CAS 701
TERM REWRITING

Hong Ni
Huan Zhang

Outline

- Motivation
- Introduction
- Rewrite Rules
- Basic Concept
- The TR Algorithm
- Examples
- Thoughts

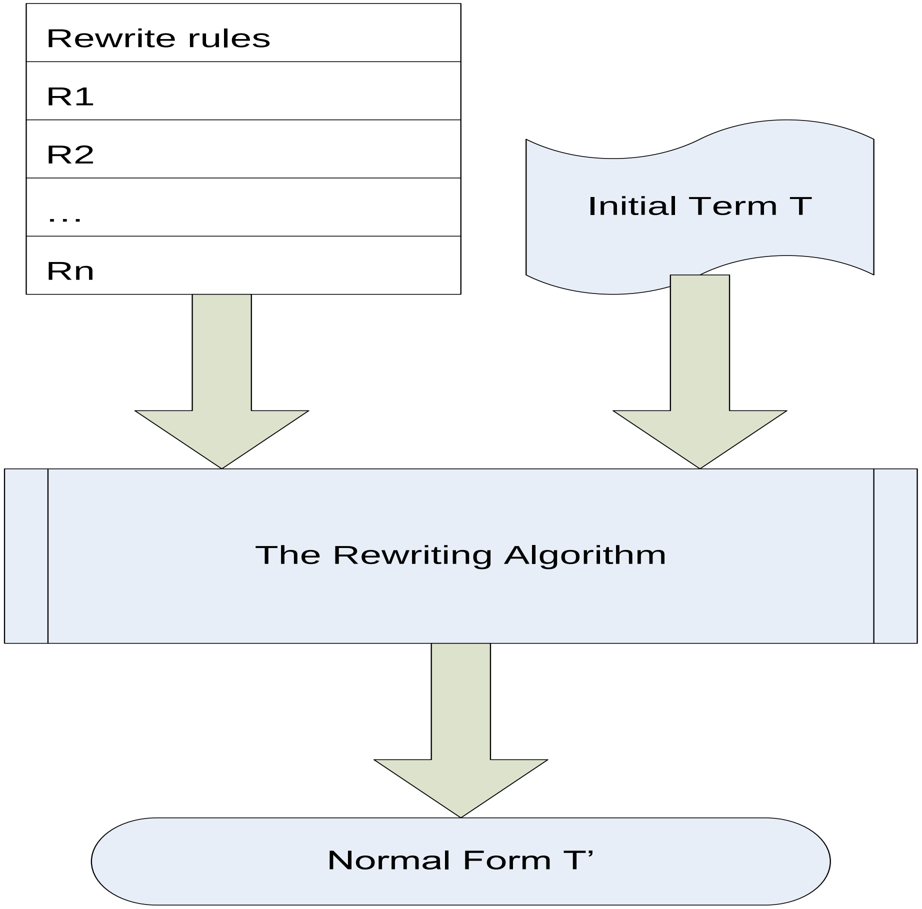
Motivations

- Suitable for computational processes based on the repeated application of simplification rules.
- Suitable for tasks like symbolic computation, program analysis and program transformation.
- Term rewriting helps to solve such tasks in a very effective and symbolic manner.

Introduction

- Term rewriting: the initial expression is simplified in a number of rules.
- There is a complex *Left-hand side* that can be simplified into the expression appearing at the *right-hand side*.
  - terms
  - variables

Introduction



Rewrite Rules

The initial term is gradually reduced…

- An initial expression that is to be simplified.
- Finding a match - there must be a *match* between the redex and the *left-hand side* of the rule.
  - *redex – **red**ucible **ex**pression*
- Replacing – the redex in the initial expression is replaced by the right-hand side of the rule.

The outcome can be called as ***normal form***.

