15-819K: Logic Programming

Lecture 21

Forward Chaining

Frank Pfenning

November 9, 2006

In this lecture we go from the view of logic programming as derived from inference rules for atomic propositions to one with explicit logical connectives. We have made this step before in order to describe the backward-chaining semantics of top-down logic programming as in Prolog. Here we instead describe the forward-chaining semantics of bottom-up logic programming. We use this to prove the correctness of an earlier example, and introduce an algorithm for unification as another example.

21.1 The Database as Context

When we try to formalize the semantics of bottom-up logic programming and saturation, one of the first questions to answer is how to represent the database from a logical point of view. Perhaps surprisingly at first, it ends up as a context of *assumptions*. We interpret a rule

$$\frac{P_1\ \mathit{true} \ldots P_n\ \mathit{true}}{P\ \mathit{true}}$$

as license to add the assumption $P\ \mathit{true}$ if we already have assumption $P_1\ \mathit{true}$ through $P_n\ \mathit{true}$. This means we actually have to turn the rule upside down to obtain the left rule

$$\frac{\Gamma, P_1\ \mathit{true}, \ldots, P_n\ \mathit{true}, P\ \mathit{true} \vdash C\ \mathit{true}}{\Gamma, P_1\ \mathit{true}, \ldots, P_n\ \mathit{true} \vdash C\ \mathit{true}}$$

Since in the case of (non-linear) intuitionistic logic, the context permits weakening and contraction, this step only represents progress if P is not

already in Γ or among the P_i. We therefore stop forward chaining if none of the inferences would make progress and say the database, represented as the context of assumptions, is saturated.

For now we will view logical deduction as ground deduction and return to the treatment of free variables in the next lecture. However, the inference rules to infer ground facts may still contain free variables. If we reify inference rules as logical implications and collect them in a fixed context Γ_0 we obtain the next version of the above rule (omitting '$true$'):

$$\frac{\begin{array}{ll}(\forall \mathbf{x}.\, P'_1 \wedge \ldots \wedge P'_n \supset P') \in \Gamma_0 & \mathrm{dom}(\theta) = \mathbf{x} \\ P'_i\theta = P_i \quad \text{for all } 1 \leq i \leq n & \mathrm{cod}(\theta) = \emptyset \qquad \Gamma, P_1, \ldots, P_n, P'\theta \vdash C\end{array}}{\Gamma, P_1, \ldots, P_n \vdash C}$$

To model saturation, we would restrict the rule to the case where $P'\theta \notin \Gamma, P_1, \ldots, P_n$. Moreover, the set of free variables in P' should be a subset of the variables in $P_1, \ldots, P_n$ so that $P'\theta$ is ground without having to guess a substitution term.

Note that the right-hand side C remains unchanged and unreferenced in the process of forward chaining. Later, when we are interested in combining forward and backward chaining we will have to pay some attention to the right-hand side. For now we leave the processes of creating the initial database and reading off an answer from the saturated database informal and concentrate on the forward chaining itself.

21.2 Even and Odd, Revisited

We revisit the two programs for checking if an given number is even in order to see how we can reason about the correctness of forward chaining programs. Recall first the definition of even, which has a natural backward chaining interpretation.

$$\frac{}{\mathsf{even}(\mathsf{z})}\ \mathsf{evz} \qquad \frac{\mathsf{even}(N)}{\mathsf{even}(\mathsf{s}(\mathsf{s}(N)))}\ \mathsf{evss}$$

Next, the rule for odd, which is our notation of the property of not being even.

$$\frac{\mathsf{odd}(\mathsf{s}(\mathsf{s}(N)))}{\mathsf{odd}(N)}$$

To see if $\mathsf{even}(n)$ we seed the database with $\mathsf{odd}(n)$, forward chain to saturation and then check if we have derived a contradiction, namely $\mathsf{odd}(\mathsf{z})$.

The correctness of the forward chaining program can be formulated as:

$\mathsf{even}(n)$ *true iff* $\mathsf{odd}(n)$ *true* $\vdash \mathsf{odd}(\mathsf{z})$ *true*

The intent is that we only use forward chaining rules for the second judgment, that is, we work entirely on the left except for an initial sequent to close off the derivation.

