15-819K: Logic Programming

Lecture 5

Backtracking

Frank Pfenning

September 12, 2006

In this lecture we refine the operational semantics further to explicit represent backtracking. We prove this formulation to be sound. From earlier examples it should be clear that it can no longer be complete.

5.1 Disjunction and Falsehood

When our aim was to explicitly represent left-to-right subgoal selection, we introduced conjunction and truth as our first logical connectives. This allowed us to make the order explicit in the propositions we were interpreting.

In this lecture we would like to make the choice of rule explicit. For this purpose, it is convenient to introduce two new logical connectives: disjunction $A \lor B$ and falsehood $\bot$. They are easily defined by their introduction rules as usual.

$$\frac{A\ \mathit{true}}{A \lor B\ \mathit{true}}\ \lor I_1 \qquad \frac{B\ \mathit{true}}{A \lor B\ \mathit{true}}\ \lor I_2 \qquad \text{No } \bot I \text{ rule}$$

We can think of falsehood as a disjunction between zero alternatives; therefore, there are zero introduction rules.

5.2 Normal Forms for Programs

Sometimes it is expedient to give the semantics of programs assuming a kind of normal form. The presentation from the previous lecture would have been slightly simpler if we had presupposed an *explicit conjunction form* for programs where each inference rule has exactly one premiss. The transformation to achieve this normal form in the presence of conjunction and truth is simple: if the rule has multiple premisses, we just conjoin them to form one premiss. If a rule has no premisses, we insert $\top$ as a premiss. It is easy to prove that this transformation preserves meaning (see Exercise 5.1).

In the semantics presented as the judgment $A \,/\, S$, every step of search is deterministic, except for selection of the rule to apply, and how to instantiate schematic rules. We will not address the latter choice in today's lecture: assume that all goals are ground, and consider for the moment only programs so that ground goals have only ground subgoals.

A simple condition on the rules that avoids any choice when encountering atomic predicates is that every ground goal matches the head of *exactly one* rule. Then, when we have a goal P the rule to apply is uniquely determined. In order to achieve this, we have to transform our program into *explicit disjunction form*. In a later lecture we will see a systematic way to create this form as part of logic program compilation. For now we are content to leave this informal and just present programs in this form.

As an example, consider the usual member predicate.

$$\frac{}{\mathsf{member}(X, [X|Ys])} \qquad \frac{\mathsf{member}(X, Ys)}{\mathsf{member}(X, [Y|Ys])}$$

There are two reasons that this predicate is not in explicit disjunction form. The first is that there is no clause for $\mathsf{member}(t, [\,])$, violating the requirement that there be exactly one clause for each ground goal. The second is that for goals $\mathsf{member}(t, [t|ys])$, both clauses apply, again violating our requirement.

We rewrite this in several steps. First, we eliminate the double occurrence of X in the first clause in favor of an explicit equality test, $X \doteq Y$.

$$\frac{X \doteq Y}{\mathsf{member}(X, [Y|Ys])} \qquad \frac{\mathsf{member}(X, Ys)}{\mathsf{member}(X, [Y|Ys])}$$

Now that the two rules have the same conclusion, we can combine them into one, using disjunction.

$$\frac{X \doteq Y \vee \mathsf{member}(X, Ys)}{\mathsf{member}(X, [Y|Ys])}$$

Finally, we add a clause for the missing case, with a premiss of falsehood.

$$\frac{\bot}{\mathsf{member}(X, [\,])} \qquad \frac{X \doteq Y \vee \mathsf{member}(X, Ys)}{\mathsf{member}(X, [Y|Ys])}$$

This program is now in explicit disjunction form, under the presupposition that the second argument to member is a list.

In Prolog, the notation for $A \vee B$ is A ; B and the notation for $\bot$ is `fail`, so the program above becomes

```
member(X, []) :- fail.
member(X, [Y | Ys]) :- X = Y ; member(X, Ys).
```

The Prolog convention is to always put whitespace around the disjunction to distinguish it more clearly from conjunction.

5.3 Equality

Transforming a program into explicit disjunction form requires equality, as we have seen in the member example. We write $s \doteq t$ for the proposition that s and t are equal, with the following introduction rule.

$$\frac{}{t \doteq t\ \mathit{true}}\ {\doteq}I$$

We will also use $s \neq t$ to denote that two terms are different as a kind of judgment.

5.4 Explicit Backtracking

Assuming the program is in explicit disjunction form, the main choice concerns how to prove $A \vee B$ as a goal. The operational semantics of Prolog prescribes that we try to solve A first and only if that fails do we try B. This suggest that in addition to the goal stack S, we add another argument F to our search judgment which records further (untried) possibilities. We refer to F as the *failure continuation* because it records what to do when the current goal A fails. We write the new judgment as $A\ /\ S\ /\ F$ and read this as: *Either A under S or F*. More formally, we will establish soundness in the form that if $A\ /\ S\ /\ F$ then $(A \wedge S) \vee F\ \mathit{true}$. This statement can also be our guide in designing the rules for the judgment.

