Lecture Notes on Unification

15-816: Linear Logic
Frank Pfenning

Lecture 19
April 2, 2012

When performing proof search in linear logic in the presence of quantifiers, we have to find ways to handle the two rules

$$\frac{\Psi \vdash M : \tau \quad \Psi \,;\, \Gamma \,;\, \Delta, A(M) \vdash C}{\Psi \,;\, \Gamma \,;\, \Delta, \forall x{:}\tau.\, A(x) \vdash C}\;\forall L \qquad \frac{\Psi \vdash M : \tau \quad \Psi \,;\, \Gamma \,;\, \Delta \vdash A(M)}{\Psi \,;\, \Gamma \,;\, \Delta \vdash \exists x{:}\tau.\, A(x)}\;\exists R$$

The difficulty here is to know which term M of type τ to use to instantiante the quantifier. This problem is not specific to linear logic but arises in all logics that permit quantification.

The general idea is simply to postpone the choice of M, just as we postponed splitting the resources in the multiplicative rules. M, in effect, becomes a *metavariable* or *logic variable* or *existential variable* whose value will be determined by constraints imposed upon it later on during proof search. Solving equational constraints imposed upon metavariables during proof search is called *unification*.

By the rule above we can see that the values of metavariables are constrained already by the first premise of both rules, $\Psi \vdash M : \tau$. For this lecture, we assume that this constraint will always be respected, but we do not explicitly discuss mechanisms to do so. Some care is needed: not only must the eventual value of X have the right type, but it must make sense in the right context Ψ. Relevant techniques are discussed in the literature [Mil92], including extensions to rich type theories where terms include binders [Mil91, NPP08].

1 Simple Terms

We assume there is a single base type ι, all variables have type ι, and function symbols all have type $\iota \times \cdots \times \iota \to \iota$, indicating their arity. It is convenient to stipulate that predicates have type $\iota \times \cdots \times \iota \to o$, where o is the type of propositions, used only in this special role. Corresponding terms have the follow form:

$$\text{Simple Terms} \quad t \quad ::= \quad x \mid f\,t \mid (t_1, t_2) \mid ()$$

where x are metavariables and f are term constructors, that is, predicate or function symbols. We write $f()$ as f, thinking of constants as nullary functions, f with a single argument just as $f(t)$, and associate pairs to the right so that (t_1, t_2, t_3) stands for $(t_1, (t_2, t_3))$. Variables will always have type ι.

2 Equality

Before we address solving equations with variables, we would like to define *equality* between simple terms. Since there is no interesting equational theory, this can be trivially stated as $t \doteq t$. Such a definition does not give rise to an algorithm for solving equations with variables, so we look instead to a structural characterization of equality. First, a backward-chaining definition. We omit variables, since they are subject to instantiation.

$$\begin{array}{l}
\mathsf{eq}(f(s), f(t)) \multimapinv \mathsf{eq}(s, t). \\
\mathsf{eq}((s_1, s_2), (t_1, t_2)) \multimapinv \mathsf{eq}(s_1, s_2) \otimes \mathsf{eq}(t_1, t_2). \\
\mathsf{eq}((), ()).
\end{array}$$

Developing an algorithm for unification from this definition leads us to a presentation of Robinson's original algorithm [Rob65], which has exponential complexity but is practical in many cases.

Here, we pursue a different path that leads us to a version of Huet's algorithm [Hue76] which actually originated from the problem of unifying higher-order terms. Knight [Kni89] has written a useful survey on various unification algorithms and their relationship.