Theorem 21.1 *If* $\mathsf{even}(n)$ *true then* $\mathsf{odd}(n)$ *true* $\vdash \mathsf{odd}(\mathsf{z})$ *true.*

Proof: By induction on the deduction $\mathcal{D}$ of $\mathsf{even}(n)$ *true*

Case: $\mathcal{D} = \dfrac{}{\mathsf{even}(\mathsf{z})}$ where $n = \mathsf{z}$.

$\mathsf{odd}(\mathsf{z}) \vdash \mathsf{odd}(\mathsf{z})$ By hypothesis rule

Case: $\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}' \\ \mathsf{even}(n')\end{array}}{\mathsf{even}(\mathsf{s}(\mathsf{s}(n')))}$ where $n = \mathsf{s}(\mathsf{s}(n'))$.

$\mathsf{odd}(n') \vdash \mathsf{odd}(\mathsf{z})$ By ind. hyp. on $\mathcal{D}'$
$\mathsf{odd}(\mathsf{s}(\mathsf{s}(n'))), \mathsf{odd}(n') \vdash \mathsf{odd}(\mathsf{z})$ By weakening
$\mathsf{odd}(\mathsf{s}(\mathsf{s}(n'))) \vdash \mathsf{odd}(\mathsf{z})$ By forward chaining

□

The last step in the second case of this proof is critical. We are trying to show that $\mathsf{odd}(\mathsf{s}(\mathsf{s}(n'))) \vdash \mathsf{odd}(\mathsf{z})$. Applying the forward chaining rule reduces this to showing $\mathsf{odd}(\mathsf{s}(\mathsf{s}(n'))), \mathsf{odd}(n') \vdash \mathsf{odd}(\mathsf{z})$. But this follows by induction hypothesis plus weakening.

This establishes a weak form of completeness of the forward chaining program in the sense that if it saturates, then $\mathsf{odd}(\mathsf{z})$ must be present in the saturated database. A second argument shows that the database must always saturate (see Exercise 21.1), and therefore the forward chaining implementation is complete in the stronger sense of terminating and yielding a contradiction whenever n is even.

For the soundness direction we need to generalize the induction hypothesis, because $\mathsf{odd}(n)$ *true* $\vdash \mathsf{odd}(\mathsf{z})$ *true* will not match the situation even after a single step on the left. The problem is that a database such as $\mathsf{odd}(n), \mathsf{odd}(\mathsf{s}(n))$ will saturate and derive $\mathsf{odd}(\mathsf{z})$, but only one of the two numbers is even. Fortunately, it is sufficient to know that there exists some even number in the context, because we seed it with a singleton.[1]

[1] I am grateful to Deepak Garg for making this observation during lecture.

Lemma 21.2 *If* $\Gamma \vdash \mathsf{odd}(\mathsf{z})$ *where* Γ *consists of assumptions of the form* $\mathsf{odd}(\_)$*, then there exists an* $\mathsf{odd}(m) \in \Gamma$ *such that* $\mathsf{even}(m)$*.*

Proof: By induction on the structure of the given derivation $\mathcal{E}$.

Case: $\mathcal{E} = \dfrac{}{\Gamma', \mathsf{odd}(\mathsf{z}) \vdash \mathsf{odd}(\mathsf{z})}$ where $\Gamma = (\Gamma', \mathsf{odd}(\mathsf{z}))$.

$\mathsf{even}(\mathsf{z})$	By rule
Choose $m = \mathsf{z}$	$\mathsf{odd}(\mathsf{z}) \in \Gamma$

Case: $\mathcal{E} = \dfrac{\mathcal{E}'}{\Gamma', \mathsf{odd}(\mathsf{s}(\mathsf{s}(n))), \mathsf{odd}(n) \vdash \mathsf{odd}(\mathsf{z})}{\Gamma', \mathsf{odd}(\mathsf{s}(\mathsf{s}(n))) \vdash \mathsf{odd}(\mathsf{z})}$ where $\Gamma = (\Gamma', \mathsf{odd}(\mathsf{s}(\mathsf{s}(n))))$.

