15-819K: Logic Programming

Lecture 23

Linear Monadic Logic Programming

Frank Pfenning

November 16, 2006

In this lecture we extend the observations about forward and backward chaining from Horn logic to a rich set of linear connectives. In order to control the interaction between forward and backward chaining we use a *monad* to separate asynchronous from synchronous connectives. The result is the logic programming language LolliMon, which we illustrate with some examples.

23.1 Separating Asynchronous and Synchronous Connectives

The observations we have made about focusing, forward chaining, and backward chaining apply to various logics, including linear logic, which was indeed the origin of the notion of focusing. But it is difficult to combine forward and backward chaining and obtain a satisfactory operational semantics.

The problem can be broadly described as follows. First, if all connectives are asynchronous we obtain a clear and non-deterministically complete backward chaining semantics. Second, on the Horn fragment with only synchronous atoms we obtain a satisfactory forward chaining semantics with saturation. However, adding more synchronous connectives, or mixing synchronous with asynchronous connectives introduces a lot of uncontrolled non-determinism into the operational reading of programs. We approach general theorem proving, leaving the predictable operational behavior of Prolog behind.

We do not rule out that it may be possible to obtain a satisfactory semantics, but in this lecture we pursue a different path. We can control the interaction between asynchronous and synchronous connectives and

therefore between backward and forward chaining by creating a firewall between them called a *monad*. From the logical point of view a monad is a so-called *lax modal operator*. We will explain its laws below. For now we just note that we write $\{-\}$ for the modal operator.

In a complex linear logic proposition we can identify the places where we change from asynchronous to synchronous connectives or vice versa. At the first transition we require the explicit monadic firewall; the second transition is simply an inclusion. There is one exception, namely in a linear implication we switch sides (if the implication appears on the right the antecedent appears on the left), so we have to switch from asynchronous to synchronous propositions to remain in the same phase of decomposition.

$$\begin{array}{lrcl} \text{Asynch} & A & ::= & P \mid A_1 \mathbin{\&} A_2 \mid \top \mid S_1 \multimap A_2 \mid A_1 \supset A_2 \mid \forall x.\, A \mid \{S\} \\ \text{Synch} & S & ::= & Q \mid S_1 \oplus S_2 \mid \mathbf{0} \mid S_1 \otimes S_2 \mid \mathbf{1} \mid !A \mid \exists x.\, S \mid A \end{array}$$

Recall that P stands for asynchronous atoms and Q for synchronous atoms. We observe that the exponential modality of linear logic, !, has a special status: $!A$ is synchronous but the subformula does not continue the synchronous phase, but is asynchronous. Similarly, $A_1 \supset A_2$ is asynchronous, but A_1 in the antecedent which we might expect to be synchronous is in fact asynchronous. We will briefly explain this phenomenon later in the lecture.

We have been inclusive here, omitting only $S_1 \otimes! A_2$ and $\mathbf{s} \doteq \mathbf{t}$ which were necessary for residuation. The first is easy to add or define. Explicit equality is omitted here because in this lecture we concentrate on propositional and modal aspects of computation (see Exercise 23.2).

23.2 The Lax Modality

In order for the lax modality to have the right effect in separating the forward and backward chaining phases, we need to give it a proper logical foundation. In this section we just present this logical meaning; later we justify its role in the operational semantics. Since the semantics here is fully general, and does not depend on the division into asynchronous and synchronous operators, we just use B and C to stand for arbitrary proposition in linear logic, augmented by the lax modality.

In addition to the usual judgment $B\ true$ we have a new judgment on propositions, $B\ lax$. This can given be many different readings. Let us think of it for now as "*B is true subject to some (abstract) constraint*". This means

that $B\ lax$ is a *weaker* judgment then $B\ true$.

$$\frac{\Delta \Vdash B\ true}{\Delta \Vdash B\ lax}\ \text{lax}$$

This is a rule concerned only with the judgment, not with the proposition which remains unchanged. Of course, this rule is not invertible, or truth and lax truth would coincide. As in the prior sections on linear logic we omit the unrestricted context from the rules since it is generally just propagated from conclusion to premisses.

