CS323: AI (Logic Programming) **Spring 2016**

Lecture 10: Feb. 06, 2016

Lecturer: K.R. Chowdhary *: Professor of CS (VF)*

10.1 Interpretation of Horn clauses in Rule-chaining

A typical Horn clause problem has the form of:

1. a set of clauses which defines a problem domain and,
2. a theorem which consists of (a) hypotheses represented by assertions $A_1 \leftarrow, \dots, A_n \leftarrow$ and (b) a conclusion which is negated and represented by a denial $\leftarrow B_1, \dots, B_m$.

For our reasoning using forward and backward chaining, we consider the family-tree of *Mauryan Dunasty* (300 B.C. India) as shown in figure 10.1.

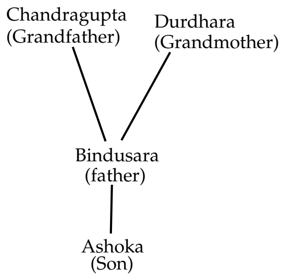

Figure 10.1: Mauryan Dynasty family-tree.

In backward-chaining, we reason backwards from the conclusion, repeatedly reducing goals to subgoals until eventually all subgoals are solved directly by the original assertions. In forward-chaining problem-solving, we reason forwards from the hypotheses, repeatedly deriving new assertions from old ones until eventually the original goal is solved directly by derived assertions.

The problem of showing that *chandragupta* is a grandparent of *ashoka* can be solved either backward-chaining or forward-chaining. In forward-chaining, we start with the following assertions:

father(bindusara, asoka) $\leftarrow$

father(chandragupta, bindusara) $\leftarrow$

Also, we shall use the clause $parent(x, y) \leftarrow father(x, y)$ to derive new assertions,

$parent(chandragupta, bindusara) \leftarrow$

$parent(bindusara, ashoka) \leftarrow$

Continuing forward-chaining we derive, from the definition of grandparent, the new assertion,

$grandparent(chandragupta, ashoka) \leftarrow$

which matches the original goal.

Reasoning using backward-chaining, we start with the original goal, which shows that chandragupta is a grandparent of ashoka,

$\leftarrow grandparent(chandragupta, ashoka)$

and use the definition of grandparent to derive two new subgoals,

$\leftarrow parent(chandragupta, z), parent(z, ashoka),$

by denying that any z is both a child of chandragupta and a parent of ashoka. Continuing backward-chaining and considering both subgoals (either one at a time or both simultaneously), we use the clause,

$$parent(x, y) \leftarrow father(x, y)$$

to replace the subproblem $parent(chandragupta, z)$ by $father(chandragupta, z)$ and the subproblem $parent(z, ashoka)$ by $father\ (z, ashoka)$. The symbol "$\leftarrow$" is read as "if". The newly derived subproblems are solved compatibly by assertions which determine "bindusara" as the desired value of z.

In both the backward-chaining and forward-chaining solutions of the grandparent problem, we have mentioned the derivation of only those clauses which directly contribute to the ultimate solution. In addition to the derivation of relevant clauses, it is often unavoidable, during the course of searching for a solution, to derive assertions or subgoals which do not contribute to the solution. For example, in the forward-chaining search for a solution to the grandparent problem, it is possible to derive the irrelevant assertions as,

$parent(durdhara, bindusara) \leftarrow$

$male(chandragupta) \leftarrow$

Also, in backward-chaining search it is possible to replace the subproblem $parent(chandragupta, z)$ by the irrelevant and unsolvable subproblem $mother(chandragupta, z)$.

There are proof procedures which understand logic in backward-chaining, e.g., model elimination, resolution, and interconnectivity graphs. These proof procedures operate with the clausal form of predicate logic and deal with both Horn clauses and non-Horn clauses. Among clausal proof procedures, the connection graph procedure is able to mix backward and forward reasoning.

The terminology we used here - the backward-chaining, is also called "top-down". Given a grammar formulated in clausal form, top-down parsing algorithm generates a sentence to its original form, i.e., the assertions. The forward-chaining is also called "bottom-up", where we start from the assertions and try to reach to the goals.

10.2 Path Finding Algorithm

Consider an algorithm A, with C_1, C_2 control components, and L_1, L_2 as logic components, which can often be analyzed in different ways.

$$A = L_1 + C_1 = L_2 + C_2. \tag{10.1}$$

Some of the behavior determined by C_1 in one analysis might be determined by the logic component L_2 in another analysis. This has significance for understanding the relationship between programming style and execution facilities. In the short term sophisticated behavior can be obtained by employing simple execution strategies and by writing complicated programs. In the longer term the same behavior may be obtained from simpler programs by using more sophisticated execution strategies.

A path-finding problem illustrates a situation in which the same algorithm can be analyzed in different ways. Consider the problem of finding a path from vertex a to vertex z in the directed graph shown in figure 10.2.

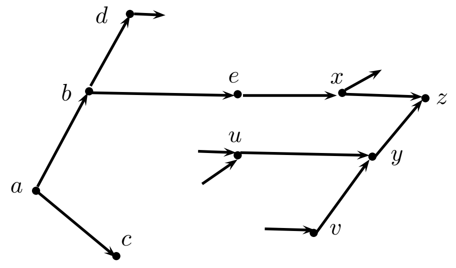

Figure 10.2: Directed graph path finding.

In one analysis, we can employ a predicate $go(x)$ which states that it is possible to go to x. The problem of going from a to z is then represented by two clauses. One asserts that it is possible to go to a. The other denies that it is possible to go to z. The arc directed from a to b is represented by a clause which states that it is possible to go to b if it is possible to go to a. Different control strategies determine different path-finding algorithms. Forward search from the initial node a is forward-chaining based reasoning from the initial assertion $go(a) \leftarrow$ (see table 10.1). Backward search from the goal node z is backward reasoning from the initial goal statement $\leftarrow go(z)$ (see table 10.2).

