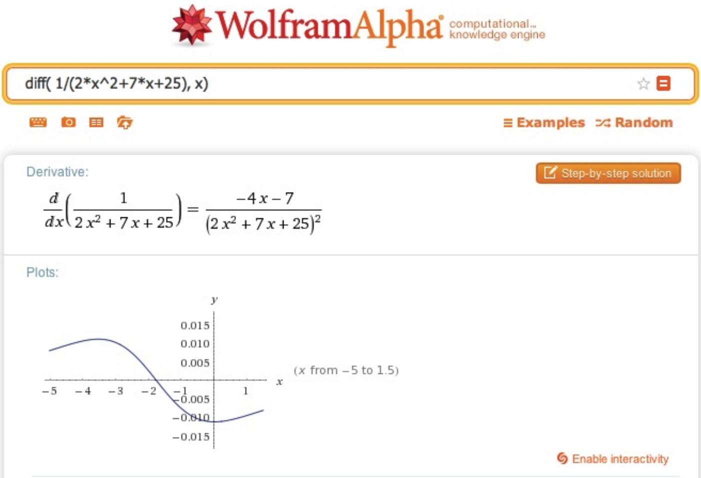
Introduction to Term Rewrite Systems and their Applications

Mark Boady

Drexel University

May 17, 2015

Motivation


WolframAlpha computational knowledge engine
diff(1/(2*x^2+7*x+25), x)
Examples
Random
Derivative:
Step-by-step solution
$\frac{d}{dx}\left(\frac{1}{2x^2+7x+25}\right) = \frac{-4x-7}{(2x^2+7x+25)^2}$
Plots:
y
0.015
0.010
0.005
−5 −4 −3 −2 −1 1
x
−0.005
−0.010
−0.015
(x from −5 to 1.5)
Enable interactivity

Overview

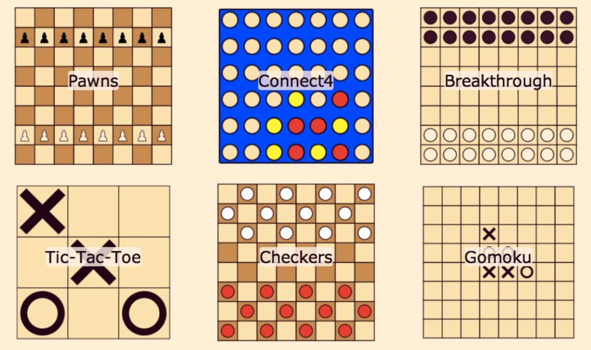
- What is a Term Rewrite System (TRS)?
- Tic-Tac-Toe
- The Maude System
- Cryptography
- Important Properties
    - Confluence
    - Termination
- Knuth-Bendix
- TRS are Turing Complete
- Example: Taking a Derivative
- Applications

What is a TRS?

- A TRS is a pair $T = (\Sigma, R)$
- The Signature, Σ, is a set of function symbols and their arity
    - Function Symbols have fixed arity
    - Arity means number of inputs
    - Constants are functions that take 0 inputs
- The Reduction Rules, R, is a collection of rules
    - $l \rightarrow r$
    - Match on pattern l and replace with pattern r
    - Patterns are made from $\Sigma \cup V$ where V is a set of variables

Board Games

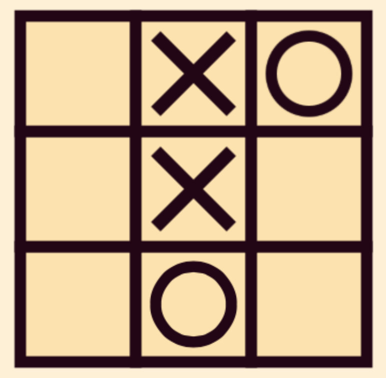
- Toss
  - http://toss.sourceforge.net
  - Model Games using TRS



Tic-Tac-Toe

- A Tic-Tac-Toe board has 9 spaces
- Each can be blank _ or have a symbol (X or O)
- The board is a term

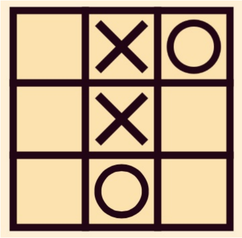
board(_,X,O,_,X,_,_,O,_)