The second property is a form of the cut principle: if we can derive $B\ lax$ we are allowed to assume $B\ true$, but only if we are deriving a lax conclusion. This is so because deriving a lax conclusion permits a constraint.

If $\Delta_B \Vdash B\ lax$ *and* $\Delta_C, B\ true \Vdash C\ lax$ *then* $\Delta_C, \Delta_B \Vdash C\ lax$.

This should always hold, which means it is admissible when formulated as a rule of inference. This is not the case if we changed the conclusion to $C\ true$ since then we could derive the converse of the first rule, and truth and lax truth collapse.

The right and left rules for the lax modality are now straightforward.

$$\frac{\Delta \Vdash B\ lax}{\Delta \Vdash \{B\}\ true}\ \{-\}R \qquad \frac{\Delta, B\ true \Vdash C\ lax}{\Delta, \{B\}\ true \Vdash C\ lax}\ \{-\}L$$

Again, the conclusion of the $\{-\}L$ rule is restricted to the form $C\ lax$; allowing $C\ true$ here would be incorrect.

However, the other left rules in the sequent calculus must now permit an arbitrary judgment J on the right-hand side, which could be either $C\ true$ or $C\ lax$ where previously in could only be $C\ true$. The prior proof of the admissibility of cut and the identity principle can be easily extended to this fragment.

In terms of focusing, the modal operator looks at first a bit odd. In a deduction (read bottom-up), we go from $\{B\}\ true$ to $B\ lax$ and from there to $B\ true$.

On the right, the first transition is asynchronous: we can always strip the modal operator. The second transition is synchronous: we may have to wait for a left rule to be applied (specifically: to go from $\{C\}\ true$ to $C\ true$ for some C) before we can prove B true.

On the left, the first transition is synchronous: we cannot remove the modal operator until the right-hand side has the form $C\ lax$. The second

one, however, is asynchronous, because $B\ true$ is a stronger assumption than $B\ lax$.

We can also see this from the proof of the identity principle.

$$\dfrac{\dfrac{\dfrac{B\ true \Vdash B\ true}{B\ true \Vdash B\ lax}\ \text{lax}}{\{B\}\ true \Vdash B\ lax}\ \{-\}L}{\{B\}\ true \Vdash \{B\}\ true}\ \{-\}R$$

There is a "silent" transition on the left-hand side from $B\ lax$ to $B\ true$. Because the lax judgment is asynchronous as an assumption, we never need to consider it on the left-hand side.

23.3 The Exponential Modality Revisited

Above we have seen that the lax modality internalizes the judgment of lax truth, which is weaker than truth. Conversely, the exponential modality $!B$ of linear logic internalizes necessary truth (written $B\ valid$), which is stronger than truth. Recall first the judgmental rule, given here with a more general right hand-side J which could be $C\ true$ or $C\ lax$.

$$\dfrac{\Gamma, B\ valid; \Delta, B\ true \Vdash J}{\Gamma, B\ valid; \Delta \Vdash J}\ \text{copy}$$

The cut-like principle encodes that B is valid if it is true, without using any any truth assumptions.

If $\Gamma; \cdot \Vdash B\ true$ *and* $\Gamma, B\ valid; \Delta \Vdash J$ *then* $\Gamma; \Delta \Vdash J$.

When we internalize validity as a modal connective we obtain the following right and left rules.

$$\dfrac{\Gamma; \cdot \Vdash B\ true}{\Gamma; \cdot \Vdash\ !B\ true}\ !R \qquad \dfrac{\Gamma, B\ valid; \Delta \Vdash J}{\Gamma; \Delta, !B\ true \Vdash J}\ !L$$

We see that the left rule expresses that if $!B\ true$ then $B\ valid$. On the right-hand side, however, we have a silent transition from $!B\ true$, through $B\ valid$ to $B\ true$.

From this we can easily derive the focusing behavior. $!B$ is synchronous on the right, but the judgment $B\ valid$ is asynchronous. We therefore never need to explicitly consider $B\ valid$ as a conclusion.

Conversely, as an assumption $!B$ is asynchronous and can be decomposed eagerly to $B\ valid$. However, the assumption $B\ valid$ is synchronous (no matter what B is), so we need keep it as as assumption that we can focus on later.

