School of informatics

Automated Reasoning

Rewrite Rules

Jacques Fleuriot

Term Rewriting

- Rewriting is a technique for replacing terms in an expression with equivalent terms
  - useful for simplification, e.g.
    - given “$x*0=0$”, we can rewrite “x+(x∗0)” to “x+0”
    - and if “$x+0=x$”, we can rewrite further to just “x”
  - uses “one-way” unification i.e. matching
- We use the notation $L \Rightarrow R$ to define a rewrite rule that replaces the term L with the term R in an expression (and not vice versa).

The Power of Rewrites

Given this set of rewrite rules:

$$0 + n \Rightarrow n \tag{1}$$
$$(0 \leq m) \Rightarrow \mathrm{True} \tag{2}$$
$$s(m) + n \Rightarrow s(m + n) \tag{3}$$
$$s(m) \leq s(n) \Rightarrow m \leq n \tag{4}$$

This statement is easily proved:

$$0 + s(0) \leq s(0) + x$$

by (1), $s(0) \leq s(0) + x$
by (3), $s(0) \leq s(0 + x)$
by (4), $0 \leq 0 + x$
by (2), True

Peano Arithmetic

The rewrites in our previous slide are part of a common foundation for the natural numbers, called Peano Arithmetic.

s is the successor function, so $\mathsf{1}$ is defined as $\mathsf{s(0)}$.

For addition and multiplication, we often have these rewrites:

$$
\begin{array}{llll}
0+x & \Rightarrow & x & (1) \\
s(x)+y & \Rightarrow & s(x+y) & (2) \\
0*x & \Rightarrow & 0 & (3) \\
s(x)*y & \Rightarrow & x*y+y & (4)
\end{array}
$$

Example:

$$
\begin{array}{lll l}
\mathsf{s(s(0))*s(0)} & = & \mathsf{s(0)*s(0)+s(0)} & \text{by (4), } [\mathsf{s(0)}/x, \mathsf{s(0)}/y] \\
 & = & \mathsf{0*s(0)+s(0)+s(0)} & \textit{by (4), } [\mathsf{0}/x, \mathsf{s(0)}/y] \\
 & = & \vdots & \\
 & = & \mathsf{s(s(0))} &
\end{array}
$$

Exercise: fill in the missing steps

In this example, the final expression is ground (contains only constants). Rewriting is useful even if this is not the case. This is called symbolic evaluation: $s(0)+s(a) \Rightarrow \ldots \Rightarrow s(s(a))$

Rewrite Rule of Inference

$$\frac{P\{t\} \qquad L \Rightarrow R \qquad L\varphi \equiv t}{P\{\mathrm{R}\varphi\}}$$

We use the notation $P\{t\}$ to mean that the expression P contains a subexpression t.

Note: rewrite rule of inference uses matching *not* unification

Example:

Given an expression	$(\mathrm{s}(A)+\mathrm{s}(0))+\mathrm{s}(B)$
and a rewrite rule	$\mathrm{s}(x)+y \Rightarrow \mathrm{s}(x+y)$
we can find	$t = \mathrm{s}(A)+\mathrm{s}(0)$
and	$\varphi = [A/x,\ \mathrm{s}(0)/y]$
Rewriting gives us	$\mathrm{s}(A+\mathrm{s}(0))+\mathrm{s}(B)$

Some Restrictions

A rewrite rule $\alpha \Rightarrow \beta$ should satisfy the following restrictions:

- α is not a variable
  - e.g. $x \Rightarrow x+1$ — if the LHS can match anything, it's very hard to control!

- $vars(\beta) \subseteq vars(\alpha)$
  - e.g. $0 \Rightarrow 0 * x$ — if we start with a ground term, we should always have a ground term

Algebraic Simplification

1. $x*0 \Rightarrow 0$
2. $1*x \Rightarrow x$
3. $x^0 \Rightarrow 1$
4. $x+0 \Rightarrow x$

Example:

$$
\begin{aligned}
& a^{\underline{2*0}} * 5 + b*0 & \\
= \; & \underline{a^0} * 5 + b*0 & \textit{by (1)} \\
= \; & \underline{1*5} + b*0 & \textit{by (3)} \\
= \; & 5 + \underline{b*0} & \textit{by (2)} \\
= \; & \underline{5 + 0} & \textit{by (1)} \\
= \; & 5 & \textit{by (4)}
\end{aligned}
$$

•Terminology: Any subexpression that can be rewritten (i.e. matches the LHS of a rewrite rule) is called a redex.
(This is short for reducible expression.)

