15-819K: Logic Programming

Lecture 6

Unification

Frank Pfenning

September 14, 2006

In this lecture we take the essential step towards making the choice of goal and rule instantiation explicit in the operational semantics. This consists of describing an algorithm for a problem called *unification* which, given two terms t and s, tries to find a substitution θ for its free variables such that $t\theta = s\theta$ if such a substitution exists. Recall that we write $t\theta$ for the result of applying the substitution θ to the term t.

6.1 Using Unification in Proof Search

Before we get to specifics of the algorithm, we consider how we use unification in proof search. Let us reconsider the (by now tired) example of unary addition

$$\frac{}{\mathsf{plus}(\mathsf{z}, N, N)}\ \mathsf{pz} \qquad\qquad \frac{\mathsf{plus}(M, N, P)}{\mathsf{plus}(\mathsf{s}(M), N, \mathsf{s}(P))}\ \mathsf{ps}$$

and an atomic goal such as

$$\mathsf{plus}(\mathsf{s}(\mathsf{z}), \mathsf{s}(\mathsf{s}(\mathsf{z})), P).$$

Clearly the conclusion of the first rule does not match this goal, but the second one does. What question do we answer to arrive at this statement?

The first attempt might be: "*There is an instance of the rule such that the conclusion matches the goal.*" When we say *instance* we mean here the result of substituting terms for the variables occuring in a rule, proposition, or term. We can see that this specification is not quite right: we need to instantiate the goal as well, since P must have the form $\mathsf{s}(P_1)$ for some as yet

unknown P_1. The subgoal in that case would be $\mathsf{plus}(\mathsf{z}, \mathsf{s}(\mathsf{s}(\mathsf{z})), P_1)$ according to the rule instantiation with z for M, $\mathsf{s}(\mathsf{s}(\mathsf{z}))$ for N, and P_1 for P.

The second attempt would therefore be: "*There is an instance of the rule and an instance of the goal such that the two are equal.*" This does not quite capture what we need either. For example, substituting $\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{z}))))$ for P in the goal and $\mathsf{s}(\mathsf{s}(\mathsf{z})))$ for P in the rule, together with the substitution for M and N from above, will also make the goal and conclusion of the rule identical, but is nonetheless wrong. The problem is that it would overcommit: using P_1 for P in the rule, on the other hand, keeps the options open. P_1 will be determined later by search and other unifications. In order to express this, we define that t_1 is *more general* than t_2 if t_1 can be instantiated to t_2.

The third attempt is therefore: "*Find the most general instance of the rule and the goal so that the conclusion of the rule is equal to the instantiated goal.*" Phrased in terms of substitutions, this says: find θ_1 and θ_2 such that $P'\theta_1 = P\theta_2$, and any other common instance of P' and P is an instance of $P'\theta_1$.

In terms of the algorithm description it is more convenient if we redefine the problem slightly in this way: "*First rename the variables in the rule so that they are disjoint from the variables in the goal. Then find a single most general substitution θ that unifies the renamed conclusion with the goal.*" Here, a unifying substitution θ is most general if any other unifying substitution is an instance of θ.

In the remainder of the lecture we will make these notions more precise and present an algorithm to compute a most general unifier. In the next lecture we show how to reformulate the operational semantics to explicitly use most general unifiers. This means that for the first time the semantics will admit free variables in goals.

6.2 Substitutions

We begin with a specification of substitutions. We use the notation $\mathrm{FV}(t)$ for the set of all free variables in a term.

$$\text{Substitutions} \quad \theta \quad ::= \quad t_1/x_1, \ldots, t_n/x_n$$

We postulate that all x_i are distinct. The order of the pairs in the substitution is irrelevant, and we consider permutations of substitutions to be equal. We denote the *domain* of θ by $\mathrm{dom}(\theta) = \{x_1, \ldots, x_n\}$. Similarly, we call the set of all variables occurring in the substitution term t_i the *codomain* and write $\mathrm{cod}(\theta) = \bigcup_i \mathrm{FV}(t_i)$.

An important general assumption is that the domain and codomain of substitutions are disjoint.

> **Assumption:** All substitutions we consider are *valid*, that is, $\mathrm{dom}(\theta) \cap \mathrm{cod}(\theta) = \emptyset$ for any substitution θ.

This is by no means the only way to proceed. A more common assumption is that all substitutions are idempotent, but we believe the above is slightly more convenient for our limited purposes.

