ConstLog: Constructive Logic

Lecture Notes on Forward Logic Programming

Frank Pfenning André Platzer

Carnegie Mellon University || Karlsruhe Institute of Technology

Lecture 18

1 Introduction

In the previous lecture we have seen backward chaining from a logical perspective, and how this can be seen as a foundation for backward-chaining logic programming languages like Prolog.

In this lecture we take a small step sideways: instead of considering all atoms to be *negative* we consider all atoms *positive*. This change that may at first glance appear to be seemingly innocuous has a rather drastic impact on the operational behavior of proof search, leading to *forward-chaining logic programming*. This is also called *bottom-up logic programming*, although the direction is for historical reasons strangely reversed from the way we consider the proof construction process. As one real application of forward chaining, we develop an algorithm for unification.

2 Reading Inference Rules from Premises to Conclusion

Over the last series of lectures, we almost always read inference rules by looking at the conclusion first and then the premises. This was so, because that is the direction of proof construction. In fact, the sequent calculus was specifically engineered by Gerhard Gentzen to have this property!

Now we will read inference rules starting with the premises. For example, assume we would like to calculate the path relation in an undirected graph, where we say there is a path from vertex x to y if there a sequence of vertices $x = x_0, x_1, \ldots, x_n = y$ such that all x_i and x_{i+1} are connected by an edge. For simplicity, let us say that the path follows $n \geq 1$ vertices.

We represent the vertices of a graph by constants, and the edge relation with a predicate $\mathsf{edge}(x, y)$ if there is an edge from x to y. Here is a specification of the path relation:

$$\frac{\mathsf{edge}(x,y)}{\mathsf{edge}(y,x)}\ \mathsf{sym} \qquad \frac{\mathsf{edge}(x,y)}{\mathsf{path}(x,y)}\ \mathsf{ep} \qquad \frac{\mathsf{path}(x,y) \quad \mathsf{path}(y,z)}{\mathsf{path}(x,z)}\ \mathsf{trans}$$

The first rule (sym) expresses we are working over an undirected graph. The second (ep) expresses that an edge represents a valid path (of length 1), and the third that the path relation is transitive.

Read from the conclusion to the premises, backward logic programming search over this specification is useless. Even just the rule sym rule will lead to an infinite loop, and the trans rule has an unknown y in the premise even if x and z are known in the conclusion.

Read from the premises to the conclusion, however, this is a decent program but only *if we avoid re-deriving facts we already know*. After a while, this program must terminate because there are at most $O(n^2)$ facts of the form $\mathsf{edge}(x, y)$ and $\mathsf{path}(x, y)$ that could be derived. Inference constructs a collection of derived facts called *database*. When inference reaches the point where any additional inference only infers facts that are already in the database, then we say the program has *reached saturation* and it halts. At this point we can answer any specific query simply by looking it up in the database. Using the above rules forward exhaustively computes a database consisting of the symmetric transitive closure of edge.

3 Saturation

As another example, we first consider the usual bottom-up specifications of $\mathsf{even}(n)$ and $\mathsf{odd}(n)$ for unary numbers.

$$\frac{}{\mathsf{even}(\mathsf{z})}\ \mathsf{ev}_z \qquad \frac{\mathsf{odd}(N)}{\mathsf{even}(\mathsf{s}(N))}\ \mathsf{ev}_s \qquad \frac{\mathsf{even}(N)}{\mathsf{odd}(\mathsf{s}(N))}\ \mathsf{od}_s$$

Trying to read these from the premise to the conclusion does not work: these rules would create an unbounded database with facts

$$\mathsf{even}(\mathsf{z}), \mathsf{odd}(\mathsf{s}(\mathsf{z})), \mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{z}))), \ldots$$

But if we view these rules as *introduction rules* we can derive *elimination rules* that work in the other direction, using what we already know!

$$\frac{\mathsf{even}(\mathsf{s}(N))}{\mathsf{odd}(N)}\ \mathsf{ev}'_s \qquad \frac{\mathsf{odd}(\mathsf{s}(N))}{\mathsf{even}(N)}\ \mathsf{od}'_s \qquad \frac{\mathsf{odd}(\mathsf{z})}{C}\ \mathsf{od}'_z$$

Note that there is no rule for $\mathsf{even}(\mathsf{z})$ because we cannot extract any information from that: the rule ev_z has no premises. In the last rule we have derived a contradiction,

coming from the fact that no rule concludes $\mathsf{odd}(\mathsf{z})$ (similar to no rule just proving $\bot$). Consequently, any proposition C would follow from $\mathsf{odd}(\mathsf{z})$. Now in a forward chaining program, we should conclude one specific proposition for C to avoid swamping the database with useless consequences. Unfortunately, $\uparrow\bot$ is not part of the chaining fragment (for good reason), so we use a new atom, no, for C, in the best tradition of the Prolog top level and consider no as finding a contradiction.

