

Term Rewriting Systems

Arvind
Laboratory for Computer Science
M.I.T.

Lecture 22

<http://www.csg.lcs.mit.edu/6.827>

Outline

- Motivation for rewriting
- TRS Syntax
 - applicative TRS
- Some properties of TRS's
 - Strong normalization
 - Confluence
- Some special TRS's
 - underlined TRS
 - orthogonal TRS
 - Recursive Program Schemes (RPS)
 - Applicative RPS

<http://www.csg.lcs.mit.edu/6.827>

Equational Specifications

$$E \quad \begin{array}{l} A(x,0) = x \\ A(x,S(y)) = S(A(x,y)) \\ M(x,0) = 0 \\ M(x,S(y)) = A(M(x,y),x) \end{array}$$

E is an *equational specification* of natural numbers.

An equation is between *terms*

The *signature* Σ for E

Function symbol	Arity	aka Constants
0	0	
S	1	
A	2	
M	2	

<http://www.csg.lcs.mit.edu/6.827>



Equational Theory

" $E \Vdash t = s$ " means that $t = s$ can be derived from the equations in E by the following rules:

Substitution:

$$E \Vdash t(x_1, \dots, x_n) = s(x_1, \dots, x_n)$$

$$E \Vdash t(t_1, \dots, t_n) = s(t_1, \dots, t_n)$$

Forming Contexts:

$$E \Vdash t = s \ \& \ C[] \text{ is a context}$$

$$E \Vdash C[t] = C[s]$$

Symmetry, Reflexivity and Transitivity of "=" :

$$E \Vdash t = s \Rightarrow E \Vdash s = t$$

$$E \Vdash t = t$$

$$E \Vdash t = s \ \& \ E \Vdash s = t' \Rightarrow E \Vdash t = t'$$

<http://www.csg.lcs.mit.edu/6.827>



Decision Procedure

Is there a procedure to decide

if $E \Vdash t1 = t2$

In general , NO!

The notion of *reduction* or *rewriting* was originally developed to understand questions regarding decision procedures.

<http://www.csg.lcs.mit.edu/6.827>



Term Rewriting Systems (TRS)

A TRS is a (Σ, R)

where Σ is a *signature* and

R is a set of *rewrite rules* for terms over Σ

R	$A(x,0)$	$\rightarrow x$
	$A(x,S(y))$	$\rightarrow S(A(x,y))$
	$M(x,0)$	$\rightarrow 0$
	$M(x,S(y))$	$\rightarrow A(M(x,y),x)$

Σ for R

Function symbol	Arity	
0	0	aka Constants
S	1	
A	2	
M	2	

$A(S(0),S(0)) \rightarrow$

<http://www.csg.lcs.mit.edu/6.827>



Syntax: *Terms*

A *signature* Σ consists of a set of constants, function symbols and infinitely many variables.

terms over Σ

$$t = x \mid c \mid F^k(t_1, \dots, t_k)$$

variable
constant
application

Open term: A term that contains a variable.

Closed term: A term without a variable.
a.k.a. *Ground term*



Rewrite Rules

$$t1 \rightarrow t2$$

1. $t1$ must not be a variable;
2. Free variables of $t2$ must be contained in the free variables of $t1$

Examples of illegal rules

$$\begin{aligned} x &\rightarrow A(x,0) \\ F(x) &\rightarrow y \end{aligned}$$

Sometimes it is convenient to disallow rules to rewrite *constants*, the 0-arity function symbols.

Variables of a rule are sometimes called the *meta variables* and *range over all terms in the signature*.



Substitution

- $A(x,0) \rightarrow x$ (1)
- $A(x,S(y)) \rightarrow S(A(x,y))$ (2)
- $M(x,0) \rightarrow 0$ (3)
- $M(x,S(y)) \rightarrow A(M(x,y),x)$ (4)

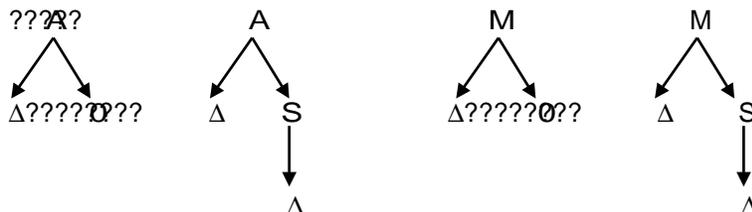
Does any rule apply to the term $M(S(S(0)),S(0))$?



