Harvard School of Engineering and Applied Sciences — CS 152: Programming Languages

Logic Programming; Dynamic Types
Section and Practice Problems

Apr 30–May 3, 2019

---

1 Logic Programming

To try playing around with Prolog, go to `http://www.swi-prolog.org/`. You will be able to use Prolog online at `https://swish.swi-prolog.org/`.

To try playing around with Datalog, you can go to either `http://abcdatalog.seas.harvard.edu/` to download a Java-based Datalog implementation, or you can go to `https://datalog.db.in.tum.de/` to use Datalog online.

Although you can use the tools above to get the answers to the section problems below very easily, work out the answers by hand (to make sure you understand the semantics of Prolog and Datalog), and then you can check your answers by using the tools to execute the programs.

(a) Consider the following Prolog program (where [] is a constant representing the empty list, $[t]$ is shorthand for $\mathsf{cons}(t, [])$ and $[t_1, t_2|t_3]$ is shorthand for $\mathsf{cons}(t_1, \mathsf{cons}(t_2, t_3))$.

$$\begin{array}{l} \mathsf{foo}([], []). \\ \mathsf{foo}([X], [X]). \\ \mathsf{foo}([X, Y|S], [Y, X|T]) \text{ :- } \mathsf{foo}(S, T). \end{array}$$

For each of the following queries, compute the substitutions that Prolog will generate, if any. (Note that there is a difference between an empty substitution, and no substitution.) If the query evaluation will not terminate, explain why.

- $\mathsf{foo}([\mathsf{a}, \mathsf{b}], X)$.
- $\mathsf{foo}([\mathsf{a}, \mathsf{b}, \mathsf{c}], X)$.
- $\mathsf{foo}([\mathsf{a}, \mathsf{b}], [\mathsf{a}, \mathsf{b}])$
- $\mathsf{foo}(X, [\mathsf{a}])$
- $\mathsf{foo}(X, Y)$.

Answer: *Intuitively,* $\mathsf{foo}(S, T)$ *holds for two lists* S *and* T *if they are the same length, and for all* i*, the* $2i$*th and* $2i+1$*th elements of* S *are equal, respectively, to the* $2i+1$*th and* $2i$*th elements of* T*.*

- $\mathsf{foo}([\mathsf{a}, \mathsf{b}], X)$.
  `X = [b,a]`
- $\mathsf{foo}([\mathsf{a}, \mathsf{b}, \mathsf{c}], X)$.
  `X = [b,a,c]`
- $\mathsf{foo}([\mathsf{a}, \mathsf{b}], [\mathsf{a}, \mathsf{b}])$
  No substitutions returned
- $\mathsf{foo}(X, [\mathsf{a}])$
  `X = [a]`

- $\mathsf{foo}(X, Y)$.
  X = [], Y = []
  X = [A, B], Y = [B, A]
  X = [A, B, C], Y = [B, A, C]
  X = [A, B, C, D], Y = [B, A, D, C]
  X = [A, B, C, D, E], Y = [B, A, D, C, E]
  X = [A, B, C, D, E, F], Y = [B, A, D, C, F, E]
  ...
  The evaluation of the query never terminates.

(b) Consider the following Datalog program.

$$
\begin{aligned}
&\mathsf{bar}(\mathsf{a}, \mathsf{b}, \mathsf{c}).\\
&\mathsf{bar}(X, Y, Z) \text{ :- } \mathsf{bar}(Y, X, Z).\\
&\mathsf{bar}(X, Y, Z) \text{ :- } \mathsf{bar}(Z, Y, X), \mathsf{quux}(X, Z).\\
&\mathsf{quux}(\mathsf{b}, \mathsf{c}).\\
&\mathsf{quux}(\mathsf{c}, \mathsf{d}).\\
&\mathsf{quux}(X, Y) \text{ :- } \mathsf{quux}(Y, X).\\
&\mathsf{quux}(X, Z) \text{ :- } \mathsf{quux}(X, Y), \mathsf{quux}(Y, Z).
\end{aligned}
$$

Find all solutions to the query $\mathsf{bar}(X, Y, Z)$.