If we want to know if, say, the fact $\mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{z}))))$ is *consistent with* the definition of the predicate, we assert it in the database of facts and saturate the database using forward inference. Because of the simple nature of these rules, each inference is forced, and we obtain a saturated database:

$$\mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{z})))), \mathsf{odd}(\mathsf{s}(\mathsf{s}(\mathsf{z}))), \mathsf{even}(\mathsf{s}(\mathsf{z})), \mathsf{odd}(\mathsf{z}), \mathsf{no}$$

This tells us that asserting that 3 is even is *inconsistent* with the definition of evenness. On the other hand, when we assert $\mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{z})))$, we learn:

$$\mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{z}))), \mathsf{odd}(\mathsf{s}(\mathsf{z})), \mathsf{even}(\mathsf{z})$$

This database is now saturated and there is no contradiction, so the assertion that 2 is even is *consistent* with the definition of evenness.

4 Forward Chaining

Our translation from rules to propositions from the previous lecture leads us to the following propositions representing the downward-reading rules for even and odd numbers:

$$\begin{array}{rcl} \Gamma_{\mathsf{eo}} & = & \forall n.\, \mathsf{even}(\mathsf{s}(n)) \supset \mathsf{odd}(n), \\ & & \forall n.\, \mathsf{odd}(\mathsf{s}(n)) \supset \mathsf{even}(n), \\ & & \mathsf{odd}(\mathsf{z}) \supset \mathsf{no} \end{array}$$

We then ask, for example,

$$\Gamma_{\mathsf{eo}}, \mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{z})))) \longrightarrow \mathsf{no}$$

to find out if the fact that 3 is even is consistent with the knowledge in Γ_{eo}.

This search is now *the exact opposite of goal-directed*, let's call is *database-directed*, because it exploits knowledge from the database. We ignore the succedent (no) and saturate the database, which are the atoms in the antecedents. Only once we have saturated the database, do we even look at the succedent and see if it is a fact in the database (or, more generally, can be proven from the database directly without further forward chaining).

For this intuition to work, we have to start by instantiating n in the first proposition with $\mathsf{s}(\mathsf{s}(\mathsf{z}))$ and then use the implication left rule to conclude $\mathsf{odd}(\mathsf{s}(\mathsf{s}(\mathsf{z}))))$. It is this process we call *forward chaining*. To formalize this, we first recall the language and rules for *backward chaining*.

Backward chaining fragment: all atoms are negative, and the only polarity shift has the form $\downarrow P^-$ for (negative) atoms P^-. So far, we have only shown the rules for the connectives in red.

$$\begin{array}{lcll} \text{Program formulas} & D^- & ::= & P^- \mid G^+ \supset D^- \mid \forall x.\, D^-(x) \mid D_1^- \wedge D_2^- \mid \top \\ \text{Programs} & \Gamma^- & ::= & \cdot \mid \Gamma^-, D^- \\ \text{Goal formulas} & G^+ & ::= & \downarrow P^- \mid G_1^+ \wedge G_2^+ \mid \top \mid \exists x.\, G^+(x) \mid G_1^+ \vee G_2^+ \mid \bot \end{array}$$

Forward chaining fragment: all atoms are positive, and the only polarity shift has the form $\uparrow P^+$ for such a positive atom P^+ to shift it to negative polarity (changes in red).

$$\begin{array}{lcll} \text{Program formulas} & D^- & ::= & \uparrow P^+ \mid G^+ \supset D^- \mid \forall x.\, D^-(x) \mid D_1^- \wedge D_2^- \mid \top \\ \text{Database} & \Gamma & ::= & \cdot \mid \Gamma, D^- \mid \Gamma, P^+ \\ \text{Goal formulas} & G^+ & ::= & P^+ \mid G_1^+ \wedge G_2^+ \mid \top \mid \exists x.\, G^+(x) \mid G_1^+ \vee G_2^+ \mid \bot \end{array}$$

The antecedents Γ now mix the program formulas D^- (sometimes called the IDB) and the database facts P^+ (sometimes called the EDB), while the succedent is always positive since negative atoms are not part of the forward chaining fragment. We have the following three judgments:

Backward Chaining		Forward Chaining
$\Gamma^- \xrightarrow{f} P^-$	stable sequent	$\Gamma \xrightarrow{f} C^+$
$\Gamma^-, [D^-] \xrightarrow{f} P^-$	left focus	$\Gamma, [D^-] \xrightarrow{f} C^+$
$\Gamma^- \xrightarrow{f} [G^+]$	right focus	$\Gamma \xrightarrow{f} [G^+]$

The rules for the connectives remain the same, with the exception of the order of premises in the $\supset L$ rule.[1] We also remove the rules concerned with negative atoms and add those for positive ones.

5 Example

Let's observe these rules in action on our program and goal, where all atoms are positive. We have added $\uparrow$ shifts on the right-hand side of implications as required for proper polarization of the propositions.

$$\begin{array}{rcl} \Gamma_{\mathsf{eo}} & = & \forall n.\, \mathsf{even}(\mathsf{s}(n)) \supset \uparrow \mathsf{odd}(n), \\ & & \forall n.\, \mathsf{odd}(\mathsf{s}(n)) \supset \uparrow \mathsf{even}(n), \\ & & \mathsf{odd}(\mathsf{z}) \supset \uparrow \mathsf{no} \end{array}$$

$$\Gamma_{\mathsf{eo}}, \mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{z}))))) \xrightarrow{f} \mathsf{no}$$

