Automated Reasoning

Jacques Fleuriot

September 14, 2013

Unification Algorithms

Jacques Fleuriot

Motivation

- Isabelle will often automatically figure out instantiations for terms and types when using theorems and tactics.
- How does it do this?

A First Look at Unification

Unification is **the operation of finding a common instance of two terms.**

Informally: We want to **make terms identical** by finding the most general **substitutions** of terms for variables.

Example

Can we make these terms equal by finding a common instance?

$f(x, B)$ and $f(A, y)$	Yes: $[A/x, B/y]$	instance: $f(A, B)$
$f(x, x)$ and $f(A, B)$	No.	
$f(x, x)$ and $f(y, g(y))$	No.	

Only variables may be replaced by other terms.

Note: When representing terms, we assume that $x, y, z, \ldots$ denote variables while $X, Y, Z, \ldots$ denote constants.

Matching

Problem

Given **pattern** *and* **target** *find a* **substitution** *such that:*

$$\mathit{pattern}[\mathit{substitution}] \equiv \mathit{target}$$

where $\equiv$ *means that the terms are identical.*

Example

$(s(x) + y)\,[0/x, s(0)/y] \equiv (s(0) + s(0))$.

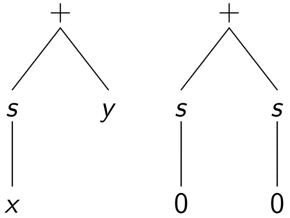
How do we find an adequate substitution?

We view matching as equation solving.

Matching (continued)

Example

$$(s(x) + y) \equiv (s(0) + s(0))$$
$$\downarrow$$
$$(s(x) \equiv s(0)) \wedge (y \equiv s(0))$$
$$\downarrow$$
$$(x \equiv 0) \wedge (y \equiv s(0))$$



The process works by decomposing the term trees.

Abbreviations

Term	Meaning
$\overrightarrow{t}$	$t_1, \ldots, t_n$ $(n \geq 1)$
$\bigwedge_i t_i$	$t_1 \wedge \cdots \wedge t_n$
$vars(t)$	the set of free variables in t
$Vars$	the set of (all) free variables.

Example

$$vars\,(f\,(x, y, g(A, z, x))) = \{x, y, z\}$$
$$vars\,(f(A, B, C)) = \{\}\,.$$

Matching as Equation Solving

- Start with *pattern* and *target* **standardised apart**:

$$vars(pattern) \cap vars(target) = \{\}$$

- Goal is to solve for $Vars(pattern)$ in equation $pattern \equiv target$
- Strategy is to use transformation rules

$$\begin{array}{cl} pattern \equiv target & \\ \downarrow & \\ \vdots & \quad \leftarrow \textbf{Transformations} \\ \downarrow & \\ x_1 \equiv s_1 \wedge \ldots \wedge x_n \equiv s_n & \end{array}$$

- Resulting substitution is $[s_1/x_1, \ldots, s_n/x_n]$.
- Transformations end in failure if no match possible

Transformation Rules for Matching

Decompose:	$s(x) + y \equiv s(0) + s(0)$ $\downarrow$ $s(x) \equiv s(0) \wedge y \equiv s(0)$	$s(x) \equiv s(0)$ $\downarrow$ $x \equiv 0$
Conflict:	$s(x) + y \equiv s(0)$ $\downarrow$ fail	Cannot match since $+$ and s conflict
Eliminate:	$(x + y \equiv s(0) + 0) \wedge y \equiv 0$ $\downarrow$ $(x + 0 \equiv s(0) + 0) \wedge y \equiv 0$	
Delete:	$x \equiv 0 \wedge (s(0) + 0 \equiv s(0) + 0)$ $\downarrow$ $x \equiv 0$	

Matching Algorithm

Assumptions: s and t are arbitrary terms and $vars(pattern) \cap vars(target) = \{\}$.