Pattern of a Rule

- $A(x,0) \rightarrow x$
- $A(x,S(y)) \rightarrow S(A(x,y))$
- $M(x,0) \rightarrow 0$
- $M(x,S(y)) \rightarrow A(M(x,y),x)$

Replace variables on the LHS by Δ



A rule applies to a term if the rule pattern matches some node in the syntax tree of the term (Δ matches any node)



Rewriting

One-step rewriting \rightarrow

Application of one rule in a context

Multiple-step rewriting

$t \equiv t_1 \rightarrow t_2 \rightarrow \dots \rightarrow t_n \equiv s$

may be rewritten as $t \twoheadrightarrow s$

Rewriting can be thought of as \twoheadrightarrow inducing a relation on terms, thus

$\twoheadrightarrow = \text{Transitive, reflexive closure of } \rightarrow$

In any semantic model, the terms t_1, t_2, \dots, t_n must have the same meaning!



Applicative TRS

A TRS that consists of a one special binary operator called *application* (Ap), and some constants.

Example: Combinatory Logic

Constants: S, K

Rewrite rules:

$$\begin{array}{l} Ap(Ap(Ap(S,x),y),z) \rightarrow Ap(Ap(x,z),Ap(y,z)) \\ Ap(Ap(K,x),y) \rightarrow x \end{array}$$


Special Notation for Applicative TRS

An infix version of Ap

$$\begin{array}{l} ((S.x).y).z \quad \rightarrow (x.z).(y.z) \\ (K.x).y \quad \rightarrow x \end{array}$$

The "." is often suppressed in programming

$$\begin{array}{l} ((S x) y) z \quad \rightarrow (x z) (y z) \\ (K x) y \quad \rightarrow x \end{array}$$

and by convention parentheses associative to the left

$$\begin{array}{l} S x y z \quad \rightarrow x z (y z) \\ K x y \quad \rightarrow x \end{array}$$

<http://www.csg.lcs.mit.edu/6.827>



The S-K Combinatory System

$$\begin{array}{l} S x y z \quad \rightarrow x z (y z) \\ K x y \quad \rightarrow x \end{array}$$

Any computable function can be expressed using S's and K's !

Example: Identity function "I x → x"

$$S K K x \rightarrow$$

<http://www.csg.lcs.mit.edu/6.827>



Mixed Notation

We can mix applicative and functional notation

$$\begin{array}{ll} S \ x \ y \ z & \rightarrow x \ z \ (y \ z) \\ K \ x \ y & \rightarrow x \\ D(x,x) & \rightarrow E \end{array}$$

The above system is very different from

$$\begin{array}{ll} S \ x \ y \ z & \rightarrow x \ z \ (y \ z) \\ K \ x \ y & \rightarrow x \\ D \ x \ x & \rightarrow E \end{array}$$

where D is a constant, that is,

$$Ap(Ap(D,x),x) \rightarrow E$$

<http://www.csg.lcs.mit.edu/6.827>



Arity - some bad terminology

A bad terminology is to say that

the "arity" of S is 3,
or the "arity" of S is variable.

S is a *constant*, or a zero arity function symbol;
Ap has arity 2, and the rewrite rule for S requires
three Ap symbols and three arguments

$$S \ t1 \ t2 \ t3 \ t4 \ t5 \rightarrow$$

<http://www.csg.lcs.mit.edu/6.827>



Normal Form

Let (Σ, R) be a TRS and t be a term

t is in *normal form* if it cannot be reduced any further.

Term t is *strongly normalizing (SN)* if every reduction sequence starting from t terminates eventually.

R is *strongly normalizing (SN)* if for all terms every reduction sequence terminates eventually.

R is *weakly normalizing (WN)* if for all terms there is some reduction sequence that terminates.



Strongly Normalizing?

$$1. \quad \begin{array}{l} \text{Arb}(x,y) \quad \rightarrow x \\ \text{Arb}(x,y) \quad \rightarrow y \end{array}$$

$$2. \quad \text{F}(0,1,x) \quad \rightarrow \text{F}(x,x,x)$$

$$3. \quad \begin{array}{l} \text{Arb}(x,y) \quad \rightarrow x \\ \text{Arb}(x,y) \quad \rightarrow y \\ \text{F}(0,1,x) \quad \rightarrow \text{F}(x,x,x) \end{array}$$



Underlined Version of a TRS

Combinatory Logic

$$\begin{array}{l} S \ x \ y \ z \ \rightarrow x \ z \ (y \ z) \\ K \ x \ y \ \rightarrow x \end{array}$$

Its *underlined version*

-- Extend the signature by S and K

$$\begin{array}{l} \underline{S} \ x \ y \ z \ \rightarrow x \ z \ (y \ z) \\ \underline{K} \ x \ y \ \rightarrow x \end{array}$$

Is the underlined version SN ?

<http://www.csg.lcs.mit.edu/6.827>



Underlined TRS

Given a TRS R , its underlined version \underline{R} is defined as follows:

1. The signature of \underline{R} contains all the symbols of R and the underlined version of each symbol of R .
2. For each rule in R , \underline{R} contains a rule gotten by replacing the left most symbol of the rule in R by its underlined version.

<http://www.csg.lcs.mit.edu/6.827>



Underlining and Development

Underline some redexes in a term.

