Automated Reasoning

Lecture 9: Rewriting

Robert Atkey
bob.atkey@ed.ac.uk

based on originals by Jacques Fleuriot

Tuesday 10th February 2015
Recap

- Previously:
  - Unification

- This time: Rewriting
  - Sets of rewrite rules
  - Termination, Confluence and Canonical Normal Forms
  - Critical Pairs
  - Knuth-Bendix Completion
  - Rewriting in Isabelle
Term Rewriting

Rewriting is a technique for replacing terms in an expression with equivalent terms.

For example, the rules:

\[ x \times 0 \Rightarrow 0 \quad x + 0 \Rightarrow x \]

can be used to simplify an expression:

\[ x + (x \times 0) \rightarrow x + 0 \rightarrow x \]

We use the notation \( L \Rightarrow R \) to define a rewrite rule that replaces the term \( L \) with the term \( R \) in an expression.

In general, rewrite rules contain (meta-)variables (e.g., \( X + 0 \Rightarrow X \)), and are instantiated using matching (one-way unification).
The Power of Rewrites

Given this set of rules:

\begin{align*}
0 + N & \implies N \quad (1) \\
(0 \leq N) & \implies \text{True} \quad (2) \\
s(M) + N & \implies s(M + N) \quad (3) \\
s(M) \leq s(N) & \implies M \leq N \quad (4)
\end{align*}

We can prove this statement:

\[
0 + s(0) \leq s(0) + x
\]

\rightarrow \quad s(0) \leq s(0) + x \quad \text{by (1)}

\rightarrow \quad s(0) \leq s(0 + x) \quad \text{by (3)}

\rightarrow \quad 0 \leq 0 + x \quad \text{by (4)}

\rightarrow \quad \text{True} \quad \text{by (2)}
Symbolic Computation

Given this set of rules:

\begin{align*}
0 + N & \Rightarrow N \\
s(M) + N & \Rightarrow s(M + N) \\
0 \times N & \Rightarrow 0 \\
s(M) \times N & \Rightarrow (M \times N) + N
\end{align*}

\((s(x)\text{ means "successor of } x\text{", } i.e.\ 1 + x)\)

We can rewrite \(2 \times x\) to \(x + x\):

\[
\begin{align*}
s(s(0)) \times x & \\
\quad \rightarrow (s(0) \times x) + x & \quad \text{by (4)} \\
\quad \rightarrow ((0 \times x) + x) + x & \quad \text{by (4)} \\
\quad \rightarrow (0 + x) + x & \quad \text{by (3)} \\
\quad \rightarrow x + x & \quad \text{by (1)}
\end{align*}
\]
**Rewrite Rule of Inference**

\[
\frac{P(t) \quad L \Rightarrow R \quad L[\theta] \equiv t}{P(R[\theta])}
\]

where \(P(t)\) means that \(P\) contains \(t\) somewhere inside it.

**Note:** rewriting uses **matching**, not unification (the substitution \(\theta\) is not applied to \(t\)).

**Example**

Given an expression \((s(a) + s(0)) + s(b)\)

and a rewrite rule \(s(X) + Y \Rightarrow s(X + Y)\)

we can find \(t = s(a) + s(0)\)

and \(\theta = [a/X, s(0)/Y]\)

\[\text{to yield } s(a + s(0)) + s(b)\]**
Notation

- **Rewrite rules**: $L \Rightarrow R$

- **Rewrite rule applications**: $s \rightarrow t$
  
  *e.g.*, $s(s(0)) \ast x \rightarrow (s(0) \ast x) + x$

- **Multiple (zero or more) rewrite rule applications**: $s \rightarrow^* t$
  
  *e.g.*, $s(s(0)) \ast x \rightarrow^* x + x$

- **Back-and-forth**:
  - $s \leftrightarrow t$ for $s \rightarrow t$ or $t \rightarrow s$
  - $s \leftrightarrow^* t$ for a chain of zero or more $u_i$ such that
    
    $s \leftrightarrow u_1 \leftrightarrow \ldots \leftrightarrow u_n \leftrightarrow t$
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 to it.