This can also be seen from the proof of the identity principle.

$$\dfrac{\dfrac{\dfrac{B\ valid; B\ true \Vdash B\ true}{B\ valid; \cdot \Vdash B\ true}\ \text{copy}}{B\ valid; \cdot \Vdash\ !B\ true}\ !R}{\cdot; !B\ true \Vdash\ !B\ true}\ !L$$

In summary, the judgment forms $B\ lax$ and $B\ valid$ interact with focusing in the sense that $B\ lax$ is synchronous as a conclusion (no matter what B is) and $B\ valid$ is asynchronous as a conclusion (again, no matter what B is). This means $B\ valid$ need never explicitly be considered as a conclusion (it immediately becomes $B\ true$). Conversely $B\ valid$ is synchronous as an assumption and $B\ lax$ is asynchronous as an assumption. This emans $B\ lax$ need never explicitly be considered as an assumption (it immediately morphs into $B\ true$).

The asynchronous behaviors are only correct because of the modal restrictions: $B\ valid$ would only appear on the right if the linear context is empty, and $B\ lax$ would only appear on the left if the conclusion is $C\ lax$ for some C.

23.4 The Lax Modality and Search

We now preview the operational behavior of LolliMon, which can be divided into multiple phases derived from focusing. We will mostly build on the intuition for forward and backward chaining from the preceding lectures, except that linear forward chaining is more complex than just saturation.

Asynchronous goal decomposition. Starting with an asynchronous goal $A\ true$, we decompose it eagerly until we we come to either $\{S\}\ true$ or $P\ true$.

Backward chaining. If the goal is now of the form $P\ true$, we can focus on an assumption and decompose it. However, the ultimate head that we focus on must match P; if we reach $\{S\}$ instead we must fail because the

left rule for $\{-\}$ is not applicable when the conclusion is $P\ true$. This turns out to be a crucial advantage of having the lax modality, because if the head were an unprotected S we could not fail, but would now have to asynchronously decompose S.

Subgoals created during backward chaining are then solved in turn, as usual for backward chaining.

Forward chaining. If the goal on the right is $\{S\}\ true$ we exploit the fact that the lax modality is asynchronous and reduce it to $S\ lax$. At this point we could either focus on the left or focus on the right.

The operational semantics now prescribes a forward chaining phase. In the presence of linearity, this is intentially not complete (see the discussion of concurrency below). Nevertheless, we focus on an assumption and break it down until the head is either $P\ true$ or $\{S'\}\ true$. In the first case we fail, because we are focusing on $P\ true$ and it does not match the conclusion (which is lax). In the case $\{S'\}\ true$ we reduce it to $S'\ true$ which is then asynchronously decomposed until we can focus on a new assumption. The left rule for the lax modality is applicable here since the right-hand side is of the form $S\ lax$.

We continue forward chaining until we reach both *saturation* and *quiescence*. Saturation means that any forward step will not add any new assumption to Γ or Δ. Quiescence means we cannot focus on any linear assumption.

Once we have reached saturation and quiescence, focusing on the left is no longer possible, so we focus on the right, $S\ lax$, which becomes $S\ true$ under focus and is decomposed until we reach an asynchronous goal A to restart a new phase.

We now consider each phase in turn, writing out the relevant judgments and formal definition.

23.5 Asynchronous Goal Decomposition

We write this judgment as $\Gamma; \Delta; \Psi \Vdash A\ true$. Here, Γ is a context of unrestricted assumptions $A\ valid$, Δ is a context of linear assumptions $A\ true$, and Ψ is a context of ordered assumptions $S\ true$. Both A in the conclusion and the propositions S in Ψ are the ones that are decomposed. The context Ψ is ordered so we can restrict rules to operate on a unique proposition, removing any non-determinism from the rules.