Note that valid substitutions cannot contain pairs x/x, since it would violate our assumption above. However, such pairs are not needed, since their action is the identity, which can also be achieve by simply omitting them.

Applying a substitution θ to a term t is easily defined compositionally.

$$\begin{array}{lcll} x\theta & = & t & \text{if } t/x \text{ in } \theta \\ y\theta & = & y & \text{if } y \notin \mathrm{dom}(\theta) \\ f(t_1, \ldots, t_n)\theta & = & f(t_1\theta, \ldots, t_n\theta) & \end{array}$$

6.3 Composing Substitutions

In the course of search for a deduction, and even in the course of solving one unification problem, we obtain information in a piecemeal fashion. This means we construct a (partial) substitution, apply it, and then construct another substitution on the result. The overall answer is then the *composition* of these two substitution. We write it as $\tau\theta$. The guiding property we need is that for any t, we have $(t\tau)\theta = t(\tau\theta)$. In order for this property to hold and maintain our general assumptions on substitutions, we specify the precondition that $\mathrm{dom}(\tau) \cap \mathrm{dom}(\theta) = \emptyset$ and also that $\mathrm{dom}(\tau) \cap \mathrm{cod}(\theta) = \emptyset$. We define the composition by going through the substitution left-to-right, until encountering the empty substitution $(\cdot)$.

$$\begin{array}{lcl} (t/x, \tau)\theta & = & t\theta/x, \tau\theta \\ (\cdot)\theta & = & \theta \end{array}$$

First, note that $\mathrm{dom}(\tau\theta) = \mathrm{dom}(\tau) \cup \mathrm{dom}(\theta)$ which is a union of disjoint domains. Second, $\mathrm{cod}(\tau\theta) = (\mathrm{cod}(\tau) - \mathrm{dom}(\theta)) \cup \mathrm{cod}(\theta)$ so that $\mathrm{cod}(\tau\theta) \cap \mathrm{dom}(\tau\theta) = \emptyset$ as can be seen by calculation. In other words, $\tau\theta$ is a valid substitution.

It is easy to verify that the desired property of composition actually holds. We will also need a corresponding property stating that composition of substitution is associative, under suitable assumptions.

Theorem 6.1 (Substitution Composition) *Assume we are given a term t and valid substitutions σ and θ with $\mathrm{dom}(\sigma) \cap \mathrm{dom}(\theta) = \emptyset$ and $\mathrm{dom}(\sigma) \cap \mathrm{cod}(\theta) = \emptyset$. Then $\sigma\theta$ is valid and*

$$(t\sigma)\theta = t(\sigma\theta)$$

Furthermore, if τ is a substitution such that $\mathrm{dom}(\tau) \cap \mathrm{dom}(\sigma) = \mathrm{dom}(\tau) \cap \mathrm{dom}(\theta) = \emptyset$ then also

$$(\tau\sigma)\theta = \tau(\sigma\theta)$$

Proof: The validity of $\sigma\theta$ has already been observed above. The first equality follows by induction on the structure of the term t, the second by induction on the structure of τ (see Exercise 6.1).

Case: $t = x$ for a variable x. Then we distinguish two subcases.

Subcase: $x \in \mathrm{dom}(\sigma)$ where $s/x \in \sigma$.

$$\begin{array}{ll} (x\sigma)\theta = s\theta & \text{By defn. of } x\sigma \\ = x(\sigma\theta) & \text{Since } s\theta/x \in \sigma\theta \end{array}$$

Subcase: $x \notin \mathrm{dom}(\sigma)$.

$$\begin{array}{ll} (x\sigma)\theta = x\theta & \text{By defn. of } x\sigma \\ = x(\sigma\theta) & \text{By defn. of } \sigma\theta \end{array}$$

Case: $t = f(t_1, \ldots, t_n)$ for terms $t_1, \ldots, t_n$.

$$\begin{array}{ll} (t\sigma)\theta = f(t_1\sigma, \ldots, t_n\sigma)\theta & \text{By defn. of } t\sigma \\ = f((t_1\sigma)\theta, \ldots, (t_n\sigma)\theta) & \text{By defn. of } f(\_)\theta \\ = f(t_1(\sigma\theta), \ldots, t_n(\sigma\theta)) & \text{By i.h., } n \text{ times} \\ = f(t_1, \ldots t_n)(\sigma\theta) = t(\sigma\theta) & \text{By defn. of } t(\sigma\theta) \end{array}$$

□

6.4 Unification

We say θ is a *unifier* of t and s if $t\theta = s\theta$. We say that θ is a *most general unifier* for t and s if it is a unifier, and for any other unifier σ there exists a substitution σ' such that $\sigma = \theta\sigma'$. In other words, a unifier is most general if any other unifier is an instance of it, where "instance" refers to the composition of substitutions.

