LRR: A Laboratory for Equational Logic Programming

Rakesh M. Verma

Department of Computer Science
University of Houston
Houston, TX 77204

Outline

- Equational Logic Programming
- LRR
- Tabling mechanism of LRR
- Some optimization techniques
- Applications of LRR
- Future Work

Equations

5x + 3y = 7

$sin^2x + cos^2x = 1$

$x + y = y + x$

Equational Specification – Insertion into a Binary Search Tree

data Tree(α) == **nil** | **node**(Tree(α) # α # Tree(α))

1. ins(i, **nil**) = **node**(**nil**, i, **nil**)

2. ins(i, **node**(l,v,r)) =

   if(i $<$ v, **node**(ins(i, l), v, r),
   node(l, v, ins(i, r)))

3. if(true, x, y) = x

4. if(false, x, y) = y

Equational specification – Reversing a List

data list(α) == **nil** | **cons**(α # list(α))

1. rev(x) = apprev(x, **nil**)

2. apprev(**nil**, x) = x

3. apprev(**cons**(x,y), z) = apprev(y, **cons**(x,z))

Equational Reasoning

S – set and $*$ – binary operation on S

- $x * x = x$
- $(x * y) * z = (y * z) * x$

To show: $x * y = y * x$

Solution: $x * y = (x * y) * (x * y)$

$= [y * (x * y)] * x = [(x * y) * x] * y$

$= [(y * x) * x] * y = [(x * x) * y] * y$

$= [(y * y)] * (x * x) = y * x$

Equations as Programs

- Cleaner and simpler semantics.

- Programs are:

  - Modular.

  - Easy to understand, modify and maintain.

- Economy of expression and easier verification.

Disadvantage - Inefficient implementations

Implementations are interpreters.

Open : Compilers for Equations

Equations as Replacement Rules

$l \to r$ instead of $l = r$.

Find instances of lhs's and replace them by corresponding instance of rhs's.

Example: ins(2, node(nil, 1, nil))
$\to_2$ if(2 $<$ 1, node(ins(2, nil), 1, nil),
node(nil, 1, ins(2, nil)))
$\to_4$ node(nil, 1, ins(2, nil))
$\to_1$ node(nil, 1, node(nil, 2, nil))

Issue: Does loss of symmetry imply loss of power?

Interpreter for Equations

Pattern Matching and Graph Reduction

- Terms are represented by labeled ordered graphs.

- Given a graph, we try to **match** the function definitions at all its vertices.

- If a match is successful, the graph is transformed.

- Transformation of graph is called **graph reduction**.

Equations as Replacement Rules – Foundational Issues

Input: an expression E, Output: normal form of E

Normal form: an expression that does not contain any instance of an lhs

<u>Example</u> Normal form of rev(**cons**(a, **cons**(b, **nil**))) = **cons**(b, **cons**(a, **nil**))

- Existence of Normal Forms
- Uniqueness of Normal Forms
- Termination: No infinite sequences of replacements. $a \to f(a)$.
- Confluence

Knuth-Bendix Completion (KBC)

No loss of power! – Knuth & Bendix '70

Example: Corresponding to the Group axioms:

$\{x + (y + z) = (x + y) + z, x + 0 = x, x + -x = 0\}$

$$Group: \begin{array}{l} 0 + x \to x \\ -x + x \to 0 \\ (x + y) + z \to x + (y + z) \\ x + 0 \to x \\ x + -x \to 0 \\ -(-x) \to x \\ -0 \to 0 \\ -(x + y) \to -y + -x \\ x + (-x + y) \to y \\ -x + (x + y) \to y \end{array}$$

First-Order theorem Proving using KBC – Hsiang '85

Confluence Results

Confluence:

$$
\begin{tikzcd}
A \arrow[r, "*"] \arrow[d, "*"'] & A' \arrow[d, dashed, "*"] \\
B \arrow[r, dashed, "*"] & B'
\end{tikzcd}
$$

Left-linear, nonoverlapping systems - Rosen ’73, O’Donnell ’77, Huet ’80

Terminating, locally-confluent systems - Newman ’42, Huet ’80

Disjoint sum of confluent systems is confluent - Toyama ’87

Goal of Equational Computations

Given a confluent system, find the normal form of a given term efficiently.