Basic Concepts

- Terms
- Substitution
- Matching

Terms

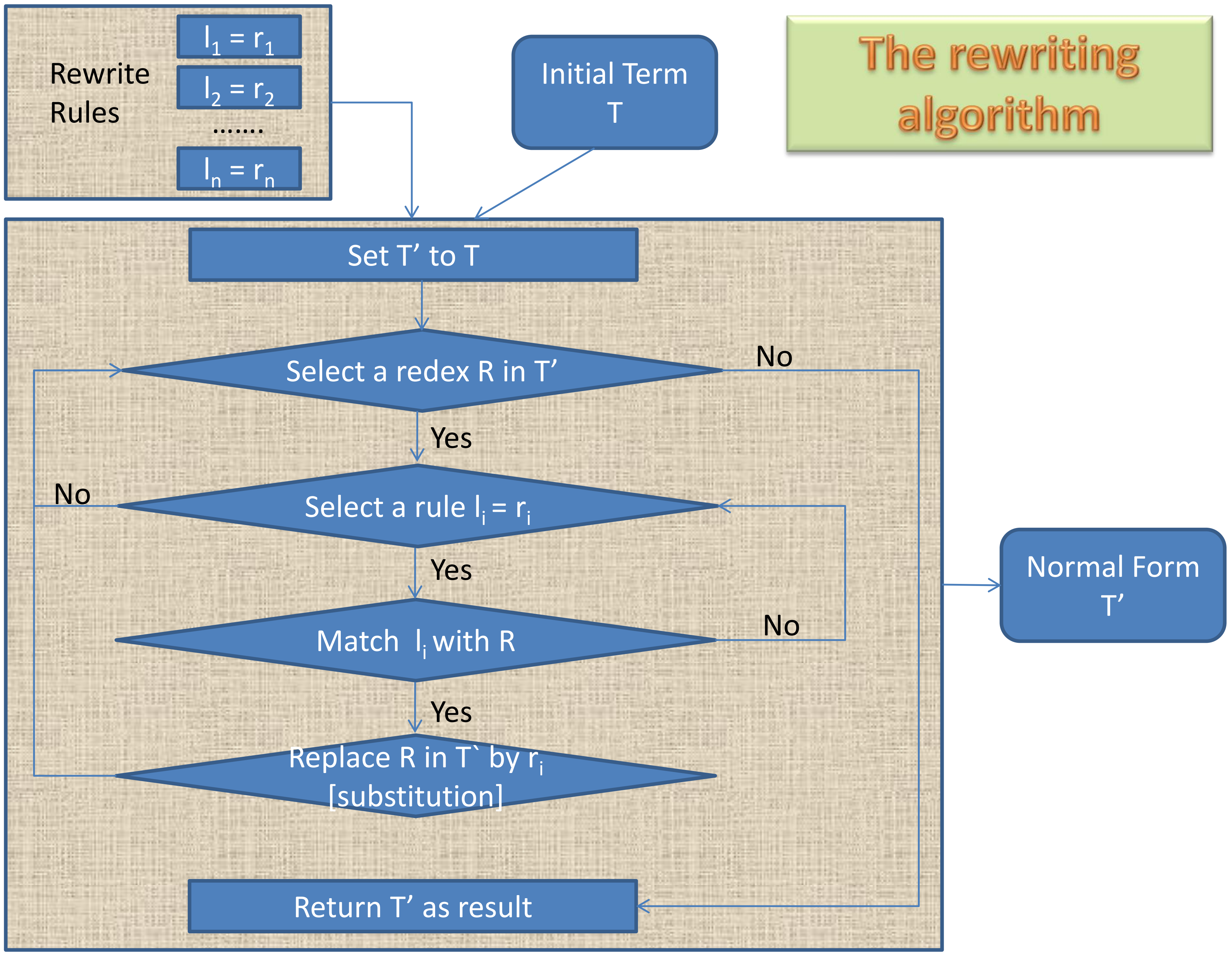
- Terms are defined in a prefix format
  - A single variable is a term, e.g. X, Y or Z
  - The function name applied to zero or more arguments is a term, e.g. add(X, Y)
- Complex hierarchical structures of arbitrary depth can be defined.

Substitution

- A substitution is an association between variables and terms.
  - For example, {X→0, Y→succ(0)}.
- Substitution can be used to create new terms from old ones.
  - For example, using the above substitution and applying it to the term mul(succ(X), Y) will yield the new term mul(succ(0), succ(0)).
- The basic idea is that variables are replaced by the term they are mapped to by the substitution.