In order to best understand Huet's algorithm we need a different presentation of equality. Fortunately, just rewriting it into a forward-chaining presentation does the trick! We *assume* that two terms are equal and signal a contradiction if that was wrong. If we are right, we just consume all assumptions and are left with the empty context. We assume that disequality between function symbols (and later variables) is primitive.

$$\begin{array}{l}
\mathsf{eq}(f(s), f(t)) \multimap \mathsf{eq}(s, t). \\
\mathsf{eq}(f(s), g(t)) \otimes f \neq g \multimap \mathbf{0}. \\
\mathsf{eq}((), ()) \multimap \mathbf{1}. \\
\mathsf{eq}((s_1, s_2), (t_1, t_2)) \multimap \mathsf{eq}(s_1, t_1) \otimes \mathsf{eq}(s_2, t_2).
\end{array}$$

This program conveniently assumes that when we assert $\mathsf{eq}(s, t)$, s and t are well-typed and have the same type. Otherwise, we would have to add further rules, such as

$$\mathsf{eq}((), (t_1, t_2)) \multimap \mathbf{0}.$$

which we reject as meaningless rather than contradictory.

3 Some Examples

Let's look at some simple unification examples to examine phenomena beyond equality.

The result of unification is usually presented as a *substitution* for the metavariables in the equations under consideration. For example,

$$\mathsf{f}(x, \mathsf{c}) \doteq \mathsf{f}(\mathsf{g}(\mathsf{c}), y)$$

has the solution $\theta = (\mathsf{g}(\mathsf{c})/x, \mathsf{c}/y)$, since

$$\mathsf{f}(x, \mathsf{c})[\theta] = \mathsf{f}(\mathsf{g}(\mathsf{c}), y)[\theta]$$

Generally, unification is defined this way: given two terms s and t, find a substitution θ such that $s[\theta] = t[\theta]$. As we will see, there are other means for presenting the answer than returning a substitution, but they must still be related to the above specification.

Sometimes, unifying substitutions are not unique. For example,

$$x \doteq y$$

has solutions

$$\begin{array}{rcl}
\theta_1 & = & y/x \\
\theta_2 & = & x/y \\
\theta_3 & = & z/x, z/y \\
\theta_4 & = & \mathsf{c}/x, \mathsf{c}/y \\
 & \ldots &
\end{array}$$

assuming that we have a constant c. Of these, there are several equivalent *most general* substitutions which make both terms equal but keep the result as general as possible. θ_1, θ_2 and θ_3 are most general in the sense that any other unifying substitution is an instance of any of them. θ_4 is not: since c is a constant, we cannot obtain θ_3 from it. And, indeed, it is an unnecessary commitment to fix both x and y to be a constant just because they are equal.

We will not develop this notion formally, because in our approach we can characterize most general solutions in a different way.

Before proposing algorithms, let's look at some further examples. Variables represent a form of communication between different parts of the term, because multiple occurrences of the same variable must be substituted by the same term. For example,

$$\mathsf{f}(x, x) \doteq \mathsf{f}(\mathsf{g}(\mathsf{c}), \mathsf{g}(\mathsf{d}))$$

must fail since $\mathsf{c} \neq \mathsf{d}$.

Another mode of failure arises from the so-called *occurs-check*:

$$x \doteq \mathsf{f}(x)$$

This equation cannot have a solution. Assume there is a unifying substitution θ. Then $x[\theta] = \mathsf{f}(x[\theta])$. But that's impossible, since the right-hand side has always one more function symbol than the left-hand side.

Sometimes this failure can be indirect, as in

$$x \doteq \mathsf{g}(y), y \doteq \mathsf{g}(x)$$

This can happen even with a single equation:

$$\mathsf{f}(x, y) \doteq \mathsf{f}(\mathsf{g}(y), \mathsf{g}(x))$$

which has no solution.

4 Constraint Simplification

We now try to turn the intuition from the example above into an algorithm for determining whether two terms are unifiable. We start with the forward chaining equality rules from before. Since we might need to perform transitive chain of inferences for situations such as

$$t_1 \doteq x, x \doteq t_2, t_3 \doteq x$$

that require t_1, t_2 and t_3 all to be equal we make all equations *persistent*. The interface to our algorithm will be:

1. Assume $!\mathsf{eq}(s,t)$ and saturate.
2. If we have derived a contradiction $\mathbf{0}$, s and t are not unifiable.
3. If we saturate without a contradiction, s and t are unifiable.

