, 25, 35, …	√	√		

- $[\![p]\!]\ \alpha\ 0 = ?$
- $[\![p]\!]\ \alpha\ 3 = ?$
- $[\![q]\!]\ \alpha\ 0 = ?$
- $[\![p \wedge q]\!]\ \alpha\ 0 = ?$
- $[\![p \vee \neg q]\!]\ \alpha\ 3 = ?$
- $[\![q \Rightarrow r]\!]\ \alpha\ 42 = ?$

Syntax and Semantics of Propositional Linear-Time Temporal Logic (PLTL) 2

$[\![\varphi]\!]\ \alpha\ t = true$ iff LTL formula φ holds in time line $\alpha : \mathbb{N} \to A \to \mathbb{B}$ at time t:

Declaration: $[\![\_]\!]$: LTL $A \to (\mathbb{N} \to A \to \mathbb{B}) \to \mathbb{N} \to \mathbb{B}$

$F\ \varphi$ is true at time t if φ is true at some time $t' \geq t$:

"Semantics of `F`":
$$[\![F\ \varphi]\!]\ \alpha\ t \equiv \exists t' : \mathbb{N} \mid t \leq t' \bullet [\![\varphi]\!]\ \alpha\ t'$$

$G\ \varphi$ is true at time t if φ is true at all times $t' \geq t$.

"Semantics of `G`":
$$[\![G\ \varphi]\!]\ \alpha\ t \equiv \forall t' : \mathbb{N} \mid t \leq t' \bullet [\![\varphi]\!]\ \alpha\ t'$$

α =

Time	p	q	r	s
0	√		√	
1	√	√		
2	√		√	
3		√		
4	√		√	
5	√	√		√
6, 16, 26, …	√		√	√
7, 17, 27, …	√	√		
8, 18, 28, …	√		√	
9, 19, 29, …	√	√	√	
10, 20, 30, …	√		√	
11, 21, 31, …	√	√		
12, 22, 32, …	√		√	
13, 23, 33, …	√	√		
14, 24, 34, …	√		√	
15, 25, 35, …	√	√		

- $[\![G\ p]\!]\ \alpha\ 0 = ?$
- $[\![G\ p]\!]\ \alpha\ 5 = ?$
- $[\![F\ q]\!]\ \alpha\ 0 = ?$
- $[\![F\ s]\!]\ \alpha\ 7 = ?$
- $[\![F\ \neg p]\!]\ \alpha\ 0 = ?$
- $[\![F\ \neg p]\!]\ \alpha\ 100 = ?$

Syntax and Semantics of Propositional Linear-Time Temporal Logic (PLTL) 3

$[\![\varphi]\!]\ \alpha\ t = true$ iff LTL formula φ holds in time line $\alpha : \mathbb{N} \to A \to \mathbb{B}$ at time t:

Declaration: $[\![\_]\!]$: LTL $A \to (\mathbb{N} \to A \to \mathbb{B}) \to \mathbb{N} \to \mathbb{B}$

$X\ \varphi$ is true at time t iff φ is true at time $t + 1$:

"Semantics of `X`":
$$[\![X\ \varphi]\!]\ \alpha\ t \equiv [\![\varphi]\!]\ \alpha\ (\mathsf{suc}\ t)$$

α =

Time	p	q	r	s
0	√		√	
1	√	√		
2	√		√	
3		√		
4	√		√	
5	√	√		√
6, 16, 26, …	√		√	√
7, 17, 27, …	√	√		
8, 18, 28, …	√		√	
9, 19, 29, …	√	√	√	
10, 20, 30, …	√		√	
11, 21, 31, …	√	√		
12, 22, 32, …	√		√	
13, 23, 33, …	√	√		
14, 24, 34, …	√		√	
15, 25, 35, …	√	√		

- $[\![X\ p]\!]\ \alpha\ 0 = ?$
- $[\![X\ q]\!]\ \alpha\ 0 = ?$
- $[\![q \wedge X\ r]\!]\ \alpha\ 1 = ?$
- $[\![G\ F\ (q \wedge X\ r)]\!]\ \alpha\ 0 = ?$
- $[\![F\ (s \wedge X\ s)]\!]\ \alpha\ 0 = ?$
- $[\![F\ (s \wedge X\ s)]\!]\ \alpha\ 10 = ?$
- $[\![G\ (q \equiv X\ r)]\!]\ \alpha\ 12 = ?$
- $[\![G\ F\ (q \equiv X\ r)]\!]\ \alpha\ 12 = ?$

Syntax and Semantics of Propositional Linear-Time Temporal Logic (PLTL) 4

$[\![\varphi]\!]\ \alpha\ t = true$ iff LTL formula φ holds in time line $\alpha : \mathbb{N} \to A \to \mathbb{B}$ at time t:

Declaration: $[\![\_]\!]$: LTL $A \to (\mathbb{N} \to A \to \mathbb{B}) \to \mathbb{N} \to \mathbb{B}$

$\varphi\ U\ \psi$ is true at time t if ψ is true at some time $t' \geq t$, and for all times t'' such that $t \leq t'' < t'$, φ is true.

Axiom "Semantics of 'U'": "until"

$$[\![\varphi\ U\ \psi]\!]\ \alpha\ t \equiv \exists t' : \mathbb{N} \mid t \leq t' \bullet [\![\psi]\!]\ \alpha\ t' \wedge \forall t'' : \mathbb{N} \mid t \leq t'' < t' \bullet [\![\varphi]\!]\ \alpha\ t''$$

- $[\![p\ U\ q]\!]\ \alpha\ 0 = ?$
- $[\![p\ U\ s]\!]\ \alpha\ 0 = ?$
- $[\![\neg s\ U \neg p]\!]\ \alpha\ 0 = ?$
- $[\![p\ U\ (q \wedge r)]\!]\ \alpha\ 42 = ?$
- $[\![p\ U\ (q \wedge s)]\!]\ \alpha\ 42 = ?$
- $[\![(p \vee r)\ U\ s]\!]\ \alpha\ 1 = ?$

$\alpha =$

Time	p	q	r	s
0	✓		✓	
1	✓	✓		
2	✓		✓	
3		✓		
4	✓		✓	
5	✓	✓		✓
6, 16, 26, ...	✓		✓	✓
7, 17, 27, ...	✓	✓		
8, 18, 28, ...	✓		✓	
9, 19, 29, ...	✓	✓	✓	
10, 20, 30, ...	✓		✓	
11, 21, 31, ...	✓	✓		
12, 22, 32, ...	✓		✓	
13, 23, 33, ...	✓	✓		
14, 24, 34, ...	✓		✓	
15, 25, 35, ...	✓	✓		

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-11-27

Frama-C and ACSL

Frama-C: `https://www.frama-c.com/`

> Frama-C is an open-source extensible and collaborative platform dedicated to source-code analysis of C software. The Frama-C analyzers assist you in various source-code-related activities, from the navigation through unfamiliar projects up to the certification of critical software.

- Platform with multiple plug-ins
- Plug-in for total correctness proofs: WP
- Specification language: ACSL "ANSI C Specificatiion Language"
  - Similar to JML
  - Based on first-order predicate logic
  - Not all ACSL features are currently supported by Frama-C and WP

Frama-C and ACSL — `https://www.frama-c.com/`

Frama-C: An industrially-used framework for C code analysis and verification

- Delegates "simple" proofs to external tools, mostly **S**atisfiability-**M**odulo-**T**heories solvers (e.g., Z3)
- Practical Program Proof = Verification Condition Generation (VCG) + SMT checking

ACSL: ANSI-**C S**pecification **Language**

- Similar to the JML — **J**ava **M**odelling **L**anguage
- But Java is more complex:
  Statements that can raise exceptions need additional postconditions for those.
- ACSL "is" standard first-order predicate logic in C syntax.
- ACSL allows definition of inductive datatypes
  — natural abstractions for specification, but rather clumsy in ACSL

  — From discrete math to C: **A big gap to bridge!**

Start reading:
`https://allan-blanchard.fr/publis/frama-c-wp-tutorial-en.pdf`

ACSL Function Contracts

Overall program correctness is based on **function contracts**, mainly:

- "**requires**": Procedure call precondition
- "**assigns**": Global variables that may be updated
- "**ensures**": Procedure call postcondition
  May refer to **\result** for the return value.

Contracts of exported functions are part of the module interface, and therefore should be in the module interface file (`*.h`).

`all_zeros.h`:

```
/*@ requires  n ≥ 0  ∧  \valid(t + (0.. n−1));
    assigns   \nothing;
    ensures   \result ≠ 0  ⇔  (∀ integer j; 0 ≤ j < n ⇒ t[j] ≡ 0);
*/
int all_zeros(int *t, int n);
```

ACSL Loop Annotations

Total correctness **While** rule:

$$\frac{\{ B \wedge Q \wedge T = t_0 \}\quad C\quad \{ Q \wedge T < t_0 \} \qquad B \wedge Q \Rightarrow T > 0}{\{ dom\ `B' \wedge Q \}\quad while\ B\ do\ C\ od\quad \{ \neg B \wedge Q \}}\ \text{prov.}\ \neg occurs(`t_0', `B, C, Q, T')$$

"**loop invariant** Q": Property always true in the following loop

- true at loop entry, at each loop iteration, at loop exit
- usually contains a generalisation of the post-condition
- may need to contain additional "sanity" conditions

"**loop assigns** *footprint*": What may be assigned to within the loop

"**loop variant** T": To prove termination:

- Integer metric T that is **strictly decreasing** at each iteration and **bounded** by 0

all_zeros

`all_zeros.c`:

```
/*@ requires  n ≥ 0  ∧  \valid(t + (0.. n−1));
    assigns   \nothing;
    ensures   \result ≠ 0  ⇔  (∀ integer j; 0 ≤ j < n ⇒ t[j] ≡ 0);
*/
int all_zeros(int *t, int n) {
  int k=0;
  /*@ loop invariant 0 ≤ k ≤ n;
      loop invariant ∀ integer j; 0 ≤ j < k ⇒ t[j] ≡ 0;
      loop assigns   k;
      loop variant   n − k;
  */
  while(k < n){
    if (t[k] ≠ 0)
      return 0;
    k++;
  }
  return 1;
}
```

findMax Attempt 1

`findMax1.c`:

```
/*@ requires n > 0;
    requires \valid(a + (0 .. n − 1));
    ensures ∀ integer i ; 0 ≤ i < n ⇒ \result ≥ a[i];
    ensures ∃ integer i ; 0 ≤ i < n ⇒ \result ≡ a[i];
 */
int findMax(int n, int a[]) {
  int i;
  /*@ loop invariant ∀ integer j ; 0 ≤ j < i ⇒ a[j] ≡ 0;
      loop invariant 0 ≤ i ≤ n;
      loop variant n − i;
  */
  for( i = 0; i < n; i++) a[i] = 0;
  return 0;
}
```

`frama-c-gui -wp findMax1.c`
`frama-c -wp findMax1.c`
`frama-c-gui -wp -wp-rte findMax1.c`
`frama-c -wp -wp-rte findMax1.c`

"RTE": Run-time exceptions (include undefined behaviour)

The *findMax* Attempt 1a

`findMax1a.c`:

```
/*@ requires n > 0;
    requires \valid(a + (0 .. n − 1));
    ensures ∀ integer i ; 0 ≤ i < n ⇒ \result ≥ a[i];
    ensures ∃ integer i ; 0 ≤ i < n ⇒ \result ≡ a[i];
 */
int findMax(int n, int a[]) {
  int i;
  /*@ loop invariant ∀ integer j ; 0 ≤ j < i ⇒ a[j] ≡ 0;
      loop invariant 0 ≤ i ≤ n;
      loop assigns i, a[0 .. n − 1];
      loop variant n − i;
  */
  for( i = 0; i < n; i++) a[i] = 0;
  return 0;
}
```

findMax Attempt 2

`findMax2.c`:

```
/*@ requires n ≥1;
    ensures ∀ integer i; 0 ≤ i < n ⇒ a[i] ≤ \result;
    ensures ∃ integer i; 0 ≤ i < n ∧ a[i] ≡ \result;
    assigns \nothing;
 */
int findMax(int n, int a[]) {
  int i;
  /*@
      loop invariant 0 ≤ i ≤ n;
      loop assigns i;
  */
  for( i = 0; i < n; i++) ;
  return 0;
}
```

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-11-29

Frama-C: Behaviours, Loop Variants

Reconsidering the *findMax* Specification

```
/*@ requires n ≥1;
    requires \valid_read(a + (0 .. n − 1));
    ensures ∀ integer i; 0 ≤ i < n ⇒ a[i] ≤ \result;
    ensures ∃ integer i; 0 ≤ i < n ∧ a[i] ≡ \result;
    assigns \nothing;
 */
int findMax(int n, int a []);
```

- "**requires** **valid_read**(a + (0 .. n − 1))" is necessary for array access (pointer dereference)
- "**assigns** **nothing**" documents that *findMax* must not have memory side-effects
- What if we wish to replace "**requires** $n \geq 1$" with "**requires** $n \geq 0$"?

  "**ensures** ∃ **integer** i; $0 \leq i < n \wedge a[i] \equiv$ **result**" would be unsatisfiable for "$n \equiv 0$"!