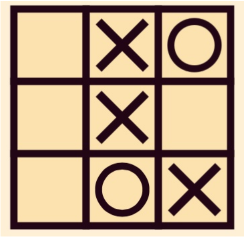


Tic-Tac-Toe

- Each move is a rewrite
- If multiple rules can match one pattern
  - Give probability to each rule
  - Select best move (guess if equal probabilities)

board(_,X,O,_,X,_,_,O,_) $\rightarrow$ board(_,X,O,_,X,_,_,O,X)

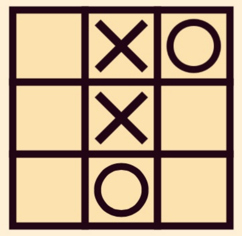


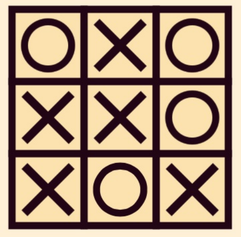


Tic-Tac-Toe

- The double-arrow $\twoheadrightarrow$ shows multiple rewrites (moves) have taken place
- The final board is in normal form
  - Normal Form: A term for which no rewrite rules match

board(_,X,O,_,X,_,_,O,_) $\twoheadrightarrow$ board(O,X,O,X,X,O,X,O,X)





Example: Addition

- A simple TRS that can add numbers
  - Positive Integers only
- Signature
  - 0 - constant Arity 0
  - S(_) - Arity 1 Successor Function
  - add(_,_) - Arity 2
- Rules
  - add$(0, a) \rightarrow a$
  - add(S(a),b)$\rightarrow$add(a,S(b))

Example: 3+2=5

- $R = \{p1 : \text{add}(0, a) \rightarrow a, p2 : \text{add(S(a),b)}\rightarrow\text{add(a,S(b))}\}$
- Reduction

$$\text{add}(S(S(S(0))), S(S(0))) \rightarrow_{p2} \text{add}(S(S(0)), S(S(S(0))))$$
$$add\,(S(S(0)), S(S(S(0)))) \rightarrow_{p2} add\,(S(0), S(S(S(S(0)))))$$
$$add\,(S(0), S(S(S(S(0))))) \rightarrow_{p2} add(0, S(S(S(S(S(0))))))$$
$$add(0, S(S(S(S(S(0)))))) \rightarrow_{p1} S(S(S(S(S(0)))))$$

- The TRS stops at $S(S(S(S(S(0)))))$
- Final term is in normal form

Maude

- Rewriting a more complex term will have many steps.
    - Multiply and Add!
    - mult(S(S(S(S(0)))),S(S(0)))
- We want to automate this process.
- The Maude System is a language for term rewriting.
- Freely Available: http://maude.cs.illinois.edu/w/index.php
    - Or google "Maude System"

Add/Mult in Maude

```
mod INTEGERS is
    sort Int .
    op 0 : -> Int .
    op S_ : Int -> Int .
    op add(_,_) : Int Int -> Int .
    op mult(_,_) : Int Int -> Int .
    vars a b  : Int .
    rl add(0,a) => a .
    rl add(S(a),b) => add(a,S(b)) .
    rl mult(0,a) => 0 .
    rl mult(S(a),b) => add(mult(a,b),b).
endm
```

- Saved as integers.fm

Add/Mult in Maude

```
Marks-MacBook-Air:maude27-osx markboady$ ./maude.* integers.fm
                    \||||||||||||||||||/
                  --- Welcome to Maude ---
                    /||||||||||||||||||\
            Maude 2.7 built: Mar  3 2014 18:07:27
            Copyright 1997-2014 SRI International
                   Wed May 13 17:41:19 2015
Maude> rewrite mult(S(S(S(S(0)))),S(S(0))) .
rewrite in INTEGERS : mult(S S S S 0,S S 0) .
rewrites: 21 in 0ms cpu (0ms real) (21000000 rewrites/second)
result Int: S S S S S S S S 0
Maude> 
```