•There is sometimes a choice:
- which subexpression to rewrite
- which rule to use

Partial Rewrite Search Tree

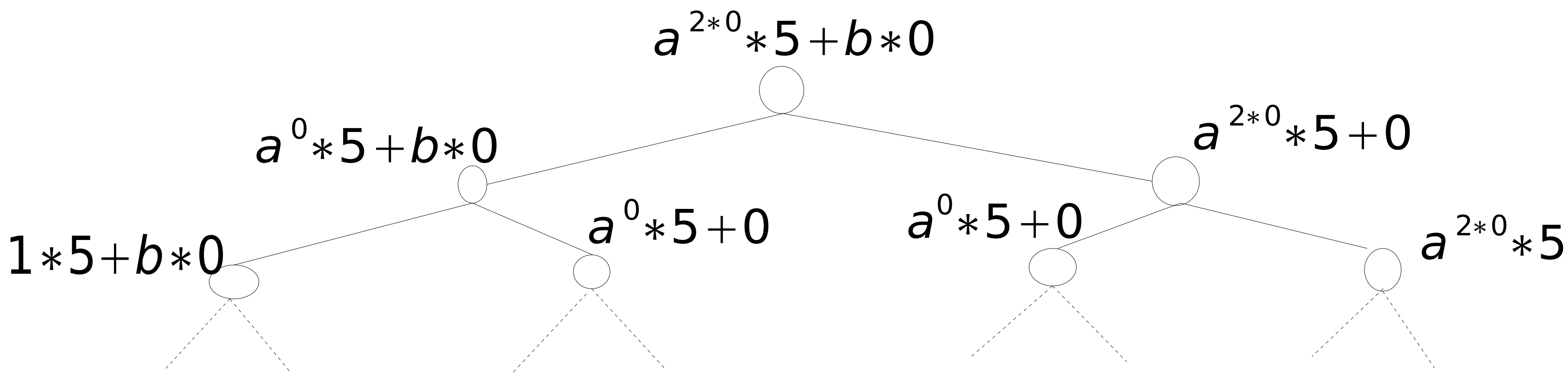
Common strategies:

- innermost (inside-out) leftmost redex (1st redex in post-order traversal)

  e.g. apply $0 * x \Rightarrow 0$ to $(\underline{0 * s(0)} + s(0)) + s(0 * 0)$

- outermost (outside-in) leftmost redex (1st redex in pre-order traversal)

  e.g. apply $x + s(y) \Rightarrow s(x + y)$ to $\underline{(0 \cdot s(0) + s(0)) + s(0)}$



Important Questions:

- Is the tree finite (does the rewriting process always end) ?
- Does it matter in which order rewrites are applied
  (or are all the leaf nodes the same) ?

Logical Interpretation

- A rewrite rule $L \Rightarrow R$ on its own is just a “replace” instruction; to be useful, it must have some logical meaning attached!
- Most commonly, a rewrite $L \Rightarrow R$ is permitted only if $L{=}R$
  - This is how Isabelle uses rewrites
  - Rewrites can instead be based on implications and other formulas (e.g. a = b mod n), but one must take great care that rewriting corresponds to logically valid steps.
- But of course, not everything that *can* be a rewrite rule *should be a rewrite rule! Rewrite sets are picked carefully:*
  - *Ideally they terminate (see next slide)*
  - *And ideally they rewrite an expression to a simplified canonical normal form (covered later in lecture)*

Termination

We say that a set of rewrites rules terminates iff:

starting with any expression, successively applying rewrite rules eventually brings us to a state where no more rewrites apply

- All the rewrite rule sets encountered so far in this lecture terminate; there is no way to loop or apply them without end
- The following rewrite rules may cause a set to be non-terminating
  - a reflexive rewrite (such as $0 \Rightarrow 0$)
  - a self-commuting rewrite (such as $x * y \Rightarrow y * x$)
  - a commutative pair (such as
    $x+(y+z) \Rightarrow (x+y)+z$ and $(x+y)+z \Rightarrow x+(y+z)$)

- An expression to which no rewrites apply is called a normal form with respect to our set of rewrites

Proving Termination

Termination can be shown by defining a natural number measure on an expression such that each rewrite rule decreases the measure.