  A different specification for that case is needed: *findMax* then has two distict **behaviours**, that can be specified separately:

"ACSL by Example": The *max_element* Algorithm — Specification

`max_element.h`:

```
#include "typedefs.h"
/*@ requires valid:   \valid_read(a + (0.. n−1));
    assigns           \nothing;
    ensures result:  0 ≤ \result ≤ n;

    behavior empty:
      assumes          n ≡ 0;
      assigns          \nothing;
      ensures result:  \result ≡ 0;
    behavior not_empty:
      assumes          0 < n;
      assigns          \nothing;
      ensures result:  0 ≤ \result < n;
      ensures upper:   ∀ integer i; 0 ≤ i < n        ⇒ a[i] ≤ a[\result];
      ensures first:   ∀ integer i; 0 ≤ i < \result ⇒ a[i] < a[\result];

    complete behaviors; disjoint behaviors;
*/
size_type max_element(const value_type* a, size_type n);
```

"ACSL by Example": The *max_element* Algorithm — Implementation

`max_element.c`:

```
#include "max_element.h"

size_type max_element(const value_type* a, size_type n)
{ if (0u < n) {
    size_type max = 0u;
    /*@ loop invariant bound: 0 ≤ i ≤ n;
        loop invariant max:   0 ≤ max < n;
        loop invariant upper: ∀ integer k; 0 ≤ k < i   ⇒ a[k] ≤ a[max];
        loop invariant first: ∀ integer k; 0 ≤ k < max ⇒ a[k] < a[max];
        loop assigns max, i;
        loop variant n−i;
    */
    for (size_type i = 1u; i < n; i++) {
      if (a[max] < a[i]) { max = i; }
    }
    return max;
  }
  return n;
}
```

ACSL By Example — Conventions

`SizeValueTypes.h`:

```
#ifndef SIZEVALUETYPES

typedef int value_type;
typedef unsigned int size_type;
typedef int bool;
#define false 0
#define true 1

#define SIZEVALUETYPES
#endif
```

`IsValidRange.h`:

```
#ifndef ISVALIDRANGE

#include "SizeValueTypes.h"
/*@ predicate IsValidRange(value_type* a, integer n)
      = (0 ≤ n) ∧ \valid(a+(0.. n−1));
*/
```

ACSL Loop Annotations

Total correctness **While** rule:

$$\frac{\{ B \wedge Q \}\ C\ \{ Q \} \quad \{ B \wedge Q \wedge T = t_0 \}\ C\ \{ T < t_0 \} \quad B \wedge Q \Rightarrow T \geq 0}{\{ dom\ `B' \wedge Q \}\ \ while\ B\ do\ C\ od\ \ \{ \neg B \wedge Q \}}\ \text{prov.}\ \neg occurs(`t_0', `B, C, Q, T')$$

"**loop invariant** Q": Property "always" true in the following loop:

- true at loop entry, at each loop iteration, at loop exit
- usually contains a generalisation of the post-condition
- may need to contain additional "sanity" conditions

"**loop assigns** *footprint*": What may be assigned to within the loop

"**loop variant** T": To prove termination:

- Integer metric T that is **strictly decreasing** at each iteration and **bounded** by 0
- Conceptually, this establishes a well-founded relation on the states encountered at start and end of loop body executions.

  $s_1 \supsetneq s_2 \quad \equiv \quad [\![T]\!]\, s_1 > [\![T]\!]\, s_2$ — (using $[\![\_]\!]$ also for expression semantics *evalV*)
- **Any** expression T for which the premises can be proven is acceptable.
- Some expressions T may make these proofs easier than others. . .

Loop Variants 1

$$\frac{\{ B \wedge Q \}\ C\ \{ Q \} \quad \{ B \wedge Q \wedge T = t_0 \}\ C\ \{ T < t_0 \} \quad B \wedge Q \Rightarrow T \geq 0}{\{ dom\ `B' \wedge Q \}\ \ while\ B\ do\ C\ od\ \ \{ \neg B \wedge Q \}}\ \text{prov.}\ \neg occurs(`t_0', `B, C, Q, T')$$

```
//@ assigns \nothing;
void f () {
  int i = 10;
  /*@ loop assigns i;
      loop variant i;  // `T`
   */
  while (i > 0)
  {
    i−−;
  }
}
```

- T needs to be some upper bound for the "number of iterations still remaining"

Loop Variants 2

$$\frac{\{ B \wedge Q \}\ C\ \{ Q \} \quad \{ B \wedge Q \wedge T = t_0 \}\ C\ \{ T < t_0 \} \quad B \wedge Q \Rightarrow T \geq 0}{\{ dom\ `B' \wedge Q \}\ \ while\ B\ do\ C\ od\ \ \{ \neg B \wedge Q \}}\ \text{prov.}\ \neg occurs(`t_0', `B, C, Q, T')$$

```
//@ assigns \nothing;
void f () {
  int i = 10;
  /*@ loop assigns i;
      loop variant i;  // `T`
   */
  while (i ≥ 0)
  {
    i−−;
  }
}
```

ACSL only requires $B \wedge Q \Rightarrow T \geq 0$

ACSL def., section "Loop Variants":

"its value at the beginning of the iteration must be nonnegative."

Loop Variants 3

$$\frac{\{ B \wedge Q \}\ C\ \{ Q \} \quad \{ B \wedge Q \wedge T = t_0 \}\ C\ \{ T < t_0 \} \quad B \wedge Q \Rightarrow T \geq 0}{\{ dom\ `B' \wedge Q \}\ \ while\ B\ do\ C\ od\ \ \{ \neg B \wedge Q \}}\ \text{prov.}\ \neg occurs(`t_0', `B, C, Q, T')$$

```
//@ assigns \nothing;
void f () {
  int i = 10;
  /*@ loop assigns i;
      loop variant i;  // `T`
   */
  while (i ≥ −1)
  {
    i−−;
  }
}
```

```
[wp] [Alt−Ergo ] Goal typed_f_loop_variant_positive : Timeout (Qed:1ms) (10s)
```

- We need $B \wedge Q \Rightarrow T \geq 0$!

Loop Variants 4

$$\frac{\{ B \wedge Q \}\ C\ \{ Q \} \quad \{ B \wedge Q \wedge T = t_0 \}\ C\ \{ T < t_0 \} \quad B \wedge Q \Rightarrow T \geq 0}{\{ dom\ `B' \wedge Q \}\ \ while\ B\ do\ C\ od\ \ \{ \neg B \wedge Q \}}\ \text{prov.}\ \neg occurs(`t_0', `B, C, Q, T')$$

```
//@ assigns \nothing;
void f () {
  int i = 10;
  /*@ loop assigns i;
      loop variant i;  // `T`  */
  while (i > 0) {
    if (i % 2 ≡ 0) { i−−; }
    else           { i = i − 3; }
  }
}
```

- T needs to be **some** upper bound for the "number of iterations still remaining"
- T does not need to be a tight upper bound!
- Simpler variants may have "faster proofs"

Loop Variants 5

$$\frac{\{B \wedge Q\}\, C\, \{Q\} \qquad \{B \wedge Q \wedge T = t_0\}\, C\, \{T < t_0\} \qquad B \wedge Q \Rightarrow T \geq 0}{\{dom\ `B' \wedge Q\}\quad while\ B\ do\ C\ od\quad \{\neg B \wedge Q\}}\ \text{prov. } \neg occurs(`t_0', `B, C, Q, T')$$

```
//@ assigns \nothing;
void f () {
  int i = 10;
  /*@ loop assigns i;
      loop variant i / 2;  // `T`   */
  while (i > 0) {
    if (i % 2 ≡ 0) { i--; }
    else           { i = i − 3; }
  }
}
```

- T needs to be **some** upper bound for the "number of iterations still remaining"
- T does not need to be a tight upper bound!
- More complex variants may have "slower proofs", or time-outs…

Loop Variants 6

$$\frac{\{B \wedge Q\}\, C\, \{Q\} \qquad \{B \wedge Q \wedge T = t_0\}\, C\, \{T < t_0\} \qquad B \wedge Q \Rightarrow T \geq 0}{\{dom\ `B' \wedge Q\}\quad while\ B\ do\ C\ od\quad \{\neg B \wedge Q\}}\ \text{prov. } \neg occurs(`t_0', `B, C, Q, T')$$

```
#define N 1000
//@ assigns \nothing;
void f () {
  int i = 0;
  /*@ loop assigns i;
      loop variant N − i;  // `T`
   */
  while (i ≤ N)
  {
    i++;
  }
}
```

- T needs to be **decreasing**, even if your counters are increasing!

Loop Variants 7

$$\frac{\{B \wedge Q\}\, C\, \{Q\} \qquad \{B \wedge Q \wedge T = t_0\}\, C\, \{T < t_0\} \qquad B \wedge Q \Rightarrow T \geq 0}{\{dom\ `B' \wedge Q\}\quad while\ B\ do\ C\ od\quad \{\neg B \wedge Q\}}\ \text{prov. } \neg occurs(`t_0', `B, C, Q, T')$$

```
//@ assigns \nothing;
void f () {
   int i = 100, k = 200;
  /*@ loop assigns i, k;
      loop variant i + k;  // `T`
   */
  while (i ≥ 0 ∧ k ≥ 0)
  {
    if( (i + k) % 2 ≡ 0 ) { i--; }
    else                  { k--; }
  }
}
```

- If your loop is not a "plain `for`-loop", several variables may be involved in the variant.

Loop Variants 8

$$\frac{\{B \wedge Q\}\, C\, \{Q\} \qquad \{B \wedge Q \wedge T = t_0\}\, C\, \{T < t_0\} \qquad B \wedge Q \Rightarrow T \geq 0}{\{dom\ `B' \wedge Q\}\quad while\ B\ do\ C\ od\quad \{\neg B \wedge Q\}}\ \text{prov. } \neg occurs(`t_0', `B, C, Q, T')$$

```
//@ assigns \nothing;
void f () {
   int i = 0, k = 10;
  /*@ loop assigns    i, k;
      loop invariant 0 ≤ i ≤ k + 1 ∧ 0 ≤ k;
      loop variant   k * (k + 1) + i;          // `T`
   */
  while (k > 0)
  {
    if( i > 0 ) { i--; }
    else        { i = k; k--; }
}}
```

- Invariants may be needed to contribute to provability of the variant.
- Finding appropriate variants can be tricky…

Loop Variants 9

$$\frac{\{B \wedge Q\}\, C\, \{Q\} \qquad \{B \wedge Q \wedge T = t_0\}\, C\, \{T < t_0\} \qquad B \wedge Q \Rightarrow T \geq 0}{\{dom\ `B' \wedge Q\}\quad while\ B\ do\ C\ od\quad \{\neg B \wedge Q\}}\ \text{prov. } \neg occurs(`t_0', `B, C, Q, T')$$

```
//@ assigns \nothing;
void f () {
   int i = 0, k = 10;
  /*@ loop assigns    i, k;
      loop invariant 0 ≤ i ≤ (k + 1) * (k + 1) ∧ 0 ≤ k;
      loop variant   k * k * (k + 1) + i;  // `T`
   */
  while (k > 0)
  {
    if( i > 0 ) { i--; }
    else        { i = k * k; k--; }
  }
}
```

- …

Loop Variants 9

$$\frac{\{B \wedge Q\}\, C\, \{Q\} \qquad \{B \wedge Q \wedge T = t_0\}\, C\, \{T < t_0\} \qquad B \wedge Q \Rightarrow T \geq 0}{\{dom\ `B' \wedge Q\}\quad while\ B\ do\ C\ od\quad \{\neg B \wedge Q\}}\ \text{prov. } \neg occurs(`t_0', `B, C, Q, T')$$

```
//@ assigns \nothing;
void f () {
   int i = 0, k = 10;
  /*@ loop assigns    ???;
      loop variant    ???;
   */
  while (k > 0)
  {
    if( i > 0 ) { i--; }
    else        { i = k * k; k--; }
  }
}
```

Loop Variants 9

$$\frac{\{B \wedge Q\}\, C\, \{Q\} \qquad \{B \wedge Q \wedge T = t_0\}\, C\, \{T < t_0\} \qquad B \wedge Q \Rightarrow T \geq 0}{\{dom\ `B' \wedge Q\}\quad while\ B\ do\ C\ od\quad \{\neg B \wedge Q\}}\ \text{prov. } \neg occurs(`t_0', `B, C, Q, T')$$

```
//@ assigns \nothing;
void f () {
   int i = 0, k = 10;
  /*@ loop assigns    i, k;
      loop invariant 0 ≤ i ≤ (k + 1) * (k + 1) ∧ 0 ≤ k;
      loop variant   k * k * (k + 1) + i;  // `T`
   */
  while (k > 0)
  {
    if( i > 0 ) { i--; }
    else        { i = k * k; k--; }
  }
}
```

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-12-01

Part 1: Midterm 2

M2.1: Alternative definition of antisymmetry (1)

Theorem "Alternative definition of antisymmetry":
$\quad$ antisymmetric $R \quad \equiv \quad \neg(\exists x \bullet \exists y \mid x \neq y \bullet x \,(R)\, y \,(R)\, x)$

Proof:

$\quad$ antisymmetric R
$\equiv\langle$ "Definition of antisymmetry" $\rangle$
$\quad R \cap R^{\smile} \subseteq \mathbb{I}$
$\equiv\langle$ "Relation inclusion" $\rangle$
$\quad \forall x \bullet \forall y \bullet x \,(R \cap R^{\smile})\, y \Rightarrow x \,(\mathbb{I})\, y$
$\equiv\langle$ "Relationship via $\mathbb{I}$" $\rangle$
$\quad \forall x \bullet \forall y \bullet x \,(R \cap R^{\smile})\, y \Rightarrow x = y$
$\equiv\langle$ "Relation intersection" $\rangle$
$\quad \forall x \bullet \forall y \bullet x \,(R)\, y \wedge x \,(R^{\smile})\, y \Rightarrow x = y$
$\equiv\langle$ "Relation converse" $\rangle$
$\quad \forall x \bullet \forall y \bullet (x \,(R)\, y \,(R)\, x) \Rightarrow x = y$
$\equiv\langle$ "Definition of $\neq$", "Contrapositive" $\rangle$
$\quad \forall x \bullet \forall y \bullet x \neq y \Rightarrow \neg(x \,(R)\, y \,(R)\, x)$
$\equiv\langle$ "Trading for $\forall$" (9.2) $\rangle$
$\quad \forall x \bullet \forall y \mid x \neq y \bullet \neg(x \,(R)\, y \,(R)\, x)$
$\equiv\langle$ "Generalised De Morgan" $\rangle$
$\quad \neg(\exists x \bullet \exists y \mid x \neq y \bullet x \,(R)\, y \,(R)\, x)$

M2.1: Alternative definition of antisymmetry (2)