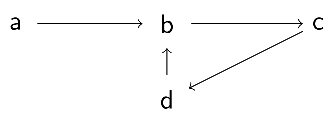
Answer: *We start by the set of facts that are known, S_0, and then given S_i we produce S_{i+1} by unifying the horn clauses with the known facts to derive new facts, and repeat until we reach a fixed point.*

$$
\begin{aligned}
S_0 &= \{\mathsf{bar}(\mathsf{a}, \mathsf{b}, \mathsf{c}).,\ \mathsf{quux}(\mathsf{b}, \mathsf{c}).,\ \mathsf{quux}(\mathsf{c}, \mathsf{d}).\}\\
S_1 &= \{\mathsf{bar}(\mathsf{a}, \mathsf{b}, \mathsf{c}).,\ \mathsf{quux}(\mathsf{b}, \mathsf{c}).,\ \mathsf{quux}(\mathsf{c}, \mathsf{d}).,\mathsf{bar}(\mathsf{b}, \mathsf{a}, \mathsf{c}).,\mathsf{quux}(\mathsf{c}, \mathsf{b}).,\mathsf{quux}(\mathsf{d}, \mathsf{c}).,\mathsf{quux}(\mathsf{b}, \mathsf{d}).\}\\
S_2 &= \{\mathsf{bar}(\mathsf{a}, \mathsf{b}, \mathsf{c}).,\ \mathsf{quux}(\mathsf{b}, \mathsf{c}).,\ \mathsf{quux}(\mathsf{c}, \mathsf{d}).,\mathsf{bar}(\mathsf{b}, \mathsf{a}, \mathsf{c}).,\mathsf{quux}(\mathsf{c}, \mathsf{b}).,\mathsf{quux}(\mathsf{d}, \mathsf{c}).,\mathsf{quux}(\mathsf{b}, \mathsf{d}).,\\
&\qquad\qquad \mathsf{bar}(\mathsf{c}, \mathsf{a}, \mathsf{b}).,\mathsf{quux}(\mathsf{d}, \mathsf{b}).,\mathsf{quux}(\mathsf{b}, \mathsf{b}).\mathsf{quux}(\mathsf{c}, \mathsf{c}).\}\\
S_3 &= \{\mathsf{bar}(\mathsf{a}, \mathsf{b}, \mathsf{c}).,\ \mathsf{quux}(\mathsf{b}, \mathsf{c}).,\ \mathsf{quux}(\mathsf{c}, \mathsf{d}).,\mathsf{bar}(\mathsf{b}, \mathsf{a}, \mathsf{c}).,\mathsf{quux}(\mathsf{c}, \mathsf{b}).,\mathsf{quux}(\mathsf{d}, \mathsf{c}).,\mathsf{quux}(\mathsf{b}, \mathsf{d}).,\\
&\qquad\qquad \mathsf{bar}(\mathsf{c}, \mathsf{a}, \mathsf{b}).,\mathsf{quux}(\mathsf{d}, \mathsf{b}).,\mathsf{quux}(\mathsf{b}, \mathsf{b}).\mathsf{quux}(\mathsf{c}, \mathsf{c}).,\mathsf{bar}(\mathsf{a}, \mathsf{c}, \mathsf{b}).\}\\
S_4 &= \{\mathsf{bar}(\mathsf{a}, \mathsf{b}, \mathsf{c}).,\ \mathsf{quux}(\mathsf{b}, \mathsf{c}).,\ \mathsf{quux}(\mathsf{c}, \mathsf{d}).,\mathsf{bar}(\mathsf{b}, \mathsf{a}, \mathsf{c}).,\mathsf{quux}(\mathsf{c}, \mathsf{b}).,\mathsf{quux}(\mathsf{d}, \mathsf{c}).,\mathsf{quux}(\mathsf{b}, \mathsf{d}).,\\
&\qquad\qquad \mathsf{bar}(\mathsf{c}, \mathsf{a}, \mathsf{b}).,\mathsf{quux}(\mathsf{d}, \mathsf{b}).,\mathsf{quux}(\mathsf{b}, \mathsf{b}).\mathsf{quux}(\mathsf{c}, \mathsf{c}).,\mathsf{bar}(\mathsf{a}, \mathsf{c}, \mathsf{b}).\}
\end{aligned}
$$