First, the rules for conjunction and truth. They do not change much, just carry along the failure continuation.

$$\frac{A\ /\ B \wedge S\ /\ F}{A \wedge B\ /\ S\ /\ F} \qquad \frac{B\ /\ S\ /\ F}{\top\ /\ (B \wedge S)\ /\ F} \qquad \frac{}{\top\ /\ \top\ /\ F}$$

The rules for atomic predicates P are also simple, because we assume the given rules for truth are in explicit disjunction form.

$$\frac{B \;/\; S \;/\; F}{P \;/\; S \;/\; F} \quad \text{for each rule} \quad \frac{B\ \mathit{true}}{P\ \mathit{true}}$$

Next, the rule for disjunction. In analogy with conjunction and truth, it is tempting to write the following two **incorrect** rules:

$$\frac{A \;/\; S \;/\; B \vee F}{A \vee B \;/\; S \;/\; F}\ \text{incorrect} \qquad \frac{B \;/\; S \;/\; F}{\bot \;/\; S \;/\; B \vee F}\ \text{incorrect}$$

Let's try to see the case in the soundness proof for the first rule. The soundness proof proceeds by induction on the structure of the deduction for $A \;/\; S \;/\; F$. For the first of the incorrect rules we would have to show that if $(A \wedge S) \vee (B \vee F)\ \mathit{true}$ then $((A \vee B) \wedge S) \vee F\ \mathit{true}$. By inversion on the rules for disjunction, we know that either $A \wedge S\ \mathit{true}$, or $B\ \mathit{true}$, or $F\ \mathit{true}$. In the middle case ($B\ \mathit{true}$), we do not have enough information to conclude that $((A \vee B) \wedge S) \vee F\ \mathit{true}$ and the proof fails (see Exercises 5.2 and 5.3).

The failure of the soundness proof also suggests the correct rule: we have to pair up the alternative B with the goal stack S and restore S it when we backtrack to consider B.

$$\frac{A \;/\; S \;/\; (B \wedge S) \vee F}{A \vee B \;/\; S \;/\; F} \qquad \frac{B \;/\; S \;/\; F}{\bot \;/\; S' \;/\; (B \wedge S) \vee F}$$

The goal stack S' in the second rule is discarded, because it applies to the goal $\bot$ which cannot succeed. Instead we restore the goal stack S saved with B.

It is worth noting explicitly that there is one case we did not cover

$$\text{no rule for } \bot \;/\; S \;/\; \bot$$

so that our overall goal fails if the current goal fails and there are no further alternatives.

Finally, we need two rules for equality, where we appeal to the equality ($s = t$) and disequality ($s \neq t$) in our languages of judgments.

$$\frac{s = t \quad \top \;/\; S \;/\; F}{s \doteq t \;/\; S \;/\; F} \qquad \frac{s \neq t \quad \bot \;/\; S \;/\; F}{s \doteq t \;/\; S \;/\; F}$$

In a Prolog implementation this will not lead to difficulties because we assume all goals are ground, so we can always tell if two terms are equal or not.

5.5 Soundness

The soundness proof goes along familiar patterns.

Theorem 5.1 *If $A \;/\; S \;/\; F$ then $(A \wedge S) \vee F$ true.*

Proof: By induction on the structure of $\mathcal{D}$ of $A \;/\; S \;/\; F$.

Case: $\mathcal{D} = \dfrac{\overset{\mathcal{D}_1}{A_1 \;/\; A_2 \wedge S \;/\; F}}{A_1 \wedge A_2 \;/\; S \;/\; F}$ where $A = A_1 \wedge A_2$.

$(A_1 \wedge (A_2 \wedge S)) \vee F$ *true*	By ind.hyp. on $\mathcal{D}_1$
$A_1 \wedge (A_2 \wedge S)$ *true* or F *true*	By inversion
$A_1 \wedge (A_2 \wedge S)$ *true*	First subcase
A_1 *true* and A_2 *true* and S *true*	By two inversions
$(A_1 \wedge A_2) \wedge S$ *true*	By two rule applications
$((A_1 \wedge A_2) \wedge S) \vee F$ *true*	By rule $(\vee I_1)$
F *true*	Second subcase
$((A_1 \wedge A_2) \wedge S) \vee F$ *true*	By rule $(\vee I_2)$

Case: $\mathcal{D} = \dfrac{\overset{\mathcal{D}_2}{A_2 \;/\; S_1 \;/\; F}}{\top \;/\; (A_2 \wedge S_1) \;/\; F}$ where $A = \top$ and $S = A_2 \wedge S_1$. Similar to the previous case.