[1] It is possible that in a combined forward/backward chaining language, there should be two forms of implication, presenting the premises in different order: forward implication $A \supset B$ proves A before assuming B, and backwards implication $B \subset A$ assumes B before proving A. This first one would be employed in forward chaining, the second one in backward chaining. Of course, logically the two are the same, but not operationally when viewed from the computation-as-proof-search perspective.

$$\frac{D^- \in \Gamma \quad \Gamma, [D^-] \xrightarrow{f} C^+}{\Gamma \xrightarrow{f} C^+}\ \mathsf{focus}L \qquad \frac{\Gamma \xrightarrow{f} [P^+]}{\Gamma \xrightarrow{f} P^+}\ \mathsf{focus}R \qquad \frac{\Gamma, P^+ \xrightarrow{f} C^+}{\Gamma, [\uparrow P^+] \xrightarrow{f} C^+}\ \uparrow L$$

$$\frac{\Gamma, [D^-(X)] \xrightarrow{f} C^+}{\Gamma, [\forall x.\, D^-(x)] \xrightarrow{f} C^+}\ \forall L^* \qquad \frac{\Gamma \xrightarrow{f} [G^+] \quad \Gamma, [D^-] \xrightarrow{f} C^+}{\Gamma, [G^+ \supset D^-] \xrightarrow{f} C^+}\ \supset L$$

no longer applicable $\left[\ \dfrac{Q^- = P^-}{\Gamma, [Q^-] \xrightarrow{f} P^-}\ \mathsf{id}^- \qquad \begin{array}{c}\text{no rule if } Q^- \neq P^- \\ \Gamma, [Q^-] \xrightarrow{f} P^-\end{array}\ \right]$

$$\frac{Q^+ \in \Gamma \quad Q^+ = P^+}{\Gamma \xrightarrow{f} [P^+]}\ \mathsf{id}^+ \qquad \begin{array}{c}\text{no rule if } Q^+ \neq P^+ \text{ for all } Q^+ \in \Gamma \\ \Gamma \xrightarrow{f} [P^+]\end{array}$$

$$\frac{\Gamma \xrightarrow{f} [G_1^+] \quad \Gamma \xrightarrow{f} [G_2^+]}{\Gamma \xrightarrow{f} [G_1^+ \wedge G_2^+]}\ \wedge R \qquad \frac{}{\Gamma \xrightarrow{f} [\top]}\ \top R$$

$$\frac{\Gamma \xrightarrow{f} [G(X)]}{\Gamma \xrightarrow{f} [\exists x. G^+(x)]}\ \exists R^* \qquad \textbf{no longer applicable} \left[\ \frac{\Gamma^- \xrightarrow{f} P^-}{\Gamma^- \xrightarrow{f} [\downarrow P^-]}\ \downarrow R\ \right]$$

$$\frac{\Gamma, [D_1^-] \xrightarrow{f} C^+}{\Gamma, [D_1^- \wedge D_2^-] \xrightarrow{f} C^+}\ \wedge L_1 \qquad \frac{\Gamma, [D_2^-] \xrightarrow{f} C^+}{\Gamma, [D_1^- \wedge D_2^-] \xrightarrow{f} C^+}\ \wedge L_2 \qquad \text{no rule for } \top L$$

$$\frac{\Gamma \xrightarrow{f} [G_1^+]}{\Gamma \xrightarrow{f} [G_1^+ \vee G_2^+]}\ \vee R_1 \qquad \frac{\Gamma \xrightarrow{f} [G_2^+]}{\Gamma \xrightarrow{f} [G_1^+ \vee G_2^+]}\ \vee R_2 \qquad \text{no rule for } \bot R$$

Figure 1: Forward chaining fragment of Horn logic

Say we focus on the first proposition in Γ_{eo}.

$$\dfrac{\dfrac{\dfrac{\vdots \qquad \vdots}{\Gamma_{\mathsf{eo}}, \mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{z})))) \xrightarrow{f} [\mathsf{even}(\mathsf{s}(N))] \quad \Gamma_{\mathsf{eo}}, \mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{z})))), [\uparrow\mathsf{odd}(N)] \xrightarrow{f} \mathsf{no}}\supset L}{\Gamma_{\mathsf{eo}}, \mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{z})))), [\mathsf{even}(\mathsf{s}(N)) \supset \uparrow\mathsf{odd}(n)] \xrightarrow{f} \mathsf{no}}\forall L}{\Gamma_{\mathsf{eo}}, \mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{z})))), [\forall n.\, \mathsf{even}(\mathsf{s}(n)) \supset \uparrow\mathsf{odd}(n)] \xrightarrow{f} \mathsf{no}}$$

Now we match $\mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{z}))))$ against $\mathsf{even}(\mathsf{s}(N))$ which succeeds with substitution $N = \mathsf{s}(\mathsf{s}(\mathsf{z}))$, which is applied globally to the partial proof which then looks as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_{\mathsf{eo}}, \mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{z})))) \xrightarrow{f} [\mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{z}))))]}\mathsf{id}^+ \qquad \dfrac{\vdots}{\Gamma_{\mathsf{eo}}, \mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{z})))), [\uparrow\mathsf{odd}(\mathsf{s}(\mathsf{s}(\mathsf{z}))))] \xrightarrow{f} \mathsf{no}}}\supset L}{\Gamma_{\mathsf{eo}}, \mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{z})))), [\mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{z}))))) \supset \uparrow\mathsf{odd}(n)] \xrightarrow{f} \mathsf{no}}\forall L}{\Gamma_{\mathsf{eo}}, \mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{z})))), [\forall n.\, \mathsf{even}(\mathsf{s}(n)) \supset \uparrow\mathsf{odd}(n)] \xrightarrow{f} \mathsf{no}}$$