Example:

1. $x*0 \Rightarrow 0$
2. $1*x \Rightarrow x$
3. $x^0 \Rightarrow 1$
4. $x+0 \Rightarrow x$

$$
\begin{array}{lll}
 & a^{\underline{2*0}} * 5 + b*0 & \mathit{measure} = 5 \\
= & a^0*5 + b*0 & \mathit{measure} = 4 \\
= & \underline{1*5} + b*0 & \mathit{measure} = 3 \\
= & 5 + \underline{b*0} & \mathit{measure} = 2 \\
= & \underline{5 + 0} & \mathit{measure} = 1 \\
= & 5 & \mathit{measure} = 0
\end{array}
$$

For this set of algebraic rewrites, define the measure of an expression as as the *count* of the number of binary operations (plus, times, or exp) it contains.

Since any rule application will decrease the measure of an expression, and since the measure cannot go past zero, this set of rewrites will always terminate.

For $a^{2*0}*5 + b*0$, one possible sequence of rewrite rules is shown at left. It terminates with normal form 5.

Notation

- We use ⇒ to indicate an application of a rewrite rule as well as the declaration of the rewrite rule; e.g. given a rule $x+0 \Rightarrow x$, we may denote the fact that 5+0 rewrites to 5 as $5+0 \Rightarrow 5$
- When considering rewrite systems, it can be useful to speak of multi-step rewrites: we use ⇒* to mean zero or more rewrite steps; e.g.

  if our set contains $a \Rightarrow b$ and $b \Rightarrow c$, we can write $a \Rightarrow^* c$;

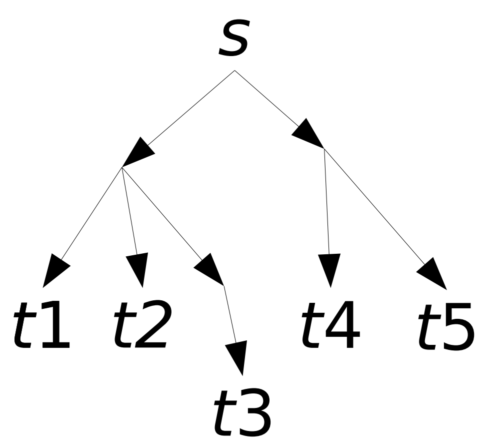
  in the previous example, $a^{2*0} * 5 + b * 0 \Rightarrow^* 5$
- We will also use the notations:

  $a \Leftrightarrow b$ for $a \Rightarrow b$ or $b \Rightarrow a$

  $a \Leftrightarrow^* b$ for there is some chain of zero or more $u_1, u_2, \ldots, u_n$ such that: $a \Leftrightarrow u_1 \Leftrightarrow u_2 \Leftrightarrow \ldots \Leftrightarrow u_n \Leftrightarrow b$
- In diagrams, we draw $\xrightarrow{*}$, or $\xleftrightarrow{*}$ to represent ⇒* and ⇔*

Canonical Normal Form

Depending on our set of rewrite rules, the order of application might affect the result.



We might have $s \Rightarrow^* t1$, $s \Rightarrow^* t2$, $s \Rightarrow^* t3$, $s \Rightarrow^* t4$, and $s \Rightarrow^* t5$, with $t1$, $t2$, $t3$, $t4$, and $t5$ normal.

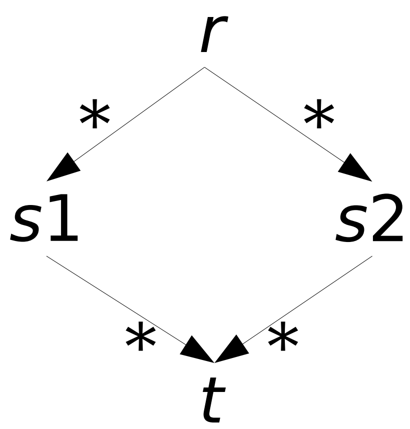
If all normal forms arising from an expression are identical, we say we have a canonical normal form of the expression.