Don’t want: “Oceania has always been at war with Eurasia” \( \rightarrow \) “Oceania has always been at war with Eastasia”

Most commonly, a rewrite \( L \Rightarrow R \) means that \( L = R \);

- Rewrites can instead be based on implications and other formulas (e.g., \( a = b \mod n \)), but care is needed to make sure that rewriting corresponds to logically valid steps.

  e.g., if \( A \Rightarrow B \) means \( A \) implies \( B \), then it is safe to rewrite \( A \) to \( B \) in \( A \wedge C \), but not in \( \neg A \wedge C \).
How to choose rewrite rules?

There are often many equalities to choose from:

\[ X + Y = Y + X \quad X + (Y + Z) = (X + Y) + Z \quad X + 0 = X \]

\[ 0 + X = X \quad 0 + (X + Y) = Y + X \quad \ldots \]

Could all be valid rewrite rules.

**But:** Not everything that can be rewrite rule should be a rewrite rule!

- Ideally, a set of rewrite rules should be **terminating**
- Ideally, they should rewrite to a **canonical normal form**
An Example: Algebraic Simplification

Rules:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule</td>
<td></td>
</tr>
<tr>
<td>$x \times 0 \Rightarrow 0$</td>
<td>(1)</td>
</tr>
<tr>
<td>$1 \times x \Rightarrow x$</td>
<td>(2)</td>
</tr>
<tr>
<td>$x^0 \Rightarrow 1$</td>
<td>(3)</td>
</tr>
<tr>
<td>$x + 0 \Rightarrow x$</td>
<td>(4)</td>
</tr>
</tbody>
</table>

Example:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Expression</td>
<td>Action</td>
</tr>
<tr>
<td>$a^{2 \times 0} \times 5 + b \times 0$</td>
<td>$\rightarrow a^{0} \times 5 + b \times 0$ by (1)</td>
</tr>
<tr>
<td>$\rightarrow 1 \times 5 + b \times 0$ by (3)</td>
<td></td>
</tr>
<tr>
<td>$\rightarrow 5 + b \times 0$ by (2)</td>
<td></td>
</tr>
<tr>
<td>$\rightarrow 5 + 0$ by (1)</td>
<td></td>
</tr>
<tr>
<td>$\rightarrow 5$ by (4)</td>
<td></td>
</tr>
</tbody>
</table>

Any subexpression that can be rewritten (i.e. matches the LHS of a rewrite rule) is called a **redex** (*reducible expression*).

The redexes used (but *not* all redexes) have been underlined above.

**Choices:** Which redex to choose? Which rule to choose?
The Rewrite Search Tree

In general, get a tree of possible rewrites:

$$a^{2*0} \cdot 5 + b \cdot 0$$

Common strategies:

- Innermost (inside-out) leftmost redex (1st redex in post-order traversal)
- Outermost (outside-in) leftmost redex (1st redex in pre-order traversal)

Important questions:

- Is the tree finite? (does the rewriting always terminate?)
- Does it matter which path we take? (is every leaf the same?)
Termination

We say that a set of rewrite rules is terminating if:

*starting with any expression, successively applying rewrite rules eventually brings us to a state where no rule applies.*

Also called *(strongly) normalizing* or *noetherian.*

All the rewrite sets so far in this lecture are terminating.

Examples of rules that can cause non-termination:

- Reflexive rules: *e.g.* $0 \Rightarrow 0$
- Self-commuting rewrites: *e.g.* $X \ast Y \Rightarrow Y \ast X$
- Commuting pairs of rewrites: *e.g.*:
  \[
  X + (Y + Z) \Rightarrow (X + Y) + Z \text{ and } (X + Y) + Z \Rightarrow X + (Y + Z)
  \]

An expression to which no rewrite rules apply is called a **normal form** (with respect to that set of rewrite rules).
Proving Termination

Termination can be shown in some cases by:

1. defining a natural number **measure** on expressions
2. such that each rewrite rule decreases the measure

Measure cannot go below zero, so any sequence will terminate.

Example:

\[
\begin{align*}
  x \ast 0 & \Rightarrow 0 \quad (1) \\
  1 \ast x & \Rightarrow x \quad (2) \\
  x^0 & \Rightarrow 1 \quad (3) \\
  x + 0 & \Rightarrow x \quad (4)
\end{align*}
\]

For these rules, define the **measure** of an expression as the number of binary operations \((+, -, \ast)\) it contains.

