15-819K: Logic Programming

Lecture 7

Lifting

Frank Pfenning

September 19, 2006

Lifting is the name for turning a search calculus with ground judgments into one employing free variables. Unification might be viewed as the result of lifting a ground equality judgment, but we never explicitly introduced such a judgment. In this lecture our goal is to lift previously given operational semantics judgments for logic programming to permit free variables and prove their soundness and completeness. Unification and the judgment to compute most general unifiers are the central tools.

7.1 Explicating Rule Application

When explicating the left-to-right subgoal selection order in the operational semantics, we needed to introduce conjunction and truth in order to expose these choices explicitly. When explicating the first-to-last strategy of clause selection and backtracking we need to introduce disjunction and falsehood. It should therefore come as no surprise that in order to explicit the details of rule application and unification we need some further logical connectives. These include at least universal quantification and implication. However, we will for the moment postpone their formal treatment in order to concentrate on the integration of unification from the previous lecture into the operational semantics.

An inference rule

$$\frac{B\ true}{P\ true}$$

is modeled logically as the proposition

$$\forall x_1 \ldots \forall x_n.\, B \supset P$$

where $\{x_1, \ldots, x_n\}$ is the set of free variables in the rule. A logic program is a collection of such propositions. According to the meaning of the universal quantifier, we can use such a proposition by instantiating the universally quantified variables with arbitrary terms. If we denote this substitution by τ with $\mathrm{dom}(\tau) = \{x_1, \ldots, x_n\}$ and $\mathrm{cod}(\tau) = \emptyset$, then instantiating the quantifiers yields $B\tau \supset P\tau\ \mathit{true}$. We can use this implication to conclude $P\tau\ \mathit{true}$ if we have a proof of $B\tau\ \mathit{true}$. Require the co-domain of τ to be empty means that the substitution is ground: there are no variables in its substitution terms. This is to correctly represent the convention that an inference rule stands for all of its ground instances.

In order to match Prolog syntax more closely, we often write $P \leftarrow B$ for $B \supset P$. Moreover, we abbreviate a whole sequence of quantifiers as $\forall \mathbf{x}.\, A$, use $\mathbf{x}$ for a set of variables.

Previously, for every rule

$$\frac{B\ \mathit{true}}{P\ \mathit{true}}$$

we add a rule

$$\frac{B\ /\ S}{P\ /\ S}$$

to the operational semantics judgment $A\ /\ S$.

Now we want to replace all these rules by a single rule. This means we have to make the program explicit as a collection Γ of propositions, representing the rules as propositions. We always assume that all members of Γ are closed, that is, $\mathrm{FV}(A) = \emptyset$ for all $A \in \Gamma$. We write this judgment as

$$\Gamma \vdash A\ /\ S$$

which means that A under stack S follows from program Γ. Rule application then has the form

$$\frac{\forall \mathbf{x}.\, P' \leftarrow B' \in \Gamma \qquad \begin{array}{l}\mathrm{dom}(\tau) = \mathbf{x} \\ \mathrm{cod}(\tau) = \emptyset \\ P'\tau = P\end{array} \qquad \Gamma \vdash B'\tau\ /\ S}{\Gamma \vdash P\ /\ S}$$

All the other rules just carry the program Γ along, since it remains fixed. We will therefore suppress it when writing the judgment.

7.2 Free Variable Deduction

We use the calculus with an explicit goal stack as the starting point. We recall the rules, omitting $\Gamma \vdash$ as promised.

$$\frac{A \mathbin{/} B \wedge S}{A \wedge B \mathbin{/} S} \qquad \frac{B \mathbin{/} S}{\top \mathbin{/} B \wedge S} \qquad \frac{}{\top \mathbin{/} \top}$$

$$\frac{\forall \mathbf{x}.\, P' \leftarrow B' \in \Gamma \quad \begin{array}{l}\mathrm{dom}(\tau) = \mathbf{x} \\ \mathrm{cod}(\tau) = \emptyset \\ P'\tau = P\end{array} \qquad B'\tau \mathbin{/} S}{P \mathbin{/} S}$$

In the free variable form we return a substitution θ, which is reminiscent of our formulation of unification. The rough idea, formalized in the next section, is that if $A \mathbin{/} S \mid \theta$ then $A\theta \mathbin{/} S\theta$. The first three rules are easily transformed.

$$\frac{A \mathbin{/} B \wedge S \mid \theta}{A \wedge B \mathbin{/} S \mid \theta} \qquad \frac{B \mathbin{/} S \mid \theta}{\top \mathbin{/} B \wedge S \mid \theta} \qquad \frac{}{\top \mathbin{/} \top \mid (\cdot)}$$