Name	**Before**	**After**	**Condition**
Decompose	$P \wedge f\left(\overrightarrow{s}\right) \equiv f\left(\overrightarrow{t}\right)$	$P \wedge \bigwedge_i s_i \equiv t_i$	
Conflict	$P \wedge f\left(\overrightarrow{s}\right) \equiv g\left(\overrightarrow{t}\right)$	*fail*	$f \not\equiv g$
Eliminate	$P \wedge x \equiv s$	$P[s/x] \wedge x \equiv s$	$x \in vars(P)$
Delete	$P \wedge s \equiv s$	P	

The algorithm terminates with success when no further rules apply and failure has not occurred.

The algorithm succeeds with a match iff there is one.

Unification

Unification is two-way matching (there is no distinction between pattern and target).

$$exp_1[subst] \equiv exp_2[subst]$$

Example

What substitution makes $(s(x) + s(0))$ and $(s(0) + y)$ identical?

$$\theta = [0/x, s(0)/y]$$

We need to add extra rules to our matching algorithm

$$\begin{array}{rl}
(s(x) + s(0)) \equiv (s(0) + y) & \\
\downarrow \text{Decompose} & \\
(s(x) \equiv s(0)) \wedge (s(0) \equiv y) & \\
\downarrow \text{Decompose} & \\
(x \equiv 0) \wedge (s(0) \equiv y) & \quad \text{From symmetry of } \equiv \\
\downarrow \text{Switch} & \\
(x \equiv 0) \wedge (y \equiv s(0)) &
\end{array}$$

New Transformation Rules

Switch

$$A \equiv x \;\downarrow\; x \equiv A$$

Switch rule applies only if *lhs* is not originally a variable

Coalesce

$$x \equiv y + 1 \wedge y \equiv x \;\downarrow\; x \equiv x + 1 \wedge y \equiv x$$

Similar to Eliminate, except both *lhs* and *rhs* are variables

Occurs Check

$$x \equiv x + 1 \;\downarrow\; \mathit{fail}$$

lhs cannot occur in *rhs*

Example

$$\begin{array}{c} f(x,x) \equiv f(y, y+1) \\ \downarrow \text{Decompose} \\ x \equiv y \wedge x \equiv y + 1 \\ \downarrow \text{Coalesce} \\ x \equiv y \wedge y \equiv y + 1 \\ \downarrow \text{Occurs check} \\ \mathit{fail} \end{array}$$

$$\begin{array}{c} P(x) \wedge x \equiv x + 1 \\ \downarrow \text{Eliminate} \\ P(x+1) \wedge x \equiv x + 1 \\ \downarrow \text{Eliminate} \\ P(x+1+1) \wedge x \equiv x + 1 \\ \downarrow \text{Eliminate} \\ \ldots \end{array}$$

Non-termination can result without the occurs check.

Unification Algorithm

Assumptions: s and t are arbitrary terms and $Vars = vars(exp_1) \cup vars(exp_2)$.

Name	Before	After	Condition
Decompose	$P \wedge f(\vec{s}) \equiv f(\vec{t})$	$P \wedge \bigwedge_i s_i \equiv t_i$	
Conflict	$P \wedge f(\vec{s}) \equiv g(\vec{t})$	fail	$f \not\equiv g$
Switch	$P \wedge s \equiv x$	$P \wedge x \equiv s$	$x \in Vars$ $s \notin Vars$
Delete	$P \wedge s \equiv s$	P	
Eliminate	$P \wedge x \equiv s$	$P[s/x] \wedge x \equiv s$	$x \in vars(P)$ $x \notin vars(s)$ $s \notin Vars$
Occurs Check	$P \wedge x \equiv s$	*fail*	$x \in vars(s)$ $s \notin Vars$
Coalesce	$P \wedge x \equiv y$	$P[y/x] \wedge x \equiv y$	$x, y \in vars(P)$ $x \not\equiv y$

- Conditions ensure that at most one rule applies to each conjunct
- Algorithm terminates with success when no further rules apply.