Theorem "Alternative definition of antisymmetry":
$\quad$ antisymmetric $R \quad \equiv \quad \neg(\exists x \bullet \exists y \mid x \neq y \bullet x \,(R)\, y \,(R)\, x)$

Proof:

$\quad \neg(\exists x \bullet \exists y \mid x \neq y \bullet x \,(R)\, y \,(R)\, x)$
$\equiv\langle$ "Definition of $\neq$", "Trading for $\exists$" $\rangle$
$\quad \neg(\exists x \bullet \exists y \mid x \,(R)\, y \,(R)\, x \bullet \neg(x = y))$
$\equiv\langle$ "Generalised De Morgan" $\rangle$
$\quad \forall x \bullet \forall y \mid x \,(R)\, y \,(R)\, x \bullet x = y$
$\equiv\langle$ "Relationship via $\mathbb{I}$" $\rangle$
$\quad \forall x \bullet \forall y \mid x \,(R)\, y \,(R)\, x \bullet x \,(\mathbb{I})\, y$
$\equiv\langle$ "Relation inclusion", "Relation intersection", "Relation converse" $\rangle$
$\quad R \cap R^{\smile} \subseteq \mathbb{I}$
$\equiv\langle$ "Definition of antisymmetry" $\rangle$
$\quad$ antisymmetric R

M2.1: Alternative definition of univalence

Theorem "Alternative definition of univalence": $\mathsf{univalent}\ R \ \equiv\ R \fatsemi {\sim}\,\mathbb{I} \subseteq {\sim}\,R$

Proof:

$R \fatsemi {\sim}\,\mathbb{I} \subseteq {\sim}\,R$

≡ ⟨ "Relation inclusion" ⟩

$\forall x \bullet \forall y \bullet x \llparenthesis R \fatsemi {\sim}\,\mathbb{I} \rrparenthesis y \Rightarrow x \llparenthesis {\sim}\,R \rrparenthesis y$

≡ ⟨ "Relation composition" ⟩

$\forall x \bullet \forall y \bullet (\exists y' \bullet x \llparenthesis R \rrparenthesis y' \llparenthesis {\sim}\,\mathbb{I} \rrparenthesis y) \Rightarrow x \llparenthesis {\sim}\,R \rrparenthesis y$

≡ ⟨ "Relation complement" ⟩

$\forall x \bullet \forall y \bullet (\exists y' \bullet x \llparenthesis R \rrparenthesis y' \wedge \neg (y' \llparenthesis \mathbb{I} \rrparenthesis y)) \Rightarrow \neg (x \llparenthesis R \rrparenthesis y)$

≡ ⟨ "Relationship via $\mathbb{I}$" ⟩

$\forall x \bullet \forall y \bullet (\exists y' \bullet x \llparenthesis R \rrparenthesis y' \wedge \neg (y' = y)) \Rightarrow \neg (x \llparenthesis R \rrparenthesis y)$

≡ ⟨ "Witness" ⟩

$\forall x \bullet \forall y \bullet \forall y' \bullet x \llparenthesis R \rrparenthesis y' \wedge \neg (y' = y) \Rightarrow \neg (x \llparenthesis R \rrparenthesis y)$

≡ ⟨ "Trading for ∀" ⟩

$\forall x \bullet \forall y \bullet \forall y' \mid x \llparenthesis R \rrparenthesis y' \bullet \neg (y' = y) \Rightarrow \neg (x \llparenthesis R \rrparenthesis y)$

≡ ⟨ "Contrapositive" ⟩

$\forall x \bullet \forall y \bullet \forall y' \mid x \llparenthesis R \rrparenthesis y' \bullet x \llparenthesis R \rrparenthesis y \Rightarrow y' = y$

≡ ⟨ "Trading for ∀", "Interchange of dummies for ∀" ⟩

$\forall y \bullet \forall z \bullet \forall x \bullet x \llparenthesis R \rrparenthesis y \wedge x \llparenthesis R \rrparenthesis z \Rightarrow y = z$

≡ ⟨ "Univalence" ⟩

$\mathsf{univalent}\ R$

M2.1: "Bounded domain"

Theorem (14.135) "Bounded domain": $\mathsf{Dom}\ R \subseteq A \ \equiv\ \mathsf{id}\ A \fatsemi R = R$

Proof:

$\mathsf{Dom}\ R \subseteq A$

≡ ⟨ "Set inclusion" ⟩

$\forall x \bullet x \in \mathsf{Dom}\ R \Rightarrow x \in A$

≡ ⟨ "Membership in `Dom`" ⟩

$\forall x \bullet (\exists y \bullet x \llparenthesis R \rrparenthesis y) \Rightarrow x \in A$

≡ ⟨ "Witness" ⟩

$\forall x \bullet \forall y \bullet x \llparenthesis R \rrparenthesis y \Rightarrow x \in A$

≡ ⟨ "Definition of ⇒ via ∧" ⟩

$\forall x \bullet \forall y \bullet x \in A \wedge x \llparenthesis R \rrparenthesis y \equiv x \llparenthesis R \rrparenthesis y$

≡ ⟨ "One-point rule for ∃", substitution ⟩

$\forall x \bullet \forall y \bullet (\exists x' \mid x = x' \bullet x' \in A \wedge x' \llparenthesis R \rrparenthesis y) \equiv x \llparenthesis R \rrparenthesis y$

≡ ⟨ "Trading for ∃" ⟩

$\forall x \bullet \forall y \bullet (\exists x' \bullet x = x' \in A \wedge x' \llparenthesis R \rrparenthesis y) \equiv x \llparenthesis R \rrparenthesis y$

≡ ⟨ "Relationship via `id`" ⟩

$\forall x \bullet \forall y \bullet (\exists x' \bullet x \llparenthesis \mathsf{id}\ A \rrparenthesis x' \llparenthesis R \rrparenthesis y) \equiv x \llparenthesis R \rrparenthesis y$

≡ ⟨ "Relation composition" ⟩

$\forall x \bullet \forall y \bullet x \llparenthesis \mathsf{id}\ A \fatsemi R \rrparenthesis y \equiv x \llparenthesis R \rrparenthesis y$

≡ ⟨ "Relation extensionality" ⟩

$\mathsf{id}\ A \fatsemi R = R$

M2.1: "Bounded range"

Theorem "Bounded range": $B \subseteq \mathsf{Ran}\ R \ \equiv\ \mathsf{id}\ B \subseteq R^{\smile} \fatsemi R$

Proof:

$B \subseteq \mathsf{Ran}\ R$

≡ ⟨ "Set inclusion" ⟩

$\forall y \bullet y \in B \Rightarrow y \in \mathsf{Ran}\ R$

≡ ⟨ "Membership in `Ran`" ⟩

$\forall y \bullet y \in B \Rightarrow (\exists x \bullet x \llparenthesis R \rrparenthesis y)$

≡ ⟨ "Idempotency of ∧" ⟩

$\forall y \bullet y \in B \Rightarrow \exists x \bullet x \llparenthesis R \rrparenthesis y \wedge x \llparenthesis R \rrparenthesis y$

≡ ⟨ "Relation converse" ⟩

$\forall y \bullet y \in B \Rightarrow \exists x \bullet y \llparenthesis R^{\smile} \rrparenthesis x \llparenthesis R \rrparenthesis y$

≡ ⟨ "Relation composition" ⟩

$\forall y \bullet y \in B \Rightarrow y \llparenthesis R^{\smile} \fatsemi R \rrparenthesis y$

≡ ⟨ "One-point rule for ∀", substitution ⟩

$\forall y \bullet \forall y' \mid y = y' \bullet y' \in B \Rightarrow y \llparenthesis R^{\smile} \fatsemi R \rrparenthesis y'$

≡ ⟨ "Trading for ∀" ⟩

$\forall y \bullet \forall y' \bullet y = y' \in B \Rightarrow y \llparenthesis R^{\smile} \fatsemi R \rrparenthesis y'$

≡ ⟨ "Relationship via `id`" ⟩

$\forall y \bullet \forall y' \bullet y \llparenthesis \mathsf{id}\ B \rrparenthesis y' \Rightarrow y \llparenthesis R^{\smile} \fatsemi R \rrparenthesis y'$

≡ ⟨ "Relation inclusion" ⟩

$\mathsf{id}\ B \subseteq R^{\smile} \fatsemi R$

M2.2: "Surjectivity of composition"

Theorem "Surjectivity of composition":
$\mathsf{surjective}\ Q \Rightarrow \mathsf{surjective}\ R \Rightarrow \mathsf{surjective}\ (Q \fatsemi R)$

Proof:

Assuming "Q" `surjective $Q$` **and using with** "Definition of surjectivity":

Assuming "R" `surjective $R$` **and using with** "Definition of surjectivity":

Using "Definition of surjectivity":

$(Q \fatsemi R)^{\smile} \fatsemi (Q \fatsemi R)$

= ⟨ "Converse of ⨾" ⟩

$R^{\smile} \fatsemi Q^{\smile} \fatsemi Q \fatsemi R$

⊇ ⟨ Monotonicity with assumption "Q" ⟩

$R^{\smile} \fatsemi \mathbb{I} \fatsemi R$

= ⟨ "Identity of ⨾" ⟩

$R^{\smile} \fatsemi R$

⊇ ⟨ Assumption "R" ⟩

$\mathbb{I}$

M2.2: "Injectivity of composition" (1)

Theorem "Injectivity of composition":
$\mathsf{injective}\ R \Rightarrow \mathsf{injective}\ S \Rightarrow \mathsf{injective}\ (R \fatsemi S)$

Proof:

Assuming `injective $R$`, `injective $S$`:

Using "Definition of injectivity":

$(R \fatsemi S) \fatsemi (R \fatsemi S)^{\smile}$

= ⟨ "Converse of ⨾" ⟩

$R \fatsemi S \fatsemi S^{\smile} \fatsemi R^{\smile}$

⊆ ⟨ Monotonicity with assumption `injective $S$` with "Definition of injectivity" ⟩

$R \fatsemi \mathbb{I} \fatsemi R^{\smile}$

= ⟨ "Identity of ⨾" ⟩

$R \fatsemi R^{\smile}$

⊆ ⟨ Assumption `injective $R$` with "Definition of injectivity" ⟩

$\mathbb{I}$

M2.2: "Injectivity of composition" (2)

Theorem "Injectivity of composition":
$\mathsf{injective}\ R \Rightarrow \mathsf{injective}\ S \Rightarrow \mathsf{injective}\ (R \fatsemi S)$

Proof:

Assuming `injective $R$`, `injective $S$`:

Using "Definition of injectivity":

$(R \fatsemi S) \fatsemi (R \fatsemi S)^{\smile}$

= ⟨ "Converse of ⨾" ⟩

$R \fatsemi (S \fatsemi S^{\smile}) \fatsemi R^{\smile}$

⊆ ⟨ "Monotonicity of ⨾" with "Monotonicity of ⨾" with assumption `injective $S$` with "Definition of injectivity" ⟩

$R \fatsemi \mathbb{I} \fatsemi R^{\smile}$

= ⟨ "Identity of ⨾" ⟩

$R \fatsemi R^{\smile}$

⊆ ⟨ Assumption `injective $R$` with "Definition of injectivity" ⟩

$\mathbb{I}$

With explicit "Monotonicity of …" invocations, all enclosing operations need to be traversed outside-in!

M2.2: "Injectivity of composition" (3)

Theorem "Injectivity of composition": $\mathsf{injective}\ R \Rightarrow \mathsf{injective}\ S \Rightarrow \mathsf{injective}\ (R \fatsemi S)$

Proof:

Assuming `injective $R$`, `injective $S$`:

$\mathsf{injective}\ (R \fatsemi S)$

≡ ⟨ "Definition of injectivity" ⟩

$(R \fatsemi S) \fatsemi (R \fatsemi S)^{\smile} \subseteq \mathbb{I}$

≡ ⟨ "Converse of ⨾" ⟩

$R \fatsemi S \fatsemi S^{\smile} \fatsemi R^{\smile} \subseteq \mathbb{I}$

⇐ ⟨ "Transitivity of ⊆" with "Monotonicity of ⨾" with "Monotonicity of ⨾" with assumption `injective $S$` with "Definition of injectivity" ⟩

$R \fatsemi \mathbb{I} \fatsemi R^{\smile} \subseteq \mathbb{I}$

≡ ⟨ "Identity of ⨾" ⟩

$R \fatsemi R^{\smile} \subseteq \mathbb{I}$

≡ ⟨ Assumption `injective $R$` with "Definition of injectivity" ⟩

true

With explicit "Monotonicity of …" invocations, all enclosing operations need to be traversed outside-in! — Here starting with "⊆"!

Transitivity theorems are (heterogeneous) mono-/anti-tonicity theorems as well!