As usual in this class, we present the algorithm to compute a most general unifier as a judgment, via a set of inference rules. The judgment has

the form $t \doteq s \mid \theta$, where we think of t and s as inputs and a most general unifier θ as the output. In order to avoid the n-ary nature of the list of arguments, we will have an auxiliary judgment $\mathbf{t} \doteq \mathbf{s} \mid \theta$ for sequences of terms $\mathbf{t}$ and $\mathbf{s}$. Notions such as application of substitution are extended to sequences of terms in the obvious way. We use $(\cdot)$ to stand for an empty sequence of terms (as well as the empty substitution, which is a sequence of term and variable pairs). In general, we will use boldface letters to stand for sequences of terms.

We first consider function terms and term sequences.

$$\frac{\mathbf{t} \doteq \mathbf{s} \mid \theta}{f(\mathbf{t}) \doteq f(\mathbf{s}) \mid \theta} \qquad \frac{}{(\cdot) \doteq (\cdot) \mid (\cdot)} \qquad \frac{t \doteq s \mid \theta_1 \quad \mathbf{t}\theta_1 \doteq \mathbf{s}\theta_1 \mid \theta_2}{(t, \mathbf{t}) \doteq (s, \mathbf{s}) \mid \theta_1\theta_2}$$

Second, the cases for variables.

$$\frac{}{x \doteq x \mid (\cdot)} \qquad \frac{x \notin \mathrm{FV}(t)}{x \doteq t \mid (t/x)} \qquad \frac{t = f(\mathbf{t}), x \notin \mathrm{FV}(t)}{t \doteq x \mid (t/x)}$$

The condition that $t = f(\mathbf{t})$ in the last rule ensures that it does not overlap with the rule for $x \doteq t$. The condition that $x \notin \mathrm{FV}(t)$ is necessary because, for example, the two terms x and $f(x)$ do not have unifier: no matter what, the substitution $f(x)\theta$ will always have one more occurrence of f than $x\theta$ and hence the two cannot be equal.

The other situations where unification fails is an equation of the form $f(\mathbf{t}) = g(\mathbf{s})$ for $f \neq g$, and two sequences of terms of unequal length. The latter can happen if function symbols are overloaded at different arities, in which case failure of unification is the correct result.

6.5 Soundness

There are a number of properties we would like to investigate regarding the unification algorithm proposed in the previous section. The first is its soundness, that is, we would like to show that the substitution θ is indeed a unifier.

Theorem 6.2 *If $t \doteq s \mid \theta$ then $t\theta = s\theta$.*

Proof: We need to generalize this to cover the auxiliary unification judgment on term sequences.

(i) If $t \doteq s \mid \theta$ then $t\theta = s\theta$.

(ii) If $\mathbf{t} \doteq \mathbf{s} \mid \theta$ then $\mathbf{t}\theta = \mathbf{s}\theta$.

The proof proceeds by mutual induction on the structure of the deduction $\mathcal{D}$ of $t \doteq s$ and $\mathcal{E}$ of $\mathbf{t} \doteq \mathbf{s}$. This means that if one judgment appears as in the premiss of a rule for the other, we can apply the appropriate induction hypothesis.

In the proof below we will occasionally refer to *equality reasoning*, which refers to properties of equality in our mathematical language of discourse, not properties of the judgment $t \doteq s$. There are also some straightforward lemmas we do not bother to prove formally, such as $t(s/x) = t$ if $x \notin \mathrm{FV}(t)$.

Case: $\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{E}\\ \mathbf{t} \doteq \mathbf{s} \mid \theta\end{array}}{f(\mathbf{t}) \doteq f(\mathbf{s}) \mid \theta}$ where $t = f(\mathbf{t})$ and $s = f(\mathbf{s})$.

$\mathbf{t}\theta = \mathbf{s}\theta$	By i.h.(ii) on $\mathcal{E}$
$f(\mathbf{t})\theta = f(\mathbf{s})\theta$	By definition of substitution

Case: $\mathcal{D} = \dfrac{}{(\cdot) \doteq (\cdot) \mid (\cdot)}$ where $\mathbf{t} = \mathbf{s} = (\cdot)$ and $\theta = (\cdot)$.