Theoretical Properties of Unification Algorithm

- The algorithm will find a unifier, if it exists.
- It returns the **most general unifier** (mgu) θ:

  $$exp_1[\theta] \equiv exp_2[\theta] \wedge \forall \phi.\ exp_1[\phi] \equiv exp_2[\phi] \rightarrow \exists \psi.\ \phi = \theta \bullet \psi.$$

  Consider $g(g(x))$ and $g(y)$. Is $[g(3)/y, 3/x]$ a unifier? Is it the mgu?
- mgu is **unique** up to alphabetic variance;
- the algorithm can easily be extended to simultaneous unification on n expressions.

Building-in Axioms

General Scheme:

$$(Ax_1 \cup Ax_2) + unif \implies Ax_1 + unif_{Ax_2}.$$

Some axioms of the theory become built into unification.

Example

Commutative-Unification

$$\begin{array}{c} x + 2 = y + 3 \\ \downarrow \\ y = 2 \land x = 3 \end{array}$$

We no longer use $\equiv$ but $=$

How do we deal with this?
We can add a new transformation rule (**Mutate rule**).

Unification Algorithm for Commutativity

Name	Before	After	Condition
Decompose	$P \wedge f(\vec{s}) = f(\vec{t})$	$P \wedge \bigwedge_i s_i = t_i$	
Conflict	$P \wedge f(\vec{s}) = g(\vec{t})$	fail	$f \neq g$
Switch	$P \wedge s = x$	$P \wedge x = s$	$x \in Vars$ $s \notin Vars$
Delete	$P \wedge s = s$	P	
Eliminate	$P \wedge x = s$	$P[s/x] \wedge x = s$	$x \in vars(P)$ $x \notin vars(s)$ $s \notin Vars$
Check	$P \wedge x = s$	*fail*	$x \in vars(s)$ $s \notin Vars$
Coalesce	$P \wedge x = y$	$P[y/x] \wedge x = y$	$x, y \in vars(P)$ $x \neq y$
Mutate	$P \wedge f(s_1, t_1) = f(s_2, t_2)$	$P \wedge s_1 = t_2 \wedge t_1 = s_2$	f is commutative

Decompose and Mutate rules overlap.

Most General Unifiers

For ordinary unification, the mgu is unique, but what happens when new rules are built-into the unification algorithm?

Multiple mgus: Commutative unification

$$x + y = A + B \longrightarrow \begin{cases} x = A \wedge y = B \\ x = B \wedge y = A \end{cases}$$

Both are equally general.

Infinitely many mgus: Associative unification $x + (y + z) = (x + y) + z$.

$$x + A = A + x \longrightarrow \begin{cases} x = A \\ x = A + A \\ x = A + A + A \\ \ldots \end{cases}$$

All independent (not unifiable).

No mgus: Build in $f(0, x) = x$ and $g(f(x, y)) = g(y)$:

$$g(x) = g(A) \longrightarrow \begin{cases} x = A \\ x = f(y_1, A) \\ x = f(y_1, f(y_2, A)) \end{cases}$$

Many unifiers but no mgu.

Types of Unification

Unitary	A single unique mgu, or none (predicate logic).
Finitary	Finite number of mgus (predicate logic with commutativity).
Infinitary	Possibly infinite number of mgus (predicate logic with associativity).
Nullary	No mgus exist, although unifiers may exist.
Undecidable	Unification not decidable — no algorithm.

Types of Unification

Axioms	Type	Decidable
nil	unitary	yes
commutative	finitary	yes
associative	infinitary	yes
assoc. + dist.	infinitary	yes
distributive	infinitary	unknown
lambda calculus	infinitary	no

Summary

- Algorithms for matching and unification.
- Unification as equation solving.
- Transformation rules for equation solving.
- Building-in axioms.
- Most general unifiers and classification.