Every rule removes a binary operation, so each rule application will reduce the overall measure of an expression.

In general: look for a **well-founded termination order** (e.g., lexicographical path ordering (LPO))
Canonical Normal Form

For some rewrite rule sets, order of application might affect result.

We might have:

\[
\begin{align*}
\text{s} & \rightarrow t_1^* \quad t_2^* \quad t_3^* \quad t_4^* \quad t_5^*
\end{align*}
\]

where all of \( t_1, t_2, t_3, t_4, t_5 \) are in normal form.

If all the normal forms are identical we can say we have a canonical normal form for \( s \).

This is a very nice property!

- Means that order of rewrite rule application doesn’t matter
- In general, means our rewrites are simplifying the expression in a canonical (safe) way.
How do we know when a set of rules yields canonical normal forms?

A set of rewrite rules is **confluent** if for all terms \( r, s_1, s_2 \) such that \( r \rightarrow^* s_1 \) and \( r \rightarrow^* s_2 \) there exists a term \( t \) such that \( s_1 \rightarrow^* t \) and \( s_2 \rightarrow^* t \).

A set of rewrite rules is **Church-Rosser** if for all terms \( s_1, s_2 \) such that \( s_1 \leftrightarrow^* s_2 \), there exists a term \( t \) such that \( s_1 \rightarrow^* t \) and \( s_2 \rightarrow^* t \).

**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 useful:

A set of rewrite rules is locally confluent if for all terms $r, s_1, s_2$ such that $r \rightarrow s_1$ and $r \rightarrow s_2$ there exists a term $t$ such that $s_1 \rightarrow^* t$ and $s_2 \rightarrow^* t$.

Theorem (Newman’s Lemma)

\[
\begin{array}{c}
  r \\
  \downarrow \\
  s_1 \\
  \downarrow \\
  t \\
  \downarrow \\
  s_2 \\
  \downarrow \\
  * \\
  \end{array}
\]

local confluence + termination = confluence

Also: local confluence is decidable (due to Knuth and Bendix)

Both theorem and the decision procedure use idea of critical pairs
Choices in Rewriting

How can choices arise in rewriting?

- Multiple rules apply to a single redex: order might matter
- Rules apply to multiple redexes:
  - if they are separate: order does not matter
  - if one contains the other: order might matter

We are interested in cases where the order matters:

<table>
<thead>
<tr>
<th>Rules</th>
<th>Rewrites</th>
<th>Critical Pair</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X^0 \Rightarrow 1$</td>
<td>$0^0$ rewrites to 0 and 1</td>
<td>$\langle 0, 1 \rangle$</td>
</tr>
<tr>
<td>$0^Y \Rightarrow 0$</td>
<td>to 1</td>
<td></td>
</tr>
<tr>
<td>$X \cdot e \Rightarrow X$</td>
<td>$(x \cdot e) \cdot z$ rewrites to $x \cdot z$ and $x \cdot (e \cdot z)$</td>
<td>$\langle x \cdot z, x \cdot (e \cdot z) \rangle$</td>
</tr>
<tr>
<td>$(X \cdot Y) \cdot Z \Rightarrow X \cdot (Y \cdot Z)$</td>
<td>$x \cdot z$ and $x \cdot (e \cdot z)$</td>
<td></td>
</tr>
</tbody>
</table>


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 $s[\theta] = L_2[\theta]$, with most general unifier $\theta$.

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

$$\langle R_1[\theta], L_1[\theta][R_2[\theta]/s[\theta]] \rangle$$ is the critical pair.

Note: the variables in the two rules should be renamed so they don’t share any variable names.

With $W \cdot e \Rightarrow W$ and $(X \cdot Y) \cdot Z \Rightarrow X \cdot (Y \cdot Z)$ we can have $\theta = [W/X, e/Y]$, any other?
Testing for Local Confluence

If we can conflate all the critical pairs, then have local confluence.

Conflation for a critical pair $\langle s_1, s_2 \rangle$ is when there is a $t$ such that $s_1 \longrightarrow^* t$ and $s_2 \longrightarrow^* t$.