M2.2: "Injectivity of composition" (4)

Theorem "Injectivity of composition":
$\mathsf{injective}\ R \Rightarrow \mathsf{injective}\ S \Rightarrow \mathsf{injective}\ (R \fatsemi S)$

Proof:

Assuming `injective $R$`, `injective $S$`:

$\mathsf{injective}\ (R \fatsemi S)$

≡ ⟨ "Definition of injectivity" ⟩

$(R \fatsemi S) \fatsemi (R \fatsemi S)^{\smile} \subseteq \mathbb{I}$

≡ ⟨ "Converse of ⨾" ⟩

$R \fatsemi S \fatsemi S^{\smile} \fatsemi R^{\smile} \subseteq \mathbb{I}$

⇐ ⟨ Antitonicity with assumption `injective $S$` with "Definition of injectivity" ⟩

$R \fatsemi \mathbb{I} \fatsemi R^{\smile} \subseteq \mathbb{I}$

≡ ⟨ "Identity of ⨾" ⟩

$R \fatsemi R^{\smile} \subseteq \mathbb{I}$

≡ ⟨ Assumption `injective $R$` with "Definition of injectivity" ⟩

true

M2.2: Theorem "M2.2a"

The following theorem statement contains an obvious invitation to use a modal role for the proof:

Theorem "M2.2a":
$Q \subseteq \mathbb{I} \Rightarrow R \cap S \fatsemi Q = (R \cap S) \fatsemi Q$

Proof:

Assuming `$Q \subseteq \mathbb{I}$`:

$R \cap S \fatsemi Q$

⊆ ⟨ "Modal rule" ⟩

$(R \fatsemi Q^{\smile} \cap S) \fatsemi Q$

⊆ ⟨ Monotonicity with assumption `$Q \subseteq \mathbb{I}$` ⟩

$(R \fatsemi \mathbb{I}^{\smile} \cap S) \fatsemi Q$

= ⟨ "Converse of $\mathbb{I}$", "Identity of ⨾" ⟩

$(R \cap S) \fatsemi Q$

⊆ ⟨ "Sub-distributivity of ⨾ over ∩" ⟩

$R \fatsemi Q \cap S \fatsemi Q$

⊆ ⟨ Monotonicity with assumption `$Q \subseteq \mathbb{I}$` ⟩

$R \fatsemi \mathbb{I} \cap S \fatsemi Q$

= ⟨ "Identity of ⨾" ⟩

$R \cap S \fatsemi Q$

Theorem "M2.2a":
$R \subseteq \mathbb{I} \Rightarrow Q \cap R \fatsemi S = R \fatsemi (Q \cap S)$

Proof:

Assuming `$R \subseteq \mathbb{I}$`:

$Q \cap R \fatsemi S$

⊆ ⟨ "Modal rule" ⟩

$R \fatsemi (R^{\smile} \fatsemi Q \cap S)$

⊆ ⟨ Monotonicity with assumption `$R \subseteq \mathbb{I}$` ⟩

$R \fatsemi (\mathbb{I}^{\smile} \fatsemi Q \cap S)$

= ⟨ "Converse of $\mathbb{I}$", "Identity of ⨾" ⟩

$R \fatsemi (Q \cap S)$

⊆ ⟨ "Sub-distributivity of ⨾ over ∩" ⟩

$R \fatsemi Q \cap R \fatsemi S$

⊆ ⟨ Monotonicity with assumption `$R \subseteq \mathbb{I}$` ⟩

$\mathbb{I} \fatsemi Q \cap R \fatsemi S$

= ⟨ "Identity of ⨾" ⟩

$Q \cap R \fatsemi S$

M2.3: Recall: The "While" Rule for Partial Correctness

The constituents of a while loop "while B do C od" are:

- The **loop condition** $B : \mathbb{B}$
- The **(loop) body** $C : Cmd$

The conventional **while rule** allows to infer only correctness statements for while loops that are in the shape of the conclusion of this inference rule, involving an **invariant** condition $Q : \mathbb{B}$:

$$\vdash \frac{`B \wedge Q \ \Rightarrow[\ C\ ]\ Q`}{`Q \ \Rightarrow[\ \mathsf{while}\ B\ \mathsf{do}\ C\ \mathsf{od}\ ]\ \neg B \wedge Q`}$$

This rule reads:

- If you can prove that execution of the loop body C starting in states satisfying the loop condition B **preserves** the invariant Q,
- then you have proof that the whole loop also preserves the invariant Q, and in addition establishes the negation of the loop condition.

M2.3: Using the "While" Rule for Partial Correctness (0)

Theorem "While-example":
Pre
⇒[INIT ;
while B
do C od ;
FINAL
]
Post

Proof:
Pre ...... Precondition
⇒[INIT] ⟨ ? ⟩
Q ...... Invariant
⇒[while B do
C
od] ⟨ "While" with subproof:
???
⇒[C] ⟨ ? ⟩
???
⟩
???
⇒[FINAL] ⟨ ? ⟩
Post ...... Postcondition

The invariant Q will be the precondition of the whole **while**-loop.

M2.3: Using the "While" Rule for Partial Correctness (1)

Theorem "While-example":
Pre
⇒[INIT ;
while B
do C od ;
FINAL
]
Post

Proof:
Pre ...... Precondition
⇒[INIT] ⟨ ? ⟩
Q ...... Invariant
⇒[while B do
C
od] ⟨ "While" with subproof:
$B \wedge Q$ ...... (1) Loop condition and invariant
⇒[C] ⟨ ? ⟩
???
⟩
???
⇒[FINAL] ⟨ ? ⟩
Post ...... Postcondition

(1): At the start of a loop body iteration, the loop condition B just checked as *true*, and we expect the invariant Q to hold.

M2.3: Using the "While" Rule for Partial Correctness (2)

Theorem "While-example":
Pre
⇒[INIT ;
while B
do C od ;
FINAL
]
Post

Proof:
Pre ...... Precondition
⇒[INIT] ⟨ ? ⟩
Q ...... Invariant
⇒[while B do
C
od] ⟨ "While" with subproof:
$B \wedge Q$ ...... (1) Loop condition and invariant
⇒[C] ⟨ ? ⟩
Q ...... (2) Invariant
⟩
???
⇒[FINAL] ⟨ ? ⟩
Post ...... Postcondition

(2): After a loop body iteration, we expect the invariant Q to still hold.
(The loop condition B may be true or false for the next check!)

M2.3: Using the "While" Rule for Partial Correctness (3)

Theorem "While-example":
Pre
⇒[INIT ;
while B
do C od ;
FINAL
]
Post

Proof:
Pre ...... Precondition
⇒[INIT] ⟨ ? ⟩
Q ...... Invariant
⇒[while B do
C
od] ⟨ "While" with subproof:
$B \wedge Q$ ...... (1) Loop condition and invariant
⇒[C] ⟨ ? ⟩
Q ...... (2) Invariant
⟩
$\neg B \wedge Q$ ...... (3) Negated loop condition, and invariant
⇒[FINAL] ⟨ ? ⟩
Post ...... Postcondition

(3): After the loop exists, the loop condition B must have become false, and we expect the invariant Q to still hold.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-12-01

Part 2: Graphs, Subgraphs, Lattices Graph Homomorphisms

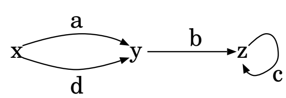
Graphs

Definition: A **graph** is a tuple $\langle V, E, src, trg \rangle$ consisting of

- a set V of ***vertices*** or ***nodes***
- a set E of ***edges*** or ***arrows***
- a mapping $src : E \to V$ that assigns each edge its ***source*** node
- a mapping $trg : E \to V$ that assigns each edge its ***target*** node

Example graph:

$$\langle\ \{x,y,z\},\ \{a,b,c,d\},\ \{\langle a,x\rangle,\langle b,z\rangle,\langle c,z\rangle,\langle d,x\rangle\},\ \{\langle a,y\rangle,\langle b,y\rangle,\langle c,z\rangle,\langle d,y\rangle\}\ \rangle$$



Graphs, Induced Subgraphs

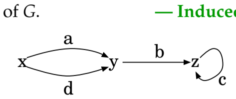
Definition: A **graph** is a tuple $\langle V, E, src, trg \rangle$ consisting of

- a set V of ***vertices*** or ***nodes***
- a set E of ***edges*** or ***arrows***
- a mapping $src : E \to V$ that assigns each edge its ***source*** node
- a mapping $trg : E \to V$ that assigns each edge its ***target*** node

Definition: Let two graphs $G_1 = \langle V_1, E_1, src_1, trg_1 \rangle$ and $G_2 = \langle V_2, E_2, src_2, trg_2 \rangle$ be given.

- G_1 is called a ***subgraph*** of G_2 iff $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$ and $src_1 \subseteq src_2$ and $trg_1 \subseteq trg_2$.

Def. and Theorem: Given a subset $V_0 \subseteq V$ of the vertex set of graph $G = \langle V, E, src, trg \rangle$, the edges incident with only nodes in V_0 are $E_0 := E \cap src^{\smile} (\!| V_0 |\!) \cap trg^{\smile} (\!| V_0 |\!)$, and then $G_0 := \langle V_0,\ E_0,\ E_0 \lhd src,\ E_0 \lhd trg \rangle$ is called the ***subgraph of*** G ***induced by*** V_0.
It is a graph, and a subgraph of G. — **Induced subgraphs are well-defined**



$a \in trg^{\smile} (\!| \{y,z\} |\!)$, but $a \notin src^{\smile} (\!| \{y,z\} |\!)$

Graphs, Subgraphs

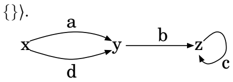
Definition: A **graph** is a tuple $\langle V, E, src, trg \rangle$ consisting of

- a set V of ***vertices*** or ***nodes***
- a set E of ***edges*** or ***arrows***
- a mapping $src : E \to V$ that assigns each edge its ***source*** node
- a mapping $trg : E \to V$ that assigns each edge its ***target*** node

Definition: Let two graphs $G_1 = \langle V_1, E_1, src_1, trg_1 \rangle$ and $G_2 = \langle V_2, E_2, src_2, trg_2 \rangle$ be given.

- G_1 is called a ***subgraph*** of G_2 iff $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$ and $src_1 \subseteq src_2$ and $trg_1 \subseteq trg_2$.
- We write $Subgraph_G$ for the set of all subgraphs of G.
- For a given graph G, we write $G_1 \sqsubseteq_G G_2$ if both G_1 and G_2 are subgraphs of G, and G_1 is a subgraph of G_2.

Theorem: $\sqsubseteq_G$ is an ordering on $Subgraph_G$.
Theorem: $\sqsubseteq_G$ has greatest element G and least element $\langle\{\},\{\},\{\},\{\}\rangle$.
Theorem: $\sqsubseteq_G$ has binary meets defined by intersection.
Theorem: $\sqsubseteq_G$ has binary joins defined by union.
Theorem: $\sqsubseteq_G$ has pseudo-complements, but not complements.



The subgraph induced by $\{y,z\}$ has the subgraph induced by $\{x\}$ as pseudo-complement, but their union is not the whole graph.

Joins and Meets

- Given an order $\sqsubseteq$, z is an "upper bound" of two elements x and y iff $x \sqsubseteq z \wedge y \sqsubseteq z$
- Given an order $\sqsubseteq$, the two elements x and y have j as "join" or "least upper bound" (lub), iff $\forall z \bullet j \sqsubseteq z \equiv x \sqsubseteq z \wedge y \sqsubseteq z$
- The order $\sqsubseteq$ "has binary joins" if for any two elements, there is a join — see "Characterisation of $\cup$" for the inclusion order $\subseteq$
- Given an order $\sqsubseteq$, the set S of elements has j as "join" or "least upper bound" (lub), iff $\forall z \bullet j \sqsubseteq z \equiv (\forall x \mid x \in S \bullet x \sqsubseteq z)$
- The order $\sqsubseteq$ "has arbitrary joins" if for any set of elements, there is a join — see "Characterisation of $\bigcup$"
- Given an order $\sqsubseteq$, the set S of elements has m as "meet" or "greatest lower bound" (glb), iff $\forall z \bullet z \sqsubseteq m \equiv (\forall x \mid x \in S \bullet z \sqsubseteq x)$
- The order $\sqsubseteq$ "has binary meets" if for any two-element set, there is a meet — see "Characterisation of $\cap$"
- The order $\sqsubseteq$ "has arbitrary meets" if for any set of elements, there is a meet.

Lattices

Definition: A **lattice** is a partial order with binary meets and joins.

Examples:

- For every graph G, its subgraphs, that is, $\langle Subgraph_G, \sqsubseteq_G \rangle$ with $\sqcap_G$ and $\sqcup_G$
- $\langle \mathbb{Z}, \leq \rangle$ with $\downarrow$ and $\uparrow$
- $\langle \mathbb{Z}, \geq \rangle$ with $\uparrow$ and $\downarrow$
- $\langle \mathbb{N}, \leq \rangle$ with $\downarrow$ and $\uparrow$
- $\langle \mathbb{N}, | \rangle$ with gcd and lcm
- $\langle \wp A, \subseteq \rangle$ with $\cap$ and $\cup$
- Equivalence relations on A ordered wrt. $\subseteq$, with $\cap$ and $(E_1 \cup E_2)^*$

Algebraic Definition: A **lattice** $\langle A, \sqcap, \sqcup \rangle$ consists of a set A with two binary operations $\sqcap$, $\sqcup$ on A such that:

- $\sqcap$ and $\sqcup$ each are idempotent, symmetric, and associative
- The absorption laws hold: $x \sqcup (x \sqcap y) = x = x \sqcap (x \sqcup y)$

A **Boolean lattice** $\langle A, \sqcap, \sqcup, \bot, \top, \sim \rangle$ in addition has least and greatest elements $\bot$ and $\top$, and a unary **complement** operation $\sim$ satisfying $\sim x \sqcap x = \bot$ and $\sim x \sqcup x = \top$.