This is a very nice property! It means that the order doesn't matter; in this example, it would mean all the *tn* are identical. In general, this property means our rewrites are simplifying the expression in a canonical (safe) way.

Church-Rosser and Confluence

How do we know if our set gives canonical normal forms?

Two definitions are helpful:



- A set of rewrite rules is confluent if:
  for all terms r, $s1$ and $s2$ such that $r \Rightarrow^* s1$ and $r \Rightarrow^* s2$ (by different sequences of rewrite rules), there exists a term t such that $s1 \Rightarrow^* t$ and $s2 \Rightarrow^* t$
- A set of rewrite rules is Church-Rosser if for all terms $s1$ and $s2$ such that $s1 \Leftrightarrow^* s2$, there exists a term t such that $s1 \Rightarrow^* t$ and $s2 \Rightarrow^* t$

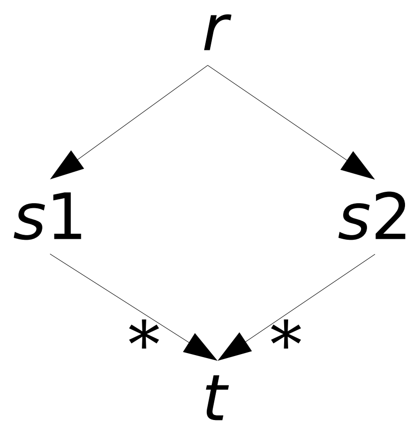
Theorem: Church-Rosser is equivalent to confluence

Theorem: for terminating rewrite sets, these properties mean that any expression will rewrite to a canonical normal form

Local Confluence

The properties of Church-Rosser and confluence can be difficult to prove. A weaker definition is very useful:

A set of rewrite rules is locally confluent if: for all terms r, $s1$ and $s2$ such that $r \Rightarrow s1$ and $r \Rightarrow s2$ (by a different rewrite rule), there exists a term t such that $s1 \Rightarrow^* t$ and $s2 \Rightarrow^* t$



Theorem:

local confluence + termination = confluence

Furthermore: local confluence is decidable (due to Knuth & Bendix)

Both the theorem and the decision procedure use the idea of critical pairs.

Choices in Rewriting

How can choices arise in rewriting?

- Multiple rewrite rules apply to a single redex: order might matter
- Rewrite rules apply to multiple redexes
  - if they are separate, order does not matter
  - but if one contains the other, the choice of order may matter

Examples:

(1) $x^0 \Rightarrow 1$
(2) $0^y \Rightarrow 0$

0^0 rewrites to: 1, by (1)
and to: 0, by (2)

$\langle 1,0 \rangle$

these are instances of what we call critical pairs

(1) $w \circ e \Rightarrow w$
(2) $(x \circ y) \circ z \Rightarrow x \circ (y \circ z)$

$(x \circ e) \circ z$ rewrites to:
$x \circ z$, by (1) and $x \circ (e \circ z)$, by (2)

$\langle x \circ z, x \circ (e \circ z) \rangle$

We are interested in the case where order matters; i.e. one subexpression is totally contained within another subexpression.

Critical Pairs

- Given two rules $L_1 \Rightarrow R_1$ and $L_2 \Rightarrow R_2$, we are concerned with the case when there exists a sub-term s of L_1 such that $s\theta \equiv L_2\theta$, with most general unifier θ

Applying these rules in different orders gives rise to a critical pair

$$\begin{tikzcd} & L_1\theta \arrow[dl, "L_1\theta \Rightarrow R_1\theta"'] \arrow[dr] & \\ R_1\theta & & L_1\theta[\ R_2\theta\ /\ s\theta\ ] \end{tikzcd}$$

$s[\theta]$ is a sub-term of $L_1\theta$

$s[\theta] \equiv L_2\theta$ and

$L_2[\theta] \Rightarrow R_2\theta$

N.B. the two rules should have no variables in common; if they do, we must rename them in order to find the mgu θ

example:

$$w \circ e \Rightarrow w$$

$$(x \circ y) \circ z \Rightarrow x \circ (y \circ z)$$

$\theta = [w/x, e/y]$, any other?