$(\cdot)\theta = (\cdot)\theta$	By equality reasoning

Case: $\mathcal{E} = \dfrac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{E}_2\\ t_1 \doteq s_1 \mid \theta_1 & \mathbf{t}_2\theta_1 \doteq \mathbf{s}_2\theta_1 \mid \theta_2\end{array}}{(t_1, \mathbf{t}_2) \doteq (s_1, \mathbf{s}_2) \mid \theta_1\theta_2}$ where $\mathbf{t} = (t_1, \mathbf{t}_2)$ and $\mathbf{s} = (s_1, \mathbf{s}_2)$ and $\theta = \theta_1\theta_2$.

$t_1\theta_1 = s_1\theta_1$	By i.h.(i) on $\mathcal{D}_1$
$(t_1\theta_1)\theta_2 = (s_1\theta_1)\theta_2$	By equality reasoning
$t_1(\theta_1\theta_2) = s_1(\theta_2\theta_2)$	By substitution composition (Theorem 6.1)
$(\mathbf{t}_2\theta_1)\theta_2 = (\mathbf{s}_2\theta_1)\theta_2$	By i.h.(ii) on $\mathcal{E}_2$
$\mathbf{t}_2(\theta_1\theta_2) = \mathbf{s}_2(\theta_1\theta_2)$	By substitution composition
$(t_1, \mathbf{t}_2)(\theta_1\theta_2) = (s_1, \mathbf{s}_2)(\theta_1\theta_2)$	By defn. of substitution

Case: $\mathcal{D} = \dfrac{}{x \doteq x \mid (\cdot)}$ where $t = s = x$ and $\theta = (\cdot)$.

$x(\cdot) = x(\cdot)$	By equality reasoning

Case: $\mathcal{D} = \dfrac{x \notin \mathrm{FV}(s)}{x \doteq s \mid (s/x)}$ where $t = x$ and $\theta = (s/x)$.

$$\begin{array}{ll} x(s/x) = s & \text{By defn. of substitution} \\ = s(s/x) & \text{Since } x \notin \mathrm{FV}(s) \end{array}$$

Case: $\mathcal{D} = \dfrac{t = f(\mathbf{t}), x \notin \mathrm{FV}(t)}{t \doteq x \mid (t/x)}$ where $s = x$ and $\theta = (t/x)$.

$$\begin{array}{ll} t(t/x) = t & \text{Since } x \notin \mathrm{FV}(t) \\ = x(t/x) & \text{By defn. of substitution} \end{array}$$

□

6.6 Completeness

Completness of the algorithm states that if s and t have a unifer then there exists a most general one according to the algorithm. We then also need to observe that the unification judgment is deterministic to see that, if interpreted as an algorithm, it will always find a most general unifier if one exists.

Theorem 6.3 *If $t\sigma = s\sigma$ then $t \doteq s \mid \theta$ and $\sigma = \theta\sigma'$ for some θ and σ'.*

Proof: As in the soundness proof, we generalize to address sequences.

(i) If $t\sigma = s\sigma$ then $t \doteq s \mid \theta$ and $\sigma = \theta\sigma'$.

(ii) If $\mathbf{t}\sigma = \mathbf{s}\sigma$ then $\mathbf{t} \doteq \mathbf{s} \mid \theta$ and $\sigma = \theta\sigma'$.

The proof proceeds by mutual induction on the structure of $t\sigma$ and $\mathbf{t}\sigma$. We proceed by distinguishing cases for t and s, as well as $\mathbf{t}$ and $\mathbf{s}$. This structure of argument is a bit unusual: mostly, we distinguish cases of the subject of our induction, be it a deduction or a syntactic object. In the situation here it is easy to make a mistake and incorrectly attempt to apply the induction hypothesis, so you should carefully examine all appeals to the induction hypothesis below to make sure you understand why they are correct.

Case: $t = f(\mathbf{t})$. In this case we distinguish subcases for s.

Subcase: $s = f(\mathbf{s})$.