Existing Approaches

Some methods for graph reduction are:

- Leftmost-outermost
- Parallel-outermost
- Leftmost-innermost
- Straight-line reduction

All these approaches are “forgetful.”

“Reinventing the Wheel”

Example 1:

Let $u \xrightarrow{n} s \xrightarrow{m} t$ and $v \xrightarrow{p} s$

Using leftmost-outermost

f(u, v) $\xrightarrow{n}$ f(s, v) $\xrightarrow{m}$ f(t, v)
$\xrightarrow{p}$ f(t, s) $\xrightarrow{m}$ f(t,t)

Example 2: Fibonacci program

Fib(0) = Fib(1) = 1

Fib(n) = Fib(n-1) + Fib(n-2)

Scenario

Given:

- An equational logic program P
- A term t

Goal: Compute normal form of t using LRR

LRR

- **TGR**: Term Graph Interpreter
- **Smaran**: Term Graph Interpreter that tables history (based on congruence closure)

LRR

- Programs consist of modules - modules can import other modules
- Terms in prefix notation

Built-in Data types

- Integers
- Floating-point numbers
- Booleans
- Characters
- Sets
- Multisets
- Strings

LRR : Features

- Compact list notation for arithmetic progressions
- Commands for interfacing

Smaran: Features

- Default: save each rewrite step
- Annotation of specific functions as "memo"
- Delete history (overriding):
  - every i steps
  - when available memory is low
- Variant detection
- AC operators

Some Optimization Techniques

- Encoding of all strings
- Minimal recursion
- Bottom-up algorithms for built-in data types
- Discrimination trees for matching
- Data structures for congruence-closure (**Smaran**)
- Dual representation of rules
- Memory allocation
- Dependency Lists (**Smaran**)
- Analysis of rules (TBI)

Congruence closure algorithm (CCA)

(Downey, Sethi & Tarjan; Nelson & Oppen; Kozen; Shostak)

Used to deduce consequences of **ground** equations.

<u>Example:</u> Given $\{f(f(a)) = a, f(f(f(a))) = a\}$

deduce $f^n(a) = a$ for all $n \geq 0$

Method

1. Construct numbered equivalence classes of terms. Terms are represented by signatures.

   Signature of $f(t_1, \ldots, t_m) = \ < f \ \#[t_1], \ldots, \ \#[t_m] >$.

2. Use equations to union equivalence classes that contain signatures representing equivalent terms.

3. Update signatures, potentially causing further unions
   ...

Example (contd.)

Initially

$\{1 : \{a\}, 2 : \{f\ 1\}, 3 : \{f\ 2\}, 4 : \{f\ 3\}\}$

After using $f(f(a)) = a$

$\{1 : \{a,\ f\ 2\}, 2 : \{f\ 1\}\}$

Applying $f(f(f(a))) = a$

$\{1 : \{a,\ f\ 1\}\}$

Solution

Observation: Although variables give rise to infinite set of ground rules, only finitely many are actually used.

Based on this observation, Chew ('80) proposed a tabling algorithm to compute normal forms.

Problem

Chew's algorithm works for a subclass of **left-linear** rewrite systems only.

Congruence-closure based Interpreter

(Chew '80 ; Verma '95)

- Identify a key signature called **unreduced signature**.

  When rule instance $A \to B$ is applied, signature of A is marked reduced.

- Each equivalence class has at most one unreduced signature

- Suffices to consider only unreduced signatures
  - To detect useful matching rules.
  - To detect normal forms.