$\mathsf{even}(m')$ for some $\mathsf{odd}(m') \in (\Gamma', \mathsf{odd}(\mathsf{s}(\mathsf{s}(n))), \mathsf{odd}(n))$
By ind. hyp. on $\mathcal{E}'$

$\mathsf{odd}(m') \in (\Gamma', \mathsf{odd}(\mathsf{s}(\mathsf{s}(n))))$	Subcase
Choose $m = m'$	Since $\mathsf{odd}(m') \in \Gamma$

$\mathsf{odd}(m') = \mathsf{odd}(n)$	Subcase
$\mathsf{even}(n)$	By equality reasoning
$\mathsf{even}(\mathsf{s}(\mathsf{s}(n)))$	By rule
Choose $m = \mathsf{s}(\mathsf{s}(n))$	Since $\mathsf{odd}(\mathsf{s}(\mathsf{s}(n))) \in \Gamma$

□

21.3 Synchronous Atoms

When studying goal-directed search as the foundation of logic programming, we found that the notion of *focusing* gave us the right model for the search behavior of the connectives. Search is goal-directed if all the connectives are *asynchronous* so they can be decomposed eagerly as goals until an atomic goal is reached. Then we focus on one assumption and break this down until it matches the conclusion. The asynchronous fragment of intuitionistic logic is defined as follows.

$$A \quad ::= \quad P \mid A_1 \wedge A_2 \mid \top \mid A_2 \supset A_1 \mid \forall x.\, A$$

We summarize the rules of the focusing system in Figure 1. So far, has been the basis of backward chaining.

$$\frac{A \in \Gamma \quad \Gamma; A \ll P}{\Gamma \vdash P}\ \text{focus}L \qquad \frac{}{\Gamma; P \ll P}\ \text{id}R \qquad \begin{array}{c}\text{no rule for } P' \neq P \\ \Gamma; P' \ll P\end{array}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B}\ \supset R \qquad \frac{\Gamma; B \ll P \quad \Gamma \vdash A}{\Gamma; A \supset B \ll P}\ \supset L$$

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \wedge R \qquad \frac{\Gamma; A \ll P}{\Gamma; A \wedge B \ll P}\ \wedge L_1 \qquad \frac{\Delta; B \ll P}{\Gamma; A \wedge B \ll P}\ \wedge L_2$$

$$\frac{}{\Gamma \vdash \top}\ \top R \qquad \text{no } \top L \text{ rule}$$

$$\frac{\Gamma \vdash A \quad x \notin \text{FV}(\Gamma)}{\Gamma \vdash \forall x.\, A}\ \forall R \qquad \frac{\Delta; A(t/x) \ll P}{\Gamma; \forall x.\, A \ll P}\ \forall L$$

Figure 1: Focused Intuitionistic Logic; Asynchronous Fragment

Forward chaining violates the goal-directed nature of search, so we need to depart from the purely asynchronous fragment. Our change is minimalistic: we introduce only *synchronous atomic propositions* Q. For the sake of economy we also interpret inference rules

$$\frac{C_1\ \mathit{true} \ldots C_n\ \mathit{true}}{Q\ \mathit{true}}$$

without conjunction, writing them with iterated implication

$$\forall \mathbf{x}.\, C_1 \supset (C_2 \supset \ldots (C_n \supset Q))$$

instead. Here, C_i are all atoms, be they asynchronous (P) or synchronous (Q).