$$\frac{\Gamma;\Delta;\cdot \Vdash A_1 \quad \Gamma;\Delta;\cdot \Vdash A_2}{\Gamma;\Delta;\cdot \Vdash A_1 \,\&\, A_2}\;\&R \qquad \frac{}{\Gamma;\Delta;\cdot \Vdash \top}\;\top R$$

$$\frac{\Gamma;\Delta;S_1 \Vdash A_2}{\Gamma;\Delta;\cdot \Vdash S_1 \multimap A_2}\;\multimap R \qquad \frac{\Gamma, A_1;\Delta;\cdot \Vdash A_2}{\Gamma;\Delta;\cdot \Vdash A_1 \supset A_2}\;\supset R$$

$$\frac{\Gamma;\Delta;\cdot \Vdash A \quad x \notin \mathrm{FV}(\Gamma,\Delta)}{\Gamma;\Delta;\cdot \Vdash \forall x.\, A}\;\forall R$$

$$\frac{\Gamma;\Delta;S_1,\Psi \Vdash P \quad \Gamma;\Delta;S_2,\Psi \Vdash P}{\Gamma;\Delta;S_1 \oplus S_2,\Psi \Vdash P}\;\oplus L \qquad \frac{}{\Gamma;\Delta;\mathbf{0},\Psi \Vdash P}\;\mathbf{0}L$$

$$\frac{\Gamma;\Delta;S_1,S_2,\Psi \Vdash P}{\Gamma;\Delta;S_1 \otimes S_2,\Psi \Vdash P}\;\otimes L \qquad \frac{\Gamma;\Delta;\Psi \Vdash P}{\Gamma;\Delta;\mathbf{1},\Psi \Vdash P}\;\mathbf{1}L$$

$$\frac{\Gamma, A;\Delta;\Psi \Vdash P}{\Gamma;\Delta;!A,\Psi \Vdash P}\;!L \qquad \frac{\Gamma;\Delta;S,\Psi \Vdash P \quad x \notin \mathrm{FV}(\Gamma,\Delta,P)}{\Gamma;\Delta;\exists x.\, S,\Psi \Vdash P}\;\exists L$$

$$\frac{\Gamma;\Delta,Q;\Psi \Vdash P}{\Gamma;\Delta;Q,\Psi \Vdash P}\;(Q)L \qquad \frac{\Gamma;\Delta,A;\Psi \Vdash P}{\Gamma;\Delta;A,\Psi \Vdash P}\;(A)L$$

$$\frac{\Gamma;\Delta \Vdash P}{\Gamma;\Delta;\cdot \Vdash P}\;(P)R \qquad \frac{\Gamma;\Delta \Vdash S\ \mathit{lax}}{\Gamma;\Delta;\cdot \Vdash \{S\}}\;\{-\}R$$

The last two rules transition to new judgments which represent so-called *neutral sequents* discussed in the next section.

23.6 Neutral Sequents

After an asynchronous decomposition has finished, we arrive at a neutral sequent. In LolliMon, there are two forms of neutral sequent, depending on whether the conclusion is true or lax. While the asynchronous decomposition is deterministic and can never fail, we now must make a choice about on which proposition to focus.

First, the case where the conclusion is $P\ \mathit{true}$. The possibilities for focusing initiate a backward chaining step.

$$\frac{\Gamma;\Delta;A \ll P \quad A \in \Gamma}{\Gamma;\Delta \Vdash P}\;\text{copy} \qquad \frac{\Gamma;\Delta;A \ll P}{\Gamma;\Delta,A \Vdash P}\;\text{focus}L \qquad \frac{\text{no focus}R\text{ rule}}{\Gamma;\Delta \Vdash P}$$

There is no way to focus on the right on an asynchronous atom since we only focus on synchronous propositions. Note that all propositions in Γ will be of the form A and therefore synchronous on the left. Similar, Δ consists of propositions that are synchronous on the left except for Q, which is analogous to allowing P on the right, and which may not be focused on.

When the conclusion is $S\ lax$ we can focus either on the left or on the right. Focusing on the left initiates a forward chaining step, focusing on the right terminates a sequence of forward chaining steps.

$$\frac{\Gamma; \Delta; A \ll S\ lax \quad A \in \Gamma}{\Gamma; \Delta \Vdash S\ lax}\ \text{copy} \qquad \frac{\Gamma; \Delta; A \ll S\ lax}{\Gamma; \Delta, A \Vdash S\ lax}\ \text{focus}L$$

$$\frac{\Gamma; \Delta \gg S\ true}{\Gamma; \Delta \Vdash S\ lax}\ \text{focus}R$$

We can see that a right-hand side $S\ lax$ means we are forward chaining, which we transition out of when when we focus in $S\ true$.