The saturated context represents the solution to the unification problem as discussed in Section 8. We reiterate the rules for equality that are relevant in the persistent setting.

$$\begin{array}{l} !\mathsf{eq}(f(s), f(t)) \multimap\, !\mathsf{eq}(s,t). \\ !\mathsf{eq}(f(s), g(t)) \otimes f \neq g \multimap \mathbf{0}. \\ !\mathsf{eq}((s_1,s_2),(t_1,t_2)) \multimap\, !\mathsf{eq}(s_1,t_1) \otimes\, !\mathsf{eq}(s_2,t_2). \end{array}$$

Next we add symmetry and transitivity.

$$\begin{array}{l} !\mathsf{eq}(s,t) \multimap\, !\mathsf{eq}(t,s) \\ !\mathsf{eq}(s,t) \otimes\, !\mathsf{eq}(t,r) \multimap\, !\mathsf{eq}(s,r) \end{array}$$

Reflexivity (encoded as $!\mathsf{eq}(s,s) \multimap \mathbf{1}$) would not be useful, since it does not change the state. But because of symmetry and transitivity, $!\mathsf{eq}(t,t)$ will be derivable for any subterm in the original equation (if they are not contradictory).

At this point we are almost finished. We can derive any structural inconsistency between the terms claimed to be unifiable. However, a single equation such as $x \doteq \mathsf{f}(x)$ would just saturate into four equations. We need to add the occurs-check. We do this with a new predicate $\mathsf{notin}(x,t)$.

$$!\mathsf{eq}(x, f(t)) \multimap\, !\mathsf{notin}(x,t)$$

This predicate now traverses the term, collecting other constraints regarding the occurs-check, or failing.

$$\begin{array}{l} !\mathsf{notin}(x, f(t)) \multimap\, !\mathsf{notin}(x,t) \\ !\mathsf{notin}(x,(t_1,t_2)) \multimap\, !\mathsf{notin}(x,t_1) \otimes\, !\mathsf{notin}(x,t_2) \\ !\mathsf{notin}(x,x) \multimap \mathbf{0} \\[1ex] !\mathsf{notin}(x,s) \otimes\, !\mathsf{eq}(s,t) \multimap\, !\mathsf{notin}(x,t) \\[1ex] !\mathsf{notin}(x,y) \otimes\, !\mathsf{notin}(y,z) \multimap\, !\mathsf{notin}(x,z) \end{array}$$

These rules are not minimal in various respects. For example, if we start with only a single equation and no notin constraint, the last rule might be redundant. And the second-to-last rule could be restricted to the case where s is a variable. The particular rules are selected for the simplicity of their correctness proof.

5 Termination

First, we note that the program is stratified in that the absence of presence of !notin cannot affect equality !eq. This means we can proceed in two phases: in the first we saturate equality, in the second we saturate the occurrence condition. A crude analysis, following Ganzinger and McAllester [**?**] yields a complexity bound of $O(n^3)$ where n is the size of the original equation. This is because we can only build $O(n^2)$ distinct equations $!\mathsf{eq}(s,t)$, since s and t must be subterm of the original equation, and the transitivity rule has therefore $O(n^3)$ so-called *prefix firings*. A more detailed analysis shows the algorithm to be implementable using a union-find data structure to maintain equivalence classes with an $O(n)$ occurs-check as a second phase, which yields a complexity of $O(n\mathrm{log}(\mathrm{n}))$ which can be improved further to $O(n\alpha(n))$, where $\alpha(n)$ is the inverse of the Ackermann function and for all practical purposes as small constant.

6 Preservation of Solutions

Next, we want to show that any of the forward-chaining rules preserve the set of unifying substitutions exactly. Let Γ be a persistent collection of equations $\mathsf{eq}(s,t)$ and $\mathsf{notin}(x,t)$. We say that a substitution for all the variables in Γ simultaneously satisfies all constraints in Γ

1. $s[\theta] = t[\theta]$ for any equation $\mathsf{eq}(s,t)$, and
2. $|x[\theta]| > |t[\theta]|$ for any constraint $\mathsf{notin}(x,t)$ where $|s|$ is the size of a term, counting up function symbols and variables.