Case: $\mathcal{D} = \dfrac{}{\top \;/\; \top \;/\; F}$ where $A = S = \top$.

$\top$ *true*	By rule $(\top I)$
$\top \wedge \top$ *true*	By rule $(\wedge I)$
$(\top \wedge \top) \vee F$ *true*	By rule $\vee I_1$

Case: $\mathcal{D} = \dfrac{\overset{\mathcal{D}'}{B \;/\; S \;/\; F}}{P \;/\; S \;/\; F}$ where $A = P$ and $\dfrac{B \textit{ true}}{P \textit{ true}}$.

$(B \wedge S) \vee F$ *true*	By ind.hyp. on $\mathcal{D}'$
B *true* and S *true*	First subcase, after inversion

$P\ \mathit{true}$	By rule
$(P \wedge S) \vee F$	By rules ($\wedge I$ and $\vee I_1$)
$F\ \mathit{true}$	Second subcase, after inversion
$(P \wedge S) \vee F$	By rule ($\vee I_2$)

Case: $\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}_1 \\ A_1 \;/\; S \;/\; (A_2 \wedge S) \vee F\end{array}}{A_1 \vee A_2 \;/\; S \;/\; F}$ where $A = A_1 \vee A_2$.

$(A_1 \wedge S) \vee ((A_2 \wedge S) \vee F)\ \mathit{true}$	By ind.hyp. on $\mathcal{D}_1$
$A_1\ \mathit{true}$ and $S\ \mathit{true}$	First subcase, after inversion
$A_1 \vee A_2\ \mathit{true}$	By rule ($\vee I_1$)
$((A_1 \vee A_2) \wedge S) \vee F\ \mathit{true}$	By rules ($\wedge I$ and $\vee I_1$)
$A_2\ \mathit{true}$ and $S\ \mathit{true}$	Second subcase, after inversion
$A_1 \vee A_2\ \mathit{true}$	By rule ($\vee I_2$)
$((A_1 \vee A_2) \wedge S) \vee F\ \mathit{true}$	By rules ($\wedge I$ and $\vee I_1$)
$F\ \mathit{true}$	Third subcase, after inversion
$((A_1 \vee A_2) \wedge S) \vee F\ \mathit{true}$	By rule ($\vee I_2$)

Case: $\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}_2 \\ A_2 \;/\; S_1 \;/\; F_0\end{array}}{\bot \;/\; S \;/\; (A_2 \wedge S_1) \vee F_0}$ where $A = \bot$ and $F = (A_2 \wedge S_1) \vee F_0$.

$(A_2 \wedge S_1) \vee F_0\ \mathit{true}$	By ind.hyp. on $\mathcal{D}_2$
$(\bot \wedge S) \vee ((A_2 \wedge S_1) \vee F_0)\ \mathit{true}$	By rule ($\vee I_2$)

Cases: The cases for equality are left to the reader (see Exercise 5.4).

□

5.6 Completeness

Of course, the given set of rules is *not* complete. For example, the single rule

$$\frac{\mathsf{diverge} \vee \top}{\mathsf{diverge}}$$

cannot be found by search, that is, there is no proof of

$$\mathsf{diverge} \;/\; \top \;/\; \bot$$

even though there is a simple proof that diverge $true$.

$$\dfrac{\dfrac{\dfrac{}{\top\ true}\ \top I}{\mathsf{diverge} \vee \top\ true}\ \vee I_2}{\mathsf{diverge}\ true}$$

However, it is interesting to consider that the set of rules is complete a weaker sense, namely that if $A \;/\; \top \;/\; \bot$ can be reduced to $\bot \;/\; S \;/\; \bot$ then there can be no proof of $A\ true$. We will not do this here (see Exercise 5.5). One way to approach this formally is to add another argument and use a four-place judgment

$$A \;/\; S \;/\; F \;/\; J$$

where J is either istrue (if a proof can be found) or isfalse (if the the attempt to find a proof fails finitely).

5.7 A Meta-Interpreter with Explicit Backtracking

Based on the idea at the end of the last section, we can turn the inference system into a Prolog program that can tell us explicitly whenever search succeeds or fails finitely.