In a lower-level operational semantic specification we would add the precondition to the focus$R$ rule that no copy or focus$L$ rule is possible, indicating saturation and quiescence.

23.7 Backward Chaining

Backward chaining occurs when the right-hand side is $P\ true$ and we are focused on a proposition on the left. We refer to the right focus judgment for implications.

$$\frac{\Gamma; \Delta; A_1 \ll P}{\Gamma; \Delta; A_1 \mathbin{\&} A_2 \ll P}\ \&L_1 \qquad \frac{\Gamma; \Delta; A_2 \ll P}{\Gamma; \Delta; A_1 \mathbin{\&} A_2 \ll P}\ \&L_2 \qquad \begin{array}{c}\text{no } \top L \text{ rule} \\ \Gamma; \Delta; \top \ll P\end{array}$$

$$\frac{\Gamma; \Delta_1; A_1 \ll P \quad \Gamma; \Delta_2 \gg S_2}{\Gamma; \Delta_1, \Delta_2; S_2 \multimap A_1 \ll P} \multimap L \qquad \frac{\Gamma; \Delta; A_1 \ll P \quad \Gamma; \cdot \gg A_2}{\Gamma; \Delta; A_2 \supset A_1 \ll P} \supset L$$

$$\frac{\Gamma; \Delta; A(t/x) \ll P}{\Gamma; \Delta; \forall x.\, A \ll P}\ \forall L \qquad \begin{array}{c}\text{no } \{-\}L \text{ rule} \\ \Gamma; \Delta; \{S\} \ll P\end{array}$$

$$\frac{}{\Gamma; \Delta; P \ll P}\ (P)L \qquad \begin{array}{c}\text{not } (P)L \text{ rule if } P' \neq P \\ \Gamma; \Delta; P' \ll P\end{array}$$

Critical is the absence of the $\{-\}L$ rule: all forward chaining rules are disallowed during backward chaining. The logical rules for the lax modality

anticipate this: the right-hand side would have to have to be a lax judgment, but here it is truth. Without the monadic protection, this could not be done in a logically justified and complete way.

The rules for right focus also belong under this heading since they are invoked to terminate forward chaining or as a subgoal in backward chaining. When the right-hand side becomes asynchronous we transition back to the asynchronous decomposition judgment.

$$\frac{\Gamma; \Delta \gg S_1}{\Gamma; \Delta \gg S_1 \oplus S_2}\ \oplus R_1 \qquad \frac{\Gamma; \Delta \gg S_2}{\Gamma; \Delta \gg S_1 \oplus S_2}\ \oplus R_2 \qquad \begin{array}{c}\text{no } \mathbf{0}R \text{ rule} \\ \Gamma; \Delta \gg \mathbf{0}\end{array}$$

$$\frac{\Gamma; \Delta_1 \gg S_1 \quad \Gamma; \Delta_2 \gg S_2}{\Gamma; \Delta_1, \Delta_2 \gg S_1 \otimes S_2}\ \otimes R \qquad \frac{}{\Gamma; \cdot \gg \mathbf{1}}\ \mathbf{1}R$$

$$\frac{\Gamma; \cdot; \cdot \Vdash A}{\Gamma; \cdot \gg\ !A}\ !R \qquad \frac{\Gamma; \Delta \gg S(t/x)}{\Gamma; \Delta \gg \exists x.\, S}\ \exists R \qquad \frac{\Gamma; \Delta; \cdot \Vdash A}{\Gamma; \Delta \gg A}\ (A)R$$

$$\frac{}{\Gamma; Q \gg Q}\ (Q)R \qquad \begin{array}{c}\text{no rule for } \Delta \neq Q \\ \Gamma; \Delta \gg Q\end{array}$$

Focusing has strong failure conditions which are important to obtain predictable behavior. These are contained in the "missing rules" for the situations

$$\begin{array}{ll}\Gamma; \Delta; \{S\} \ll P\ true & \\ \Gamma; \Delta; P' \ll P\ true & \text{for } P' \neq P \\ \Gamma; \Delta \gg Q & \text{for } \Delta \neq Q\end{array}$$

all of which fail. The first is justified via the modal laws of lax logic, the second and third by properties of focusing on synchronous and asynchronous atoms. It is also important that we cannot focus on P on the right or Q on the left, which is again due to properties of focusing.

