CS206 Lecture 13
Equation Logic and Term Rewriting

G. Sivakumar
Computer Science Department
IIT Bombay
siva@iitb.ac.in
http://www.cse.iitb.ac.in/∼siva

Tue, Jan 28, 2003

Plan for Lecture 13

- Overview of Equational Logic
- Rewrite Systems

Equational Logic

Standard example (Group Theory)

$$\begin{aligned} 0 + x &= x \\ -(x) + x &= 0 \\ (x + y) + z &= x + (y + z) \end{aligned}$$

Equational logic ("replace equals by equals")

$$0 = 0 + x = (-0 + 0) + x = -0 + (0 + x) = -0 + x$$

Validity: Is $-0 = 0$? Is $-(-(x)) = x$? (for all x)?
Satisfiability: Is there x such that $x + x = 0$?
Formal definition of **term**, **substitution**, **matching**, **unification** in future lectures!
We can give an "efficient" **decision** procedure for validity and a **semi-decision** procedure for satisfiabilty if we can find a **convergent (canonical) rewrite system** equivalent to the above equations.

Rewrite Systems

A **rule** is an "oriented" equation (one-way replacement).
Numbers are built from **constructors** $(0, s)$ and some functions $+, *, fact, gcd$ are **defined** as follows.

$$
\begin{array}{rcl}
0 + x & \rightarrow & x \\
s(x) + y & \rightarrow & s(x + y) \\
0 * x & \rightarrow & 0 \\
s(x) * y & \rightarrow & y + (x * y) \\
fact(0) & \rightarrow & s(0) \\
fact(s(x)) & \rightarrow & s(x) * fact(x) \\
gcd(0, x) & \rightarrow & x \\
gcd(x, x + y) & \rightarrow & gcd(x, y)
\end{array}
$$

Derivations and Normal Forms

$$s(0) + (0 * s(0)) \rightarrow s(0) + 0 \rightarrow s(0 + 0) \rightarrow s(0)$$

Equational Programming

(First-order) Functional Programming

- Evaluate $fact(s^2(0) * s^3(0))$
- **Matching** is the parameter-passing mechanism for applying rules.
- No backtracking if definition is **confluent**.

Logic Programing

- Solve $x * y = s^4(0)$
- Enumerate **all** answers: $\{x \mapsto s(0), y \mapsto s^4(0)\}$, $\{x \mapsto s^2(0), y \mapsto s^2(0)\}$, ...
- **Unification** is the parameter passing mechanism.
- Backtracking needed for completeness!

"Efficent" methods for the above are possible when the rewrite system has useful properties of **termination** and **confluence.**

Problems with GCD definition

$$\boxed{\begin{array}{rcl} gcd(0,x) & \rightarrow & x \\ gcd(x,x+y) & \rightarrow & gcd(x,y) \end{array}}$$

Is definition of **gcd** complete?

- Can we **simplify** $gcd(s^2(0), s^4(0))$?
- We need matching modulo $+$
- How about $gcd(s^4(0), s^2(0))$?
- We need **commutativity** of gcd.

Another Definiton

First, we define $diff(x, y) = |x - y|$ the absolute value of the difference as follows.

$$\begin{aligned} diff(x, 0) &\rightarrow x \\ diff(0, x) &\rightarrow x \\ diff(s(x), s(y)) &\rightarrow diff(x, y) \end{aligned}$$

Next, we define $smin(x, y) = 1 + min(x, y)$ as follows.

$$\begin{aligned} smin(x, 0) &\rightarrow s(0) \\ smin(0, x) &\rightarrow s(0) \\ smin(s(x), s(y)) &\rightarrow s(smin(x, y)) \end{aligned}$$

Using these two definitions one can now write $gcd(x, y)$ as follows.

$$\begin{aligned} gcd(x, 0) &\rightarrow x \\ gcd(0, x) &\rightarrow x \\ gcd(s(x), s(y)) &\rightarrow gcd(diff(x, y), smin(x, y)) \end{aligned}$$

Sample Derivation

A sample *derivation* using the rules above to compute $gcd(4, 2) = 2$ is shown below.

