Rewrite Systems

Autumn 2004

Famous Equations

$$a^2+b^2=c^2$$
$$F = ma$$
$$e^{i\pi}+1 = 0$$
$$\nabla \times E = -\partial B/\partial t$$
$$E = mc^2$$

Subject

- Equations
  - Reasoning with equations
    - solving equations
    - proving identities
  - Computing with equations
    - rewriting by pattern matching
    - goal solving by unification

Beautiful Results

- Knuth's *Critical Pair Lemma*
- Huet's "diamond proof" of *Newman's Lemma*
- Nash-William's proof of *Kruskal's Tree Theorem*

Tentative Course Outline

1. Introduction
2. Termination
3. Church-Rosser
4. Orthogonality
5. Diagrams
6. Completion
7. Saturation
8. Modularity
9. Unification
10. Induction
11. Polynomials
12. Boolean Rings
13. Extensions
14. Open Problems

Today

- Mechanics
- Introduction & Examples
- Programming
- Concepts
- Undecidability

Mechanics

- Prerequisites
- Website
- Textbook
- Homework
- Exam

Prerequisites

- A little algebra
- A little logic
- A little combinatorics
- A little computability

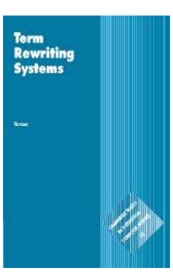
Website

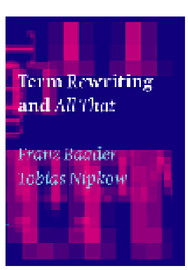
- ~nachumd/rewrite
  - registration (email address)
  - outline
  - notes (~nachumd/papers/hand-final.pdf)
  - information (links)

Information

- Course Notes
- Surveys
- Systems
- Open Problems
- Papers
- Other Links
- Other Courses
  - 1997 Mini-Course
  - 2002 Course
  - Miscellaneous

Books





Today

- Chap. 0 of Terese
- Link on my page to
  - http://assets.cambridge.org/052139/1156/sample/0521391156WS.pdf
- Beginning of Chap. 5

Introduction

- History
- Applications
- Examples
- Definitions

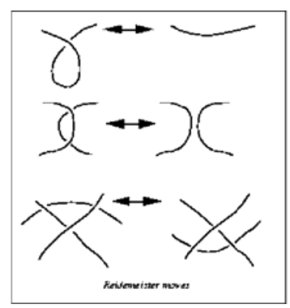
History

1914	Thue	string rewrite rules
1951	Evans	word problem in abstract algebra
1960s	Brainerd etc.	tree automata
1967	Gorn	rewriting systems
1970	Knuth&Bendix	***Simple word problems in universal algebra***
1973	Rosen	orthogonal systems
1970s	ADJ	algebraic specifications

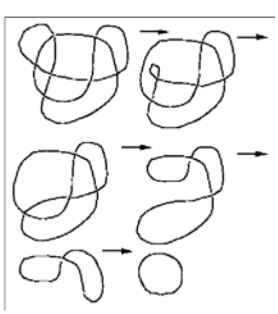
Applications

- Symbolic Computation
- Functional Programming Languages
  - Miranda, Haskell, ML, Curry, Refine, Obj
- Semantics of Programming Languages
- Automated Deduction
  - Robbins Algebra
- Verification
  - Modelling Verilog
- Hardware Synthesis
  - Bluespec @ MIT