Matching

- A matching sets as a goal to determine whether two terms can be made equal.
  - For example, the two terms mul(succ(X), Y) and mul(succ(0),succ(0)) match since we can use the substitution {X→Y, Y→succ(0)} to make them identical.
- If no such substitution can be found, the two terms cannot be matched.



The rewriting algorithm
Rewrite Rules
$l_1 = r_1$
$l_2 = r_2$
…….
$l_n = r_n$
Initial Term T
Set T' to T
Select a redex R in T'
No
Yes
Select a rule $l_i = r_i$
No
Yes
Match l_i with R
No
Yes
Replace R in T` by r_i [substitution]
Return T' as result
Normal Form T'

Normal Forms

- To get terms rewritten to a ‘simplest’ term, where this term cannot be modified any further from the rules in the rewriting system.
- Unique?
  - T = {a, b} with rules a→b, b→a. [not unique]
  - Terms can be rewritten regardless of the choice of rewriting rule to obtain the same normal form is know as confluence.

Numerals Example

Rewrite Rules

- [add1] add(0, X) = X
- [add2] add(succ(X), Y) = succ(add(X, Y))
- [mul1] mul(0, X) = 0
- [mul2] mul(succ(X), Y) = add(mul(X, Y), Y)

Initial Term T = **add(succ(succ(0)), succ(succ(0)))**

add (succ (succ(0)), succ (succ(0)))

↓ [add2]

succ (add (succ(0), succ (succ(0))))

↓ [add2]

succ (succ (add (0, succ (succ(0)))))

↓ [add1]

succ (succ (succ (succ (0))))

Logic Example

Rewrite Rules

- [double neg. Eli.] $\neg\neg p = p$
- [→ Eli.] $p \rightarrow q = \neg p \vee q$
- [De Morgan's laws] $\neg(p \wedge q) = \neg p \vee \neg q$
  $\neg(p \vee q) = \neg p \wedge \neg q$
- [Distributivity] $(p \wedge q) \vee r = (p \vee r) \wedge (q \vee r)$

Initial Term T = $\neg(\ ((p \wedge q) \vee r) \rightarrow \neg\neg m)$

$\neg(\ ((p \wedge q) \vee r) \rightarrow \neg\neg m)$

↓ [Distributivity]

$\neg(\ ((p \vee r) \wedge (q \vee r)) \rightarrow \neg\neg m)$

↓ [double neg. eli]

$\neg(\ ((p \vee r) \wedge (q \vee r)) \rightarrow m)$

↓ [→ Eli.]

$\neg(\neg((p \vee r) \wedge (q \vee r)) \vee m)$

↓ [De Morgan's Laws

$\neg\neg(p \vee r) \wedge (q \vee r) \wedge \neg m$

↓ [double neg. eli]

$(p \vee r) \wedge (q \vee r) \wedge \neg m$

Booleans Example

Initial Term T = **not (or(false, and(true, not (false))))**

Rewrite Rules

- [or1] or(true, true) = true
- [or2] or(true, false) = true
- [or3] or(false, true) = true
- [or4] or(false, false) = false
- [and1] and(true, true) = true
- [and2] and(true, false) = false
- [and3] and(false, true) = false
- [and4] and(false, false) = false
- [not1] not(true) = false
- [not2] not(false) = true

not (or(false, and(true, not (false))))

↓ [not2]

not (or(false, and(true, true)))

↓ [and1]

not (or(false, true))

↓ [or3]

not (true)

↓ [not1]

false

Extensions of Term Rewriting

- User-defined syntax
  - Relax the strict prefix format of functions and use arbitrary notation,
    - add(0, X) = X ⟹ 0 + X = X
    - and(true, false) ⟹ true & false
- Conditional rules
  - One or more conditions are attached that are first evaluated in order to determine whether the rule should be applied at all
- Traversal function
  - Reduce the number of rules

Extensions of Term Rewriting

- Term Rewriting Basics
- Knuth-Bendix completion procedure
  - An algorithm for transforming a set of equations into confluent term rewriting system. When succeeds, it has effectively solved the word problem for the specified algebra
- Lindenmayer
  - Most famously used to model the growth process of plant development

END

Thank you !