Development is a reduction of the term such that only underlined redexes are done.

Complete Development is a reduction sequence such that all the underlined redexes have been performed.

$$\begin{array}{l}
 (\underline{S} K x (\underline{K} y z)) \\
 \rightarrow (\underline{S} K x y) \quad \left| \quad \rightarrow K (\underline{K} y z) (x (\underline{K} y z)) \right. \\
 \rightarrow K y (x y) \quad \left| \quad \rightarrow K y (x (\underline{K} y z)) \right. \\
 \quad \quad \quad \left| \quad \rightarrow K y (x y) \right.
 \end{array}$$

By underlining redexes we can distinguish between old and newly created redexes in a reduction sequence.

<http://www.csg.lcs.mit.edu/6.827>



Underlined TRS

Theorem: For every TRS R , \underline{R} is strongly normalizing.

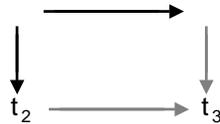
The proof is based on assigning weights to each rule such that there is a *Decreasing weight property* for each redex.

<http://www.csg.lcs.mit.edu/6.827>



Confluence *aka Church-Rosser Property*

A reduction system R is said to be *confluent (CR)*, if $t \rightarrow t_1$ and $t \rightarrow t_2$ then there exists a t_3 such that $t_1 \rightarrow t_3$ and $t_2 \rightarrow t_3$.



Fact: In a confluent system, if a term has a normal form then it is *unique*.

Are all TRS's confluent?

<http://www.csg.lcs.mit.edu/6.827>



Confluence is difficult to Prove

$$\begin{aligned} A(x,0) &\rightarrow x \\ A(x,S(y)) &\rightarrow S(A(x,y)) \\ M(x,0) &\rightarrow 0 \\ M(x,S(y)) &\rightarrow ?A(M(x,y),x) \end{aligned}$$

$$\begin{aligned} \text{Ack}(0,x) &\rightarrow S(x) \\ \text{Ack}(S(y),0) &\rightarrow \text{Ack}(x,S(0)) \\ \text{Ack}(S(x),S(y)) &\rightarrow \text{Ack}(x,\text{Ack}(S(x),y)) \end{aligned}$$

$$\begin{aligned} S \ x \ y \ z &\rightarrow x \ z \ (y \ z) \\ K \ x \ y &\rightarrow x \end{aligned}$$

<http://www.csg.lcs.mit.edu/6.827>



Orthogonal TRSs

A TRS is *Orthogonal* if it is:

1. *Left Linear*: has no multiple occurrences of a variable on the LHS of any rule, and
2. *Non Interfering*: patterns of rewrite rules are pairwise non-interfering

Theorem: An Orthogonal TRS is Confluent.

<http://www.csg.lcs.mit.edu/6.827>



Orthogonal TRS: *Examples*

$$\begin{array}{ll} A(x,0) & \rightarrow x \\ A(x,S(y)) & \rightarrow S(A(x,y)) \\ M(x,0) & \rightarrow 0 \\ M(x,S(y)) & \rightarrow ?A(M(x,y),x) \end{array}$$

$$\begin{array}{ll} \text{Ack}(0,x) & \rightarrow S(x) \\ \text{Ack}(S(y),0) & \rightarrow \text{Ack}(x,S(0)) \\ \text{Ack}(S(x),S(y)) & \rightarrow \text{Ack}(x,\text{Ack}(S(x),y)) \end{array}$$

$$\begin{array}{ll} S\ x\ y\ z & \rightarrow x\ z\ (y\ z) \\ K\ x\ y & \rightarrow x \end{array}$$

<http://www.csg.lcs.mit.edu/6.827>



Recursive Program Scheme (RPS)

An RPS is a TRS such that

$G = \{ G_1, \dots, G_n \}$ are base functions with non-interfering rules

$F = \{ F_1, \dots, F_m \}$ are user-defined functions such that

1. $G \cap F = \emptyset$

2? There is at most one rule for each F_i in F

$$F_i(x_1, \dots, x_k) = t_i$$

where each x_i is distinct and each t_i is built from x_1, \dots, x_k , and symbols from F and G

Fact: An RPS is an orthogonal TRS.

\Rightarrow RPS is confluent!

<http://www.csg.lcs.mit.edu/6.827>



Applicative RPS

It is the same as a functional RPS except that it is defined using applicative format.

We can generate an applicative TRS R^{ap} from a functional TRS R as follows:

For each rule $t_1 \rightarrow t_2$ in R , R^{ap} contains the rule $t_1^{ap} \rightarrow t_2^{ap}$ where t^{ap} means

$$F(t_1, \dots, t_n)^{ap} \Rightarrow F(t_1^{ap}, \dots, t_n^{ap})$$

Theorem: If R is confluent then so is R^{ap} .

<http://www.csg.lcs.mit.edu/6.827>