Example: Given $h(b, a_1, b)$ and the rules

$a_1 \to a_2 \to a$

$h(x, a, x) \to g(x)$

$g(x) \to x$

Initially:

3: $\{h$ 2 1 2$\}$

2: $\{b\}$

1: $\{a_1\}$

After one iteration:

3: $\{h$ 2 1 2$\}$

2: $\{b\}$

1: $\{a_1, a_2(*)\}$

Finally: $\{$2: $\{h$ 2 1 2, g 2, $b(*)\}$, 1: $\{a_1, a_2, a(*)\}\}$

Group Example

# TN	# Red	T.Time	History	# Class	# Sig
59	23	0.30 (i)	19	62	63

TN - Number of Terms Normalized

Red - Total Number of Reductions

T.Time - Total Time

History - Number of Time History was Used

Class - Total Number of Classes, # Sig - Total Number of Signatures

Program	Time
REVEAL	0.7
SMARAN-REVEAL	0.5
TGR-REVEAL	0.3

Some Experiments with Smaran

Test Case	# Red	# Comp	Time	Reps	# Class	# Sig
fib(500.0)	1002	1998	0.0 (p) 0.22 (n)	4554	1007	2493
fib(1000.0)	20002	3998	0.0 (p) 0.79 (n)	2534	2007	4991
huff(500)	167322	83577	0.1 (p) 12.06 (n)	13874	86529	253849
huff(700)	327585	163675	0.15 (p) 31.71 (n)	10330	167806	495389
ins(600)	360001	180899	0.0 (p) 17.17 (n)	20966	180305	540305
ins(800)	640001	321199	0.0 (p) 34.74 (n)	18422	320405	960405
qsort(400)	402603	160401	0.0 (p) 18.49 (n)	21774	163205	563806
qsort(500)	628253	250501	0.01 (p) 29.17 (n)	21537	254005	879756
rev(100000)	300004	300001	0.02 (p) 15.44 (n)	19430	300009	600010
rev(200000)	600004	600001	0.01 (p) 39.12 (n)	15337	600009	1200010
sieve(4000,500)	325376	328875	0.0 (p) 16.47 (n)	19755	163496	488371
sieve(6000,800)	651314	656529	0.01 (p) 39.4 (n)	16530	326838	977367
tiso	200	138	0.0 (p) 0.05 (n)	4000	411	388

Red - Number of Reductions, # Comp - Number of Computations

Time - Time taken for Normalization, Reps - Reductions Per Second

Class - Total Number of Classes, # Sig - Total Number of Signatures

(p) - Parse Time, (n) - Normalize Time

Note: Initialization Time for all the above Test cases was Zero.

Experiments with Smaran (including delete option)

Test Case	# Red	# Comp	Time	Reps	# Class	# Sig
qsort(700)	1229554	491072	2.74 (i) 1.93 (p) 81.12 (n)	15157	140640	658708
sieve(10000,1200)	1576038	1584835	0.72 (i) 0.29 (p) 103.01 (n)	15299	82543	238213
rev(400000)	1200006	1200004	9.95 (i) 35.59 (p) 99.66 (n)	12041	910013	1140016

