Logic Programming

Theory Lecture 4: Proof Search for Definite Clause Predicate Logic

Alex Simpson

School of Informatics

24th October 2013

Recap (Lecture 3): Definite clause predicate logic

A *definite clause* is a formula of one of the two shapes below

$$B \qquad \text{(a Prolog } \textit{fact } B\,.\,)$$

$$A_1 \wedge \cdots \wedge A_k \ \rightarrow B \qquad \text{(a Prolog } \textit{rule } B \,\texttt{:-}\, A_1, \ldots, A_k.)$$

where $A_1, \ldots, A_k, B$ are all *atomic formulas*.

A *logic program* is a list $F_1, \ldots, F_n$ of definite clauses

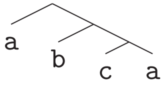
A *goal* is a list $G_1, \ldots, G_m$ of atomic formulas.

The job of the system is to ascertain whether the logical consequence below holds.

$$\forall \mathit{Vars}(F_1).\, F_1, \ldots, \forall \mathit{Vars}(F_n).\, F_n \models \exists \mathit{Vars}(G_1, \ldots, G_m).\, G_1 \wedge \cdots \wedge G_m$$

Example: Leaf-labelled binary trees

E.g.



`nd(lf(a),nd(lf(b),nd(lf(c),lf(a))))`

The `path` predicate :

```
path(a,nd(lf(a),nd(lf(b),nd(lf(c),lf(d)))), [l])
path(b, nd(lf(a),nd(lf(b),nd(lf(c),lf(d)))), [r,l])
path(c, nd(lf(a),nd(lf(b),nd(lf(c),lf(d)))), [r,r,l])
path(a, nd(lf(a),nd(lf(b),nd(lf(c),lf(d)))), [r,r,r])
```

Logic program for path

Prolog notation

```
path(X,lf(X),[]).
path(X,nd(T,_),[l|P]) :- path(X,T,P).
path(X,nd(_,T),[r|P]) :- path(X,T,P).
```

Logical notation (with explicit universal quantification)

$$\begin{array}{rrcl}
\forall \mathtt{X}. & & & \mathtt{path(X,lf(X),[])} \\
\forall \mathtt{X,T,U,P}. & \mathtt{path(X,T,P)} & \rightarrow & \mathtt{path(X,nd(T,U),[l|P])} \\
\forall \mathtt{X,S,T,P}. & \mathtt{path(X,T,P)} & \rightarrow & \mathtt{path(X,nd(S,T),[r|P])}
\end{array}$$

Example goal

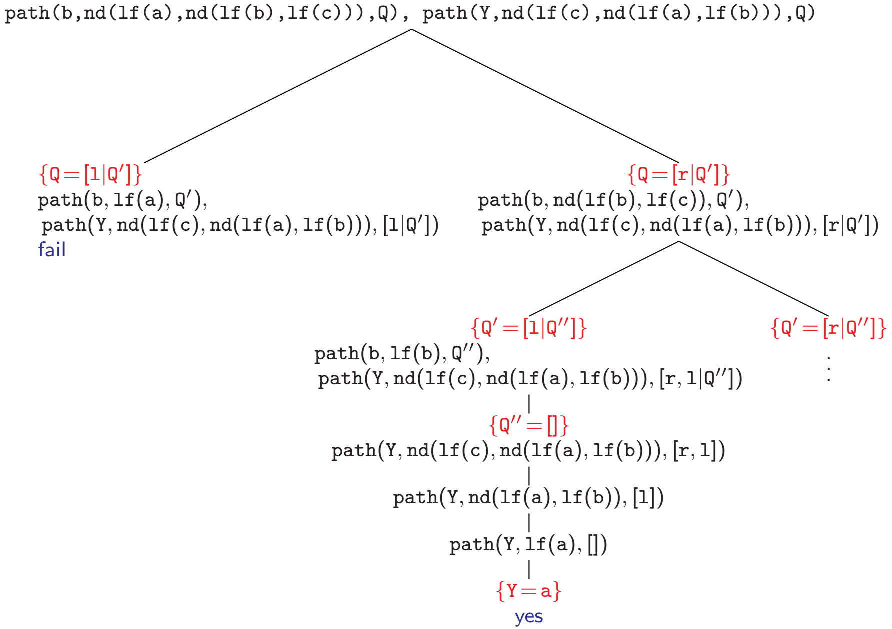
Prolog notation

```
path(b,nd(lf(a),nd(lf(b),lf(c))),Q), path(Y,nd(lf(c),nd(lf(a),lf(b))),Q)
```