Add/Mult in Maude

```
Maude> debug rewrite mult(S(S(S(S(0)))),S(S(0))) .
rewrite in INTEGERS : mult(S S S S 0,S S 0) .
Debug(1)> step .
************ rule
rl mult(S a,b) => add(mult(a,b),b) .
a --> S S S 0
b --> S S 0
mult(S S S S 0,S S 0)
--->
add(mult(S S S 0,S S 0),S S 0)
Debug(1)> step .
************ rule
rl mult(S a,b) => add(mult(a,b),b) .
a --> S S 0
b --> S S 0
mult(S S S 0,S S 0)
--->
add(mult(S S 0,S S 0),S S 0)
```

Cryptography

- NQ Vault is a popular encryption app for Andriod and iOS
- Video and Image files were encrypted by
  - Static 8-bit key is selected for all files
  - XOR first 128 bytes of file with key
- This is trivial to decrypt
  - There are only 255 possible keys to try
- It is important to prove how well your ecryption method works

Cryptography

- Reachability Analysis
  - Given two terms, is it possible to get from one to the other
- Timbuk
  - http://www.irisa.fr/celtique/genet/timbuk/
- Lande Project
  - Proving properties of cryptography systems
  - Can a potential intruder get secret information?
  - http://www.irisa.fr/celtique/genet/crypto.html
- RAVAJ
  - Security testing for Java bytecode
  - http://www.irisa.fr/lande/genet/RAVAJ

Reachability

- An encryption method is defined by an equational system
- Is there a way to use the equations to get some one term to another?
  - path between $a(b+c)$ and $ab+ac$
- Universal Word Problem
  - Given two terms s, t and a set of equations E can we make $s = t$?
- Knuth-Bendix Algorithm

Knuth-Bendix

- Possible Solution:

  1. Make E into a TRS
  2. rewrite $s \twoheadrightarrow s'$ to normal form
  3. rewrite $t \twoheadrightarrow t'$ to normal form
  4. If s' and t' are exactly the same then $s = t$

Knuth-Bendix

- XOR:

$$A \oplus 0 = A$$
$$A \oplus A = 0$$
$$(A \oplus B) \oplus C = A \oplus (B \oplus C)$$

- If an attacker has the encrypted message $E = M \oplus K$ can they recover M
  - If E=M under equational rules
- In this case, as long as the attacker can guess K

Knuth-Bendix

- A TRS with two properties can answer this question
- Confluence
  - If mutiple rules match a term, which is picked does not change outcome
  - One input would have 2 or more possible outputs without this
- Termination
  - For any input term, the TRS will terminate at a normal form
- If both these properties hold, then
  - $a \twoheadrightarrow_E a'$
  - $b \twoheadrightarrow_E b'$
  - if $a' \equiv b'$ then $a = b$ under equational system E

Knuth-Bendix

- Knuth-Bendix Completion is an algorithm to answer the Universal World Problem
- Inputs: Σ and E where E is an equational System and sorting
- Outputs:
  - $T = (\Sigma, R)$ where T is confluent and terminating
  - or Failure if termination is impossible
  - or Loops infinitely

Algorithm

- We start with a set of equations

$$
\begin{aligned}
A \oplus A &= 0 \\
A \oplus 0 &= A \\
(A \oplus B) \oplus C &= A \oplus (B \oplus C)
\end{aligned}
$$

- Select one equation from the set $(A \oplus A = 0)$
- Decide which direction to place arrow
  - $A \oplus A \to 0$
  - $0 \to A \oplus A$

Termination

- We want to place the arrow so that TRS always terminates
- Introduce a sorting on terms, with minimal element
- if $l \rightarrow r$ means $l > r$ in the sorting, then it will terminate
- If every reduction moves the term closer to the minimal element, then it must terminate
- We will pick
  - $A \oplus A \rightarrow 0$
- A constant will be the minimal element

Confluence

- We also need a confluent system so we can compare the results
- Assume the second rule we pick is
  - $(A \oplus B) \oplus C \rightarrow A \oplus (B \oplus C)$
- This overlaps with $A \oplus A \rightarrow 0$ to make
  - $(A \oplus A) \oplus C$
- What happens if we try to rewrite this?