In the right branch of the proof we now lose focus, and we have reached a stable sequent, adding another fact to the database.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_{\mathsf{eo}}, \mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{z})))) \xrightarrow{f} [\mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{z}))))]}\mathsf{id}^+ \qquad \dfrac{\dfrac{\vdots}{\Gamma_{\mathsf{eo}}, \mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{z})))), \mathsf{odd}(\mathsf{s}(\mathsf{z}))) \xrightarrow{f} \mathsf{no}}\uparrow L}{\Gamma_{\mathsf{eo}}, \mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{z})))), [\uparrow\mathsf{odd}(\mathsf{s}(\mathsf{s}(\mathsf{z}))))] \xrightarrow{f} \mathsf{no}}}\supset L}{\Gamma_{\mathsf{eo}}, \mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{z})))), [\mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{z}))))) \supset \uparrow\mathsf{odd}(n)] \xrightarrow{f} \mathsf{no}}\forall L}{\Gamma_{\mathsf{eo}}, \mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{z})))), [\forall n.\, \mathsf{even}(\mathsf{s}(n)) \supset \uparrow\mathsf{odd}(n)] \xrightarrow{f} \mathsf{no}}$$

This will continue until we add $\mathsf{odd}(\mathsf{z})$ and then add no to the database. At this point the goal looks like

$$\dfrac{\vdots}{\Gamma_{\mathsf{eo}}, \ldots, \mathsf{no} \xrightarrow{f} \mathsf{no}}$$

Recall that all atoms are positive, so we can now focus on the succedent and complete the proof.

$$\dfrac{\dfrac{}{\Gamma_{\mathsf{eo}}, \ldots, \mathsf{no} \xrightarrow{f} [\mathsf{no}]}\mathsf{id}^+}{\Gamma_{\mathsf{eo}}, \ldots, \mathsf{no} \xrightarrow{f} \mathsf{no}}\mathsf{focus}R$$

We can also see what happens if we focus on the "wrong" antecedent, that is, one that we cannot use with forward chaining.

$$\dfrac{\dfrac{\textbf{fails} \qquad\qquad \vdots}{\Gamma_{\mathsf{eo}}, \mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{z})))) \xrightarrow{f} [\mathsf{odd}(\mathsf{s}(N))] \quad \Gamma_{\mathsf{eo}}, \mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{z})))), [\uparrow\mathsf{even}(N)] \xrightarrow{f} \mathsf{no}}\,{\supset}L}{\dfrac{\Gamma_{\mathsf{eo}}, \mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{z})))), [\mathsf{odd}(\mathsf{s}(N)) \supset \uparrow\mathsf{even}(N)] \xrightarrow{f} \mathsf{no}}{\Gamma_{\mathsf{eo}}, \mathsf{even}(\mathsf{s}(\mathsf{s}(\mathsf{s}(\mathsf{z})))), [\forall n.\, \mathsf{odd}(\mathsf{s}(n)) \supset \uparrow\mathsf{even}(n)] \xrightarrow{f} \mathsf{no}}\,\forall L}$$

Experience shows that, generally, the reduction in the search space with forward chaining is not quite as drastic as with backward chaining. This is because there is only one goal (the succedent) but many clauses in the database (the antecedents). Nevertheless, there are many algorithms more easily described with forward chaining than with backward chaining. We will show one in Section 7 on unification. Forward chaining is capable of describing many algorithms very succinctly and in powerful ways!

6 Comparing Backward and Forward Chaining

Consider a simple transitivity example:

$$a, a \supset b, b \supset c \longrightarrow c$$

In backward chaining we polarize all atoms negatively, so we obtain

$$a^-, {\downarrow}a^- \supset b^-, {\downarrow}b^- \supset c^- \xrightarrow{f} c^-$$

where the (unary) shift operators binds tightest.

Without backward chaining there are different proofs. In particular, we could apply $\supset L$ to $a \supset b$, or to $b \supset c$. With backward chaining, there is only one possible proof: at each choice point, when we focus, only on possibility will succeed and the others will fail immediately. You should convince yourself that this is the case. We define

$\Gamma_0 = (a^-, \downarrow a^- \supset b^-, \downarrow b^- \supset c^-)$

$$\dfrac{\dfrac{\dfrac{}{\Gamma_0, [c^-] \xrightarrow{f} c^-}\,\mathsf{id}^- \qquad \dfrac{\dfrac{\dfrac{}{\Gamma_0, [b^-] \xrightarrow{f} b^-}\,\mathsf{id}^- \qquad \dfrac{\dfrac{\dfrac{}{\Gamma_0, [a^-] \xrightarrow{f} a^-}\,\mathsf{id}^-}{\Gamma_0 \xrightarrow{f} a^-}\,\mathsf{focus}L}{\Gamma_0 \xrightarrow{f} [\downarrow a^-]}\downarrow R}{\Gamma_0, [\downarrow a^- \supset b^-] \xrightarrow{f} b^-}\supset L}{\Gamma_0 \xrightarrow{f} b^-}\,\mathsf{focus}L}{\Gamma_0 \xrightarrow{f} [\downarrow b^-]}\downarrow R}{\Gamma_0, [\downarrow b^- \supset c^-] \xrightarrow{f} c^-}\supset L}{\Gamma_0 \xrightarrow{f} c^-}\,\mathsf{focus}L$$