Since a goal can now be synchronous, we have the opportunity to focus on the right, if the right-hand side is a synchronous proposition (so far only Q). When a synchronous atomic proposition is in right focus, we succeed if the same proposition is in Γ; otherwise we fail.

$$\frac{\Gamma \gg Q}{\Gamma \vdash Q}\ \text{focus}R \qquad \frac{Q \in \Gamma}{\Gamma \gg Q}\ \text{id}L \qquad \begin{array}{c}\text{no rule for } Q \notin \Gamma \\ \Gamma \gg Q\end{array}$$

We also have to re-evaluate the left rule for implication.

$$\frac{\Gamma; B \ll P \quad \Gamma \vdash A}{\Gamma; A \supset B \ll P} \supset L$$

Strictly speaking, the focus should continue on both subformulas. However, when all propositions are asynchronous, we immediately lose right focus, so we short-circuited the step from $\Gamma \gg A$ to $\Gamma \vdash A$. Now that we have synchronous proposition, the rule needs to change to

$$\frac{\Gamma; B \ll P \quad \Gamma \gg A}{\Gamma; A \supset B \ll P} \supset L$$

and we add a rule

$$\frac{\Gamma \vdash A \quad A \neq Q}{\Gamma \gg A} \text{blur}R$$

A similar phenomenon arises on the left: when focused on a proposition that is asynchronous on the right (and not asynchronous on the left), we lose focus but we cannot fail as in the case of P.

$$\frac{\Gamma, Q \vdash P}{\Gamma; Q \ll A} \text{blur}L$$

Furthermore, all rules need to be generalized to allow either synchronous or asynchronous atoms on the right, which we write as C.

These considerations lead to the following rules, where have omitted the rules for conjunction, truth, and universal quantification. They are only changed in that the conclusion in the left rules can now be an arbitrary C, that is, a P or Q.

$$\frac{A \in \Gamma, A \neq Q \quad \Gamma; A \ll C}{\Gamma \vdash C} \text{focus}L \qquad \frac{}{\Gamma; P \ll P} \text{id}R \qquad \begin{array}{c}\text{no rule for } P \neq C \\ \Gamma; P \ll C\end{array}$$

$$\frac{\Gamma, A_1 \vdash A_2}{\Gamma \vdash A_1 \supset A_2} \supset R \qquad \frac{\Gamma; A_1 \ll C \quad \Gamma \gg A_2}{\Gamma; A_2 \supset A_1 \ll C} \supset L$$

$$\frac{\Gamma \gg Q}{\Gamma \vdash Q} \text{focus}R \qquad \frac{Q \in \Gamma}{\Gamma \gg Q} \text{id}L \qquad \begin{array}{c}\text{no rule for } Q \notin \Gamma \\ \Gamma \gg Q\end{array}$$

$$\frac{\Gamma \vdash A \quad A \neq Q}{\Gamma \gg A} \text{blur}R \qquad \frac{\Gamma, Q \vdash C}{\Gamma; Q \ll C} \text{blur}L$$

This system is sound and complete with respect to ordinary intuitionistic logic, no matter which predicates are designated as synchronous and asynchronous. As before, this can be proved via a theorem showing the admissibility of various forms of cut for focused derivations. However, it is important that all occurrences of a predicate have the same status, otherwise the system may become incomplete.

21.4 Pure Forward Chaining

In pure forward chaining all atomic predicates are considered synchronous. What kind of operational semantics can we assign to this system? In a situation $\Gamma \vdash A$ we first break down A (which, after all, is asynchronous) until we arrive at Q. At this point we have to prove $\Gamma \vdash Q$. There are two potentially applicable rules, focus$R$ and focus$L$. Let us consider these possibilities.

The focus$R$ rule will always be immediately preceded by id$L$, which would complete the derivation. So it comes down to a check if Q is in Γ.

The focus$L$ rule focuses on a program clause or fact in Γ. Consider the case of a propositional Horn clause $Q_1 \supset \ldots \supset Q_n \supset Q'$. We apply $\supset L$ rule, with premisses $\Gamma; Q_2 \supset \ldots \supset Q_n \supset Q' \ll C$ and $\Gamma \gg Q_1$. The only rule applicable to the second premiss is id$L$, so Q_1 must be in the database Γ. We continue this process and note that $Q_2, \ldots, Q_n$ must all be in the database Γ already. In the last step the first premiss is $\Gamma; Q' \ll Q$ which transitions to $\Gamma, Q' \vdash Q$.