Red - Number of Reductions, # Comp - Number of Computations

Time - Time taken for Normalization, Reps - Reductions Per Second

Class - Total Number of Classes, # Sig - Total Number of Signatures

Rules for Sieve

$$from(x, y) \to ffrom(> (y, 0), x, y)$$

$$ffrom(false, x, y) \to nil$$

$$ffrom(true, x, y) \to : (x, from(+(x, 1), -(y, 1)))$$

$$filter(n, nil) \to nil$$

$$filter(n, : (x, l)) \to ffilt(= (\%(x, n), 0), n, x, l)$$

$$ffilt(true, n, x, l) \to filter(n, l)$$

$$ffilt(false, n, x, l) \to : (x, filter(n, l))$$

$$sieve(x, 0) \to nil$$

$$sieve(nil, y) \to nil$$

$$sieve(: (x, l), y) \to fsieve(> (y, 0), x, l, y)$$

$$fsieve(false, x, l, y) \to nil$$

$$fsieve(true, x, l, y) \to : (x, sieve(filter(x, l), -(y, 1)))$$

Rules for Quick sort

$sort(nil) \to nil$

$sort(:(x,y)) \to apnd(sort(smaller(x,y)), apnd(:(x,nil), sort(larger(x,y))))$

$smaller(x,nil) \to nil$

$smaller(x,:(y,z)) \to f(<(x,y),x,y,z)$

$f(true,x,y,z) \to smaller(x,z)$

$f(false,x,y,z) \to :(y, smaller(x,z))$

$larger(x,nil) \to nil$

$larger(x,:(y,z)) \to g(<(x,y),x,y,z)$

$g(true,x,y,z) \to :(y, larger(x,z))$

$g(false,x,y,z) \to larger(x,z)$

$apnd(nil,x) \to x$

$apnd(:(x,y),z) \to :(x, apnd(y,z))$

$list(x) \to h(>(x,0),x)$

$h(true,x) \to :(x, list(-(x,1)))$

$h(false,x) \to nil$

Rules for Huffman and Binary Tree Isomorphism

BuildHuff(:(Huff(weight1,tree1),nil)) = tree1

BuildHuff(:(x,:(y,remainder))) =

BuildHuff(Insert(remainder,Combine(x,y)))

Insert(nil,item) = :(item,nil)

Insert(:(Huff(weight2,tree2),remainder),Huff(weight1,tree1))
=

f(¡(weight1,weight2),weight1,tree1,weight2,tree2,remainder)

f(true,weight1,tree1,weight2,tree2,remainder) =

:(Huff(weight1,tree1),:(Huff(weight2,tree2),remainder))

f(false,weight1,tree1,weight2,tree2,remainder) =

:(Huff(weight2,tree2),Insert(remainder,Huff(weight1,tree1)))

Combine(Huff(weight1,tree1),Huff(weight2,tree2)) =

Huff(+(weight1,weight2),:(tree1,tree2))

dsc(nil) = 0

dsc(:(l,v,r))= +(1, +(dsc(l),dsc(r)))

at(v,v) = true

at(nil,:(l,w,r)) = false

```
at(:(l,w,r),nil) = false

at(:(l,v,r),:(l1,w,r1)) =
          ft(=(dsc(:(l,v,r)),dsc(:(l1,w,r1))),l,r,l1,r1)

ft(true,l,r,l1,r1) =
        or(and(at(l,l1),at(r,r1)),and(at(l,r1),at(r,l1)))

ft(false,l,r,l1,r1) = false
```

Some experiments with Reveal

Test Case	# DT	# Dup	# Var	N
assoc	30	75 (71.4 %)	13 (12.4 %)	105
furtin1	215	393 (64.6 %)	90 (14.8 %)	608
group	120	499 (80.6 %)	69 (11.1 %)	619
groupl	120	499 (80.6 %)	69 (11.1 %)	619
groupr	76	259 (77.3 %)	38 (11.3 %)	335
huntingc	819	3721 (81.9 %)	1729 (38.1 %)	4540
hunting2c	402	1613 (80.0 %)	191 (9.5 %)	2015
hunting2cn	348	1271 (78,5 %)	165 (10.2 %)	1619
ternary	1020	3374 (76.7 %)	271 (6.2 %)	4394

DT - Number of Distinct Terms

Dup - Number of Duplicates

Var - Number of Variants

N - Total Number of Terms

Note: Commutativity of $+$ not considered for hunting test cases.

How does Tabling Help?

- Can improve termination

- Can improve efficiency

Theorem Proving Application – Boolean Algebra

$\vee$ – boolean OR, $*$ – boolean AND, $\oplus$ – exclusive OR.

Note: $*$ and $\oplus$ are AC operators.

$$
\begin{aligned}
x \vee y &\rightarrow x * y \oplus x \oplus y && (1)\\
x \Rightarrow y &\rightarrow x * y \oplus x \oplus 1 && (2)\\
\neg(x) &\rightarrow x \oplus 1 && (3)\\
x \oplus 0 &\rightarrow x && (4)\\
x \oplus x &\rightarrow 0 && (5)\\
x * 1 &\rightarrow x && (6)\\
x * x &\rightarrow x && (7)\\
x * 0 &\rightarrow 0 && (8)\\
x * (y \oplus z) &\rightarrow x * y \oplus x * z && (9)
\end{aligned}
$$

Example: $\neg(\phi)$, where

$\phi = (p \vee q) * (\neg q \vee s) * (\neg s \vee p) * (\neg p \vee r) * (\neg r \vee \neg p \vee t) \Rightarrow (t * r)$

(Hsiang '85)

The End

Thank you!