Let's go back to our sequent

$$a, a \supset b, b \supset c \longrightarrow c$$

For forward chaining we polarize all the atoms positively, so we get:

$$a^+, a^+ \supset \uparrow b^+, b^+ \supset \uparrow c^+ \xrightarrow{f} c^+$$

Now we must focus on $a^+ \supset \uparrow b^+$ first, then $b^+ \supset \uparrow c^+$, then c^+. All other attempts at focusing will either fail, or conclude a fact that is already in the database. We abbreviate $\Gamma_1 = (a^+, a^+ \supset \uparrow b^+, b^+ \supset \uparrow c^+)$

$$\dfrac{\dfrac{\dfrac{a^+ \in \Gamma}{\Gamma_1 \xrightarrow{f} [a^+]}\,\mathsf{id}^+ \qquad \dfrac{\dfrac{\dfrac{}{\Gamma_1, [b^+] \xrightarrow{f} b^+}\,\mathsf{id}^+ \qquad \dfrac{\dfrac{\dfrac{}{\Gamma_1, b^+, c^+ \xrightarrow{f} [c^+]}\,\mathsf{id}^+}{\Gamma_1, b^+, c^+ \xrightarrow{f} c^+}\,\mathsf{focus}R}{\Gamma_1, b^+, [\uparrow c^+] \xrightarrow{f} c^+}\uparrow L}{\Gamma_1, b^+, [b^+ \supset \uparrow c^+] \xrightarrow{f} c^+}\supset L}{\Gamma_1, b^+ \xrightarrow{f} c^+}\,\mathsf{focus}L}{\Gamma_1, [\uparrow b^+] \xrightarrow{f} c^+}\uparrow L}{\Gamma_1, [a^+ \supset \uparrow b^+] \xrightarrow{f} c^+}\supset L}{\Gamma_1 \xrightarrow{f} c^+}\,\mathsf{focus}L$$

7 Unification

As a major and significant example of forward chaining, which is similar to many realistic applications of Datalog, we use *unification*. So far, we have just treated it informally, despite its complexity and logical significance.

We describe the unification algorithm by a set of rules concerning a predicate $t \doteq s$ for (first-order) terms t and s. This set of rules can be translated to a collection of propositions Γ_u where all atoms are positive. We assert a new equality, adding it as an antecedent, and then saturate the database. If it produces no, then the new equality is inconsistent with all the information we already had. Otherwise, the new saturated database represents the "solution" and shows consistency.

We begin with two rules that compare the function symbol at the head of the two terms. We write $\bar{t}$ for a sequence of terms.

$$\frac{f(\bar{t}) \doteq f(\bar{s})}{\bar{t} \doteq \bar{s}}\ \mathsf{con}_{=} \qquad \frac{f(\bar{t}) \doteq g(\bar{s}) \quad f \neq g}{\mathsf{no}}\ \mathsf{con}_{\neq}$$

The first rule expresses that if the term $f(\bar{t})$ is equal to the term $f(\bar{s})$ then their respective sequences of arguments must be equal. This means that function symbols are "uninterpreted": they are used as data constructors, not to stand for any mathematical functions.

The second rule expresses that if the data constructors are different, then the terms are not equal. In other words, if we know they are equal, then this is a contradiction.

These two rules also capture constants, since we think of a constant c as a function symbol $c()$, with the empty sequence of arguments.

Now we need four rules for comparing sequences of arguments.

$$\frac{(t, \bar{t}) \doteq (s, \bar{s})}{t \doteq s}\ \mathsf{seq}_{=1} \qquad \frac{(t, \bar{t}) \doteq (s, \bar{s})}{\bar{t} \doteq \bar{s}}\ \mathsf{seq}_{=2}$$

$$\frac{() \doteq (s, \bar{s})}{\mathsf{no}}\ \mathsf{seq}_{\neq 1} \qquad \frac{(t, \bar{t}) \doteq ()}{\mathsf{no}}\ \mathsf{seq}_{\neq 2}$$

Note that there is not rule for $() \doteq ()$, because such an equality, while true, contains no information to extract.