Confluence

- Path 1

$$(A \oplus A) \oplus C \rightarrow 0 \oplus C$$

- Path 2

$$(A \oplus A) \oplus C \rightarrow A \oplus (A \oplus C)$$

- These aren't equivalent, so we need to add an equation

$$0 \oplus C = A \oplus (A \oplus C)$$

- Through repeated applications of this method, the system will learn

$$A \oplus (A \oplus C) \rightarrow C$$

Knuth-Bendix

- Knuth Bendix Algorithm Overview
  - Inputs: Equations E, Signature Σ, sorting
- Steps:
  1. Pick an Equation $a = b$ from E
     1. if $a \equiv b$ discard
     2. otherwise orient using sorting to $l \to r$
     3. Fail if can't be ordered
  2. Add any pattern overlaps back into E as equations
  3. Repeat until $E = \emptyset$
- If this algorithm succeeds, then it generates a TRS that is confluent and terminating.

Turing Complete

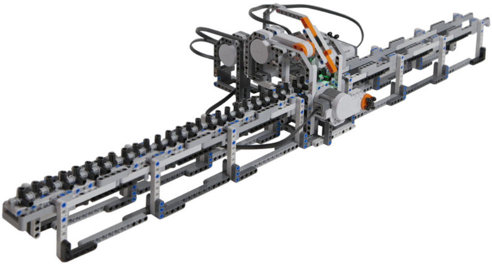
- Turing machines are simple machines that can simulate any real-world computer
- A system is Turing complete if it can simulate a Turing Machine
- C++, Java, and Haskell are all Turing Equivalent
  - Any program written for one of these languages can also be written in any other
- In Short: A Turing complete system can do anything you expect from a real-world computer

Turing Machine

- Turing machines are simple machines that can simulate any real-world computer
- A Turing Machine has:
  - A tape of infinite length
  - A set of characters Σ that can be written/read from the tape
  - A set of states Q the machine can be in
  - An input value written on the tape

Lego Turing Machine

- http://www.legoturingmachine.org



Turing as TRS

- We want to simulate a Turing machine as a TRS
- Each tape symbol is a function of one input.
- Special functions L and R for infinite blank space
- Each state is a 1 input function
- Example:
    - if a tape looks like $\cdots 0110 \cdots$ and is it state q_0 reading first 1
    - term looks like $L(0(q_0(1(1(0(R))))))$

Turing as TRS

- Each Transition is a reduction rule
- Example:
  - In state q_2 if you read a 1 write 0 and move right and go to q_3
  - $A(q_2(1(B))) \rightarrow A(0(q_3(B)))$
- Special rules for Spaces
  - $q_1(R) \rightarrow q_1(\_(R))$
- We can simulate any Turing Machine as a TRS
- TRS are Turing Complete

Example: Taking a Derivative

- Simplification: Assume only differential variable is x
- $\Sigma = \{\frac{d}{dx}, (\_)^{\_}, \_\,\_, \_ + \_, x, \cdots\}$
- $V = \{C :: integer, A, B\}$
- Derivative Rules:

$$\begin{aligned}
\frac{d}{dx}C &\to 0 \\
\frac{d}{dx}x &\to 1 \\
\frac{d}{dx}(A)^{B} &\to B\frac{dA}{dx}(A)^{B-1} \\
\frac{d}{dx}(A+B) &\to \frac{dA}{dx} + \frac{dB}{dx} \\
\frac{d}{dx}(AB) &\to B\frac{dA}{dx} + A\frac{dB}{dx}
\end{aligned}$$

Example: Taking a Derivative

$$
\begin{aligned}
\frac{d(2x^2+7x+25)^{-1}}{dx} &\to -1\left(\frac{d}{dx}(2x^2+7x+25)\right)(2x^2+7x+25)^{-2} \\
&\twoheadrightarrow \frac{-\frac{d}{dx}(2x^2) - \frac{d}{dx}(7x) - \frac{d}{dx}(25)}{(2x^2+7x+25)^2} \\
&\twoheadrightarrow \frac{-2\frac{d}{dx}x^2 - x^2\frac{d}{dx}2 - x\frac{d}{dx}7 - 7\frac{d}{dx}x - \frac{d}{dx}25}{(2x^2+7x+25)^2} \\
&\twoheadrightarrow \frac{-2\frac{d}{dx}x^2 - 7\frac{d}{dx}x}{(2x^2+7x+25)^2} \\
&\twoheadrightarrow \frac{-4x\frac{d}{dx}x - 7\frac{d}{dx}x}{(2x^2+7x+25)^2} \\
&\twoheadrightarrow \frac{-4x-7}{(2x^2+7x+25)^2}
\end{aligned}
$$