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-12-04

Temporal Logic and Model Checking

Temporal Logics for Specification of Reactive and Distributed Systems

- **Reactive Systems**: No clear input-output relation
  - Operating systems
  - Embedded systems
  - Network protocols
- Specification techniques: **Temporal logics**
  - Rich choice of temporal logics — multiple classification criteria
  - Some important logics are (polynomial-time) decidable — **Model checking**

Reading More about Temporal Logics

- E. Allen Emerson: **Temporal and Modal Logic**, pages 995–1072 of Jan van Leeuwen (ed.): **Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics**, Elsevier Science Publishers B. V., 1990
  `https://doi.org/10.1016/B978-0-444-88074-1.50021-4`
  Thode Library Bookstacks: QA 76 .H279 1990
  "Post-print"? linked on Wikipedia:
  `https://profs.info.uaic.ro/~masalagiu/pub/handbook3.pdf`
- Michael R. A. Huth and Mark D. Ryan: **Logic in Computer Science, Modelling and Reasoning about Systems, 2nd edition**, Cambridge University Press 2004,
  Thode Library Bookstacks: QA 76.9 .L63H88 2004

Modal Logics

- Original philosophical motivation: Express different **modalities**:
  The proposition "Napoleon was victorious at Waterloo"
  - is false in this world,
  - but could be true in another world.
- Typical modal operators:
  - "**possibly**": $\diamond\, p$ — "it is imaginable that p holds" "diamond p"
  - "**necessarily**": $\Box\, p$ — "it is not imaginable that p doesn't hold" "box p"
- Kripke (1963): "**possible world semantics**" (orig. Kanger 1957)

Temporal Logics

- Prior (1955): **Tense Logic** — notation still customary today
  - instead of $\diamond\, p$ now temporally: $F\, p$ — "p will eventually be true"
  - instead of $\Box\, p$ now temporally: $G\, p$ — "p will always be true"
- Two kinds of applications: Temporal logics are used
  - in AI, to let programs reason about the world,
  - in software technology, to let the world reason about programs
- Pnueli (1977): "**The Temporal Logic of Programs**":
  Argues for using temporal logics as tool for specification and verification, in particular for **reactive systems** such as operating systems and network protocols

Propositional Logics versus First-order Predicate Logics

- **Temporal Propositional Logics**:
  - Classical junctors: $\wedge$, $\vee$, $\neg$
  - Temporal operators: F , G
- Extension to **temporal predicate logics**
  - variable, constant, function and predicate symbols as usual
  - uninterpreted / partially interpreted / fully interpreted
  - local/global variables
  - sometimes **restrictions on permitted formulae**
    with respect to the interaction between quantifiers and temporal operators, e.g.:
    $$(\forall y : G\,(P(y))) \Leftrightarrow (G\,(\forall y : P(y)))$$
    "Formula of Barcan" — "highly undecidable" logics

Linear Time versus Branching Time

This distinction is mainly semantic, but also reflected in syntax

- **Linear Time**:
  - At any point only **one** possible future
- **Branching Time**:
  - At any point **multiple** possible futures

Both approaches are used in software technology

Further Aspects of Time

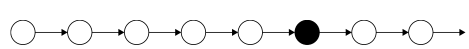
- **Time Points versus Time Intervals**
  - Some properties are easier to formulate using intervals.
- **Discrete Time versus Continuous Time**
  - Continuous (or dense) time first considered in philosophy
  - Possible application in real time systems
- **Future Only versus Also Past**
  - Philosophical approaches: Past at least as important as future
  - Software: Frequently only future
  - Past operators are frequently useful in compositional specifications.

Classification of Temporal Logics — Summary

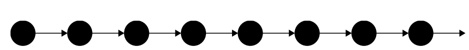
- **Propositional logics** — first-order predicate logics
- **Endogeneous time** (**global**) — exogeneous time (compositional)
- **Linear time** — **branching time**
- **Time points** — time intervals
- **Discrete time** — continuous time
- **Future** — **also past**

Temporal Operators of Linear-Time Propositional Logic

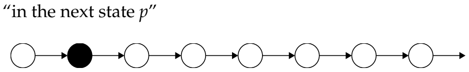
- $F\, p$ — "eventually p"



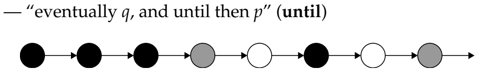
- $G\, p$ — "always p"



- $X\, p$ — "in the next state p"



- $p\; U\; q$ — "eventually q, and until then p" (**until**)



Propositional Linear-Time Temporal Logic — Syntax

Definition: The set of formulae of **propositional linear-time temporal logic** is the smallest set generated by the following rules:

- every atomic proposition $P : AP$ is a formula;
- if p and q are formulae, then $p \wedge q$ and $\neg p$ are formulae, too;
- if p and q are formulae, then $p\ U\ q$ and $X\ p$ formulae, too.

Abbreviations:

$p \vee q$	$:\equiv$	$\neg(\neg p \wedge \neg q)$	$F\ p$	$:\equiv$	$true\ U\ p$	
$p \Rightarrow q$	$:\equiv$	$\neg p \vee q$	$G\ p$	$:\equiv$	$\neg F\ \neg p$	
$p \Leftrightarrow q$	$:\equiv$	$(p \Rightarrow q) \wedge (q \Rightarrow p)$	$F^{\infty}\ p$	$:\equiv$	$G\ F\ p$	— **“infinitely often”**
$true$	$:\equiv$	$p \vee \neg p$	$G^{\infty}\ p$	$:\equiv$	$F\ G\ p$	— **“almost everywhere”**
$false$	$:\equiv$	$\neg true$	$p\ B\ q$	$:\equiv$	$\neg((\neg p)\ U\ q)$	— “p before q”

Syntax and Semantics of Propositional Linear-Time Temporal Logic (PLTL) 1

$[\![\ \varphi\ ]\!]\ \alpha\ t = true$ iff LTL formula φ holds in time line $\alpha : \mathbb{N} \to A \to \mathbb{B}$ at time t:

Declaration: $[\![\_]\!] : \mathsf{LTL}\ A \to (\mathbb{N} \to A \to \mathbb{B}) \to \mathbb{N} \to \mathbb{B}$

An atomic proposition p is true at time t iff the time line contains, at time t, a state in which p is true:

“Semantics of LTL atoms”: $[\![\ ‘\ p\ ]\!]\ \alpha\ t\ \equiv\ \alpha\ t\ p$

“Semantics of LTL $\neg$”: $[\![\ \neg'\ \varphi\ ]\!]\ \alpha\ t\ \equiv\ \neg\ [\![\ \varphi\ ]\!]\ \alpha\ t$

“Semantics of LTL $\wedge$”: $[\![\ \varphi\ \wedge'\ \psi\ ]\!]\ \alpha\ t\ \equiv\ [\![\ \varphi\ ]\!]\ \alpha\ t\ \wedge\ [\![\ \psi\ ]\!]\ \alpha\ t$

“Semantics of LTL $\vee$”: $[\![\ \varphi\ \vee'\ \psi\ ]\!]\ \alpha\ t\ \equiv\ [\![\ \varphi\ ]\!]\ \alpha\ t\ \vee\ [\![\ \psi\ ]\!]\ \alpha\ t$

“Semantics of LTL $\Rightarrow$”: $[\![\ \varphi\ \Rightarrow'\ \psi\ ]\!]\ \alpha\ t\ \equiv\ [\![\ \varphi\ ]\!]\ \alpha\ t\ \Rightarrow\ [\![\ \psi\ ]\!]\ \alpha\ t$

- $[\![\ p\ ]\!]\ \alpha\ 0\ =\ ?$
- $[\![\ p\ ]\!]\ \alpha\ 3\ =\ ?$
- $[\![\ q\ ]\!]\ \alpha\ 0\ =\ ?$
- $[\![\ p \wedge q\ ]\!]\ \alpha\ 0\ =\ ?$
- $[\![\ p \vee \neg q\ ]\!]\ \alpha\ 3\ =\ ?$
- $[\![\ q \Rightarrow r\ ]\!]\ \alpha\ 42\ =\ ?$

$\alpha\ =$

Time	p	q	r	s
0	✓		✓	
1	✓	✓		
2	✓		✓	
3		✓		
4	✓		✓	
5	✓	✓		✓
6, 16, 26, ...	✓		✓	✓
7, 17, 27, ...	✓	✓		
8, 18, 28, ...	✓		✓	
9, 19, 29, ...	✓	✓	✓	
10, 20, 30, ...	✓		✓	
11, 21, 31, ...	✓	✓		
12, 22, 32, ...	✓		✓	
13, 23, 33, ...	✓	✓		
14, 24, 34, ...	✓		✓	
15, 25, 35, ...	✓	✓		

Syntax and Semantics of Propositional Linear-Time Temporal Logic (PLTL) 2

$[\![\ \varphi\ ]\!]\ \alpha\ t = true$ iff LTL formula φ holds in time line $\alpha : \mathbb{N} \to A \to \mathbb{B}$ at time t:

Declaration: $[\![\_]\!] : \mathsf{LTL}\ A \to (\mathbb{N} \to A \to \mathbb{B}) \to \mathbb{N} \to \mathbb{B}$

$F\ \varphi$ is true at time t if φ is true at some time $t' \geq t$:

“Semantics of `$F$`”:

$[\![\ F\ \varphi\ ]\!]\ \alpha\ t\ \equiv\ \exists\ t' : \mathbb{N}\ |\ t\ \leq\ t'\ \bullet\ [\![\ \varphi\ ]\!]\ \alpha\ t'$

$G\ \varphi$ is true at time t if φ is true at all times $t' \geq t$.

“Semantics of `$G$`”:

$[\![\ G\ \varphi\ ]\!]\ \alpha\ t\ \equiv\ \forall\ t' : \mathbb{N}\ |\ t\ \leq\ t'\ \bullet\ [\![\ \varphi\ ]\!]\ \alpha\ t'$

- $[\![\ G\ p\ ]\!]\ \alpha\ 0\ =\ ?$
- $[\![\ G\ p\ ]\!]\ \alpha\ 5\ =\ ?$
- $[\![\ F\ q\ ]\!]\ \alpha\ 0\ =\ ?$
- $[\![\ F\ s\ ]\!]\ \alpha\ 7\ =\ ?$
- $[\![\ F\ \neg p\ ]\!]\ \alpha\ 0\ =\ ?$
- $[\![\ F\ \neg p\ ]\!]\ \alpha\ 100\ =\ ?$

$\alpha\ =$

Time	p	q	r	s
0	✓		✓	
1	✓	✓		
2	✓		✓	
3		✓		
4	✓		✓	
5	✓	✓		✓
6, 16, 26, ...	✓		✓	✓
7, 17, 27, ...	✓	✓		
8, 18, 28, ...	✓		✓	
9, 19, 29, ...	✓	✓	✓	
10, 20, 30, ...	✓		✓	
11, 21, 31, ...	✓	✓		
12, 22, 32, ...	✓		✓	
13, 23, 33, ...	✓	✓		
14, 24, 34, ...	✓		✓	
15, 25, 35, ...	✓	✓		

Syntax and Semantics of Propositional Linear-Time Temporal Logic (PLTL) 3

$[\![\ \varphi\ ]\!]\ \alpha\ t = true$ iff LTL formula φ holds in time line $\alpha : \mathbb{N} \to A \to \mathbb{B}$ at time t:

Declaration: $[\![\_]\!] : \mathsf{LTL}\ A \to (\mathbb{N} \to A \to \mathbb{B}) \to \mathbb{N} \to \mathbb{B}$

$X\ \varphi$ is true at time t iff φ is true at time $t + 1$:

“Semantics of `$X$`”:

$[\![\ X\ \varphi\ ]\!]\ \alpha\ t\ \equiv\ [\![\ \varphi\ ]\!]\ \alpha\ (\mathsf{suc}\ t)$

- $[\![\ X\ p\ ]\!]\ \alpha\ 0\ =\ ?$
- $[\![\ X\ q\ ]\!]\ \alpha\ 0\ =\ ?$
- $[\![\ q\ \wedge\ X\ r\ ]\!]\ \alpha\ 1\ =\ ?$
- $[\![\ G\ F\ (q\ \wedge\ X\ r)\ ]\!]\ \alpha\ 0\ =\ ?$
- $[\![\ F\ (s\ \wedge\ X\ s)\ ]\!]\ \alpha\ 0\ =\ ?$
- $[\![\ F\ (s\ \wedge\ X\ s)\ ]\!]\ \alpha\ 10\ =\ ?$
- $[\![\ G\ (q\ \equiv\ X\ r)\ ]\!]\ \alpha\ 12\ =\ ?$
- $[\![\ G\ F\ (q\ \equiv\ X\ r)\ ]\!]\ \alpha\ 12\ =\ ?$

$\alpha\ =$

Time	p	q	r	s
0	✓		✓	
1	✓	✓		
2	✓		✓	
3		✓		
4	✓		✓	
5	✓	✓		✓
6, 16, 26, ...	✓		✓	✓
7, 17, 27, ...	✓	✓		
8, 18, 28, ...	✓		✓	
9, 19, 29, ...	✓	✓	✓	
10, 20, 30, ...	✓		✓	
11, 21, 31, ...	✓	✓		
12, 22, 32, ...	✓		✓	
13, 23, 33, ...	✓	✓		
14, 24, 34, ...	✓		✓	
15, 25, 35, ...	✓	✓		

Syntax and Semantics of Propositional Linear-Time Temporal Logic (PLTL) 4

$[\![\ \varphi\ ]\!]\ \alpha\ t = true$ iff LTL formula φ holds in time line $\alpha : \mathbb{N} \to A \to \mathbb{B}$ at time t:

Declaration: $[\![\_]\!] : \mathsf{LTL}\ A \to (\mathbb{N} \to A \to \mathbb{B}) \to \mathbb{N} \to \mathbb{B}$

$\varphi\ U\ \psi$ is true at time t if ψ is true at some time $t' \geq t$, and for all times t'' such that $t \leq t'' < t'$, φ is true.