We write $\theta \models \Gamma$. Initially we just have a single equation $\mathsf{eq}(s,t)$, so $\theta \models \mathsf{eq}(s,t)$ iff θ is a unifier for s and t.

We want to show that the exact set of solutions is preserved under all forward chaining steps.

Theorem 1 (Preservation of Unifiers) *For all substitutions θ and all forward-chaining transitions $\Gamma \to \Gamma'$, we have that $\theta \models \Gamma$ iff $\theta \models \Gamma'$.*

Proof: Since all propositions are persistent, any substitution satisfying Γ' will trivially satisfy Γ.

Now assume $\theta \models \Gamma$. It remains to show that $\theta \models P$ for any new constraints that are added as part of the transition. By the definition of equality, this is very easy to see for the following structural rules:

$$\begin{array}{l}
!\mathsf{eq}(f(s), f(t)) \multimap !\mathsf{eq}(s,t). \\
!\mathsf{eq}(f(s), g(t)) \otimes f \neq g \multimap \mathbf{0}. \\
!\mathsf{eq}((s_1,s_2),(t_1,t_2)) \multimap !\mathsf{eq}(s_1,t_1) \otimes !\mathsf{eq}(s_2,t_2). \\
!\mathsf{eq}(s,t) \multimap !\mathsf{eq}(t,s) \\
!\mathsf{eq}(s,t) \otimes !\mathsf{eq}(t,r) \multimap !\mathsf{eq}(s,r)
\end{array}$$

Note that in the second rule, the assumption $\theta \models \Gamma$ is contradictory, so the conclusion follows trivially.

Next, we consider

$$!\mathsf{eq}(x, f(t)) \multimap !\mathsf{notin}(x,t)$$

By assumption $x[\theta] = f(t)[\theta]$, so $|x[\theta]| = |f(t[\theta])| = |t[\theta]| + 1$, so the $|x[\theta]| > |t[\theta]|$.

In the next two rules

$$\begin{array}{l}
!\mathsf{notin}(x, f(t)) \multimap !\mathsf{notin}(x,t) \\
!\mathsf{notin}(x,(t_1,t_2)) \multimap !\mathsf{notin}(x,t_1) \otimes !\mathsf{notin}(x,t_2)
\end{array}$$

the property is trivially preserved, since the second argument of $\mathsf{notin}(x,t)$ in the conclusion is subterm of the one in the premises.

The assumption $\theta \models \mathsf{notin}(x,x)$ is contradictory, since $|x[\theta]| = |x[\theta]|$, so

$$!\mathsf{notin}(x,x) \multimap \mathbf{0}$$

trivially preserves all solutions (of which there are none).

Finally, for the rules

$$\begin{array}{l}
!\mathsf{notin}(x,s) \otimes !\mathsf{eq}(s,t) \multimap !\mathsf{notin}(x,t) \\
!\mathsf{notin}(x,y) \otimes !\mathsf{notin}(y,z) \multimap !\mathsf{notin}(x,z)
\end{array}$$

we exploit that $s[\theta] = t[\theta]$ because $\theta \models \mathsf{eq}(s,t)$. So $|x[\theta]| > |s[\theta]| = |t[\theta]|$. A similar argument applies to the second rule. □

7 Unsatisfiability of Inconsistent States

So far we know that forward-chaining will always saturate, and that it preserves the set of unifying substitutions exactly. We say that Γ *is inconsistent*, if $(\Gamma\ ;\ \cdot) \to (\Gamma\ ;\ \mathbf{0})$. Taking some liberty with notation, we write $\Gamma \to \mathbf{0}$.