Logical notation (with explicit existential quantification)

$\exists$`Q,Y.path(b,nd(lf(a),nd(lf(b),lf(c))),Q)` $\wedge$ `path(Y,nd(lf(c),nd(lf(a),lf(b))),Q)`

The next slide depicts (relevant parts of) the search tree for this goal.


path(b,nd(lf(a),nd(lf(b),lf(c))),Q), path(Y,nd(lf(c),nd(lf(a),lf(b))),Q)
{Q=[l|Q′]}
path(b, lf(a), Q′),
path(Y, nd(lf(c), nd(lf(a), lf(b))), [l|Q′])
fail
{Q=[r|Q′]}
path(b, nd(lf(b), lf(c)), Q′),
path(Y, nd(lf(c), nd(lf(a), lf(b))), [r|Q′])
{Q′=[l|Q″]}
path(b, lf(b), Q″),
path(Y, nd(lf(c), nd(lf(a), lf(b))), [r, l|Q″])
{Q′=[r|Q″]}
⋮
{Q″=[]}
path(Y, nd(lf(c), nd(lf(a), lf(b))), [r, l])
path(Y, nd(lf(a), lf(b)), [l])
path(Y, lf(a), [])
{Y=a}
yes

Result of example query

The query

$\exists$`Q,Y . path(b,nd(lf(a),nd(lf(b),lf(c))),Q)` $\wedge$ `path(Y,nd(lf(c),nd(lf(a),lf(b))),Q)`

succeeds, returning the result of the combined substitution

$$\{\,\mathtt{Q} = [\mathtt{r}|\mathtt{Q}']\,\}\{\,\mathtt{Q}' = [\mathtt{l}|\mathtt{Q}'']\,\}\{\,\mathtt{Q}'' = []\,\}\{\,\mathtt{Y} = \mathtt{a}\,\}$$

which is the substitution

$$\Big\{\,\mathtt{Q} = \mathtt{[r,l]},\ \mathtt{Y} = \mathtt{a}\,\Big\}$$

The resulting substitution provides *witnessing terms* for the existentially quantified variables. That is, the following formula is a logical consequence of the program.

`path(b,nd(lf(a),nd(lf(b),lf(c))),[r,l])` $\wedge$ `path(a,nd(lf(c),nd(lf(a),lf(b))),[r,l])`

Substitutions

A substitution is an assignment of terms to variables:

$$\{\, \mathtt{X}_1 = t_1 \,,\, \ldots \,,\, \mathtt{X}_\mathtt{n} = t_n \,\}$$

where the variables $\mathtt{X}_1, \ldots, \mathtt{X}_\mathtt{n}$ are all distinct.

Other notation often used for substitutions is:

$$\begin{gathered} \{\, \mathtt{X}_1 \backslash t_1 \,,\, \ldots \,,\, \mathtt{X}_\mathtt{n} \backslash t_n \,\} \\ \{\, \mathtt{X}_1 / t_1 \,,\, \ldots \,,\, \mathtt{X}_\mathtt{n} / t_n \,\} \\ \{\, t_1 / \mathtt{X}_1 \,,\, \ldots \,,\, t_n / \mathtt{X}_\mathtt{n} \,\} \end{gathered}$$

Note that the last two are mutually inconsistent.

Applying substitutions

A substitution $\{ \mathtt{X_1} = t_1 , \ldots , \mathtt{X_n} = t_n \}$ is *applied* to a term t or a formula F by replacing all the (free) occurrences of the variables $\mathtt{X_1}, \ldots, \mathtt{X_n}$ with the terms $t_1, \ldots, t_n$ respectively.

We write:

$$t \{ \mathtt{X_1} = t_1 , \ldots , \mathtt{X_n} = t_n \}$$
$$F \{ \mathtt{X_1} = t_1 , \ldots , \mathtt{X_n} = t_n \}$$

for the resulting substituted term and formula.