Axiom “Semantics of `$U$`”: —— “until”

$[\![\ \varphi\ U\ \psi\ ]\!]\ \alpha\ t$

$\equiv\ \exists\ t' : \mathbb{N}\ |\ t\ \leq\ t'$

$\bullet\ [\![\ \psi\ ]\!]\ \alpha\ t'$

$\wedge\ \forall\ t'' : \mathbb{N}\ |\ t\ \leq\ t''\ <\ t'\ \bullet\ [\![\ \varphi\ ]\!]\ \alpha\ t''$

- $[\![\ p\ U\ q\ ]\!]\ \alpha\ 0\ =\ ?$
- $[\![\ p\ U\ s\ ]\!]\ \alpha\ 0\ =\ ?$
- $[\![\ \neg\ s\ U\ \neg\ p\ ]\!]\ \alpha\ 0\ =\ ?$
- $[\![\ p\ U\ (q\ \wedge\ r)\ ]\!]\ \alpha\ 42\ =\ ?$
- $[\![\ p\ U\ (q\ \wedge\ s)\ ]\!]\ \alpha\ 42\ =\ ?$
- $[\![\ (p\ \vee r)\ U\ s\ ]\!]\ \alpha\ 1\ =\ ?$

$\alpha\ =$

Time	p	q	r	s
0	✓		✓	
1	✓	✓		
2	✓		✓	
3		✓		
4	✓		✓	
5	✓	✓		✓
6, 16, 26, ...	✓		✓	✓
7, 17, 27, ...	✓	✓		
8, 18, 28, ...	✓		✓	
9, 19, 29, ...	✓	✓	✓	
10, 20, 30, ...	✓		✓	
11, 21, 31, ...	✓	✓		
12, 22, 32, ...	✓		✓	
13, 23, 33, ...	✓	✓		
14, 24, 34, ...	✓		✓	
15, 25, 35, ...	✓	✓		

Important Valid Formulae

$\models G\ \neg p \Leftrightarrow \neg F\ p$ $\qquad \models G^{\infty}\ \neg p \Leftrightarrow \neg F^{\infty}\ p$ $\qquad \models X\ \neg p \Leftrightarrow \neg X\ p$

$\models F\ \neg p \Leftrightarrow \neg G\ p$ $\qquad \models F^{\infty}\ \neg p \Leftrightarrow \neg G^{\infty}\ p$ $\qquad \models ((\neg p)\ U\ q) \Leftrightarrow \neg(p\ B\ q)$

<u>Idempotencies</u>

- $\models F\ F\ p \Leftrightarrow F\ p$
- $\models G\ G\ p \Leftrightarrow G\ p$
- $\models F^{\infty}\ F^{\infty}\ p \Leftrightarrow F^{\infty}\ p$
- $\models G^{\infty}\ G^{\infty}\ p \Leftrightarrow G^{\infty}\ p$

<u>Implications</u>

- $\models p \Rightarrow F\ p$ $\qquad \models G\ p \Rightarrow p$
- $\models X\ p \Rightarrow F\ p$ $\qquad \models G\ p \Rightarrow X\ p$
- $\models G\ p \Rightarrow F\ p$ $\qquad \models G\ p \Rightarrow X\ G\ p$
- $\models p\ U\ q \Rightarrow F\ q$ $\qquad \models G^{\infty}\ q \Rightarrow F^{\infty}\ q$

$\models X\ F\ p \Leftrightarrow F\ X\ p$ $\qquad \models X\ G\ p \Leftrightarrow G\ X\ p$ $\qquad \models ((X\ p)\ U\ (X\ q)) \Leftrightarrow X\ (p\ U\ q)$

$\models\ \ F^{\infty}\ p\ \Leftrightarrow\ X\ F^{\infty}\ p\ \Leftrightarrow\ F\ F^{\infty}\ p\ \Leftrightarrow\ G\ F^{\infty}\ p\ \Leftrightarrow\ F^{\infty}\ F^{\infty}\ p\ \Leftrightarrow\ G^{\infty}\ F^{\infty}\ p$

$\models\ \ G^{\infty}\ p\ \Leftrightarrow\ X\ G^{\infty}\ p\ \Leftrightarrow\ F\ G^{\infty}\ p\ \Leftrightarrow\ G\ G^{\infty}\ p\ \Leftrightarrow\ F^{\infty}\ G^{\infty}\ p\ \Leftrightarrow\ G^{\infty}\ G^{\infty}\ p$

(considering $\Leftrightarrow$ to be conjunctional)

Interplay between Junctors and Temporal Operators

$\models F\ (p \vee q) \Leftrightarrow (F\ p \vee F\ q)$ $\qquad \models G\ (p \wedge q) \Leftrightarrow (G\ p \wedge G\ q)$

$\models F^{\infty}\ (p \vee q) \Leftrightarrow (F^{\infty}\ p \vee F^{\infty}\ q)$ $\qquad \models G^{\infty}\ (p \wedge q) \Leftrightarrow (G^{\infty}\ p \wedge G^{\infty}\ q)$

$\models p\ U\ (q \vee r) \Leftrightarrow (p\ U\ q \vee p\ U\ r)$ $\qquad \models (p \wedge q)\ U\ r \Leftrightarrow (p\ U\ r \wedge q\ U\ r)$

$\models X\ (p \vee q) \Leftrightarrow (X\ p \vee X\ q)$ $\qquad \models X\ (p \Rightarrow q) \Leftrightarrow (X\ p \Rightarrow X\ q)$

$\models X\ (p \wedge q) \Leftrightarrow (X\ p \wedge X\ q)$ $\qquad \models X\ (p \Leftrightarrow q) \Leftrightarrow (X\ p \Leftrightarrow X\ q)$

$\models (G\ p \vee G\ q) \Rightarrow G\ (p \vee q)$ $\qquad \models F\ (p \wedge q) \Rightarrow F\ p \wedge F\ q$

$\models (G^{\infty}\ p \vee G^{\infty}\ q) \Rightarrow G^{\infty}\ (p \vee q)$ $\qquad \models F^{\infty}\ (p \wedge q) \Rightarrow F^{\infty}\ p \wedge F^{\infty}\ q$

$\models ((p\ U\ r) \vee (q\ U\ r)) \Rightarrow ((p \vee q)\ U\ r)$ $\qquad \models (p\ U\ (q \wedge r)) \Rightarrow ((p\ U\ q) \wedge (p\ U\ r))$

Monotonicity and Fixpoint Characterisations

$\models G\ (p \Rightarrow q) \Rightarrow (F\ p \Rightarrow F\ q)$ $\qquad \models G\ (p \Rightarrow q) \Rightarrow (F^{\infty}\ p \Rightarrow F^{\infty}\ q)$

$\models G\ (p \Rightarrow q) \Rightarrow (G\ p \Rightarrow G\ q)$ $\qquad \models G\ (p \Rightarrow q) \Rightarrow (G^{\infty}\ p \Rightarrow G^{\infty}\ q)$

$\models G\ (p \Rightarrow q) \Rightarrow ((p\ U\ r) \Rightarrow (q\ U\ r))$ $\qquad \models G\ (p \Rightarrow q) \Rightarrow ((r\ U\ p) \Rightarrow (r\ U\ q))$

$\models G\ (p \Rightarrow q) \Rightarrow (X\ p \Rightarrow X\ q)$

Fixpoint Characterisations:

$\models F\ p \Leftrightarrow p \vee X\ F\ p$ $\qquad \models (p\ U\ q) \Leftrightarrow q \vee (p \wedge X\ (p\ U\ q))$

$\models G\ p \Leftrightarrow p \wedge X\ G\ p$ $\qquad \models (p\ B\ q) \Leftrightarrow \neg q \wedge (p \vee X\ (p\ B\ q))$

Variants of the Basic Temporal Operators

- $p\ U\ q$, until now, is known as “**strong until**”:
  There is a future state q, and until then p.
- Alternative notations: $p\ U_s\ q$ or $p\ U_{\exists}\ q$.
- **Weak until** $p\ U_w\ q$ or $p\ U_{\forall}\ q$:
  p **holds as long as** q **does not hold — if necessary, forever.**
- $x \models p\ U_{\forall}\ q$ iff for all $j : \mathbb{N}$ we have $x^j \models p$ as far as for all $k \leq j$ we have $x^k \models \neg q$.

We have:

- $\models p\ U_{\exists}\ q \Leftrightarrow p\ U_{\forall}\ q \wedge F\ q$
- $\models p\ U_{\forall}\ q \Leftrightarrow (p\ U_{\exists}\ q \vee G\ p) \Leftrightarrow (p\ U_{\exists}\ q \vee G\ (p \wedge \neg q))$

Past

Until now, all operators are future-related — explicitly:

- $F^{+}\ p$ — “in the future, eventually p”
- $G^{+}\ p$ — “in the future, always p”
- $X^{+}\ p$ — “in the next state p”
- $p\ U^{+}\ q$ — “in the future, eventually q, and until then p”

Purely future-oriented propositional linear-time temporal logic —
Propositional Linear-time Temporal Logic / Future: PLTLF

Corresponding past-oriented operators (originally P, H, and S for **since**):

- $F^{-}\ p$ — “in the past at some point p”
- $G^{-}\ p$ — “in the past, always p”
- $X^{-}_{\exists}\ p$ — “in the previous state we had p”
- $p\ U^{-}\ q$ — “in the past at some point q, and since then p”

Logic only with past-oriented operators: PLTLP; with both: PLTLB.

Safety

- Safety properties: "nothing bad happens"
- Invariance properties: every finite prefix of the execution satisfies the invariance condition
- in PLTLB: initially equivalent to $G\ p$ for a past formula p: "nothing bad has happened until now" must always be true.
- Every formula constructed from past operators, $\wedge$, $\vee$, G and U_w is a safety property, e.g.:

$$(p\ U_w\ q) \equiv_i G\ (G^{-} p \vee F^{-}(q \wedge X^{-} G^{-} p))$$

Exercise!

Safety Examples

- **Partial correctness** wrt. precondition φ and postcondition ψ: If a program (with start label l_0 and halting label l_h) starts executing in a state satisfying the precondition φ and terminates, the the terminating state satisfies the postcondition ψ:

$$\mathsf{at}l_0 \wedge \varphi \Rightarrow G\ (\mathsf{at}l_h \Rightarrow \psi)$$

This is initially equivalent to:

$$G\ (F^{-}(\neg(\mathsf{at}l_0 \wedge \varphi) \wedge X^{-}_{w} \mathit{false}) \vee G\ (\mathsf{at}l_h \Rightarrow \psi))$$

and therefore a safety property.

- **Mutual Exclusion**: $G\ (\neg(\mathsf{atCS}_1 \wedge \mathsf{atCS}_2))$
- **Deadlock-freeness**: $G\ (\mathsf{enabled}_1 \vee \ldots \vee \mathsf{enabled}_m)$

Liveness

- **Liveness**: "Something good will still happen (often enough)"
- p is an "**invincible**" past formula iff every finite sequence x has a finite extension x' such that p holds in the last state of x':

$$[\![p]\!]\ x'\ (\mathit{length} x') \equiv \mathit{true}$$

- A **pure liveness property** is a PLTLB formula that is initially equivalent to a formula $F\ p$, $G\ F\ p$ or $F\ G\ p$, where p is an invincible past formula
- If p is a pure liveness property, then every finite sequence x can be extended to a finite or infinite sequence x' such that $(x', 0) \models p$
- **Temporal implication** $G\ (p \Rightarrow F\ q)$ (where p and q are past formulae) is a generic liveness property

Propositional Branching-time Temporal Logic

- The "Computational Tree Logic" CTL, and its generalisation CTL*
- Low complexity of CTL
- CTL model checking (SMV)

Time Structures for Branching Time

Definition: A **time structure** $M = (S, R, L)$ consists of

- a **state set** S,
- a total **time step relation** $R : S \leftrightarrow S$ (for every time point there is at least one successor)
- a **marking** $L : S \to \mathbb{P}\ AP$, mapping each state s to the set of atomic propositions true in s.

Therefore M is a node-labelled directed graph. M is

- **acyclic** iff $R^{+} \cap \mathbb{I} = \{\}$,
- **tree-like** iff M is acyclic and R is injective (every state has at most one predecessor)
- a **tree** iff M is tree-like and there is a **root node** (a node without predecessors from which all nodes are reachable). □

Tree property is not essential! Cyclic graphs can be "unravelled" to infinite trees.

Syntax of the "Computational Tree Logic" CTL

State formulae are generated by the following rules:

(S1) Every atomic proposition P is a state formula.
(S2) If p and q are state formulae, then so are $p \wedge q$ and $\neg p$.
(S3a) If p is a **state formula**, then $\mathsf{E}\ X\ p$ and $\mathsf{A}\ X\ p$ are state formulae.

$\mathsf{E}\ X\ p$ — in some possible future, $X\ p$

$\mathsf{A}\ X\ p$ — in all possible futures, $X\ p$

(S3b) If p and q are **state formulae**, then $\mathsf{E}\ (p\ U\ q)$ and $\mathsf{A}\ (p\ U\ q)$ are state formulae.

$\mathsf{E}\ (p\ U\ q)$ — in some possible future, $(p\ U\ q)$

$\mathsf{A}\ (p\ U\ q)$ — in all possible futures, $(p\ U\ q)$

Abbreviations in CTL: $\mathsf{E}\ F\ p :\equiv \mathsf{E}\ (\mathit{true}\ U\ p)$ $\quad \mathsf{A}\ G\ p :\equiv \neg\mathsf{E}\ F\ \neg p$
$\mathsf{A}\ F\ p :\equiv \mathsf{A}\ (\mathit{true}\ U\ p)$ $\quad \mathsf{E}\ G\ p :\equiv \neg\mathsf{A}\ F\ \neg p$

CTL: Strict alternation between E / A and X, U, F, G
CTL*: Direct nesting of X, U, F, G allowed

CTL Specification Patterns

- $E\ F\ (\mathit{started} \wedge \neg \mathit{ready})$
- $A\ G\ (\mathit{requested} \Rightarrow A\ F\ \ \mathit{acknowledged})$
- $A\ G\ (A\ F\ \ \mathit{enabled})$
- $A\ F\ (A\ G\ \ \mathit{deadlock})$
- $A\ G\ (E\ F\ \ \mathit{restart})$
- $A\ G\ (\mathit{floor} = 2 \wedge \mathit{direction} = \mathit{up} \wedge \mathit{ButtonPressed}5 \Rightarrow A\ [\mathit{direction} = \mathit{up}\ U\ \mathit{floor} = 5])$
- $A\ G\ (\mathit{floor} = 3 \wedge \mathit{idle} \wedge \mathit{door} = \mathit{closed} \Rightarrow E\ G\ (\mathit{floor} = 3 \wedge \mathit{idle} \wedge \mathit{door} = \mathit{closed}))$

Small Models Theorem for CTL

Theorem: Let p_0 be a CTL formula of length n. Then the following statements are equivalent:

- p_0 is satisfiable.
- p_0 has an infinite tree model with finite branching degree in $\mathcal{O}(n)$.
- p_0 has a finite model of size $\leq n \cdot 2^n$.

Theorem: The satisfiability test for CTL is DEXPTIME complete.

Why is this useful?

Synthesis of correct-by-construction automata!
(For satisfiable specifications...)

Model Checking

The **Model Checking Problem**:

$$M \stackrel{?}{\models} p$$

I.e., is a given finite structure M a model for a given temporal logic formula p?