In summary, applying a left focus rule to a clause $Q_1 \supset \ldots \supset Q_n \supset Q'$ reduces the sequent $\Gamma \vdash Q$ to $\Gamma, Q' \vdash Q$ if $Q_i \in \Gamma$ for $1 \leq i \leq n$. This is exactly the forward chaining step from before.

Overall, we can either right focus to see if the goal Q has already been proved, or apply a forward chaining step. A reasonable strategy is to saturate (repeated applying left focus until we make no more progress) and then apply right focus to see if Q has been deduced. The failure of left focus can be enforced by replacing the blur$L$ rule by

$$\frac{\Gamma, Q \vdash C \quad Q \notin \Gamma}{\Gamma; Q \ll C}\ \text{blur}L'.$$

This remains complete due to the admissibility of contraction, that is, if $\Gamma, A, A \vdash C$ then $\Gamma, A \vdash C$. This analysis also suggests that the left rule for implication is more perspicuous if written with the premisses reversed in

case the goal on the right-hand side is synchronous.

$$\frac{\Gamma \gg A_1 \quad \Gamma; A_2 \ll Q}{\Gamma; A_1 \supset A_2 \ll Q}\ \supset L$$

In the case of a Horn clause this means we first check if $A_1 = Q_1$ is in Γ and then proceed to analyze the rest of the clause.

21.5 Matching and Ground Bottom-Up Logic Programming

The analysis of forward chaining becomes only slightly more complicated if we allow the Horn clauses $\forall \mathbf{x}.\, Q_1 \supset \ldots \supset Q_n \supset Q'$ to be quantified as long as we restrict the variables in the head Q' of the clause to be a subset of the variables in $Q_1, \ldots, Q_n$. Then right focusing must return a substitution θ and we have the following rules, specialized to the Horn fragment with only synchronous atoms.

$$\frac{A \in \Gamma, A \neq Q' \quad \Gamma; A \ll Q}{\Gamma \vdash Q}\ \text{focus}L$$

$$\frac{\Gamma \gg Q_1 \mid \theta \quad \Gamma; A_2\theta \ll Q}{\Gamma; Q_1 \supset A_2 \ll Q}\ \supset L \qquad \frac{\Gamma; A(X/x) \ll Q \quad X \notin \text{FV}(\Gamma, A, Q)}{\Gamma; \forall x.\, A \ll Q}\ \forall L$$

$$\frac{\Gamma \gg Q \mid (\cdot)}{\Gamma \vdash Q}\ \text{focus}R \qquad \frac{Q' \in \Gamma \quad Q' = Q\theta}{\Gamma \gg Q \mid \theta}\ \text{id}L \qquad \begin{array}{c}\text{no rule if no such } Q', \theta \\ \Gamma \gg Q \mid \theta\end{array}$$

$$\frac{\Gamma, Q' \vdash Q \quad Q' \notin \Gamma}{\Gamma; Q' \ll Q}\ \text{blur}L'$$

By the restriction on free variables, the Q' in the blur$L$ cannot contain any free variables. The blur$R$ rule cannot apply in the Horn fragment. We also assumed for simplicity that the overall goal Q in $\Gamma \vdash Q$ is closed.

So on the Horn fragment under the common restriction that all variables in the head of a clause must appear in its body, bottom-up logic programming corresponds exactly to treating all atomic propositions as synchronous. The operational semantics requires matching, instead of full unification, because the database Γ consists only of ground facts.

21.6 Combining Forward and Backward Chaining

The focusing rules for the language given ealier (that is, all asynchronous connectives plus synchronous atoms) are sound and complete with respect

to the truth judgment and therefore a potential basis for combining forward and backward chaining. Backward chaining applies to asynchronous atoms and forward chaining to synchronous atoms. For the propositional case this is straightforward, but in the case of quantifiers it becomes difficult. We consider quantifiers in the next lecture and the propositional case briefly here, by example.

We look at the behavior of

$$C_1, C_1 \supset C_2, C_2 \supset C_3 \vdash C_3$$

for various assignment of C_1, C_2, and C_3 as synchronous or asynchronous. Interestingly, no matter what we do, in the focusing system there is exactly one proof of this sequent.