Application: Symbolic Computation

- Mathematic/Wolfram Alpha
  - http://www.wolfram.com/mathematica/
- Maple Computer Algebra System
  - http://www.maplesoft.com
- Both Maple and Mathematic allow you to create your own TRS
- Matlab
  - http://www.mathworks.com
- SymPy - Symbolic Computation Library for Python
  - http://www.sympy.org

```
>>> from sympy import *
>>> x, y, z = symbols('x,y,z')
>>> ((x + y)*(x - y)).expand(basic=True)
x**2 - y**2
>>> ((x + y + z)**2).expand(basic=True)
x**2 + 2*x*y + 2*x*z + y**2 + 2*y*z + z**2
```

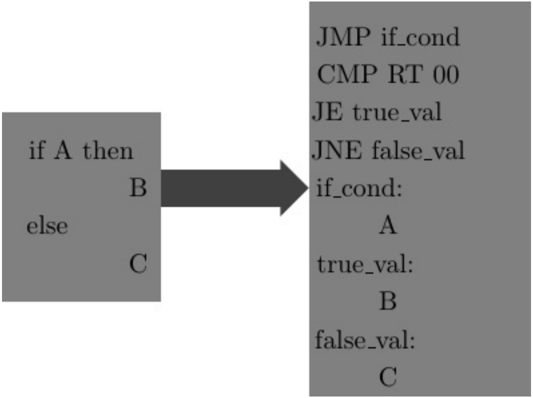
Programming Languages

- The Maude System allows for the creation of TRS
  - Even allows for object oriented systems
- PURE programming language based on TRS
  - http://purelang.bitbucket.org
  - Dynamically Typed

```
> f + g = \x -> f x + g x
    if nargs f > 0 && nargs g > 0;
> f - g = \x -> f x - g x
    if nargs f > 0 && nargs g > 0;
> f x = 2*x+1; g x = x*x; h x = 3;
> map (f+g-h) (1..10);
[1,6,13,22,33,46,61,78,97,118]
```

Compiling

- We can think of the translation before a programming language and it's compiled code as a series of rewrites



KITTeL Termination Analysis

- Available from: https://github.com/s-falke/kittel-koat
- Termination Analysis of C Programs Using Compiler Intermediate Languages. RTA 2011
- Termination Analysis of Imperative Programs Using Bitvector Arithmetic. VSTTE 2012
- Alternating Runtime and Size Complexity Analysis of Integer Programs. TACAS 2014

```
int power(int x, int y) {
    int r = 1;
    while (y > 0) {
        r = r * x;
        y = y - 1;
    }
    return r;
}
```

$$\mathsf{state}_{\mathsf{start}}(v_x, v_y, v_{y.0}, v_{r.0}) \to \mathsf{state}_{\mathsf{entry}_{\mathsf{in}}}(v_x, v_y, v_{y.0}, v_{r.0})$$

$$\mathsf{state}_{\mathsf{entry}_{\mathsf{in}}}(v_x, v_y, v_{y.0}, v_{r.0}) \to \mathsf{state}_{\mathsf{bb1}_{\mathsf{in}}}(v_x, v_y, v_y, 1)$$

$$\mathsf{state}_{\mathsf{bb1}_{\mathsf{in}}}(v_x, v_y, v_{y.0}, v_{r.0}) \to \mathsf{state}_{\mathsf{bb}_{\mathsf{in}}}(v_x, v_y, v_{y.0}, v_{r.0})\ [\![v_{y.0} > 0]\!]$$