At this point we have enough rules to decide *equality*, but not yet enough to implement unification. Consider

$$f(X, X) \doteq f(c(), d())$$

This problem must fail, since X cannot be equal to $c()$ and $d()$ simultaneously, but the rules so far do not account for this. The simple device of stating symmetry and transitivity of equality will solve this particular problem.

$$\frac{t \doteq s}{s \doteq t}\ \mathsf{sym} \qquad \frac{t \doteq s \quad s \doteq r}{t \doteq r}\ \mathsf{trans}$$

Now we deduce:

$f(X, X) \doteq f(c(), d())$	given
$(X, X) \doteq (c(), d())$	by rule $\text{con}_{=}$
$X \doteq c()$	by rule $\text{seq}_{=1}$
$(X) \doteq (d())$	by rule $\text{seq}_{=2}$
$X \doteq d()$	by rule $\text{seq}_{=1}$
$c() \doteq X$	by rule sym
$c() \doteq d()$	by rule trans
no	by rule $\text{con}_{\neq}$

One could make these rules more efficient, for example, by restricting some terms in symmetry and transitivity to be variables.

At this point we have arrived at Prolog-style unification. Unfortunately, we know that this is unsound, because an equation such as

$$X \doteq f(X)$$

is not recognized as inconsistent by Prolog. We can incorporate this by adding some rules for the occurs-check that discover such inconsistencies.

$$\frac{X \doteq f(\bar{t})}{X \notin f(\bar{t})} \qquad \frac{X \notin X}{\text{no}}$$

$$\frac{X \notin f(\bar{t})}{X \notin \bar{t}} \qquad \frac{X \notin (t, \bar{t})}{X \notin t} \qquad \frac{X \notin (t, \bar{t})}{X \notin \bar{t}}$$

$$\frac{X \notin Y \quad Y \doteq t}{X \notin t}$$

The last rule is necessary to obtain contradictions from problems such as

$$X \doteq f(Y), Y \doteq g(X)$$

With a few optimization, these rules can be seen to define Huet's algorithm for (first-order) unification [Hue76]. This proceeds in two stages: in the first stage we saturate the equalities, and once they are saturated we perform the occurs-check. If implemented correctly, this will have complexity $O(n \log(n))$, where n is the size of the input problem. Robinson's original unification algorithm [Rob71] in contrast was exponential in the size of input, although it performed quite well in applications [CB83].

8 From Propositions to Rules of Inference

As we have seen in this lecture and also already in the last lecture, we can translate inference rules to propositions and then use either forward or backward chaining to specify an operational semantics for proof search.

Question: can we go the other way? That is, can we take a *proposition* and turn it into inference rules? Another way to pose the question: can we take advantage of the chaining semantics to *compile* program propositions into "big-step" inference rules so we don't have to play through focusing all the time?

Let's try with the example of the type of the S combinator:

$$(a \supset (b \supset c)) \supset ((a \supset b) \supset (a \supset c))$$

In order to prove this, we first apply inversion as far as we can and arrive at the sequent

$$a, a \supset b, a \supset (b \supset c) \longrightarrow c$$

Now we have to decide on a polarization. We'll first try all negative and then all positive.

Polarizing all atoms **negative** as in backward logic programming gives:

$$a^-, {\downarrow}a^- \supset b^-, {\downarrow}a^- \supset ({\downarrow}b^- \supset c^-) \xrightarrow{f} c^-$$

Which propositions could we focus on? Not on c^- in the succedent—there is no rule for that in the backward chaining fragment. But we can focus on each of the antecedents since they are all negative propositions. In each case we imagine what would happen if we focused on the proposition, not knowing the remaining antecedents Γ^- or the conclusion P^-. But note that since antecedents are persistent, all propositions in $\Gamma_0 = (a^-, {\downarrow}a^- \supset b^-, {\downarrow}a^- \supset ({\downarrow}b^- \supset c^-))$ will always be in the antecedent.

$$\dfrac{\dfrac{a^- = P^-}{\Gamma^-, [a^-] \xrightarrow{f} P^-}\,\mathsf{id}^-}{\Gamma^- \xrightarrow{f} P^-}\,\mathsf{focus}L$$

So for the focus on a^- to succeed, the right-hand side P^- must be equal to a^-. This derivation gives use the derived rule

$$\dfrac{}{\Gamma^- \xrightarrow{f} a^-}\,R_1$$

Focusing on the second proposition:

$$\dfrac{\dfrac{\dfrac{P^- = b^-}{\Gamma^-, [b^-] \xrightarrow{f} P^-}\,\mathsf{id}^- \quad \dfrac{\Gamma^- \xrightarrow{f} a^-}{\Gamma^- \xrightarrow{f} [{\downarrow}a^-]}\,{\downarrow}R}{\Gamma^-, [{\downarrow}a^- \supset b^-] \xrightarrow{f} P^-}\,{\supset}L}{\Gamma^- \xrightarrow{f} P^-}\,\mathsf{focus}L$$

We see $P^- = b^-$ and we have one stable subgoal that will be the premise of the derived rule.

$$\frac{\Gamma^- \xrightarrow{f} a^-}{\Gamma^- \xrightarrow{f} b^-}\, R_2$$

Finally, focusing on the third antecedent:

$$\dfrac{\dfrac{\dfrac{\dfrac{c^- = P^-}{\Gamma^-, [c^-] \xrightarrow{f} P^-}\,\mathsf{id}^- \quad \dfrac{\Gamma^- \xrightarrow{f} b^-}{\Gamma^- \xrightarrow{f} [\downarrow b^-]}\,\downarrow R}{\Gamma^-, [\downarrow b^- \supset c^-] \xrightarrow{f} P^-}\,\supset L \quad \dfrac{\Gamma^- \xrightarrow{f} a^-}{\Gamma^- \xrightarrow{f} [\downarrow a^-]}\,\downarrow R}{\Gamma^-, [\downarrow a^- \supset (\downarrow b^- \supset c^-)] \xrightarrow{f} P^-}\,\supset L}{\Gamma^- \xrightarrow{f} P^-}\,\mathsf{focus}L$$