Technical restriction on substitutions

In logic programming, the application of the substitution $\{\, \mathtt{X}_1 = t_1 \,,\, \ldots \,,\, \mathtt{X}_n = t_n \,\}$ can be restricted only to terms t and formulas F for which

$$\begin{aligned} (\mathit{Vars}(t) \cup \{\mathtt{X}_1, \ldots, \mathtt{X}_n\}) \cap \mathit{Vars}(t_1, \ldots, t_n) \;&=\; \emptyset \\ (\mathit{Vars}(F) \cup \{\mathtt{X}_1, \ldots, \mathtt{X}_n\}) \cap \mathit{Vars}(t_1, \ldots, t_n) \;&=\; \emptyset \end{aligned}$$

Informally, the restriction says that variables in the new terms being substituted in must be "fresh".

The restriction is technically motivated. We shall comment further on the effects of this restriction in purple. Such comments are for the benefit of the mathematically inquisitive.

Example

The result of the substitution

$$\mathtt{f(X, g(Y, h(X)))} \{ \mathtt{X} = \mathtt{h(Z)}\, ,\ \mathtt{Y} = \mathtt{f(W, W)} \}$$

is the term

$$\mathtt{f(h(Z), g(f(W, W), h(h(Z))))}$$

Note how the substitutions for `X` and `Y` are performed simultaneously.

Example (continued)

Substitutions are performed in sequence in a composite substitution:

$$\mathtt{f(X, g(W, h(X)))}\,\{\,\mathtt{X} = \mathtt{h(Y)}\,\}\,\{\,\mathtt{Y} = \mathtt{f(Z, Z)}\,\}$$

($\mathtt{W}$ is not allowed to be $\mathtt{Y}$ or $\mathtt{Z}$ here). The result of this is

$$\mathtt{f(h(f(Z, Z)), g(W, h(h(f(Z, Z)))))}$$

The composite substitution $\{\,\mathtt{X} = \mathtt{h(Y)}\,\}\,\{\,\mathtt{Y} = \mathtt{f(Z, Z)}\,\}$ has the same effect (on terms not containing `Y,Z`) as the substitution:

$$\{\,\mathtt{X} = \mathtt{h(f(Z, Z))}\}$$

We write:

$$\{\,\mathtt{X} = \mathtt{h(Y)}\,\}\,\{\,\mathtt{Y} = \mathtt{f(Z, Z)}\,\} \;=\; \{\,\mathtt{X} = \mathtt{h(f(Z, Z))}\,\}$$

Composition of substitutions

In general, given substitutions θ_1 and θ_2, one can (when there is no clash of variables) define a *composite* substitution

$$\theta_1\,\theta_2$$

that satisfies, for every term t (for which both sides are legitimate),

$$t\,(\theta_1\,\theta_2) \;=\; (t\,\theta_1)\,\theta_2$$

Let $\{\}$ be the *identity* substitution, which has no effect. That is $t\,\{\} \;=\; t$.

Composition is an example of a (partial) *monoid*, i.e.,

$$\begin{aligned} \{\}\,\theta \;=\; \theta \;=\; \theta\,\{\} \\ \theta_1\,(\theta_2\,\theta_3) \;=\; (\theta_1\,\theta_2)\,\theta_3 \end{aligned}$$

Accordingly, we can write $\theta_1\,\theta_2\,\theta_3$ without ambiguity.

Generality of substitutions

We say that a substitution θ_2 is *more general than* a substitution θ_1, notation

$$\theta_1 \preccurlyeq \theta_2$$

if there exists a substitution θ such that

$$\theta_1 \;=\; \theta_2\,\theta$$

Example

$$\{\,\mathtt{X} = \mathtt{f(a)}\,,\,\mathtt{Y} = \mathtt{g(a,b)}\,,\,\mathtt{W} = \mathtt{f(b)}\,\} \;\preccurlyeq\; \{\,\mathtt{X} = \mathtt{f(Z)}\,,\,\mathtt{Y} = \mathtt{g(Z,V)}\,\}$$

because

$$\{\,\mathtt{X} = \mathtt{f(a)}\,,\,\mathtt{Y} = \mathtt{g(a,b)}\,,\,\mathtt{W} = \mathtt{f(b)}\} \;=\; \{\,\mathtt{X} = \mathtt{f(Z)}\,,\,\mathtt{Y} = \mathtt{g(Z,V)}\}\{\,\mathtt{Z} = \mathtt{a}\,,\,\mathtt{V} = \mathtt{b}\,,\,\mathtt{W} = \mathtt{f(b)}\,\}$$