Critical pair: $< R_1\theta\ ,\ L_1\theta[R_2\theta\ /\ s\theta] >$

Testing for Local Confluence

We are interested in whether we can *conflate* all the possible critical pairs i.e. given $r \Rightarrow s1$ and $r \Rightarrow s2$, for expressions r, s1, and s2, can we find a t such that $s1 \Rightarrow^* t$ and $s2 \Rightarrow^* t$?

The following algorithm is used to test for local confluence:

Assumption: Set of rewrite rules R is known to be terminating

Find all critical pairs of all pairs of rules in R

For each critical pair <*a*, *b*>, check that it is *joinable* (or *conflatable*)

find a normal form a' of a

find a normal form b' of b

check that $a' \equiv b'$;
FAIL if they are different

Establishing Local Confluence

Sometimes non-locally confluent (i.e. test fails)

$$\begin{array}{lcl} x * e & \Rightarrow & x \\ f * x & \Rightarrow & x \end{array}$$ not locally confluent since $\langle f, e \rangle$ does not conflate

However, we can add a new rule, $f \Rightarrow e$ to make this critical pair joinable

CARE:

Adding the new rule

- must preserve termination
- might give rise to new critical pairs

need to start over again and check local confluence

Knuth-Bendix Completion

Set of rewrite rules **R**, known to be terminating

1. *let* $i = 0$ *and* $R_1 = R$
2. *increment* i *by* 1
3. *find all critical pairs (c.p.s) of all pairs of rules in* R_i
4. *for all c.p.s.* $\langle a, b \rangle$,
   - *find a normal form* a' *of* a
   - *find a normal form* b' *of* b
5. *if* $a' \not\equiv b'$, *extend the set* R_i *to* R_{i+1} *as follows:*

   if $R_i \cup \{a' \Rightarrow b'\}$ *is terminating, let*

   $R_{i+1} = R_i \cup \{a' \Rightarrow b'\}$ *and goto step 2*

   if $R_i \cup \{b' \Rightarrow a'\}$ *is terminating, let*

   $R_{i+1} = R_i \cup \{b' \Rightarrow a'\}$ *and goto step 2*

   if neither is terminating then exit with FAIL
6. *let* $R^* = R_i$

IF the above procedure terminates without failure THEN R^* *is confluent*

Rewriting in Isabelle

- In Isabelle, the powerful tactic simp performs rewriting
- Many useful lemmas already added to the simplifier – hence power of simp and auto.
- You can control rewrite rules used by simplifier
  - To add, delete, or use only a limited set, in a single proof step, write:

    `apply (simp add: eq1 .. eqn)` (or `del:` or `only:`)

  - To add a lemma permanently as a rewrite rule, insert `[simp]` after its name when you are defining it
  - A lemma P that is not of the form $L=R$ will be interpreted implicitly as $P=True$ when used as a rewrite rule
  - Simp rule `xxx_def` will expand a definition (of `xxx`)

More on rewriting in Isabelle

- **Conditional rewriting**: a lemma with assumptions is applied if the simplifier can prove the assumptions
  `[| P1; ... ; Pn |] ==> l = r`
- **Ordered rewriting**: a lexicographical (dictionary) ordering can be used to prevent endless loops, e.g. to prevent:
  a + b ⇒ b + a ⇒ a + b ⇒ . , ,
- More control:
- `apply (simp (no_asm_simp) ...)`
  Simplify only the conclusion (the "…" might be one or more modifiers (`add`,`del`,`only`), or nothing at all)
- `apply (simp (no_asm_use) ...)`
  Simplify everything, but without using the assumptions
- `Apply (simp (no_asm) ...)`
  Simplify only the conclusion, without using the assumptions

Summary

- Rewrite rules are a powerful technique
  for automated reasoning
- A rule set gives canonical normal forms if it is
  - (1) terminating; and
  - (2) locally confluent
- We show (1) by finding a monotonic measure
- We show (2) using critical pairs and the Knuth-Bendix procedure to try to make confluent set from a non-confluent set. (This does not always work.)
- In Isabelle, we have a lot of control over the rewrites.