Term Rewriting Systems

Arvind
Laboratory for Computer Science
M.I.T.

December 2, 2002

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

Equational Specifications

E

$A(x,0) = x$
$A(x,S(y)) = S(A(x,y))$
$M(x,0) = 0$
$M(x,S(y)) = A(M(x,y),x)$

E is an *equational specification* of natural numbers.

An equation is between *terms*

The *signature* Σ?for E

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

Equational Theory

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

Substitution:

E |-- $t\,(x1, \ldots, xn) = s\,(x1, \ldots, xn)$

E |-- $t\,(t1, \ldots, tn) = s\,(t1, \ldots, tn)$

Forming Contexts:

E |-- $t = s$ & C[ ] is a context

E |-- C[t] = C[s]

Symmetry, Reflexivity and *Transitivity* of "=" *:*

E |-- $t = s$ $\Rightarrow$ E |-- $s = t$
E |-- $t = t$
E |-- $t = s$ & E |-- $s = t'$ $\Rightarrow$ E |-- $t = t'$

Decision Procedure

Is there a procedure to decide

if E |-- *t1* = *t2*

In general , NO!

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

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)	→	x
A(x,S(y))	→	S(A(x,y))
M(x,0)	→	0
M(x,S(y))	→	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)) →

Syntax: *Terms*

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

terms over Σ

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

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 in the free variables of *t1*

Examples of illegal rules

$x \rightarrow A(x,0)$

$F(x) \rightarrow y$

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

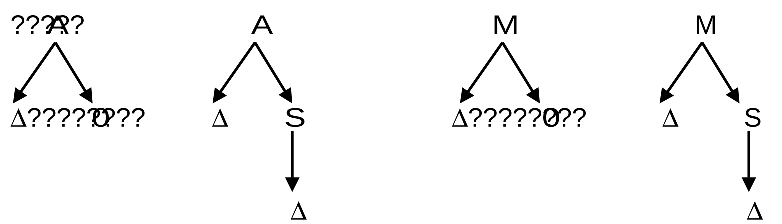
A(x,0)	$\rightarrow$ x	(1)
A(x,S(y))	$\rightarrow$ S(A(x,y))	(2)
M(x,0)	$\rightarrow$ 0	(3)
M(x,S(y))	$\rightarrow$ A(M(x,y),x)	(4)

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

$t \equiv t1 \rightarrow t2 \rightarrow \ldots \rightarrow tn \equiv s$

may be rewritten as $t \twoheadrightarrow s$

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

$\twoheadrightarrow$ = *Transitive*, *reflexive closure* of $\rightarrow$

In any semantic model, the terms t1, t2, ..., tn 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:

Ap(Ap(Ap(S,x),y),z) $\rightarrow$ Ap(Ap(x,z),Ap(y,z))
Ap(Ap(K,x),y) $\rightarrow$ x

Special Notation for Applicative TRS

An infix version of Ap

((S.x).y).z → (x.z).(y.z)
(K.x).y → x

The "." is often suppressed in programming

((S x) y) z → (x z) (y z)
(K x) y → x

and by convention parentheses associative to the left

S x y z → x z (y z)
K x y → x

The S-K Combinatory System

S x y z → x z (y z)
K x y → x

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

Example: Identity function "I x → x"

S K K x →

Mixed Notation

We can mix applicative and functional notation

S x y z → x z (y z)
K x y → x
D(x,x) → E

The above system is very different from

S x y z → x z (y z)
K x y → x
D x x → E

where D is a constant, that is,

Ap(Ap(D,x),x) → E

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 →

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. Arb(x,y) → x
   Arb(x,y) → y

2. F(0,1,x) → F(x,x,x)

3. Arb(x,y) → x
   Arb(x,y) → y
   F(0,1,x) → F(x,x,x)

Underlined Version of a TRS

Combinatory Logic

$$S\ x\ y\ z \rightarrow x\ z\ (y\ z)$$
$$K\ x\ y \rightarrow x$$

Its *underlined version*

-- Extend the signature by $\underline{S}$ and $\underline{K}$

$$\underline{S}\ x\ y\ z \rightarrow x\ z\ (y\ z)$$
$$\underline{K}\ x\ y \rightarrow x$$

Is the underlined version SN ?

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.

L22-21
Arvind

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.

$(\underline{S}\ \ K\ \ x\ \ (\underline{K}\ \ y\ \ z))$

$\rightarrow (\underline{S}\ \ K\ \ x\ \ y)$	$\rightarrow K\ \ (\underline{K}\ \ y\ \ z\ )\ \ (x\ \ (\underline{K}\ \ y\ \ z\ ))$
$\rightarrow K\ y\ \ (x\ \ y)$	$\rightarrow K\ \ y\ \ (x\ \ (\underline{K}\ \ y\ \ z))$
	$\rightarrow K\ y\ \ (x\ \ y)$

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

L22-22
Arvind

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.

Confluence *aka Church-Rosser Property*

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

$$\begin{tikzcd} t \arrow[r, twoheadrightarrow] \arrow[d] & t_1 \arrow[d] \\ t_2 \arrow[r, twoheadrightarrow] & t_3 \end{tikzcd}$$

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

Are all TRS's confluent?

Confluence is difficult to Prove

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)

Ack(0,x) → S(x)
Ack(S(y),0) → Ack(x,S(0))
Ack(S(x),S(y)) → Ack(x,Ack(S(x),y))

S x y z → x z (y z)
K x y → x

L22-25
Arvind

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.

December 2, 2002 http://www.csg.lcs.mit.edu/6.827

L22-26
Arvind

Orthogonal TRS: *Examples*

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)

Ack(0,x)	→ S(x)
Ack(S(y),0)	→ Ack(x,S(0))
Ack(S(x),S(y))	→ Ack(x,Ack(S(x),y))

S x y z	→ x z (y z)
K x y	→ x

December 2, 2002 http://www.csg.lcs.mit.edu/6.827

Recursive Program Scheme (RPS)

An RPS is a TRS such that

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

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

1. $G \cap F = \Phi$
2. There is at most one rule for each F_i in F
   $F_i (x_1, \ldots, x_k) = t_i$
   where each x_i is distinct and each t_i is built from $x_1,\ldots,x_k$, and symbols from F and G

Fact: An RPS is an orthogonal TRS.

$\Rightarrow$ RPS is confluent!

Applicative RPS

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

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

For each rule $t1 \rightarrow t2$ in R, R^{ap} contains the rule $t1^{ap} \rightarrow t2^{ap}$ where t^{ap} means

$F\ (\ t1,\ldots,\ tn\ )^{ap} \quad \Rightarrow F\ t1^{ap} \ldots\ tn^{ap}$

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