$f(\mathbf{t})\sigma = f(\mathbf{s})\sigma$	Assumption
$\mathbf{t}\sigma = \mathbf{s}\sigma$	By defn. of substitution
$\mathbf{t} \doteq \mathbf{s} \mid \theta$ and $\sigma = \theta\sigma'$ for some θ and σ'	By i.h.(ii) on $\mathbf{t}\sigma$
$f(\mathbf{t}) \doteq f(\mathbf{s}) \mid \theta$	By rule

Subcase: $s = g(\mathbf{s})$ for $f \neq g$. This subcase is impossible:

$f(\mathbf{t})\sigma = g(\mathbf{s})\sigma$	Assumption
Contradiction	By defn. of substitution

Subcase: $s = x$.

$f(\mathbf{t})\sigma = x\sigma$	Assumption
$\sigma = (f(\mathbf{t})\sigma/x, \sigma')$ for some σ'	By defn. of subst. and reordering
$x \notin \mathrm{FV}(f(\mathbf{t}))$	Otherwise $f(\mathbf{t})\sigma \neq x\sigma$
$f(\mathbf{t}) = x \mid (f(\mathbf{t})/x)$ so we let $\theta = (f(\mathbf{t})/x)$	By rule
$\sigma = (f(\mathbf{t})\sigma/x, \sigma')$	See above
$= (f(\mathbf{t})\sigma'/x, \sigma')$	Since $x \notin \mathrm{FV}(f(\mathbf{t}))$
$= (f(\mathbf{t})/x)\sigma'$	By defn. of substitution
$= \theta\sigma'$	Since $\theta = (f(\mathbf{t})/x)$

Case: $t = x$. In this case we also distinguish subcases for s and proceed symmetrically to the above.

Case: $\mathbf{t} = (\cdot)$. In this case we distinguish cases for $\mathbf{s}$.

Subcase: $\mathbf{s} = (\cdot)$.

$(\cdot) \doteq (\cdot) \mid (\cdot)$	By rule
$\sigma = (\cdot)\sigma$	By defn. of substitution

Subcase: $\mathbf{s} = (s_1, \mathbf{s}_2)$. This case is impossible:

$(\cdot)\sigma = (s_1, \mathbf{s}_2)\sigma$	Assumption
Contradiction	By definition of substitution

Case: $\mathbf{t} = (t_1, \mathbf{t}_2)$. Again, we distinguish two subcases.

Subcase: $\mathbf{s} = (\cdot)$. This case is impossible, like the symmetric case above.

Subcase: $\mathbf{s} = (s_1, \mathbf{s}_2)$.

$(t_1, \mathbf{t}_2)\sigma = (s_1, \mathbf{s}_2)\sigma$	Assumption
$t_1\sigma = s_1\sigma$ and $\mathbf{t}_2\sigma = \mathbf{s}_2\sigma$	By defn. of substitution

$t_1 \doteq s_1 \mid \theta_1$ and
$\sigma = \theta_1\sigma_1'$ for some θ_1 and σ_1' By i.h.(i) on $t_1\sigma$
$\mathbf{t}_2(\theta_1\sigma_1') = \mathbf{s}_2(\theta_1\sigma_1')$ By equality reasoning
$(\mathbf{t}_2\theta_1)\sigma_1' = (\mathbf{s}_2\theta_1)\sigma_1'$ By subst. composition (Theorem 6.1)
$\mathbf{t}_2\theta_1 \doteq \mathbf{s}_2\theta_1 \mid \theta_2$ and
$\sigma_1' = \theta_2\sigma_2'$ for some θ_2 and σ_2' By i.h.(ii) on $\mathbf{t}_2\sigma$ $(= (\mathbf{t}_2\theta_1)\sigma_1')$
$(t_1, \mathbf{t}_2) \doteq (s_1, \mathbf{s}_2) \mid \theta_1\theta_2$ By rule
$\sigma = \theta_1\sigma_1' = \theta_1(\theta_2\sigma_2')$ By equality reasoning
$= (\theta_1\theta_2)\sigma_2'$ By subsitution composition (Theorem 6.1)

□

It is worth observing that a proof by mutual induction on the structure of t and $\mathbf{t}$ would fail here (see Exercise 6.2).

An alternative way we can state the first induction hypothesis is:

> For all r, s, t, and σ such that $r = t\sigma = s\sigma$, there exists a θ and a σ' such that $t \doteq s \mid \theta$ and $\sigma = \theta\sigma'$.

The the proof is by induction on the structure of r, although the case we distinguish still concern the structure of s and t.