Properties of generality

The relation $\preccurlyeq$ is a *preorder*, that is:

$$\theta \preccurlyeq \theta$$

$$\theta_1 \preccurlyeq \theta_2 \wedge \theta_2 \preccurlyeq \theta_3 \rightarrow \theta_1 \preccurlyeq \theta_3$$

We say that substitutions θ_1 and θ_2 are *equivalent* if

$$\theta_1 \preccurlyeq \theta_2 \wedge \theta_2 \preccurlyeq \theta_1$$

θ_1 and θ_2 are equivalent if and only if there exists a *renaming*, that is, a substitution

$$\{ \mathtt{X}_1 = \mathtt{Y}_1 , \dots , \mathtt{X}_n = \mathtt{Y}_n \}$$

where the variables $\mathtt{Y}_1, \dots, \mathtt{Y}_n$ are all distinct, such that

$$\theta_1 = \theta_2 \{ \mathtt{X}_1 = \mathtt{Y}_1 , \dots , \mathtt{X}_n = \mathtt{Y}_n \}$$

Most general unifiers

A *unifier* for two terms t_1, t_2 is a substitution θ such that

$$t_1\,\theta \;=\; t_2\,\theta$$

We say that t_1, t_2 are *unifiable* if there exists a unifier.

Theorem.
If t_1, t_2 are unifiable then they have a *most general unifier*. That is, there exists a unifier θ satisfying

for every unifier θ', it holds that $\theta' \preccurlyeq \theta$.

Proof. Find θ using the algorithm in Programming Lecture 2. $\square$

Note that most general unifiers are unique up to equivalence.

Similarly, most general unifiers exist for unifiable formulas A_1, A_2.

Example search tree revisited

For the goal

$$\mathtt{path(b, nd(lf(b), lf(c)), Q'),\ path(Y, nd(lf(c), nd(lf(a), lf(b))), [r|Q'])}$$

We unify the head atomic formula $\mathtt{path(b, nd(lf(b), lf(c)), Q')}$ with the right-hand-side of the rule

$$\mathtt{path(X, T, P)} \quad \longrightarrow \quad \mathtt{path(X, nd(T, U), [l|P])}$$

Giving the most general unifier

$$\theta \;=\; \{\, \mathtt{X = b}\,,\, \mathtt{T = lf(b)}\,,\, \mathtt{U = lf(c)}\,,\, \mathtt{Q' = [l|Q'']}\,,\, \mathtt{P = Q''}\,\} \qquad\qquad (*)$$

We replace the goal with

$$\begin{aligned} &\mathtt{path(X, T, P)}\,\theta,\ \mathtt{path(Y, nd(lf(c), nd(lf(a), lf(b))), [r|Q'])}\,\theta \\ =\quad &\mathtt{path(b, lf(b), Q''),\ path(Y, nd(lf(c), nd(lf(a), lf(b))), [r, l|Q''])} \end{aligned}$$

and we record θ restricted to goal variables $\{\,\mathtt{Q' = [l|Q'']}\,\}$

Building the search tree in general

At each node in the search tree, the *current goal* is a list $G_1, \dots, G_m$ of atomic formulas.

1. If G_1 unifies with some axiom B (a Prolog fact) in the program then generate a new goal

   $$G_2\theta, \dots, G_m\theta \ ,$$

   where θ is the most general unifier of G_1 and B.

   Record the effect of the substitution θ on variables occuring in $G_1, \dots, G_m$.

Building the search tree in general (continued)

2. If G_1 unifies with the right-hand side B of an axiom (a Prolog rule) of the form

   $$A_1 \wedge \cdots \wedge A_k \ \rightarrow \ B$$

   in the program, then generate a new goal of the form:

   $$A_1\theta, \ldots, A_k\theta, G_2\theta, \ldots, G_m\theta$$

   where θ is the most general unifier of G_1 and B.
   Again, record the effect of the substitution θ on variables occuring in $G_1, \ldots, G_m$.