Since S_3 and S_4 are the same (i.e., applying the rules to S_3 doesn't derive any new facts) we have a fixed point. So all solutions to the query $\mathsf{bar}(X, Y, Z)$? *are:*

$$
\begin{aligned}
&\mathsf{bar}(\mathsf{a}, \mathsf{b}, \mathsf{c}).\\
&\mathsf{bar}(\mathsf{b}, \mathsf{a}, \mathsf{c}).\\
&\mathsf{bar}(\mathsf{c}, \mathsf{a}, \mathsf{b}).\\
&\mathsf{bar}(\mathsf{a}, \mathsf{c}, \mathsf{b}).
\end{aligned}
$$

(c) Suppose that we represent a directed graph using the predicates $\mathsf{edge}(X, Y)$ to indicate that there is an edge from node X to node Y. For example, the following graph is represented by the following facts:



node(a).
node(b).
node(c).
node(d).
edge(a, b).
edge(b, c).
edge(c, d).
edge(d, b).

(i) Write a Datalog program that computes reachable(X, Y), where reachable(X, Y) holds if there is a path (of zero or more edges) from X to Y.

Answer:

reachable(X, X) :- node(X).
reachable(X, Y) :- edge(X, Z), reachable(Z, Y).

Note that we can't just use the clause reachable(X, X).*, as that would not bind variable X in the body of clause, which violates the requirements of Datalog. That is, X is reachable from itself only if X is a node.*

(ii) Write a Datalog program that computes sameSCC(X, Y), where sameSCC(X, Y) holds if nodes X and node Y are in the same strongly connected component. (Hint: use the predicate reachable.)

Answer: *Two nodes a and b are in the same strongly connected component if and only if there is a path from a to b, and a path from b to a.*

sameSCC(X, Y) :- reachable(X, Y), reachable(Y, X).

For our example graph above, node a *is in its own strongly connected component, but nodes* b*,* c*, and* d *are in the same SCC. So the result of the query* sameSCC(X, Y)? *is the following:*

```
sameSCC(a, a)
sameSCC(b, b)
sameSCC(b, c)
sameSCC(b, d)
sameSCC(c, b)
sameSCC(c, c)
sameSCC(c, d)
sameSCC(d, b)
sameSCC(d, c)
sameSCC(d, d)
```

2 Dynamic types and contracts

This material will be covered in Lecture on Tuesday April 30; we include it here so that later sections can be focused on comprehensive review.

(a) To make sure you understand the operational semantics of dynamic types and exceptions, show the execution of the following program under the semantics of Section 1 of the Lecture 25 notes.

$$\begin{array}{l}\textsf{let } f = \lambda x.\, 42 + x \textsf{ in} \\ \textsf{let } g = \lambda y.\, (y\ \textsf{true}) + 42 \textsf{ in} \\ g\ f\end{array}$$

Answer:

$$\begin{array}{ll} & \textit{let } f = \lambda x.\, 42 + x \textit{ in let } g = \lambda y.\, (y\ \textsf{true}) + 42 \textit{ in } g\ f \\ \longrightarrow & \textit{let } g = \lambda y.\, (y\ \textsf{true}) + 42 \textit{ in } g\ (\lambda x.\, 42 + x) \\ \longrightarrow & (\lambda y.\, (y\ \textsf{true}) + 42)\ (\lambda x.\, 42 + x) \\ \longrightarrow & ((\lambda x.\, 42 + x)\ \textsf{true}) + 42 \\ \longrightarrow & (42 + \textsf{true}) + 42 \\ \longrightarrow & \textsf{Err} + 42 \\ \longrightarrow & \textsf{Err} \end{array}$$