- The model checking problem for propositional temporal logics is **decidable**.
- The model checking problem for PLTL(F,X) is PSPACE-complete.
- The model checking problem for PLTL(F) ist NP-complete.
- The model checking problem for CTL* is PSPACE-complete.
- The model checking problem for CTL is solvable in **deterministic polynomial time**.

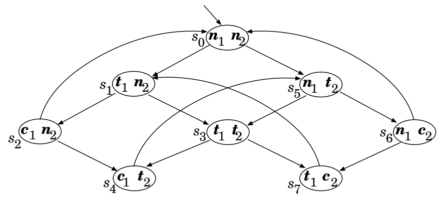
A CTL Model Checker: SMV

- Developed since 1992 at Carnegie Mellon University
- OBDD-based symbolic model checking for CTL
- Finite datatypes: Booleans, enumeration types, finite arrays
- Model description: Arbitrary propositional-logic formulae allowed
- Safe model description: Parallel assignments
- Original motivation: **hardware description**

```
MODULE main
VAR
  request : boolean;
  status : {ready, busy};
ASSIGN
  init(status) := ready;
  next(status) :=
      case
        request : busy;
        1 : {ready, busy};
      esac;
SPEC
  AG(request → AF status=busy)
```

SMV Example from [Huth, Ryan]: Mutual Exclusion

Two processes, each with three states: "n": non-critical, "t": trying, "c": critical.
First protocol:



Safety	$\Phi_1 :\equiv A\ G\ \neg(c_1 \wedge c_2)$
Liveness	$\Phi_2 :\equiv A\ G\ (t_1 \Rightarrow A\ F\ c_1)$
Non-blocking	$\Phi_3 :\equiv A\ G\ (n_1 \Rightarrow E\ X\ t_1)$
No strict sequencing	$\Phi_4 :\equiv E\ F\ (c_1 \wedge E\ [c_1\ U\ (\neg c_1 \wedge E\ [\neg c_2\ U\ c_1])])$

First Translation into SMV Input Language

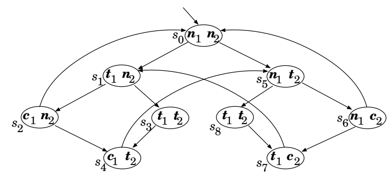
```
MODULE main
VAR
  p1 : {n, t, c};
  p2 : {n, t, c};
ASSIGN
  init(p1) := n;
  init(p2) := n;
TRANS
  (next(p2) = p2 & ((p1 = n → next(p1) = t) &
                    (p1 = t → next(p1) = c) &
                    (p1 = c → next(p1) = n)))  |
  (next(p1) = p1 & ((p2 = n → next(p2) = t) &
                    (p2 = t → next(p2) = c) &
                    (p2 = c → next(p2) = n)))
TRANS next(p1) = c → next(p2) ≠ c

SPEC  AG !(p1=c & p2=c)
SPEC  AG (p1=t → AF p1=c)
SPEC  AG (p1=n → EX p1=t)
SPEC  EF (p1=c & E[p1=c U (p1≠c & E[ p2≠c U p1=c])])
```

SMV Output

```
-- specification  AG (!(p1 = c & p2 = c))  is  true
-- specification  AG (p1 = t → AF p1 = c)  is  false
-- as demonstrated by the following execution sequence
 state  1.1:
p1 = n,  p2 = n

-- loop starts  here --
 state  1.2:
p1 = t

 state  1.3:
p2 = t

 state  1.4:
p2 = c

 state  1.5:
p2 = n

-- specification  AG (p1 = n → EX p1 = t)  is  true
-- specification  EF (p1 = c  & E(p1 = c U (p1 ≠  c  & E(p2  ...   is  true
```

Mutual Exclusion — continued

Safety	$\Phi_1 :\equiv A\ G\ \neg(c_1 \wedge c_2)$
Liveness	$\Phi_2 :\equiv A\ G\ (t_1 \Rightarrow A\ F\ c_1)$
Non-blocking	$\Phi_3 :\equiv A\ G\ (n_1 \Rightarrow E\ X\ t_1)$
No strict sequencing	$\Phi_4 :\equiv E\ F\ (c_1 \wedge E\ [c_1\ U\ (\neg c_1 \wedge E\ [\neg c_2\ U\ c_1])])$



That can even be synthesised from the specification!

Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-12-06

Part 1: Graph Homomorphisms, Categories

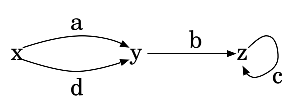
Recall: Graphs

Definition: A **graph** is a tuple $\langle V, E, src, trg \rangle$ consisting of

- a set V of ***vertices*** or ***nodes***
- a set E of ***edges*** or ***arrows***
- a mapping $src : E \twoheadrightarrow V$ that assigns each edge its ***source*** node
- a mapping $trg : E \twoheadrightarrow V$ that assigns each edge its ***target*** node

Example graph:

$$\langle\ \{x, y, z\},\ \{a, b, c, d\},\ \{\langle a, x\rangle, \langle b, z\rangle, \langle c, z\rangle, \langle d, x\rangle\},\ \{\langle a, y\rangle, \langle b, y\rangle, \langle c, z\rangle, \langle d, y\rangle\}\ \rangle$$



Graphs as Structures over Signature *sigGraph*

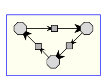
A **signature** is a tuple $\Sigma = (\mathcal{S}, \mathcal{F}, \mathcal{R})$ consisting of

- a set $\mathcal{S}$ of **sorts**
- a set $\mathcal{F}$ of **function symbols** $f : s_1 \times \dots \times s_n \to t$
- a set $\mathcal{R}$ of **relation symbols** $r : s_1 \times \dots \times s_n \leftrightarrow t$

A Σ**-structure** $\mathcal{A}$ consists of:

- for every sort $s : \mathcal{S}$, a **carrier** $s^{\mathcal{A}}$, and
- for every function symbol $f : s_1 \times \dots \times s_n \to t$ a **mapping** $f^{\mathcal{A}} : s_1^{\mathcal{A}} \times \dots \times s_n^{\mathcal{A}} \to t^{\mathcal{A}}$.
- for every relation symbol $r : s_1 \times \dots \times s_n \leftrightarrow t$ a **relation** $r^{\mathcal{A}} : s_1^{\mathcal{A}} \times \dots \times s_n^{\mathcal{A}} \leftrightarrow t^{\mathcal{A}}$.

$sigGraph :\equiv\ \langle$ **sorts:** $\mathcal{V}, \mathcal{E}$
ops: $src, trg : \mathcal{E} \to \mathcal{V}$
$\rangle$



The signature graph of *sigGraph*: $\mathcal{E} \underset{trg}{\overset{src}{\rightrightarrows}} \mathcal{V}$

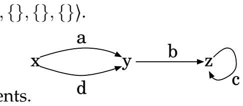
Signatures, as mathematical objects, are of a similar kind as graphs!

Recall: Subgraphs

Definition: Let two graphs $G_1 = \langle V_1, E_1, src_1, trg_1 \rangle$ and $G_2 = \langle V_2, E_2, src_2, trg_2 \rangle$ be given.

- G_1 is called a ***subgraph*** of G_2 iff $V_1 \subseteq V_2$ and $E_1 \subseteq E_2$ and $src_1 \subseteq src_2$ and $trg_1 \subseteq trg_2$.
- We write $Subgraph_G$ for the set of all subgraphs of G.
- For a given graph G, we write $G_1 \sqsubseteq_G G_2$ if both G_1 and G_2 are subgraphs of G, and G_1 is a subgraph of G_2.

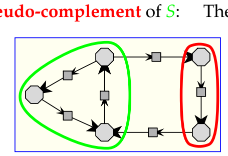
Theorem: $\sqsubseteq_G$ is an ordering on $Subgraph_G$.
Theorem: $\sqsubseteq_G$ has greatest element G and least element $\langle \{\}, \{\}, \{\}, \{\} \rangle$.
Theorem: $\sqsubseteq_G$ has binary meets defined by intersection.
Theorem: $\sqsubseteq_G$ has binary joins defined by union.
Theorem: $\sqsubseteq_G$ has pseudo-complements, but not complements.

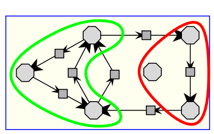


The subgraph induced by $\{y, z\}$ has the subgraph induced by $\{x\}$ as pseudo-complement, but their union is not the whole graph.

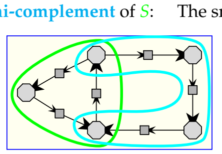
Pseudo- and Semi-Complements of a Subgraph

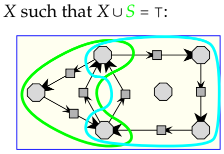
Pseudo-complement of S: The largest X such that $X \cap S = \bot$:





Semi-complement of S: The smallest X such that $X \cup S = \top$:





Graph Homomorphisms

Definition: Let two graphs $G_1 = \langle V_1, E_1, src_1, trg_1 \rangle$ and $G_2 = \langle V_2, E_2, src_2, trg_2 \rangle$ be given.
A pair $\Phi = \langle \Phi_V, \Phi_E \rangle$ is called a **graph homomorphism from** G_1 **to** G_2 iff

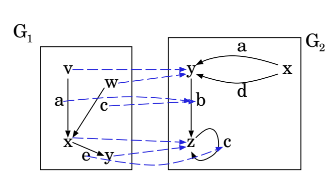
- $\Phi_V \in V_1 \twoheadrightarrow V_2$ and $\Phi_E \in E_1 \twoheadrightarrow E_2$
- $\Phi_E \mathbin{⨾} src_2 = src_1 \mathbin{⨾} \Phi_V$ and $\Phi_E \mathbin{⨾} trg_2 = trg_1 \mathbin{⨾} \Phi_V$

Homomorphisms are **"structure-preserving mappings"**.

(Mappings; Total and univalent)

Graph homomorphisms can:

- Identify different structure elements
  — not injective
- Not cover the target completely
  — not surjective

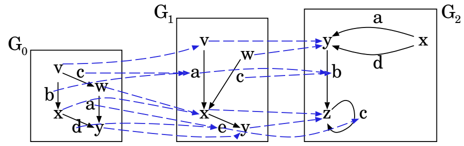


Graph Homomorphisms Compose

Definition: Let two graphs $G_1 = \langle V_1, E_1, src_1, trg_1 \rangle$ and $G_2 = \langle V_2, E_2, src_2, trg_2 \rangle$ be given.
A pair $\Phi = \langle \Phi_V, \Phi_E \rangle$ is called a **graph homomorphism from** G_1 **to** G_2 iff

- $\Phi_V \in V_1 \twoheadrightarrow V_2$ and $\Phi_E \in E_1 \twoheadrightarrow E_2$
- $\Phi_E ; src_2 = src_1 ; \Phi_V$ and $\Phi_E ; trg_2 = trg_1 ; \Phi_V$

Definition and theorem: Let three graphs G_0, G_1, and G_2 be given.
Let $\Phi = \langle \Phi_V, \Phi_E \rangle$ be a graph homomorphism from G_0 to G_1 and $\Psi = \langle \Psi_V, \Psi_E \rangle$ be a graph homomorphism from G_1 to G_2.
Then their **composition** $\Phi ; \Psi = \langle \Phi_V ; \Psi_V, \Phi_E ; \Psi_E \rangle$ is a graph homomorphism from G_0 to G_2.



Definition and theorem: The **identity graph homomorphism** $\mathbb{I} = \langle \mathrm{id}\ V, \mathrm{id}\ E \rangle$ is well-defined, and is "the" identity for graph homomorphism composition.

Graph Homomorphisms Compose — and Form a Category

Graph homomorphisms have

- source and target graphs,
- associative composition $;$ of consecutive homomorphisms,
- identity homomorphisms $\mathbb{I}$ (satisfying the identity laws).

That is, graphs with graph homomorphisms form a **category**.

In particular:

- Ψ is an inverse of Φ iff $\Phi ; \Psi = \mathbb{I}$ and $\Psi ; \Phi = \mathbb{I}$.
- $\Phi = \langle \Phi_V, \Phi_E \rangle$ has an inverse iff it is bijective, that is, iff both Φ_V and Φ_E are bijective. The inverse of Φ is then $\langle \Phi_V{}^{\smile}, \Phi_E{}^{\smile} \rangle$.

(Category theory is the source of the words "functor", "monad", "arrow", etc. in the context of Haskell.)

Categories

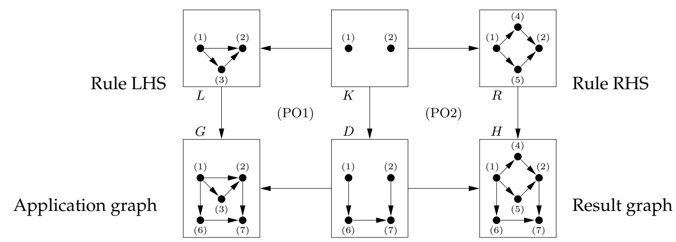
A **category** C consists of:

- a collection of **objects**
- for every two objects $\mathcal{A}$ and $\mathcal{B}$ a **homset** containing **morphisms** $f : \mathcal{A} \to \mathcal{B}$
- associative **composition** "$;$" of morphisms, defined for $\mathcal{A} \xrightarrow{f} \mathcal{B} \xrightarrow{g} \mathcal{C}$, with $(f ; g) : \mathcal{A} \to \mathcal{C}$
- for every object $\mathcal{A}$ an **identity** morphism $\mathbb{I}_{\mathcal{A}}$ which is both a right and left unit for composition.

Categorial Graph Transformation

Graphs with graph homomorphisms form a **category** — category theory is **re-usable theory**!

Using category-theoretical concepts, various **graph transformation** mechanisms are defined; these are used for system modelling and model transformation.