The rule for atomic goals requires a bit of thought. In order to avoid a conflict between the names of the variables in the rule, and the names of variables in the goal, we apply a so-called *renaming substitution*. A renaming substitution ρ has the form $y_1/x_1, \ldots, y_n/x_n$ where all the x_i and y_i are distinct. We will always use ρ to denote renaming substitutions. In Prolog terminology we say that we *copy the clause*, instantiating its variables with fresh variables.

$$\frac{\forall \mathbf{x}.\, P' \leftarrow B' \in \Gamma \quad \begin{array}{l}\mathrm{dom}(\rho) = \mathbf{x} \\ \mathrm{cod}(\rho) \cap \mathrm{FV}(P/S) = \emptyset \\ P'\rho \doteq P \mid \theta_1\end{array} \qquad B'\rho\theta_1 \mathbin{/} S\theta_1 \mid \theta_2}{P \mathbin{/} S \mid \theta_1\theta_2}$$

7.3 Soundness

The soundness of the lifted calculus is a relatively straightforward property. We would like to say that if $P \mathbin{/} S \mid \theta$ then $P\theta \mathbin{/} S\theta$. However, the co-domain of θ may contain free variables, so the latter may not be well defined. We therefore have to admit an arbitrary grounding substitution σ' to be composed with θ. In the proof, we also need to extend σ' to account

for additional variables. We write $\sigma'' \supseteq \sigma'$ for an extension of σ' with some additional pairs t/x for ground terms t.

Theorem 7.1 *If $A \;/\; S \mid \theta$ then for any substitution σ' with $\mathrm{FV}((A/S)\theta\sigma') = \emptyset$ we have $A\theta\sigma' \;/\; S\theta\sigma'$.*

Proof: By induction on the structure of $\mathcal{D}$ of $P \;/\; S \mid \theta$.

Cases: The first three rule for conjunction and truth are straightforward and omitted here.

Case: $\mathcal{D} = \dfrac{\forall \mathbf{x}.\, P' \leftarrow B' \in \Gamma \qquad \begin{array}{l}\mathrm{dom}(\rho) = \mathbf{x} \\ \mathrm{cod}(\rho) \cap \mathrm{FV}(P/S) = \emptyset \\ P'\rho \doteq P \mid \theta_1\end{array} \qquad \begin{array}{c}\mathcal{D}' \\ B'\rho\theta_1 \;/\; S\theta_1 \mid \theta_2\end{array}}{P \;/\; S \mid \theta_1\theta_2}$

where $A = P$ and $\theta = \theta_1\theta_2$.

$\mathrm{FV}(P(\theta_1\theta_2)\sigma') = \mathrm{FV}(S(\theta_1\theta_2)\sigma') = \emptyset$	Assumption
Choose $\sigma'' \supseteq \sigma'$ such that $\mathrm{FV}((B'\rho\theta_1)\theta_2\sigma'') = \emptyset$	
$(B'\rho\theta_1)\theta_2\sigma'' \;/\; (S\theta_1)\theta_2\sigma''$	By i.h. on $\mathcal{D}'$
$B'(\rho\theta_1\theta_2\sigma'') \;/\; S\theta_1\theta_2\sigma''$	By assoc. of composition
$P'\rho\theta_1 = P\theta_1$	By soundness of unification
$P'\rho\theta_1\theta_2\sigma'' = P\theta_1\theta_2\sigma''$	By equality reasoning
$P'(\rho\theta_1\theta_2\sigma'') = P\theta_1\theta_2\sigma''$	By assoc. of composition
$P\theta_1\theta_2\sigma'' \;/\; S\theta_1\theta_2\sigma''$	By rule (using $\tau = \rho\theta_1\theta_2\sigma''$)
$P(\theta_1\theta_2)\sigma'' \;/\; S(\theta_1\theta_2)\sigma''$	By assoc. of composition
$P(\theta_1\theta_2)\sigma' \;/\; S(\theta_1\theta_2)\sigma'$	Since $\sigma' \subseteq \sigma''$ and $\mathrm{FV}(P(\theta_1\theta_2)\sigma') = \mathrm{FV}(S(\theta_1\theta_2)\sigma') = \emptyset$

□

The fact that we allow an arbitrary grounding substitution in the statement of the soundness theorem is not just technical device. It means that if there are free variables left in the answer substitution θ, then any instance of θ is also a valid answer. For example, if we ask `append([1,2,3], Ys, Zs)` and obtain the answer `Zs = [1,2,3|Ys]` then just by substituting `[4,5]` for `Ys` we can conclude `append([1,2,3], [4,5], [1,2,3,4,5])` without any further search.