If all predicates are asynchronous,

$$P_1, P_1 \supset P_2, P_2 \supset P_3 \vdash P_3,$$

we must focus on $P_2 \supset P_3$. One step of backward chaining generates the subgoal

$$P_1, P_1 \supset P_2, P_2 \supset P_3 \vdash P_2.$$

Now we must focus on $P_1 \supset P_2$ and obtain the subgoal

$$P_1, P_1 \supset P_2, P_2 \supset P_3 \vdash P_1$$

which succeeds by focusing on P_1 on the left. No other proof paths are possible.

If all predicates are synchronous,

$$Q_1, Q_1 \supset Q_2, Q_2 \supset Q_3 \vdash Q_3,$$

we must focus on $Q_1 \supset Q_2$ because only Q_1 is directly available in the context. Focusing on Q_1 is prohibited because it is synchronous on the right, and therefore asynchronous on the left. We obtain the subgoal

$$Q_1, Q_2, Q_1 \supset Q_2, Q_2 \supset Q_3 \vdash Q_3.$$

Now we must focus on $Q_2 \supset Q_3$. Focusing on $Q_1 \supset Q_2$ would fail since Q_2 is already in the context, and focusing on Q_3 on the right would fail since Q_3 is not yet in the context. This second forward chaining step now generates

$$Q_1, Q_2, Q_3, Q_1 \supset Q_2, Q_2 \supset Q_3 \vdash Q_3.$$

At this point we can only focus on the right, which completes the proof.

Now consider a mixed situation

$$Q_1, Q_1 \supset P_2, P_2 \supset Q_3 \vdash Q_3.$$

Focusing on $Q_1 \supset P_2$ will fail, because the right-hand side does not match P_2. The only successful option is to focus on $P_2 \supset Q_3$ which generates two subgoals

$$Q_1, Q_1 \supset P_2, P_2 \supset Q_3 \vdash P_2$$

and

$$Q_1, Q_3, Q_1 \supset P_2, P_2 \supset Q_3 \vdash Q_3.$$

For the first we focus on $Q_1 \supset P_2$ and finish, for the second we focus on the right and complete the proof.

You are asked to consider other mixed situations in Exercise 21.2.

21.7 Beyond the Horn Fragment

The focusing rules are more general than the Horn fragment, but there is no particular difficulty in adopting the operational semantics as presented here, as long as we exclude the difficulties that arise due to quantification. In a later lecture we will consider adding other synchronous connectives (falsehood, disjunction, and existential quantification).

Here, we make only one remark about conjunction and truth. In case of intuitionistic logic they can safely be considered to be synchronous *and* asynchronous. This means we can add the rules

$$\frac{\Gamma \gg A_1 \quad \Gamma \gg A_2}{\Gamma \gg A_1 \wedge A_2} \qquad \frac{}{\Gamma \gg \top}$$

This allows us to write Horn clauses as $\forall \mathbf{x}.\, Q_1 \wedge \ldots \wedge Q_n \supset Q'$ without any change in the operational behavior compared with $\forall \mathbf{x}.\, Q_1 \supset \ldots \supset Q_n \supset Q'$.

We could similarly add some left rules, but this would require an additional judgment form to break down connectives that are asynchronous on the left, which we postpone to a later lecture.

21.8 Unification via Bottom-Up Logic Programming

We close this lecture with another example of bottom-up logic programming, namely and implementation of unification.

The implementation of unification by bottom-up logic programming illustrates a common type of program where saturation can be employed to great advantage. This is similar to the even example where we reason by contradication for a decidable predicate. Here, the predicate is *non-unifiability* of two terms s and t, as well as non-unifiability of sequences of terms $\mathbf{s}$ and $\mathbf{t}$ as an auxiliary predicate. In order to show that two terms are non-unifiable we assume they are unifiable, saturate, and then test the resulting saturated database for inconsistent information. We write $s \doteq t$ and $\mathbf{s} \doteq \mathbf{t}$ for the two equality relations.