Theorem 2 (Unsatisfiability of Inconsistent States) *If Γ is inconsistent (that is, $\Gamma \to \mathbf{0}$) then Γ is unsatisfiable (that is, there is no θ such that $\theta \models \Gamma$).*

Proof: Here are the two rules that can produce a contradiction:

$$\begin{array}{l} !\mathsf{eq}(f(s), g(t)) \otimes f \neq g \multimap \mathbf{0}. \\ !\mathsf{notin}(x, x) \multimap \mathbf{0}. \end{array}$$

But there are no θ, s, and t such that $f(s)[\theta] = g(t)[\theta]$ when $f \neq g$. Similarly, there are no θ and x such that $|x[\theta]| > |x[\theta]|$. □

8 Satisfiability of Consistent Saturated States

The last piece of the puzzle is more difficult. We need to show that if we reach saturation, and there is no contradiction, then the set of equations is simultaneously unifiable. Clearly, this is necessary. For example, if we had forgotten the occurs-check altogether, we would still have termination and preservation, but there are final states for which there are not unifying substitutions. In essence, we need to devise a method to read off a unifying substitution from a saturated state. By preservation, we know this will be a correct answer to the original equation.

A first naive attempt we be to take an arbitrary equation in the saturated state, say, $\mathsf{eq}(x, t)$ and postulate the substitution t/x. This won't work, however, because in equations such as $\mathsf{eq}(x, f(y)), \mathsf{eq}(y, c)$ the substitution $f(y)/x, c/y$ does not actually unify the first equation! It would end up as $\mathsf{eq}(f(y), c)$. This shows that we first have to build up a substitution θ_1 for y, here c/y, then then set $\theta_2 = (f(y)[\theta_1]/x, \theta_1) = (f(c)/x, c/y)$. But why can we always find some order in which to construct the final substitution? Here is where the occurs-check comes in crucially.

We say that a substitution on a set of variables X is a *partial solution for Γ on X* if for any $x \in X$, we have $x[\theta] = t[\theta]$ for any equation $\mathsf{eq}(x, t)$ or $\mathsf{eq}(t, x)$, and $|x[\theta]| > |t[\theta]|$ for any constraint $\mathsf{notin}(x, t)$. Clearly, the empty substitution is a partial solution on the empty set of variables, and a partial solution on *all* variables is a unifier.

Assume we already have a substitution θ_k that is a partial solution for Γ on a set of variables X_k. If it covers all variables already, we are done and have a unifier. Otherwise we construct a substitution θ_{k+1} in the following way.

The notin relation is irreflexive and transitive in a saturated state that's not contradictory, because we have explicit rules to that effect. We can therefore pick a variable z such that $\mathsf{notin}(z, x)$ implies that $x \in X_k$. Now define $Y_k = \{y \mid \mathsf{eq}(z, y) \in \Gamma\}$. There exists a term t_0 such that for all non-variable s and equations $\mathsf{eq}(y_i, s)$ we have $s[\theta_k] = t_0$. To show this we use an auxiliary induction on the number of such equations. If there are zero, we pick a fresh variable w and set $t_0 = w$. If there is only one such equation, pick $t_0 = s$. If there are two such equations with s and s' we can pick either one, because $s[\theta_k] = s'[\theta_k]$. We obtain this by constructing an explicit proof of equality of these terms by using the fact that the state is saturated and that θ_k is a partial solution on X_k.

Now define the substitution θ_{k+1} as the extension of θ_k with t_0/y for all $y \in Y_k$. Then θ_{k+1} is a partial solution for Γ on $X_{k+1} = X_k \cup Y_k$.

Since we pick up at least one variable on each step, this process must finish after a finite number of steps, with the final X_k being a unifier, if we start $X_0 = \{\,\}$ and θ_0 the empty substitution.