An algorithm to test for local confluence (assuming termination):

1. Find all the critical pairs in $R$
2. For each critical pair $\langle s_1, s_2 \rangle$:
   2.1 Find a normal form $s_1'$ of $s_1$;
   2.2 Find a normal form $s_2'$ of $s_2$;
   2.3 Check $s_1' = s_2'$, if not then fail.
Establishing Local Confluence

Sometimes a set of rules is not locally confluent

\[ X \cdot e \Rightarrow X \]
\[ f \cdot X \Rightarrow X \]

is not locally confluent: \( \langle f, e \rangle \) does not conflate.

We can add the rule \( f \Rightarrow e \) to make this critical pair joinable.

However, adding new rules requires care:

- Must preserve termination
- Might give rise to new critical pairs (need to check local confluence again)
Knuth-Bendix Completion Algorithm

Start with a set $R$ of terminating rewrite rules

While there are non-conflatable critical pairs in $R$:

1. Take a critical pair $\langle s_1, s_2 \rangle$ in $R$
2. Normalise $s_1$ to $s'_1$ and $s_2$ to $s'_2$ (and we know $s'_1 \neq s'_2$)
3. if $R \cup \{ s'_1 \Rightarrow s'_2 \}$ is terminating then
   $$ R := R \cup \{ s'_1 \Rightarrow s'_2 \} $$
   else if $R \cup \{ s'_2 \Rightarrow s'_1 \}$ is terminating then
   $$ R := R \cup \{ s'_2 \Rightarrow s'_1 \} $$
   else Fail

* If this terminates then we have a locally confluent and terminating (and hence confluent) rewrite set (it may not terminate!)

* Depends on the termination check: define a measure and use that to test for termination.
Rewriting in Isabelle

Isabelle has two rules for primitive rewriting (useful with `erule`):

\[
\text{subst} : \quad [s = t; P \, s] \implies P \, t
\]
\[
\text{ssubst} : \quad [t = s; P \, s] \implies P \, t
\]

The `P` is matched against the term using higher-order unification.

There is also a tactic that rewrites using a theorem:

- `apply (subst `theorem)`: rewrites goal using `theorem`
- `apply (subst (asm) `theorem)`: rewrites assumptions using `theorem`
- `apply (subst (`i_1` `i_2`...) `theorem)`: rewrites goal at positions $i_1$, $i_2$, ...
- `apply (subst (asm) (`i_1` `i_2`...) `theorem)`: rewrites assumptions at positions $i_1$, $i_2$, ...

Working out what the right positions are is essentially just trial and error, and can be quite brittle.
The Isabelle Simplifier

The `simp` tactic does automatic rewriting, using a database of rules.

Adding `[simp]` after a lemma (or theorem) name when declaring it adds that lemma to the simplifier’s database.

▶ If it is not an equality, then it is treated as $P = \text{True}$.

Many rules are already added — so it often appears quite magical.

Control over the rules:

▶ `simp add: ...` `del: ...`
▶ `simp only: ...
▶ `simp (no_asm)` – ignore assumptions
▶ `simp (no_asm_simp)` – use assumps, but do not rewrite them
▶ `simp (no_asm_use)` – rewrite assumps, don’t use them

Every definition generates a rewrite rule `defnname_def`
The Isabelle Simplifier

The Isabelle simplifier also has more bells and whistles:

1. Conditional rewriting:

$$[P_1, \ldots, P_n] \implies s = t$$

if it can prove $P_1, \ldots, P_n$

2. Ordered rewriting: a lexicographical (dictionary) ordering is used to prevent (some) endless loops:

$$a + b \rightarrow b + a \rightarrow a + b \ldots$$

3. Case splitting:

$$?P \text{ (case } ?x \text{ of True } \Rightarrow ?f_1 \mid \text{False } \Rightarrow ?f_2)$$

$$= ((?x = \text{True } \Rightarrow ?P ?f_1) \land (?x = \text{False } \Rightarrow ?P ?f_2))$$

Applies when there is an explicit case split in the goal

More information: Isabelle tutorial Section 3.1, Chapter 9.
Summary

- Rewriting (Bundy Ch. 9)
  - Rewriting expressions using rules
  - Termination (by strictly decreasing measure)
  - Local confluence
  - Local confluence + Termination = Confluence
  - Canonical Normal Forms
  - Critical Pairs and Knuth-Bendix Completion
  - Rewriting in Isabelle (Isabelle Tutorial, Section 3.1)

- Next time: Inductive Proof in Isabelle