Unfortunately, in the presence of free variables built-in extra-logical Prolog predicates such as disequality, negation-as-failure, or cut destroy this property (in addition to other problems with soundness).

7.4 Completeness

Completness follows the blueprint in the completeness proof for unification. In the literature this is often called the *lifting lemma*, showing that if there is ground deduction of a judgment, there must be a more general free variable deduction.

The first try at a lifting lemma, in analogy with a similar completness property for unification, might be:

If $A\sigma$ / $S\sigma$ then A / $S \mid \theta$ and $\sigma = \theta\sigma'$ for some θ and σ'.

This does not quite work for a technical reason: during proof search additional variables are introduced which could appear in the domain of θ (and therefore in the domain of $\theta\sigma'$), while σ does not provide a substitution term for them.

We can overcome this inaccuracy by just postulating that additional term/variable pairs can be dropped, written as $\sigma \subseteq \theta\sigma'$. In the theorem and proof below we always assume that substitutions τ and σ, possibly subscripted or primed, are *ground substitutions*, that is, their co-domain is empty.

Theorem 7.2 *If $A\sigma$ / $S\sigma$ for ground $A\sigma$, $S\sigma$, and σ, then A / $S \mid \theta$ and $\sigma \subseteq \theta\sigma'$ for some θ and ground σ'.*

Proof: The proof is by induction on the structure of the given deduction of $A\sigma$ / $S\sigma$.

Cases: The cases for the three rule for conjunction and truth are straightforward and omitted here.

Case: $\mathcal{D} = \dfrac{\begin{array}{lll} & \mathrm{dom}(\tau) = \mathbf{x} & \\ & \mathrm{cod}(\tau) = \emptyset & \mathcal{D}' \\ \forall \mathbf{x}.\, P' \leftarrow B' \in \Gamma & P'\tau = P\sigma & B'\tau \;/\; S\sigma \end{array}}{P\sigma \;/\; S\sigma}$ where $A\sigma = P\sigma$.

$\tau = \rho\tau'$ for some renaming ρ and substitution τ'	
with $\mathrm{dom}(\rho) = \mathbf{x}$, $\mathrm{cod}(\rho) = \mathrm{dom}(\tau')$, and	
$\mathrm{dom}(\tau') \cap \mathrm{dom}(\sigma) = \emptyset$	Choosing fresh vars.
(τ', σ) a valid substitution	By disjoint domains
$(P'\rho)\tau' = (P'\rho)(\tau', \sigma)$	$\mathrm{dom}(\sigma) \cap \mathrm{FV}(P'\rho) = \emptyset$
$S\sigma = S(\tau', \sigma)$	$\mathrm{dom}(\tau') \cap \mathrm{FV}(S) = \emptyset$
$P\sigma = P(\tau', \sigma)$	$\mathrm{dom}(\tau') \cap \mathrm{FV}(P) = \emptyset$

$P'\tau = P\sigma$	Given premiss
$P'\rho(\tau', \sigma) = P(\tau', \sigma)$	By equality reasoning
$P'\rho \doteq P \mid \theta_1$ and	
$(\tau', \sigma) = \theta_1\sigma'_1$ for some θ_1 and σ'_1	By completness of unification
$B'\tau \;/\; S\sigma$	Given subderivation $\mathcal{D}'$
$B'\tau = B'\rho\tau'$	By equality reasoning
$= B'\rho(\tau', \sigma)$	$\mathrm{dom}(\sigma) \cap \mathrm{FV}(B'\rho) = \emptyset$
$= B'\rho(\theta_1\sigma'_1)$	By equality reasoning
$= (B'\rho\theta_1)\sigma'_1$	By assoc. of composition
$S\sigma = S(\tau', \sigma) = S(\theta_1\sigma'_1)$	By equality reasoning
$= (S\theta_1)\sigma'_1$	By assoc. of composition
$B'\rho\theta_1 \;/\; S\theta_1 \mid \theta_2$ and	
$\sigma'_1 \subseteq \theta_2\sigma'_2$ for some θ_2 and σ'_2	By i.h. on $\mathcal{D}'$
$P \;/\; S \mid \theta_1\theta_2$	By rule
$\sigma \subseteq (\tau', \sigma) = \theta_1\sigma'_1$	By equality reasoning
$\subseteq \theta_1(\theta_2\sigma'_2)$	$\mathrm{cod}(\theta_1\sigma'_1) = \emptyset$
$= (\theta_1\theta_2)\sigma'_2$	By assoc. of composition

□

7.5 Occurs-Check Revisited

Now that the semantics of proof search with free variables has been clarified, we return to the issue that Prolog omits the occurs-check as mentioned in the last lecture. Instead, it builds circular terms when encountering problems such as $X \doteq f(X)$.