$$gcd(s^4(0), s^2(0))$$

$$\rightarrow gcd(diff(s^3(0), s(0)), smin(s^3(0), s(0)))$$

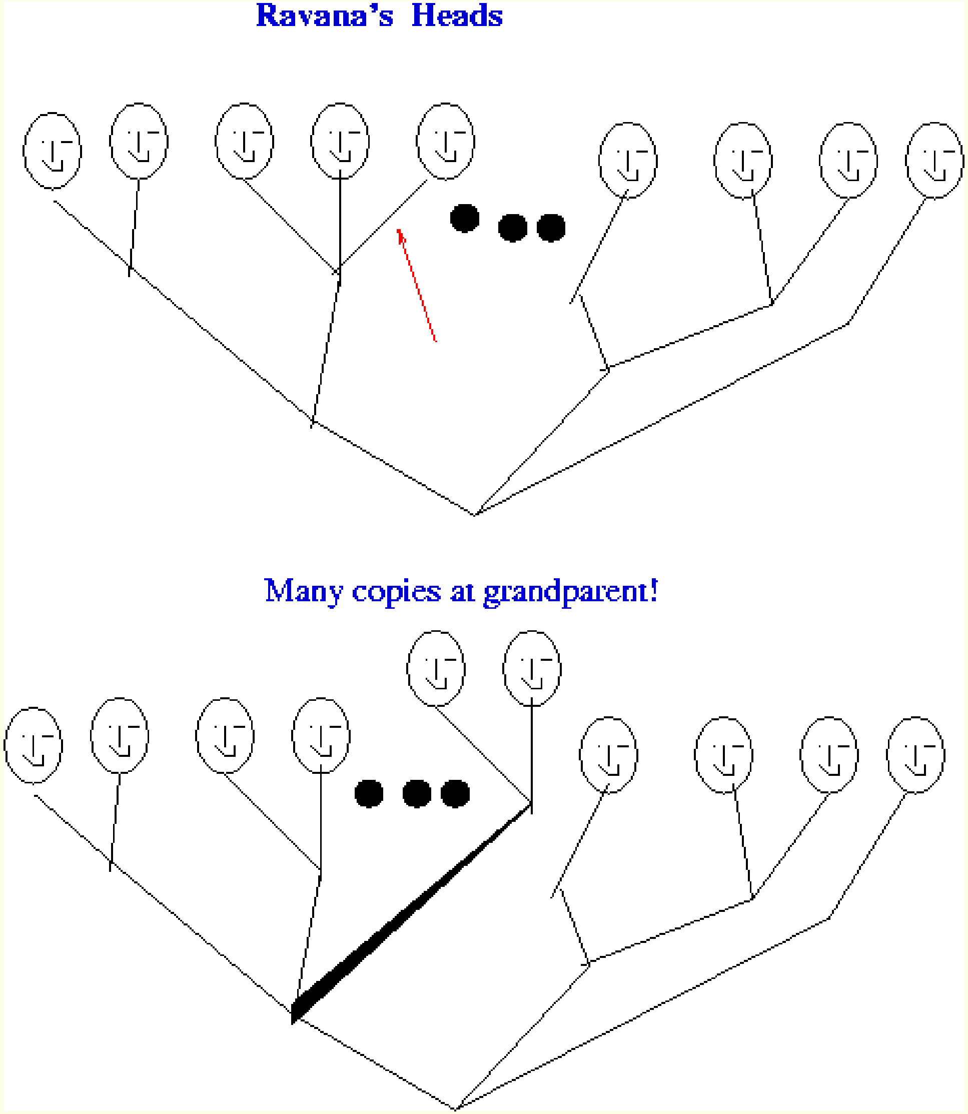
$$\rightarrow^* gcd(s^2(0), s^2(0))$$

$$\rightarrow^* gcd(0, s^2(0)) \rightarrow s^2(0)$$

Interesting Questions

1. Is there any term $gcd(m, n)$ that has more than one normal form?
2. Is the definition *sufficiently complete*? That is, does every term of the form $gcd(m, n)$ where m, n evaluate to a normal form which is a natural number?
3. Can we prove properties of gcd like commutativity $gcd(x, y) = gcd(y, x)$ for any natural numbers x, y?
4. Is there any term $gcd(m, n)$ for which there is some infinite derivation sequence $gcd(m, n) \longrightarrow t_1 \longrightarrow t_2 ...$?