Since there are a number of ways a contradiction can arise, we also introduce an explicit proposition contra to indicate a contradiction. In a later lecture we will see that we can in fact use $\bot$ with a sound logical interpretation, but that would requires us to go beyond the current setting.

The way to think about the rules is via the laws governing equality. We start with symmetry, transitivity, and reflexivity. Reflexivity actually gives us no new information (we already know the two terms are equal), so there is no forward rule for it.

$$\frac{s \doteq t}{t \doteq s} \qquad \frac{s \doteq t \quad t \doteq r}{s \doteq r}$$

Not all instances of transitivity are actually required (see Exercise 21.3); restricting it can lead to an improvement in the running time of the algorithm. Next, the congruence rules for constructors and sequences.

$$\frac{f(\mathbf{s}) \doteq f(\mathbf{t})}{\mathbf{s} \doteq \mathbf{t}} \qquad \frac{(s, \mathbf{s}) \doteq (t, \mathbf{t})}{s \doteq t} \qquad \frac{(s, \mathbf{s}) \doteq (t, \mathbf{t})}{\mathbf{s} \doteq \mathbf{t}} \qquad \begin{array}{c}(\cdot) \doteq (\cdot) \\ \text{no rule}\end{array}$$

There is no rule for $(\cdot) \doteq (\cdot)$ since this fact does not yield any new information. However, we have rules that note contradictory information by concluding contra.

$$\frac{f(\mathbf{s}) \doteq g(\mathbf{t}) \quad f \neq g}{\text{contra}} \qquad \frac{(\cdot) \doteq (t, \mathbf{t})}{\text{contra}} \qquad \frac{(s, \mathbf{s}) \doteq (\cdot)}{\text{contra}}$$

Even in the presence of variables, the rules so far will saturate, closing a given equality under its consequences. We consider $f(x, g(b)) \doteq f(a, g(x))$ as an example and show the generated consequences, omitting any identi-

ties $t \doteq t$ and intermediate steps relating a sequences to their elements.

$f(x, g(b)) \doteq f(a, g(x))$	Assumption
$f(a, g(x)) \doteq f(x, g(b))$	Symmetry
$x \doteq a$	Congruence
$g(b) \doteq g(x)$	Congruence
$a \doteq x$	Symmetry
$g(x) \doteq g(b)$	Symmetry
$b \doteq x$	Congruence
$x \doteq b$	Symmetry
$b \doteq a$	Transitivity
$a \doteq b$	Symmetry
contra	Clash $b \neq a$

The only point missing from the overall strategy to is to generate a contradiction due to a failure of the occurs-check. For this we have two new forms of propositions, $x \notin t$ and $x \notin \mathbf{t}$ which we use to propagate occurrence information.

$$\frac{x \doteq f(\mathbf{t})}{x \notin \mathbf{t}}$$

$$\frac{x \notin (t, \mathbf{t})}{x \notin t} \qquad \frac{x \notin (t, \mathbf{t})}{x \notin \mathbf{t}} \qquad \begin{array}{c} x \notin (\cdot) \\ \text{no rule} \end{array}$$

$$\frac{x \notin f(\mathbf{t})}{x \notin \mathbf{t}} \qquad \frac{x \notin x}{\mathsf{contra}} \qquad \begin{array}{c} x \notin y, x \neq y \\ \text{no rule} \end{array}$$

$$\frac{x \notin t \quad t \doteq s}{x \notin s}$$

The last rule is necessary so that, for example, the set $x \doteq f(y), y \doteq f(x)$ can be recognized as contradictory.

Let us apply the McAllester meta-complexity result. In the completed database, any two subterms of the original unification problem may be set equal, so we have $O(n^2)$ possibilities. Transitivity has $O(n^3)$ prefix firings, so a cursory analysis yields $O(n^3)$ complexity. This is better than the exponential complexity of Robinson's algorithm, but still far worse then the

linear time lower bound. Both the algorithm and the analysis can be refined in a number of ways. For example, we can restrict the uses of symmetry and transitivity to obtain better bounds, and we can postpone the use of non-occurrence to a second pass over a database saturated by the other rules.