To understand why, we reconsider the `append` program.

```
append(nil, Ys, Ys).
append([X|Xs], Ys, [X|Zs]) :- append(Xs, Ys, Zs).
```

In order to append two ground lists (for simplicity we assume of the same length), we would issue a query

```
?- append([x1, ..., xn], [y1, ..., yn], Zs).
```

The fact that the first clause does not apply is discovered in one or two steps, because nil clashes with cons. We then copy the second clause and unify `[X|Xs] = [`$x_1, \ldots, x_n$`]`. Assuming all the x_i are integers, this operation will still take $O(n)$ operations because of the occurs-check when unifying `Xs = [`$x_2, \ldots, x_n$`]`. Similarly, unifying `Y = [`$y_1, \ldots, y_n$`]` would

take $O(n)$ steps to perform the occurs-check. Finally the unification in the last argument `[X|Zs1] = Zs` just takes constant time.

Then the recursive call looks like

```
?- append([x2, ..., xn], [y1, ..., yn], Zs1).
```

which again takes $O(n)$ operations. Overall, we will recurse $O(n)$ times, performing $O(n)$ operations on each call, giving us a complexity of $O(n^2)$. Obviously, this is unacceptable for a simple operations such as appending two lists, which should be $O(n)$.

We can see that the complexity of this implementation is almost entirely due to the occurs-check. If we do not perform it, then a query such as the one in our example will be $O(n)$.

However, I feel the price of soundness is too high. Fortunately, in practice, the occurs-check can often be eliminated in a sound interpreter or compiler. The first reason is that in the presence of mode information, we may know that some argument in the goal are ground, that is, contain no variables. In that case, the occurs-check is entirely superfluous.

Another reason is slightly more subtle. As we can see from the operational semantics, we copy the clause (and the clause head) by applying a renaming substitution. The variables in the renamed clause head, $P'\rho$ are entirely new and are not allowed do not appear in the goal P or the goal stack S. As a result, we can omit the occurs-check when we first encounter a variable in the clause head, because that variable couldn't possible occur in the goal.

However, we have to be careful for the second occurrence of a variable. Consider the goal

```
?- append([], [1|Xs], Xs).
```

Clearly, this should fail because there is no term t such that $[1|t] = t$. If we unify with the clause head `append([], Ys, Ys)`, the first unification `Ys = [1|Xs]` can be done without the occurs-check.

However, after the substitution for `Ys` has been carried out, the third argument to append yields the problem `[1|Xs] = Xs` which can only be solved correctly if the occurs-check is carried out.

If your version of Prolog does not have switch to enable sound unification to be used in its operational semantics, you can achieve the same effect using the built-in `unify_with_occurs_check/2`. For example, we can rewrite `append` to the following sound, but ugly program.

```
append(nil, Ys, Zs) :- unify_with_occurs_check(Ys, Zs).
append([X|Xs], Ys, [Z|Zs]) :-
        unify_with_occurs_check(X, Z),
        append(Xs, Ys, Zs).
```

7.6 Historical Notes

The basic idea of lifting to go from a ground deduction to one with free variables goes back to Robinson's seminal work on unification and resolution [3], albeit in the context of theorem proving rather than logic programming. The most influential early paper on the theory of logic programming is by Van Emden and Kowalski [2], who introduced several model-theoretic notions that I have replaced here by more flexible proof-theoretic definitions and relate them to each other. An important completeness result regarding the subgoal selection strategy was presented by Apt and Van Emden [1] which can be seen as an analogue to the completeness result we presented.

7.7 Exercises

Exercise 7.1 *Give ground and free variable forms of deduction in a formulation without an explicit goal stack, but with an explicit program, and show soundness and completeness of the free variable version.*

Exercise 7.2 *Fill in the missing cases in the soundness proof for free variable deduction.*

Exercise 7.3 *Give three concrete counterexamples showing that the substitution property for free variables in an answer substitution fails in the presence of disequality on non-ground goals, negation-as-failure, and cut.*

Exercise 7.4 *Fill in the missing cases in the completeness proof for free variable deduction.*

Exercise 7.5 *Analyze the complexity of appending two ground lists of integers of length n and k given the optimization that the first occurrence of a variable in a clause head does not require an occurs-check. Then analyze the complexity if it is known that the first two argument to* append *are ground. Which one is more efficient?*

7.8 References

[1] Krzysztof R. Apt and M. H. Van Emden. Contributions to the theory of logic programming. *Journal of the ACM*, 29(3):841–862, July 1982.

[2] M. H. Van Emden and R. A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the ACM*, 23(4):733–742, October 1976.

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