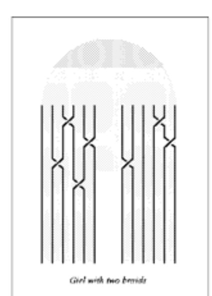
Knot Equivalences



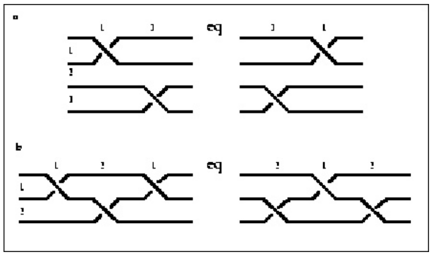
Knot Moves



Braids



Braid Equivalences



Abstract Rewriting

An *abstract rewriting system* is composed of

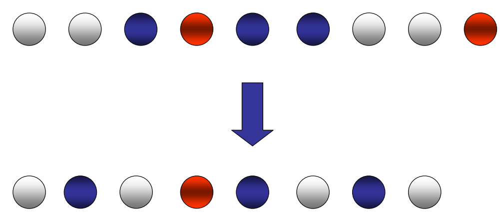
- elements T
- binary relation $\rightarrow \subseteq T \times T$
  - there may be several relations $\rightarrow$, $\rightarrow$, ...
  - **labelled transition system**

String Rewriting

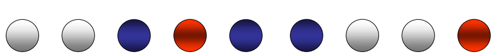
A *string rewriting system* is composed of

- alphabet Σ
  - defines set Σ^* of words
- rules $R \subseteq \Sigma^* \times \Sigma^*$
  - define rewrite relation $\rightarrow$

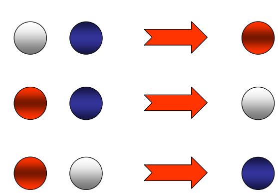
Transitions



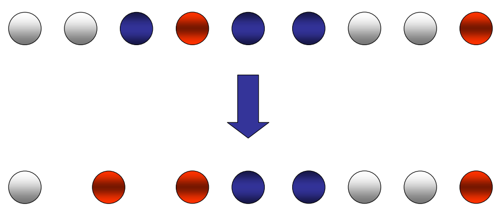
Marble State



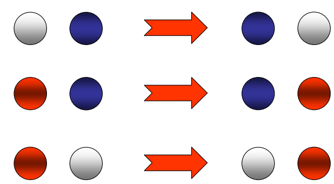
Simple Rules



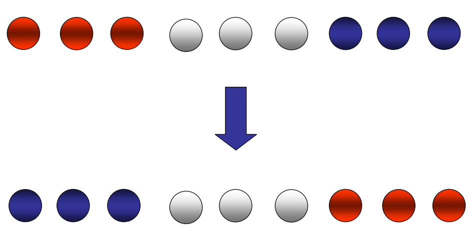
Marble Move



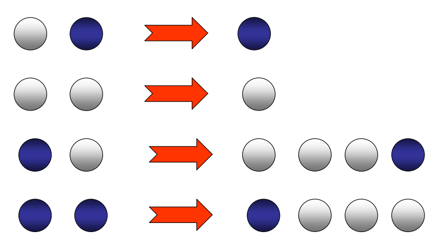
Flag Problem



Dutch National Flag



Non-trivial Problem



Markov System

$$
\begin{array}{rcl}
a+a^{-1} & \rightarrow & \epsilon \\
b+b^{-1} & \rightarrow & \epsilon \\
c+c^{-1} & \rightarrow & \epsilon \\
b+a & \rightarrow & +ab \\
c+a & \rightarrow & +ac \\
a+b & \rightarrow & +ba \\
-ba & \rightarrow & -ab \\
a+c & \rightarrow & +ca \\
-ca & \rightarrow & -ac \\
+ & \rightarrow & - \\
- & \rightarrow & \epsilon
\end{array}
$$

Hydra vs. Hercules

Terms

- Signature Σ=(S,#,X)
  - S set of symbols
  - arity function #:$S \to N$
  - variables X
- Defines set of first-order terms (with variables)

Term Rewriting

A *term rewriting system* is composed of

- signature Σ defining terms T
- rules $R \subseteq T \times T$
  - define rewrite relation $\to$

Example

- Σ = ({+,s,0},#,{x,y,...})
  #(+)=2, #(s)=1, #(0)=0
  - +/2, s/1, 0/0
- R={ +(s(x),y)$\to$s(+(x,y)),
  +(0,x)$\to$x }

Pattern Matching

- Left side of rules are applied if they match a subterm
- If match, replace with corresponding right side