This form of presentation of unification has become standard practice. It does not explicitly compute a unifier, but for unifiable terms it computes a kind of graph where the nodes in the original term (when viewed as a dag) are the nodes, and nodes are related by explicit equalities. From this a unifier can be read off by looking up the equivalence classes of the variables in the original unification problem.

Another nice property of unification, shared by many other saturation-based algorithms, is that it is *incremental*. This means that equations can be added one by one, and the database saturated every time, starting from the previously saturated one. If the equations ever become contradictory, contra is derived.

Here is a sketch how this might be used in the implementation of a logic programming engine. We have a constraint store, initially empty. Whenever logic programming search would call unification to obtain a unifier, we instead assume the equation into the database and saturate it. If we obtain a contradiction, unification fails and we have to backtrack. If not, we continue with the resulting constraint store. A neat thing about this implementation is that we never explicitly need to compute and apply a most general unifier: any goal we consider is always with respect to a saturated (and therefore consistent) set of equations.

21.9 Historical Notes

Although the notion that atoms may be synchronous or asynchronous, at the programmer's discretion, is relatively old [1], I believe that the observation connecting forward chaining to synchronous atoms is relatively recent [2], and was made in the setting of general theorem proving. An alternative approach to combining forward and backward chaining in logic programming using a monad [5] will be the subject of a later lecture.

The view of unification as a forward reasoning process similar to the one described here is due to Huet [3], although he maintained equivalence classes of terms much more efficiently than our naive specification, using the well-known union-find algorithm to arrive at an almost linear algorithm. Huet's basic idea was later refined by Martelli and Montanari [6] and in a different way by Paterson and Wegman [7] to obtain linear time

algorithms for unification.

The idea to handle unification problems via a store of constraints, to be updated and queried during computation, goes back to constraint logic programming [4]. It was elevated to logical status by Saraswat [8], although the connection to focusing, forward and backward chaining was not recognized at the time.

21.10 Exercises

Exercise 21.1 *Prove that the database initialized with* $\mathsf{odd}(n)$ *for some* n *and closed under forward application of the rule*

$$\frac{\mathsf{odd}(\mathsf{s}(\mathsf{s}(N)))}{\mathsf{odd}(N)}$$

will always saturate in a finite number of steps.

Exercise 21.2 *Consider two other ways to assign atoms to be synchronous or asynchronous for the sequent*

$$C_1, C_1 \supset C_2, C_2 \supset C_3 \vdash C_3$$

from Section 21.6 and show that there exists a unique proof in each case.

Exercise 21.3 *Improve the bottom-up unification algorithm by analyzing more carefully which instances of symmetry and transitivity are really needed. You may ignore the occurs-check, which we assume could be done in a second pass after the other rules saturate. What kind of McAllester complexity does your analysis yield?*

21.11 References

[1] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *Journal of Logic and Computation*, 2(3):297–347, 1992.

[2] Kaustuv Chaudhuri, Frank Pfenning, and Greg Price. A logical characterization of forward and backward chaining in the inverse method. In U. Furbach and N. Shnakar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06)*, pages 97–111, Seattle, Washington, August 2006. Springer LNCS 4130.

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

[4] Joxan Jaffar and Jean-Louis Lassez. Constraint logic programming. In *Proceedings of the Fourteenth Annual ACM Symposium on Principles of Programming Languages*, pages 111–119, Munich, Germany, January 1987. ACM Press.

[5] Pablo López, Frank Pfenning, Jeff Polakow, and Kevin Watkins. Monadic concurrent linear logic programming. In A.Felty, editor, *Proceedings of the 7th International Symposium on Principles and Practice of Declarative Programming (PPDP'05)*, pages 35–46, Lisbon, Portugal, July 2005. ACM Press.

[6] Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems*, 4(2):258–282, April 1982.

[7] M. S. Paterson and M. N. Wegman. Linear unification. *Journal of Computer and System Sciences*, 16(2):158–167, April 1978.

[8] Vijay Saraswat. *Concurrent Constraint Programming*. MIT Press, 1991. ACM Doctoral Dissertation Award Series.