We read off $P^- = c^-$ and the two stable sequents that are the premises of the derived rules:

$$\frac{\Gamma^- \xrightarrow{f} b^- \quad \Gamma^- \xrightarrow{f} a^-}{\Gamma^- \xrightarrow{f} c^-}\, R_3$$

Summarizing the three rules:

$$\frac{}{\Gamma^- \xrightarrow{f} a^-}\, R_1 \qquad \frac{\Gamma^- \xrightarrow{f} a^-}{\Gamma^- \xrightarrow{f} b^-}\, R_2 \qquad \frac{\Gamma^- \xrightarrow{f} b^- \quad \Gamma^- \xrightarrow{f} a^-}{\Gamma^- \xrightarrow{f} c^-}\, R_3$$

To see how they prove our original sequent

$$a^-, \downarrow a^- \supset b^-, \downarrow a^- \supset (\downarrow b^- \supset c^-) \xrightarrow{f} c^-$$

we first take all the negative propositions away from the antecedents. That's because *instead* of focusing on them, we should be using one of the derived rules R_1, R_2, R_3. Here is the resulting proof

$$\dfrac{\dfrac{\dfrac{}{\cdot \xrightarrow{f} a^-}\, R_1}{\cdot \xrightarrow{f} b^-}\, R_2 \quad \dfrac{}{\cdot \xrightarrow{f} a^-}\, R_1}{\cdot \xrightarrow{f} c^-}\, R_3$$

Of course, this is much shorter and more efficient than the proof using the explicit focusing rules, essentially because the focusing effort was only needed once in order to derive the rule, instead of once per usage.

Circling back: let's make all atoms **positive** as in forward-chaining:

$$a^+, a^+ \supset \uparrow b^+, a^+ \supset (b^+ \supset \uparrow c^+) \xrightarrow{f} c^+$$

This time, we cannot focus on a^+ in the antecedent, but on the other two propositions and also on the succedent. Let's do each in turn. Again, remember that $\Gamma_0 = (a^+ \supset \uparrow b^+, a^+ \supset (b^+ \supset \uparrow c^+))$ is a part of every antecedent in a proof. The succedent can be any positive proposition G^+.

$$\dfrac{\dfrac{\dfrac{a^+ \in \Gamma}{\Gamma \xrightarrow{f} [a^+]}\,\mathsf{id}^+ \quad \dfrac{\Gamma, b^+ \xrightarrow{f} G^+}{\Gamma, [\uparrow b^+] \xrightarrow{f} G^+}\,\uparrow L}{\Gamma, [a^+ \supset \uparrow b^+] \xrightarrow{f} G^+}\,\supset L}{\Gamma \xrightarrow{f} G^+}\,\mathsf{focus}L$$

Here, $a^+ \in \Gamma$ should be seen as a constraint, so we just write $\Gamma = (\Gamma', a^+)$ and obtain the rule

$$\dfrac{\Gamma', a^+, b^+ \xrightarrow{f} G^+}{\Gamma', a^+ \xrightarrow{f} G^+}\,S_1$$

For the second proposition:

$$\dfrac{\dfrac{\dfrac{a^+ \in \Gamma}{\Gamma \xrightarrow{f} [a^+]}\,\mathsf{id}^+ \quad \dfrac{\dfrac{b^+ \in \Gamma}{\Gamma \xrightarrow{f} [b^+]}\,\mathsf{id}^+ \quad \dfrac{\Gamma, c^+ \xrightarrow{f} G^+}{\Gamma, [\uparrow c^+] \xrightarrow{f} G^+}\,\uparrow L}{\Gamma, [b^+ \supset \uparrow c^+] \xrightarrow{f} G^+}\,\supset L}{\Gamma, [a^+ \supset (b^+ \supset \uparrow c^+)] \xrightarrow{f} G^+}\,\supset L}{\Gamma \xrightarrow{f} G^+}\,\mathsf{focus}L$$

Again, collecting membership constraints we see $\Gamma = (\Gamma', a^+, b^+)$ and get

$$\dfrac{\Gamma', a^+, b^+, c^+ \xrightarrow{f} G^+}{\Gamma', a^+, b^+ \xrightarrow{f} G^+}\,S_2$$

Finally, we can focus on the succedent:

$$\dfrac{\dfrac{c^+ \in \Gamma}{\Gamma \xrightarrow{f} [c^+]}\,\mathsf{id}^+}{\Gamma \xrightarrow{f} c^+}\,\mathsf{focus}R$$

which gives us

$$\frac{}{\Gamma', c^+ \xrightarrow{f} c^+}\, S_3$$

In summary:

$$\frac{\Gamma', a^+, b^+ \xrightarrow{f} G^+}{\Gamma', a^+ \xrightarrow{f} G^+}\, S_1 \qquad \frac{\Gamma', a^+, b^+, c^+ \xrightarrow{f} G^+}{\Gamma', a^+, b^+ \xrightarrow{f} G^+}\, S_2 \qquad \frac{}{\Gamma', c^+ \xrightarrow{f} c^+}\, S_3$$