$$\mathsf{state}_{\mathsf{bb1}_{\mathsf{in}}}(v_x, v_y, v_{y.0}, v_{r.0}) \to \mathsf{state}_{\mathsf{return}_{\mathsf{in}}}(v_x, v_y, v_{y.0}, v_{r.0})\ [\![v_{y.0} \leq 0]\!]$$

$$\mathsf{state}_{\mathsf{bb}_{\mathsf{in}}}(v_x, v_y, v_{y.0}, v_{r.0}) \to \mathsf{state}_{\mathsf{bb1}_{\mathsf{in}}}(v_x, v_y, v_{y.0} - 1, v_{r.0} * v_x)$$

$$\mathsf{state}_{\mathsf{return}_{\mathsf{in}}}(v_x, v_y, v_{y.0}, v_{r.0}) \to \mathsf{state}_{\mathsf{stop}}(v_x, v_y, v_{y.0}, v_{r.0})$$

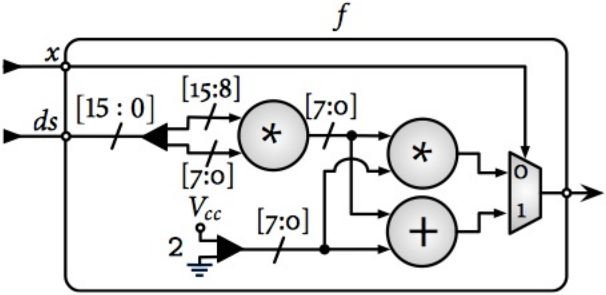
CλaSH Circuit Design

- Available from: http://www.clash-lang.org
- Generates VHDL (Hardware Description) from Haskell Functional Programming
- Using Rewriting to Synthesize Functional Languages to Digital Circuits. Trends in Functional Programming (TFP) May 2013
- Digital Circuits in CλaSH: Functional Specifications and Type-Directed Synthesis. PhD thesis, University of Twente, Enschede, The Netherlands, January 2015.
- N Queens on an FPGA: Mathematics, Programming, or Both?. In: Communicating Processes Architectures 2014

Haskell

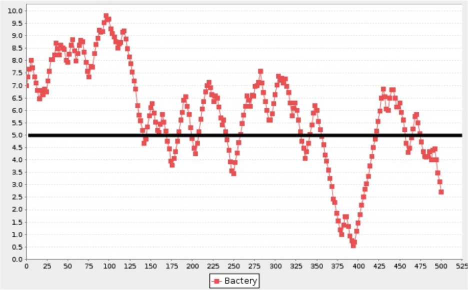
```
data Bool = False | True

f :: Bool → (Int8, Int8) → Int8
f x (a,b) = if x then y + 2 else y * 2
  where
    y = a * b
```



Biological Modeling

- Stochastic Multilevel Multiset Rewriting
  - Proceedings of the 9th International Conference on Computational Methods in Systems Biology (CMSB '11)
  - Mathematical Structures in Computer Science. 2013
- Model of Bacterium searching for food source
- Optimal Food source along line at 5
- Bacterium can spin or move forward



Formal Proofs

- The ACL2 Sedan Theorem Prover
  - http://acl2s.ccs.neu.edu/acl2s/doc/
- Example from http://www.ccs.neu.edu/home/riccardo/courses/csu290-sp09/lect22-acl2.pdf
- Uses Simplification and Induction to prove theories about code
- Simplification done using rewriting

Proof by Induction

```
ACL2 > (defun rev (x)
    (if (endp x)
        NIL
        (app (rev (cdr x)) (list (car x)))))
...
ACL2 > (defthm true-listp-rev
    (true-listp (rev x)))
...
But simplification reduces this to T, us-
ing the :definition REV and the :executable-
counterpart of TRUE-LISTP.
That completes the proof of *1.
Q.E.D.
```

Conclusions

- Term Rewriting Systems provide a very simple model of computation
- A TRS is composed of
  - Signature: how terms can be written
  - Rewrite Rules: how terms can be transformed
- Important Properties
  - Confuence
  - Termination
- Knuth-Bendix - Makes a TRS from an Equational System
- TRS are Turing Complete
- This model has a wide variety of applications

Thank You

Thank You.

Questions?