Recall the important assumption that all goals are ground, and that the program is in explicit disjunction form.

```
solve(true, true, _, istrue).
solve(true, (A , S), F, J) :- solve(A, S, F, J).
solve((A , B), S, F, J) :- solve(A, (B, S), F, J).
solve(fail, _, fail, isfalse).
solve(fail, _, ((B , S) ; F), J) :- solve(B, S, F, J).
solve((A ; B), S, F, J) :- solve(A, S, ((B , S) ; F), J).
solve((X = Y), S, F, J) :- X = Y, solve(true, S, F, J).
solve((X = Y), S, F, J) :- X \= Y, solve(fail, S, F, J).
solve(P, S, F, J) :- clause(P, B), solve(B, S, F, J).

% top level interface
solve(A, J) :- solve(A, true, fail, J).
```

Given a program in explicit disjunction form, such as

```
member(X, []) :- fail.
member(X, [Y|Ys]) :- X = Y ; member(X, Ys).
```

we can now ask

```
?- solve(member(1, [2,3,4]), J).
J = isfalse;

?- solve(member(1, [1,2,1,4]), J).
J = istrue;
```

Each query will succeed only once since our meta-interpreter only searches for the first solution (see Exercise 5.6).

5.8 Abstract Machines

The meta-interpreter in which both subgoal selection and backtracking are explicit comes close to the specification of an abstract machine. In order to see how the inference rule can be seen as transition rules, we consider $A \;/\; S \;/\; F$ as the state of the machine. Each rule for this judgment has only one premiss, so each rule, when read from the conclusion to the premiss can be seen as a transition rule for an abstract machine.

Examining the rules we can see that for every state there is a unique state transition, with the following exceptions:

1. A state $\top \;/\; \top \;/\; F$ is final since there is no premiss for the maching rule. The computation finishes.

2. A state $\bot \;/\; S \;/\; \bot$ is final since there is no rule that applies. The computation fails.

3. A state $P \;/\; S \;/\; F$ applies a unique transition (by the requirement that there be a unique rule for every atomic goal P), although how to find that rule instance remains informal.

In the next lecture we make the process of rule application more precise, and we also admit goals with free variables as in Prolog.

5.9 Historical Notes

The explicit disjunction form is a pre-cursor of Clark's *iff-completion* of a program [2]. This idea is quite general, is useful in compilation of logic programs, and can be applied to much richer logic programming languages than Horn logic [1].

Early logic programming theory generally did not make backtracking explicit. Some references will appear in the next lecture, since some of the intrinsic interest arises from the notion of substitution and unification.

5.10 Exercises

Exercise 5.1 *Prove that if we replace every rule*

$$\frac{B_1\ \mathit{true} \quad \ldots \quad B_m\ \mathit{true}}{P\ \mathit{true}}$$

by

$$\frac{B_1 \wedge \cdots \wedge B_m\ \mathit{true}}{P\ \mathit{true}}$$

to achieve the explicit conjunction form, *the original and revised specification are strongly equivalent in the sense that there is a bijection between the proofs in the two formulations for each atomic proposition. Read the empty conjunction (*$m = 0$*) as* $\top$.

Exercise 5.2 *Give a counterexample to show that the failure in the soundness proof for the first incorrect disjunction rule is not just a failure in the proof: the system is actually unsound with respect to logical truth.*

Exercise 5.3 *Investigate if the second incorrect rule for disjunction*

$$\frac{B \mathrel{/} S \mathrel{/} F}{\bot \mathrel{/} S \mathrel{/} B \vee F}\ \mathit{incorrect}$$

also leads to a failure in the soundness proof. If so, give a counterexample. If not, discuss in what sense this rule is nonetheless incorrect.

Exercise 5.4 *Complete the soundness proof by giving the cases for equality.*

Exercise 5.5 *Investigate weaker notions of completeness for the backtracking semantics, as mentioned at the end of the section of completeness.*

Exercise 5.6 *Rewrite the meta-interpreter so it counts the number of proofs of a goal instead of returning just an indication of whether it is true or false. You may make all the same assumptions that* `solve(`A`,` S`,` F`,` J`)` *makes.*

Exercise 5.7 *Revisit the example from Lecture 3*

$$\frac{}{\mathsf{digit}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{z}))))))))))} \qquad \frac{\mathsf{digit}(\mathsf{s}(N))}{\mathsf{digit}(N)}$$

and prove more formally now that any query `?- digit(`n`)` *for* $n > 9$ *will not terminate. You should begin by rewriting the program into explicit disjunction form. Please be clear what you are doing the induction over (if you use induction), and explain in which way your theorem captures the statement above.*

Exercise 5.8 *If we were not concerned about space usage or efficiency, we could write a breadth-first interpreter instead of backtracking. Specify such an interpreter using the judgmental style and prove that it is sound and complete with respect to logical truth. Translate your interpreter into a Prolog program.*

5.11 References

[1] Iliano Cervesato. Proof-theoretic foundation of compilation in logic programming languages. In J. Jaffar, editor, *Proceedings of the Joint International Conference and Symposium on Logic Programming (JICSLP'98)*, pages 115–129, Manchester, England, June 1998. MIT Press.

[2] Keith L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, New York, 1978.