Basics

- substitution: $f(t_1,\ldots,t_n)^\sigma = f(t_1^\sigma,\ldots,t_n^\sigma)$ homomorphism on the term algebra
- context C[_] is a term with a hole ❑
  $C[t]=C[\_]^\sigma$ where $❑^\sigma$=t, x^σ=x

Semantics

- abstract reduction system (T,$\to_R$) where $\to_R$ is the smallest rewrite relation containing R
- replaces "equals for equals"
- a relation S on terms is a rewrite relation iff
  - t S u implies t^σ S u^σ for any substitution σ
  - t S u implies C[t] S C[u] for any context C[_]

Normal Form

- Element to which no rule applies
- Questions
  - Existence
  - Uniqueness

Symbolic Computation

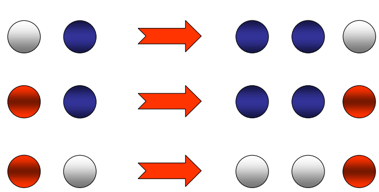
Disjunctive Normal Form

$$\neg\neg x \to x$$
$$\neg(x \wedge y) \to (\neg x) \vee (\neg y)$$
$$\neg(x \vee y) \to (\neg x) \wedge (\neg y)$$
$$x \wedge (y \vee z) \to (x \wedge y) \vee (x \wedge z)$$
$$(y \vee z) \wedge x \to (y \wedge x) \vee (z \wedge x)$$

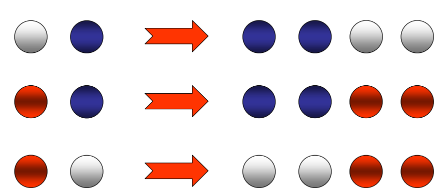
Termination

- No endless sequence of rewrites

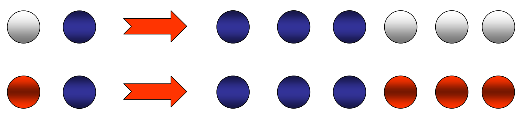
Growth Problem



Duplication Problem



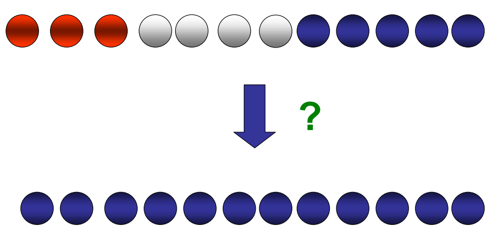
Toyama's Problem



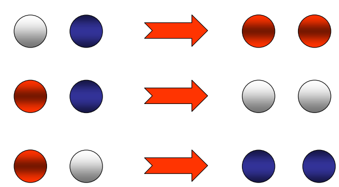
Confluence

- Order of rewrites doesn't matter

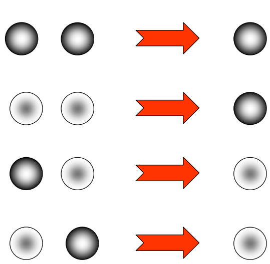
Chameleon Isle



Chamelion Problem



Urn System



Programming

Quotient Equations

minus(x,0) = x
minus(s(x),s(y)) = minus(x,y)

quot(0,s(y)) = 0
quot(s(x),s(y)) = s(quot(minus(x,y),s(y)))

Quotient Program

minus(x,0) → x
minus(s(x),s(y)) → minus(x,y)

quot(0,s(y)) → 0
quot(s(x),s(y)) → s(quot(minus(x,y),s(y)))

Append & Reverse

□@z → z
(x:y)@z → x:(y@z)

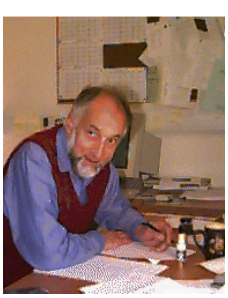
$□^r$ → □
$(x:y)^r$ → y^r@(x:□)

Haskell

```
reverse :: [[a]] -> [a]

reverse (x:xs) = (reverse xs) ++ [x]
reverse [] = []
```