Termination Puzzle



Termination

A rewrite system $\mathcal{R} = \{l_i \rightarrow r_i\}$ is terminating if there is no term t_1 such that an infinte chain

$$t_1 \rightarrow t_2 \rightarrow \ldots$$

of rewrite steps is possible using $\mathcal{R}$.
How to prove **termination**?

- **Well-founded** orderings on terms.
- **Simplification** Orderings
  - Subterm Property $(u[t] > t)$
  - Monotonicity Property ($t > s$ implies $u[t] > u[s]$)
- **Stability** under substitutions
  ($t > s$ implies for all σ, $t\sigma > s\sigma$)

Other desirable properties (totality on ground terms, maximality). Designing such orderings is quite challenging.

Unique Normal Form (Confluence)

Consider some rules for Propositional Logic.

$$\begin{array}{rcl} x \vee 0 & \rightarrow & x \\ x \wedge 1 & \rightarrow & x \\ x \wedge \neg(x) & \rightarrow & 0 \\ x \vee (y \wedge z) & \rightarrow & (x \vee y) \wedge (x \vee z) \end{array}$$

Clausal form (Disjunctive Normal Form).
The formula $x \vee (y \wedge \neg(y))$ has two normal forms x and $(x \vee y) \wedge (x \vee \neg(y))$.
Resolution based methods will **resolve** the two clauses in $(x \vee y) \wedge (x \vee \neg(y))$.
How to fix for rewriting?
Add this as a new rule?
$(x \vee y) \wedge (x \vee \neg(y)) \rightarrow x$
No, More problems!

Data Types using Rewrite Systems

Quite easy to model and reason about many data types. nil and $\cdot$ constructors for list, $empty$ and $push$ constructors for stack.

$$\begin{array}{rcl} top(push(x,y)) & \rightarrow & x \\ pop(push(x,y)) & \rightarrow & y \\ append(nil,Z) & \rightarrow & Z \\ append(X\cdot Y,Z) & \rightarrow & X\cdot append(Y,Z) \\ rev(nil) & \rightarrow & nil \\ rev(X\cdot Y) & \rightarrow & append(rev(Y),X\cdot nil) \end{array}$$

Are properties such as **associativity** of append or $rev(rev(X)) = X$ valid? (equational proofs exist?).

Inductive Properties and Proofs

Example of series summation

$$\boxed{\begin{array}{rcl} ssum(0) & \rightarrow & 0 \\ ssum(s(x)) & \rightarrow & s(x) + ssum(x) \end{array}}$$

Can we prove

$$s^2(0) * ssum(x) = s(x) * x$$

Two methods

- Structural Induction using **cover sets**
- **Inductionless Induction**

  Add "inductive theorem" as a rule and check if we can generate a **contradiction** (equality between different constructors such as $0 = 1$).

Associativity and Commutativity

Many useful functions are AC.

$$\boxed{\begin{array}{rcl} x+y & \rightarrow & y+x \\ (x+y)+z & \rightarrow & x+(y+z) \end{array}}$$

We cannot add the first as explicit rule (why?)
Also, we do want the rule $x * x \longrightarrow x$ to apply to $(p+(q+r))*(r+(q+p))$ if $+$ is AC.
Use **flattening** $(p+q+r)$ (messes-up orderings!) and **AC-matching**.

Conditional Rules

Many functions are not easy to write using **unconditional** rules. Consider $<$ over integers (constructors $0, s, p$).

$$
\begin{array}{rcll}
spx & \rightarrow & x & \\
psx & \rightarrow & x & \\
s(x) < s(y) & \rightarrow & x < y & \\
p(x) < p(y) & \rightarrow & x < y & \\
0 < 0 & \rightarrow & false & \\
0 < s(0) & \rightarrow & true & \\
0 < s(x) & \rightarrow & true & if \;\; 0 < x = true
\end{array}
$$

Not complete. But rest of rules are similar.

Note: Last rule above cannot be applied without doing a (recursive) **validity** proof!

Proving termination and confluence quite a challenge (my Ph.D. thesis was in this area).

Challenge: Do this without conditional rules.