Bidirectional search from both the initial node and the goal node is the combination of backward and forward reasoning. Whether the path-finding algorithm investigates one path at a time (depth-first) or develops all paths simultaneously (breadth-first) is a matter of search strategy used to investigate alternatives.

Table 10.1: Forward-chaining.

$go(a) \leftarrow$
$go(b) \leftarrow go(a)$
$go(c) \leftarrow go(a)$
$go(d) \leftarrow go(b)$
$go(e) \leftarrow go(b)$
$go(x) \leftarrow go(e)$
$go(z) \leftarrow go(x)$
...

Table 10.2: Backward-chaining.

$\leftarrow go(z)$
$go(z) \leftarrow go(x)$
$go(z) \leftarrow go(y)$
$go(x) \leftarrow go(e)$
$go(y) \leftarrow go(u)$
$go(y) \leftarrow go(v)$
$go(e) \leftarrow go(b)$
$go(b) \leftarrow go(a)$
...

10.3 Expressing Control Information

The distinction between backward-chaining and forward-chaining based execution can be expressed in a graphical notation using arrows to indicate the flow of control. The same notation can be used to represent different combinations of backward-chaining and forward-chaining based execution. The notation does not, however, aim to provide a complete language for expressing useful control information.

The arrows are attached to atoms in clauses to indicate the direction of transmission of processing activity from one clause to other clause. For every pair of matching atoms in the initial set of clauses (i.e., one atom in the conclusion of a clause and the other among the conditions of a clause), and there is an arrow directed from one atom to the other. For backward-chained reasoning, arrows are directed from goals to assertions. For the grandparent problem, we have the graph shown in figure 10.3.

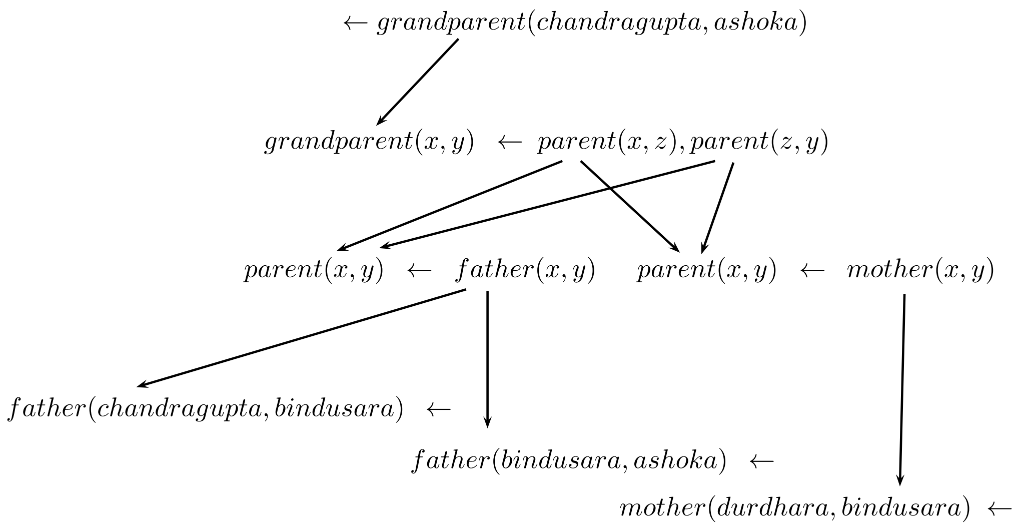
$\leftarrow grandparent(chandragupta, ashoka)$
$grandparent(x, y) \leftarrow parent(x, z), parent(z, y)$
$parent(x, y) \leftarrow father(x, y)$
$parent(x, y) \leftarrow mother(x, y)$
$father(chandragupta, bindusara) \leftarrow$
$father(bindusara, ashoka) \leftarrow$
$mother(durdhara, bindusara) \leftarrow$

Figure 10.3: Control-flow for backward-chaining .

The processing activity starts with the initial goal statement. It transmits activity to the body of the grandparent procedure, whose procedure calls, in turn, activate the parenthood definitions. Finally, the assertions in the knowledge-base passively accepts processing activity without transmitting it to other clauses.

For forward-chaining execution, arrows are directed from assertions to goals (see figure 10.4). The processing activity originates with the database of initial assertions (i.e., bottom of the graph in this figure).

$$\begin{tikzcd}
 & & \leftarrow grandparent(chandragupta, ashoka) & & \\
 & grandparent(x, y) \leftarrow \arrow[ur] & parent(x, z), & parent(z, y) & \\
 & parent(x, y) \leftarrow \arrow[ur] \arrow[urr] & father(x, y) & parent(x, y) \leftarrow \arrow[ul] \arrow[u] & mother(x, y) \\
father(chandragupta, bindusara) \leftarrow \arrow[urr] & & & & \\
 & father(bindusara, ashoka) \leftarrow \arrow[uur] & & & \\
 & & & & mother(durdhara, bindusara) \leftarrow \arrow[uuu]
\end{tikzcd}$$

Figure 10.4: Control-flow for forward-chaining.

They transmit activity to the parenthood definitions, which, in turn, activate the grandparent definition. Processing terminates when it reaches all the conditions in the passive initial goal statement, at the top of the graph.

The grandparent definition can be used in a combination of backward-chaining and forward-chaining methods. Using numbers to indicate sequencing, we can represent different combinations of backward-chaining and forward-chaining reasoning. For the sake of simplicity, we only show the control notation associated with the grandparent definition.