With these three rules we can drop Γ_0 since their only purpose would be to focus on them—and that has been replaced by the derived rules S_1, S_2, S_3. Our big-step proof becomes:

$$\cfrac{\cfrac{\cfrac{}{a^+, b^+, c^+ \xrightarrow{f} c^+}\, S_3}{a^+, b^+ \xrightarrow{f} c^+}\, S_2}{a^+ \xrightarrow{f} c^+}\, S_1$$

Note that we have kept a^+ among the antecedents (instead of trying to turn it into a derived rule) since we cannot focus on a positive atom (or any positive proposition, for that matter) in the antecedent.

9 Polarization: A Brief Roadmap

A critical component in understanding the various fragments and operational interpretations we have seen is *polarization* [Lau99]. We started from the inversion strategy.

Negative Propositions are those with invertible right rules, that is, their right rule can be applied to the succedent without considering any other choice and search remains complete.

Positive Propositions are those with invertible left rules, that is, their left rule can be applied to the antecedent without considering any other choice and search remains complete.

In order for *every* proposition to have a polarized version we need the so-called *shift* operators $\uparrow$ and $\downarrow$ to embed opposite polarity propositions by explictly passing between the two classes of propositions. We get:

$$\begin{array}{llll} \text{Neg. props.} & A^- & ::= & P^- \mid B^+ \supset A^- \mid \forall x.\, A^-(x) \mid A_1^- \wedge A_2^- \mid \top \mid \uparrow B^+ \\ \text{Pos. props.} & B^+ & ::= & P^+ \mid B_1^+ \wedge B_2^+ \mid \top \mid \exists x.\, B^+(x) \mid B_1^+ \vee B_2^+ \mid \bot \mid \downarrow A^- \end{array}$$

Atoms can be either negative (P^-) or positive (P^+). During polarization we can choose the polarization of each atom p freely, but must assign the same polarity uniformly to

each occurrence of p. Conjunction and truth have invertible left and right rules, so they appear in both rows.

Chaining is the opposite of inversion: we focus on one particular negative antecedent or positive succedent and continue to apply rules only to the single proposition in focus until the focusing phase is interrupted by a shift, changing the polarity of the proposition.

Chaining by itself is complete for proof search as long as the shifts are restricted such that we only have $B^- ::= \cdots \mid \downarrow P^-$ and $A^- ::= \cdots \mid \uparrow P^+$. The language so restricted is (an insignificant extension of) Horn logic.

In case all atoms are negative, chaining is called *backward chaining* (also called *top-down logic programming*), which is a goal-directed proof search strategy and the foundation of Prolog [Kow88].

In case all atoms are positive, chaining is called *forward chaining* (also called *bottom-up logic programming* [NR91]), which is a saturation-based proof search strategy and the foundation of the programming language Datalog.

There is the possibility of allowing both positive and negative atoms, but the resulting mixed chaining logic programming language has never been deeply investigated, as far as I am aware. However, chaining is complete for this language, so it is a plausible candidate for an interesting and expressive language.

If we allow arbitrary polarized propositions, then chaining alone is insufficient: positive (non-atomic) propositions can show up in the antecedents, and negative (non-atomic) propositions in the succedent. When such a proposition is encountered, we apply inversion until we once again reach a *stable sequent* which is characterized with only negative propositions and positive atoms as antecedents, and positive propositions and negative atoms as succedents.

Focusing = chaining + inversion was first discovered by Andreoli [And92], with two caveats: (1) his propositions were not implicitly polarized, and (2) focusing was defined for *linear logic* [Gir87], which we will only see later in this course. However, focusing (and also the chaining-only fragment based on Horn logic) has been remarkably robust in that it applies to a large number of reasonable substructural, modal, and other logics, both intuitionistic and classical.

References

[And92] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):197–347, 1992.

[CB83] Jacques Corbin and Michel Bidoit. Rehabilitation of Robinson's unification algorithm. In *Information Processing 83*, volume 9, pages 909–914, 1983.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Hue76] Gérard Huet. *Résolution d'équations dans des langages d'ordre* $1, 2, \ldots, \omega$. PhD thesis, Université Paris VII, September 1976.

[Kow88] Robert A. Kowalski. The early years of logic programming. *Communications of the ACM*, 31(1):38–43, 1988.

[Lau99] Olivier Laurent. Polarized proof-nets: Proof-nets for LC. In J.-Y. Girard, editor, *Proceedings of the 4th International Conference on Typed Lambda Calculi and Applications (TLCA 1999)*, pages 213–227, L'Aquila, Italy, April 1999. Springer LNCS 1581.

[NR91] Jeff Naughton and Raghu Ramakrishnan. Bottom-up evaluation of logic programs. In J.-L. Lassez and G. Plotkin, editors, *Computational Logic. Essays in Honor of Alan Robinson*, pages 640–700. MIT Press, Cambridge, Massachusetts, 1991.

[Rob71] J. A. Robinson. Computational logic: The unification computation. *Machine Intelligence*, 6:63–72, 1971.