The goal of the proof search procedure is to find a branch ending in a leaf with the *empty* goal list. Whence Prolog returns the composite substitution along this branch, which provides a substitution witnessing the original query.

A subtle point

In the above, G_1 is always unified with an atomic formula B, where B is either itself one of the program axioms F_i, or it is the right-hand side of a rule F_i. In either case, the variables in F_i are implictly universally quantified. Similarly, the variables in $G_1, \ldots, G_m$ are implicitly existentially quantified.

Because of this, it does not matter what the variables are called. The meaning of the clause is unaffected by a different choice of variable names.

However, for the unification in the proof search to work correctly, it is essential that the variables names in F_i and $G_1, \ldots, G_m$ are chosen so that they do not overlap.

This motivates the choice of variable names in the most general unifier (*) on the Example Search Tree Revisited slide.

Prolog proof search

Prolog proof search is *depth first*:

- The search always moves to an *unvisited* (i.e., not previously visited) *child* (i.e., immediately below) node of the current node, whenever such a node exists.
- If there is no such node, the search backtracks to the most recently visited node from which such an unvisited child node is available.

Prolog proof search follows *program order*.

- Child nodes are visited in the order that the axioms (Prolog rules) that determine the child node appear in the program (i.e., from left to right in the trees as we are drawing them).

The strategy here is identical to that for propositional Prolog

Main points today

substitutions

unifiers and most general unifiers

search tree for definite clause predicate logic

Prolog proof search strategy

Appendix 1: Restriction on application of substitutions

We would like to have that

$$\{\mathtt{X} = \mathtt{a}\,,\ \mathtt{Y} = \mathtt{a}\} \preccurlyeq \{\mathtt{X} = \mathtt{Z}\,,\ \mathtt{Y} = \mathtt{Z}\}$$

via the expected composition of substitutions

$$\{\mathtt{X} = \mathtt{a}\,,\ \mathtt{Y} = \mathtt{a}\} = \{\mathtt{X} = \mathtt{Z}\,,\ \mathtt{Y} = \mathtt{Z}\}\{\mathtt{Z} = \mathtt{a}\}$$

But if we were to allow the application of $\{\mathtt{X} = \mathtt{Z}\,,\ \mathtt{Y} = \mathtt{Z}\}$ to terms containing $\mathtt{Z}$ we would get, for example,

$$\begin{aligned} \mathtt{f}(\mathtt{X},\mathtt{Y},\mathtt{Z})\{\mathtt{X} = \mathtt{Z}\,,\ \mathtt{Y} = \mathtt{Z}\}\{\mathtt{Z} = \mathtt{a}\} &= \mathtt{f}(\mathtt{Z},\mathtt{Z},\mathtt{Z})\{\mathtt{Z} = \mathtt{a}\} \\ &= \mathtt{f}(\mathtt{a},\mathtt{a},\mathtt{a}) \end{aligned}$$

$$\mathtt{f}(\mathtt{X},\mathtt{Y},\mathtt{Z})\{\mathtt{X} = \mathtt{a}\,,\ \mathtt{Y} = \mathtt{a}\} = \mathtt{f}(\mathtt{a},\mathtt{a},\mathtt{Z})$$

Appendix 2: Non-overlapping variables requirement

Consider the simple logic program

```
p(a,X)
```

and the simple query `p(X,b)` .

The query should succeed with substitution $\{\,\mathtt{X} = \mathtt{a}\,\}$

As written, the goal `p(X,b)` and the axiom `p(a,X)` do not unify, because there is no consistent assignment for the variable `X`.

It is thus necessary to rename variables so that the variables in the program do not overlap with variables in the goal.

The unification of `p(X,b)` and the axiom $\mathtt{p}(\mathtt{a}, \mathtt{X}')$ does succeed with most general unifier $\{\,\mathtt{X} = \mathtt{a}\,,\ \mathtt{X}' = \mathtt{b}\,\}$, and so the correct substitution $\{\,\mathtt{X} = \mathtt{a}\,\}$ is returned.