Pushouts — A Typical Categorial "Universal Construction"

Pushouts can be seen as a generalisation of unions/joins:

Recall "Characterisation of $\cup$":

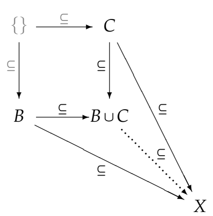
$B \cup C$ is **union** of sets B and C iff

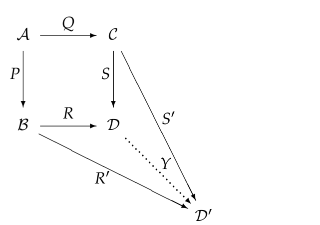
$\forall\ X \bullet\ B \subseteq X \wedge C \subseteq X \ \equiv\ B \cup C \subseteq X$

$\langle \xrightarrow{R} \mathcal{D} \xleftarrow{S} \rangle$ is **pushout** of span "$\mathcal{B} \xleftarrow{P} \mathcal{A} \xrightarrow{Q} \mathcal{C}$" iff

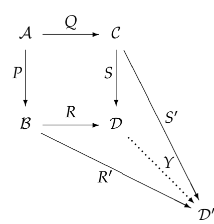
$P ; R = Q ; S \ \wedge\ \forall \langle \xrightarrow{R'} \mathcal{D}' \xleftarrow{S'} \rangle \ \mid\ P ; R' = Q ; S'$

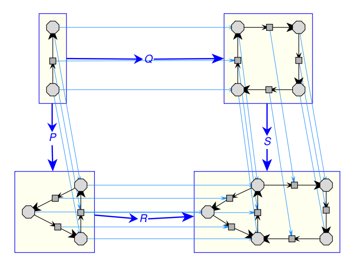
$\bullet\ \exists\ Y : D \to D' \ \bullet\ R ; Y = R' \ \wedge\ S ; Y = S'$





Pushouts of Graph Homomorphisms: "Gluing"



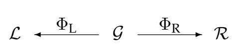


Such a pushout can be understood as:

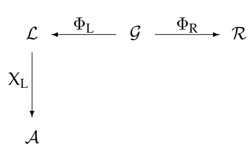
gluing $\mathcal{B}$ and $\mathcal{C}$ together "along the interface $\xleftarrow{P} \mathcal{A} \xrightarrow{Q}$".

Double-Pushout Rewriting

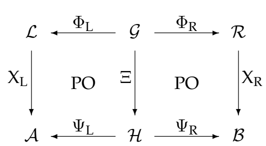
Rule:



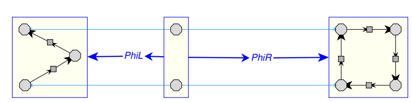
Redex:



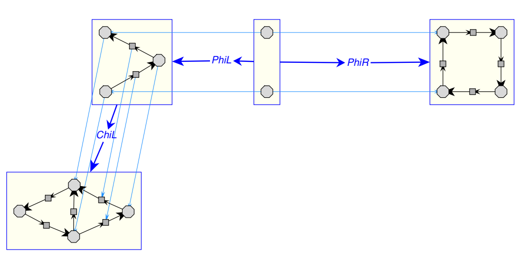
Rewriting step:



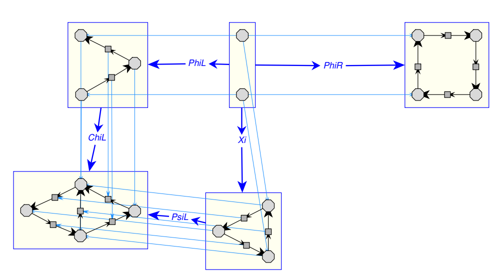
Example Double-Pushout Rewriting Step: Rule



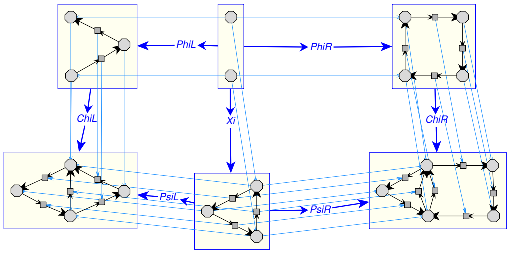
Example Double-Pushout Rewriting Step: Redex



Example Double-Pushout Rewriting Step: Host



Example Double-Pushout Rewriting Step: Result



The Power of Double-Pushout Rewriting

- **easy to understand**
- **easy to implement**
- can { delete / identify / add } **precisely specified** items
- **cannot** duplicate or delete **loosely specified** items
  — no **"subgraph variables"**

DPO graph rewriting is the most widely used graph transformation formalism.

- Describing evolution/execution of systems modelled as graphs
- Defining model transformations (e.g., of UML diagrams) for system development

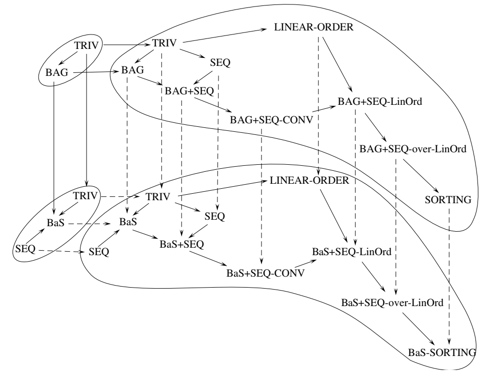
The Power of Gluing

- Gluing via pushouts (or more general colimits) works in many intersting categories
- A component specifications consists of a signature and axioms
- Such component specifications form a category; specification homomorphism can structure comples specifications:

```
spec BinRel is
  sort E
  op _le_ : E, E → Boolean
  end-spec
```
→
```
spec PreOrder is
  import BinRel
  axiom reflexivity is x le x
  axiom transitivity is
    x le y ∧ y le z ⟹ x le z
  end-spec
```
↓
```
spec Antisymmetry is
  import BinRel
  axiom antisymmetry is
    x le y ∧ y le x ⟹ x = z
  end-spec
```

- Specification homomorphism can also be used for **refinement** —
  this method is used for **correct-by-construction software development**

Refining Bags to Sets in Sorting [Smith 1998]



Logical Reasoning for Computer Science

COMPSCI 2LC3

McMaster University, Fall 2023

Wolfram Kahl

2023-12-06

Part 2: Conclusion

Organisation

Extra TA office hours — **Details to be announced — current plan:**

- Thursday, Dec. 7th, 1:00 to 4:00 p.m. — online only: Course help channel
- Friday, Dec. 8th, 1:00 to 4:00 p.m. — room TBA
- Saturday, Dec. 9th, 1:00 to 4:00 p.m. — room TBA
- Sunday, Dec. 10th, 1:00 to 4:00 p.m. — room TBA (if there is demand)
- Monday, Dec. 11th, 1:00 to 4:00 p.m. — room TBA

The **final exam** covers the whole course. Expect questions that combine several topics.

- COMPSCI 2LC3 on Avenue and CalcCheck$_{\text{Web}}$ remains active throughout term 2.
- Collected lecture slides will be posted under "General".
- Please fill in the course experience surveys for **all your courses!**
  ⟶ mcmaster.bluera.com/mcmaster

Proofs — (Simplified) Inference Rules —- See LADM p. 133, "Using Z" ch. 2&3

"Natural Deduction" — A Presentation of Logic for Mathematical Study of Logic

$\dfrac{P \land Q}{P}$ $\land$-Elim$_1$ $\qquad$ $\dfrac{P \land Q}{Q}$ $\land$-Elim$_2$ $\qquad$ $\dfrac{\forall\, x \bullet P}{P[x := E]}$ Instantiation ($\forall$-Elim)

$\dfrac{P}{P \lor Q}$ $\lor$-Intro$_1$ $\qquad$ $\dfrac{Q}{P \lor Q}$ $\lor$-Intro$_2$ $\qquad$ $\dfrac{P[x := E]}{\exists\, x \bullet P}$ $\exists$-Intro

$\dfrac{P \Rightarrow Q \quad P}{Q}$ $\Rightarrow$-Elim $\qquad$ $\dfrac{P \quad Q}{P \land Q}$ $\land$-Intro $\qquad$ $\dfrac{P}{\forall\, x \bullet P}$ $\forall$-Intro (prov. x not free in assumptions)

$\dfrac{\begin{matrix}\ulcorner P \urcorner \\ \vdots \\ Q\end{matrix}}{P \Rightarrow Q}$ $\Rightarrow$-Intro $\qquad$ $\dfrac{P \lor Q \quad \begin{matrix}\ulcorner P \urcorner \\ \vdots \\ R\end{matrix} \quad \begin{matrix}\ulcorner Q \urcorner \\ \vdots \\ R\end{matrix}}{R}$ $\lor$-Elim $\qquad$ $\dfrac{(\exists\, x \bullet P) \quad \begin{matrix}\ulcorner P \urcorner \\ \vdots \\ R\end{matrix}}{R}$ $\exists$-Elim (prov. x not free in R, assumptions)

About Natural Deduction

Example proof (using the inference rules as shown in Using Z):

$$\dfrac{\dfrac{\ulcorner \exists x : a \bullet p \Rightarrow q \urcorner^{[1]} \qquad \dfrac{\ulcorner x \in a \urcorner^{[3]} \qquad \dfrac{\ulcorner p \Rightarrow q \urcorner^{[3]} \qquad \dfrac{\ulcorner \forall x : a \bullet p \urcorner^{[2]} \quad \ulcorner x \in a \urcorner^{[3]}}{p}\ \forall\text{-elim}}{q}\ \Rightarrow\text{-elim}}{\exists x : a \bullet q}\ \exists\text{-intro}}{\dfrac{\exists x : a \bullet q}{(\forall x : a \bullet p) \Rightarrow (\exists x : a \bullet q)}\ \Rightarrow\text{-intro}^{[2]}}\ \exists\text{-elim}^{[3]}}{(\exists x : a \bullet p \Rightarrow q) \quad \Rightarrow \quad ((\forall x : a \bullet p) \Rightarrow (\exists x : a \bullet q))}\ \Rightarrow\text{-intro}^{[1]}$$

- Each formula construction C has:
  - **Introduction rule(s):** How to prove a C-formula?
  - **Elimination rule(s):** How to use a C-formula to prove something else?
- Tactical theorem provers (Coq, Isabelle) provide methods to (virtually) construct such trees piecewise from all directions
- Several of the Natural Deduction inference rules correspond
  - to LADM Metatheorems or proof methods,
  - to CalcCheck proof structures.

Writing Proofs

- Natural deduction was designed as a variant of **sequent calculus** that closely corresponds to the "natural" way of reasoning used in traditional mathematics.
- As such, natural deduction rules constitute building blocks of proof strategies.
- Natural deduction inference trees are **not normally used for proof presentation.**
- CalcCheck structured proofs are **readable formalisations** of conventional informal proof presentation patterns.
- If you wish to write prose proofs, you still need to get the right proof structure first — **think CalcCheck!**
- For proofs, **informality as such is not a value.**
  Rigorous (informal) proofs (e.g. in LADM)
  strive to "make the eventual formalisation effort minimal".
- There is value to **readable proofs**, no matter whether formal or informal.
- There is value to **formal, machine-checkable proofs**,
  especially in the software context,
  where the world of mathematics is not watching.

Strive for readable formal proofs!

Proofs for Software

- **Partial correctness**: Verifying essential functionality
- **Total correctness**: Verifying also termination
- Absence of **run-time errors** imposes additional preconditions on commands
- Termination is typically dealt with separately requires a **well-founded** "termination order".

These are supported by tools like Frama-C, VeriFast, Key, . . . :

- Hoare calculus inference rules are turned into **Verification Condition Generation**
- Many simple verification conditions can be proved using SMT solvers (Satisfiability Modulo Theories) — Z3, veriT, . . .
- More complex properties may need human assitance:
  Proof assistants: Isabelle, Coq, PVS, Agda, . . .
- Pointer structures require an extension of Hoare logic: **Separation Logic**

Industry has more and more **formal methods jobs**!

- Legacy C/C++ code needs to be analysed for issues
- Legacy C/C++ code bases are still growing. . .

Mathematical Programming Languages

- **Software is a mathematical artefact**
- **Functional programming languages** and **logic programming languages** aim to make expression in mathematical manner easier
- Among reasonably-widespread programming languages. **Haskell** is "the most mathematical"
- **Dependently-typed logics** (e.g., Coq, Lean, PVS, Agda) make it possible to express mathematics in a natural way:
  - For a matrix $M : \mathbb{R}^{3\times 4}$, the element access $M_{5,6}$ raises a **type error**
  - A simple graph (V, E) can consist of a **type** V and a relation $E : V \leftrightarrow V$.
- **Dependently-typed programming languages** (e.g., Agda, Idris)
  - contain dependently-typed logics — "proofs are programs, too"
  - make it possible to express functional specifications via the type system — "formulae as types": **Curry-Howard correspondence**
  - A program that has not been proven correct wrt. the stated specification does not even compile.

Continued Use of Logical Reasoning

- **COMPSCI 2AC3 Automata and Computability**
  — formal languages, grammars, finite automata, transition relations, Kleene algebra! acceptance predicates, ...
- **COMPSCI 2SD3 Concurrent Systems Design**
  —**correctness of concurrent programs, may use temporal logic**
- **COMPSCI 2DB3 Databases**
  — n-ary relations, relation**al** algebra; functional dependencies
- COMPSCI **3MI3 Principles of Programming Languages**
  — Programming paradigms, including functional programming; mathematical understanding of prog. language constructs, semantics
- **3RA3 Software Requirements**
  — Capturing **precisely** what the customer wants, formalisation
- COMPSCI **3EA3 Software and System Correctness**
  — Formal specifications, validation, verification
- COMPSCI **4FP3 Advanced Functional Programming**

Concluding Remarks

- How do I find proofs? — There is no general recipe
- Proving is somewhat like doing puzzles — **practice helps**
- **Proofs** are especially **important for software** — and much care is needed!
- Be aware of **types**, both in programming, and in mathematics
- Be aware of **variable binding** — in quantification, local variables, formal parameters
- Strive to use **abstraction** to **avoid variable binding**
  — e.g., using relation algebra instead of predicate logic
- When designing **data representations**, **think mathematics**: **Subsets, relations, functions, injectivity**, ...
- **Thinking mathematics in programming** is easiest in functional languages, e.g., **Haskell**, OCaml
- **Specify formally!** — **Design for provability!**
- **When doing software, think logics and discrete mathematics!**