(b) Modify the program from question (a) by adding appropriate error handlers (i.e., expressions of the form try e_1 catch $x.\, e_2$ to catch the type error and return the integer 42 as the final result of the program. There are multiple places in the program where you can insert an error handler to achieve the desired result. Show three variations and their executions. (Note that the semantics for the execution of your programs is from Part 2 (Exception handling) of the Lecture 22 notes.)

Answer: *Here is a version where we add an error handler in the body of function f.*

$$\begin{array}{l}\textit{let } f = \lambda x.\, (\textsf{try } (42 + x) \textsf{ catch } z.\, 0) \textit{ in} \\ \textit{let } g = \lambda y.\, (y\ \textsf{true}) + 42 \textit{ in} \\ g\ f\end{array}$$

$$\begin{array}{ll} & \textit{let } f = \lambda x.\, (\textsf{try } (42 + x) \textsf{ catch } z.\, 0) \textit{ in let } g = \lambda y.\, (y\ \textsf{true}) + 42 \textit{ in } g\ f \\ \longrightarrow & \textit{let } g = \lambda y.\, (y\ \textsf{true}) + 42 \textit{ in } g\ (\lambda x.\, (\textsf{try } (42 + x) \textsf{ catch } z.\, 0)) \\ \longrightarrow & (\lambda y.\, (y\ \textsf{true}) + 42)\ (\lambda x.\, (\textsf{try } (42 + x) \textsf{ catch } z.\, 0)) \\ \longrightarrow & ((\lambda x.\, (\textsf{try } (42 + x) \textsf{ catch } z.\, 0))\ \textsf{true}) + 42 \\ \longrightarrow & (\textsf{try } (42 + \textsf{true}) \textsf{ catch } z.\, 0) + 42 \\ \longrightarrow & (\textsf{try } (\textsf{Err } 1) \textsf{ catch } z.\, 0) + 42 \\ \longrightarrow & 0 + 42 \\ \longrightarrow & 42 \end{array}$$

Here's another version, where we add an error handler in the body of function g.

$$\begin{array}{l} \textit{let } f = \lambda x.\, 42 + x \textit{ in} \\ \textit{let } g = \lambda y.\, (\mathsf{try}\ (y\ \mathsf{true})\ \mathsf{catch}\ z.\, 0) + 42 \textit{ in} \\ g\ f \end{array}$$

$$\begin{array}{l} \quad \textit{let } f = \lambda x.\, 42 + x \textit{ in let } g = \lambda y.\, (\mathsf{try}\ (y\ \mathsf{true})\ \mathsf{catch}\ z.\, 0) + 42 \textit{ in } g\ f \\ \longrightarrow \textit{let } g = \lambda y.\, (\mathsf{try}\ (y\ \mathsf{true})\ \mathsf{catch}\ z.\, 0) + 42 \textit{ in } g\ (\lambda x.\, 42 + x) \\ \longrightarrow (\lambda y.\, (\mathsf{try}\ (y\ \mathsf{true})\ \mathsf{catch}\ z.\, 0) + 42)\ (\lambda x.\, 42 + x) \\ \longrightarrow (\mathsf{try}\ (\lambda x.\, 42 + x)\ \mathsf{true})\ \mathsf{catch}\ z.\, 0) + 42 \\ \longrightarrow (\mathsf{try}\ (42 + \mathsf{true})\ \mathsf{catch}\ z.\, 0) + 42 \\ \longrightarrow (\mathsf{try}\ (\mathsf{Err}\ 1)\ \mathsf{catch}\ z.\, 0) + 42 \\ \longrightarrow 0 + 42 \\ \longrightarrow 42 \end{array}$$