ML



- Robin Milner
- Logic for Computable Functions
  Stanford & Edinburgh 1972-1995
- Meta-Language
  - Theorem proving
  - Type system
  - Higher-order functions

Pattern Matching

```
fun length nil  = 0
  | length (x::s) = 1 + length(s);
```

List Functions

- Reverse a list

```
fun reverse nil = nil
  | reverse (x::xs) =
        append ((reverse xs), [x]);
```

- Append lists

```
fun append(nil, ys) = ys
  | append(x::xs, ys) = x :: append(xs, ys);
```


Value Declarations

- General form

  val <pat> = <exp>

- Examples

```
val myTuple = ("Conrad", "Lorenz");
val (x,y) = myTuple;
val myList = [1, 2, 3, 4];
val x::rest = myList;
```


Types in ML

$f : A \to B$ means
for every $x \in A$,

$$f(x) = \begin{cases} \text{some element } y=f(x) \in B \\ \text{run forever} \\ \text{terminate with an exception} \end{cases}$$

In words, "if $f(x)$ terminates normally, then $f(x) \in B$."

Compound Types

- Tuples
  - (4, 5, "noxious") : int * int * string
- Lists
  - nil
  - 1 :: [2, 3, 4]   infix cons notation
- Records
  - {name = "Fido", hungry=true}
    : {name : string, hungry : bool}

Higher-Order

- Apply function to every element of list

```
fun map (f, nil) = nil
|   map (f, x::xs) = f(x) :: map (f,xs);
```

map (fn x => x+1, [1,2,3]); $\Rightarrow$ [2,3,4]

Higher-Order Functions

- Tactic is a function
- Method for combining tactics is a function on functions
- Example:

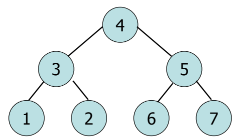
  $f(tactic_1, tactic_2) =$
  λ formula. try $tactic_1$(formula)
  else $tactic_2$ (formula)

Datatypes

- Recursively defined data structure

  datatype tree = leaf of int | node of int*tree*tree

node(4, node(3,leaf(1), leaf(2)),
node(5,leaf(6), leaf(7))
)



- Recursive function

  fun sum (leaf n) = n
  | sum (node(n,t1,t2)) = n + sum(t1) + sum(t2)

Automated Deduction

Equational Reasoning

- Reflexivity

  x=x
- Commutativity

  x=y => y=x
- Transitivity

  x=y & y=z => x=z
- Functional Reflexivity

  x=x' & y=y' => f(x,y)=f(x',y')

Robbins Algebra

$$\neg(\neg(x \vee y) \vee \neg(x \vee \neg y)) = x$$

Robbins Algebras are Boolean

- 60-year old conjecture
- 20 years of computer attempts
- Solved by McCune in 1996
- 8 days on Unix workstation
- 50,000 equations inferred
- 2,500,000 attempted rewrites
- 12-step proof (of main lemma)

Word Problems

- Given an equational theory E
- Does an equation g=d follow?
- Does an identity s=t follow?

Undecidable Problem

ah = ha
oh = ho
at = ta
ot = to
tai = it
hoi = ih
that = itht

Undecidable Problem

abaabb = bbaaba
aababba = bbaaaba
abaaabb = abbabaa
bbbaabbaaba = bbbaabbaaaa
aaaabbaaba = bbaaaa

Turing Machines

- **Deterministic or nondeterministic, TM**
- **One-way infinite tape**
- **Represent instantaneous description (with machine state at position of read head) of TM tape as a word with $ at right end**

Turing Machine System

TM transition	**SRS rule**
$q,a \mapsto p,c,R$	**$qa \to cp$**
$q,B \mapsto p,c,R$	**$q\# \to cp\#$**
$q,a \mapsto p,c,L$	**$xqa \to pxc$**
	(every tape symbol x)

Homework #1

- Sorting program
  - Numbers: 0, (s 0), (s (s 0)),...
- Run my interpreter: hw1.scm
- Input: "your-sort-program.scm"
- Check that output is sorted
  - (0 (s 0) (s (s 0)) (s (s (s 0))))