6.7 Termination

From the completeness argument in the previous section we can see that if given t and s the deduction of $t \doteq s \mid \theta$ is bounded by the structure of the common instance $r = t\theta = s\theta$. Since the rules furthermore have no non-determinism and the occurs-checks in the variable/term and term/variable cases also just traverse subterms of r, it means a unifier (if it exists) can be found in time proportional to the size of r.

Unfortunately, this means that this unification algorithm is exponential in the size of t and s. For example, the only unifier for

$$g(x_0, x_1, x_2, \ldots, x_n) \doteq g(f(x_1, x_1), f(x_2, x_2), f(x_3, x_3), \ldots a)$$

has 2^n occurrences of a.

Nevertheless, it is this exponential algorithm with a small, but significant modification that is used in Prolog implementations. This modification (which make Prolog unsound from the logical perspective!) is to omit the check $x \notin \mathrm{FV}(t)$ in the variable/term and term/variable cases and construct a circular term. This means that the variable/term case in unification

is constant time, because in an implementation we just change a pointer associated with the variable to point to the term. This is of crucial importance, since unification in Prolog models parameter-passing from other languages (thinking of the predicate as a procedure), and it is not acceptable to take time proportional to the size of the argument to invoke a procedure.

This observation notwithstanding, the worst-case complexity of the algorithm in Prolog is still exponential in the size of the input terms, but it is linear in the size of the result of unification. The latter fact appears to be what rescues this algorithm in practice, together with its straightforward behavior which is important for Prolog programmers.

All of this does not tell us what happens if we pass terms to our unification algorithm that do *not* have a unifier. It is not even obvious that the given rules terminate in that case (see Exercise 6.3). Fortunately, in practice most non-unifiable terms result in a clash between function symbols rather quickly.

6.8 Historical Notes

Unification was originally developed by Robinson [7] together with resolution as a proof search principle. Both of these critically influenced the early designs of Prolog, the first logic programming language. Similar computations were described before, but not studied in their own right (see [1] for more on the history of unification).

It is possible to improve the complexity of unification to linear in the size of the input terms if a different representation for the terms and substitutions is chosen, such as a set of multi-equations [4, 5] or dag structures with parent pointers [6]. These and similar algorithms are important in some applications [3], although in logic programming and general theorem proving, minor variants of Robinson's original algorithm are prevalent.

Most modern versions of Prolog support sound unification, either as a separate predicate `unify_with_occurs_check/2` or even as an optional part of the basic execution mechanism[1]. Given advanced compilation technology, I have been quoted figures of 10% to 15% overhead for using sound unification, but I have not found a definitive study confirming this. We will return to the necessary optimization in a later lecture.

Another way out is to declare that the bug is a feature, and Prolog is really a constraint programming language over rational trees, which requires a small modification of the unification algorithm to ensure termination in

[1] for example, in Amzi!Prolog

the presence of circular terms [2] but still avoids the occurs-check. The price to be paid is that the connection to the predicate calculus is lost, and that popular reasoning techniques such as induction are much more difficult to apply in the presence of infinite terms.

6.9 Exercises

Exercise 6.1 *Prove $\tau(\sigma\theta) = (\tau\sigma)\theta$ under the conditions stated in Theorem 6.1.*

Exercise 6.2 *Show precisely where and why the attempt to prove completeness of the rules for unification by mutual induction over the structure of t and $\mathbf{t}$ (instead of $t\sigma$ and $\mathbf{t}\sigma$) would fail.*

Exercise 6.3 *Show that the rules for unification terminate no matter whether given unifiable or non-unifiable terms t and s. Together with soundness, completeness, and determinacy of the rules this means that they constitute a decision procedure for finding a most general unifier if it exists.*

6.10 References

[1] Franz Baader and Wayne Snyder. Unification theory. In J.A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 8, pages 447–532. Elsevier and MIT Press, 2001.

[2] Joxan Jaffar. Efficient unification over infinite terms. *New Generation Computing*, 2(3):207–219, 1984.

[3] Kevin Knight. Unification: A multi-disciplinary survey. *ACM Computing Surveys*, 2(1):93–124, March 1989.

[4] Alberto Martelli and Ugo Montanari. Unification in linear time and space: A structured presentation. Internal Report B76-16, Istituto di Elaborazione delle Informazione, Consiglio Nazionale delle Ricerche, Pisa, Italy, July 1976.

[5] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, April 1982.

[6] M. S. Paterson and M. N. Wegman. Linear unification. *Journal of Computer and System Sciences*, 16(2):158–167, April 1978.

[7] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.