Finally, here is a version where we put the error handling code at the top level.

$$\begin{array}{l} \textit{let } f = \lambda x.\, 42 + x \textit{ in} \\ \textit{let } g = \lambda y.\, (y\ \mathsf{true}) + 42 \textit{ in} \\ \mathsf{try}\ g\ f\ \mathsf{catch}\ z.\, 42 \end{array}$$

$$\begin{array}{l} \quad \textit{let } f = \lambda x.\, 42 + x \textit{ in let } g = \lambda y.\, (y\ \mathsf{true}) + 42 \textit{ in } \mathsf{try}\ g\ f\ \mathsf{catch}\ z.\, 42 \\ \longrightarrow \textit{let } g = \lambda y.\, (y\ \mathsf{true}) + 42 \textit{ in } \mathsf{try}\ g\ (\lambda x.\, 42 + x)\ \mathsf{catch}\ z.\, 42 \\ \longrightarrow \mathsf{try}\ (\lambda y.\, (y\ \mathsf{true}) + 42)\ (\lambda x.\, 42 + x)\ \mathsf{catch}\ z.\, 42 \\ \longrightarrow \mathsf{try}\ (((\lambda x.\, 42 + x)\ \mathsf{true}) + 42)\ \mathsf{catch}\ z.\, 42 \\ \longrightarrow \mathsf{try}\ ((42 + \mathsf{true}) + 42)\ \mathsf{catch}\ z.\, 42 \\ \longrightarrow \mathsf{try}\ ((\mathsf{Err}\ 1) + 42)\ \mathsf{catch}\ z.\, 42 \\ \longrightarrow \mathsf{try}\ (\mathsf{Err}\ 1)\ \mathsf{catch}\ z.\, 42 \\ \longrightarrow 42 \end{array}$$

(c) Modify the program from question (a) by adding appropriate dynamic type checks to raise the error as early as possible. When does your program detect the error?

Answer: *This version of the code adds dynamic type checks on all arguments and on results of function applications.*

$$\begin{array}{l} \textit{let } f = \lambda x.\, \mathsf{if}\ (\mathsf{is\_int?}\ \ x)\ \mathsf{then}\ 42 + x\ \mathsf{else\ raise}\ 3 \textit{ in} \\ \textit{let } g = \lambda y.\, \mathsf{if}\ (\mathsf{is\_fun?}\ \ y)\ \mathsf{then} \\ \qquad\qquad\qquad \textit{let } y' = (y\ \mathsf{true}) \textit{ in } \mathsf{if}\ (\mathsf{is\_int?}\ \ y')\ \mathsf{then}\ y' + 42\ \mathsf{else\ raise}\ 3 \\ \qquad\quad \mathsf{else} \\ \qquad\qquad\qquad \mathsf{raise}\ 3 \textit{ in} \\ \textit{let } a = g\ f \textit{ in } \mathsf{if}\ (\mathsf{is\_int?}\ \ a)\ \mathsf{then}\ a\ \mathsf{else\ raise}\ 3 \end{array}$$

The execution detects the error as soon as function f is invoked, i.e., $\mathsf{is\_int?}\ x$ *evaluates to false when x is replaced with* true*.*

(d) Modify the program from question (a) by adding contracts that specify the types of the input and output of f and g. Show the execution of the modified program.

Answer:

$$\textit{let } f = \mathsf{monitor}(\lambda x.\, 42 + x, \mathsf{is\_int?} \longmapsto \mathsf{is\_int?}\,) \textit{ in}$$
$$\textit{let } g = \mathsf{monitor}(\lambda y.\, (y\ \mathsf{true}) + 42, (\mathsf{is\_bool?} \longmapsto \mathsf{is\_int?}\,) \longmapsto \mathsf{is\_int?}\,) \textit{ in}$$
$$g\ f$$