Theorem 3 (Satisfiability of Consistent Saturated States) *If Γ is saturated and consistent (that is, does not generate $\mathbf{0}$), then there exists a substitution θ on the variables in Γ such that $\theta \models \Gamma$.*

Proof: Following the proof sketch above, by an induction on the number of variables in the saturated state. □

9 Example Revisited

Here is a standard example which illustrates that the complexity of unification, if we are expected to fully write out the unifying substitution as a term, is exponential.

$$\begin{array}{l} x_0 \doteq \mathsf{f}(x_1, x_1), \\ x_1 \doteq \mathsf{f}(x_2, x_2), \\ \ldots, \\ x_{n-1} \doteq \mathsf{f}(x_n, x_n) \end{array}$$

In the unifying substitution term for x_0, there are 2^{n-1} occurrences of f. However, when seen as a directed acyclic graph, it is rather compact and linear in n. The saturating forward-chaining algorithm captures the sharing that is necessary to obtain a polynomial bound.

If we express each equation as $\mathsf{eq}(x_i, \mathsf{f}(x_{i+1}, x_{i+1}))$, the set will saturate rather quickly. We obtain its symmetric form and also $\mathsf{eq}(x_i, x_i)$ by transitivity, as well as $\mathsf{eq}(\mathsf{f}(x_{i+1}, x_{i+1}), \mathsf{f}(x_{i+1}, x_{i+1}))$. There are also some uninteresting intermediate equations, such as $\mathsf{eq}((x_{i+1}, x_{i+1}), (x_{i+1}, x_{i+1}))$.

Next we start phase two, generating and saturating the occurs-check constraints. At first, we generate (essentially) $\mathsf{notin}(x_i, x_{i+1})$. Using transitivity, we obtain $\mathsf{notin}(x_i, x_j)$ whenever $0 \leq i < j \leq n$, plus a few uninteresting ones like $\mathsf{notin}(x_i, \mathsf{f}(x_{j+1}, x_{j+1}))$ for $i < j$. This shows unifiability in polynomial time (based on the results by Ganzinger and McAllester), even if the unifier has exponential size if written out.

10 Incrementality

During proof search, every focusing phase will generate a set of equations that match clause heads against atomic propositions in the contexts. Some care must be taken to make sure appropriately fresh variables are generated when universal ($\forall L$) or existential ($\exists R$) quantifiers are traversed, but otherwise we can essentially just add any new equations to the context and saturate immediately. If the equations are inconsistent, we fail and backtrack. If they are consistent, we keep the context in its saturated state and continue, just adding further equations and re-saturating each time. This guarantees that we are never going down a branch that is impossible, because saturation of the equations only checks unifiability without any overcommitment.

This kind of structure generalizes smoothly when the equations are subject to an equational theory. The resource semantics presented in Lecture 23 provides an example of this kind.

Exercises

Exercise 1 (Optimizations) The rules we give generate more equations than necessary to guarantee unifiability for a saturated state. Consider some optimization to generate fewer persistent constraints $\mathsf{eq}(s,t)$ and $\mathsf{notin}(x,s)$, or have fewer applicable rules even if the final set of constraints is the same.

Argue informally for the correctness of your optimizations.

Exercise 2 (Satisfiability of Saturated State) A crucial step in the proof of satisfiability for a consistent saturated state (Theorem 3) is the following:

> *If there are two such equations with s and s' we can pick either one, because $s[\theta_k] = s'[\theta_k]$. We obtain this by constructing an explicit proof of equality of these terms by using the fact that the state is saturated and that θ_k is a partial solution on X_k.*

Carry out this argument rigorously. If you need additional invariants for the state or the algorithm to recover a unifier from the saturated state, please make them explicit.

References

[Hue76] Gérard Huet. *Résolution d'équations dans des langages d'ordre* $1, 2, \ldots, \omega$. PhD thesis, Université Paris VII, September 1976.

[Kni89] Kevin Knight. Unification: A multi-disciplinary survey. *ACM Computing Surveys*, 2(1):93–124, March 1989.

[Mil91] Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.

[Mil92] Dale Miller. Unification under a mixed prefix. *Journal of Symbolic Computation*, 14:321–358, 1992.

[NPP08] Aleksandar Nanevski, Frank Pfenning, and Brigitte Pientka. Contextual modal type theory. *Transactions on Computational Logic*, 9(3), 2008.

[Rob65] J. A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.