23.8 Forward Chaining

Forward chaining takes place when we focus on the left while the right-hand side is a lax judgment $S\ lax$. We can forward chain only if the ultimate head of the clause is a lax modality, which provides for a clean separation of the two phases with a logical foundation, and could not be easily justified otherwise as far as I can see.

The rules are mostly carbon copies of the left rules applied for forward chaining, except in the order of the premisses in the implication rules and,

of course, the rules for P and $\{S\}$.

$$\frac{\Gamma; \Delta; A_1 \ll S\ lax}{\Gamma; \Delta; A_1 \mathbin{\&} A_2 \ll S\ lax}\ \&L_1 \qquad \frac{\Gamma; \Delta; A_2 \ll S\ lax}{\Gamma; \Delta; A_1 \mathbin{\&} A_2 \ll S\ lax}\ \&L_2 \qquad \frac{\text{no } \top L \text{ rule}}{\Gamma; \Delta; \top \ll S\ lax}$$

$$\frac{\Gamma; \Delta_1 \gg S_1 \quad \Gamma; \Delta_1; A_2 \ll S\ lax}{\Gamma; \Delta_1, \Delta_2; S_1 \multimap A_2 \ll S\ lax}\ \multimap L \qquad \frac{\Gamma; \cdot \gg A_1 \quad \Gamma; \Delta; A_2 \ll S\ lax}{\Gamma; \Delta; A_1 \supset A_2 \ll S\ lax}\ \supset L$$

$$\frac{\Gamma; \Delta; A(t/x) \ll S\ lax}{\Gamma; \Delta; \forall x.\, A \ll S\ lax}\ \forall L \qquad \frac{\text{no } (P)L \text{ rule}}{\Gamma; \Delta; P \ll S\ lax}$$

$$\frac{\Gamma; \Delta; S' \Vdash S\ lax}{\Gamma; \Delta; \{S'\} \ll S\ lax}\ \{-\}L$$

We see $\Gamma; \Delta; P \ll S\ lax$ as an additional failure mode, due to focusing.

The rules for asynchronous decomposition of S' on the left once the focusing phase has been completed, are carbon copies of the rules when the conclusion is $P\ true$.

$$\frac{\Gamma; \Delta; S_1, \Psi \Vdash S\ lax \quad \Gamma; \Delta; S_2, \Psi \Vdash S\ lax}{\Gamma; \Delta; S_1 \oplus S_2, \Psi \Vdash S\ lax}\ \oplus L \qquad \frac{}{\Gamma; \Delta; \mathbf{0}, \Psi \Vdash S\ lax}\ \mathbf{0}L$$

$$\frac{\Gamma; \Delta; S_1, S_2, \Psi \Vdash S\ lax}{\Gamma; \Delta; S_1 \otimes S_2, \Psi \Vdash S\ lax}\ \otimes L \qquad \frac{\Gamma; \Delta; \Psi \Vdash S\ lax}{\Gamma; \Delta; \mathbf{1}, \Psi \Vdash S\ lax}\ \mathbf{1}L$$

$$\frac{\Gamma, A; \Delta; \Psi \Vdash S\ lax}{\Gamma; \Delta; !A, \Psi \Vdash S\ lax}\ !L \qquad \frac{\Gamma; \Delta; S', \Psi \Vdash S\ lax \quad x \notin \mathrm{FV}(\Gamma, \Delta, S)}{\Gamma; \Delta; \exists x.\, S', \Psi \Vdash S\ lax}\ \exists L$$

$$\frac{\Gamma; \Delta, Q; \Psi \Vdash S\ lax}{\Gamma; \Delta; Q, \Psi \Vdash S\ lax}\ (Q)L \qquad \frac{\Gamma; \Delta, A; \Psi \Vdash S\ lax}{\Gamma; \Delta; A, \Psi \Vdash S\ lax}\ (A)L$$

23.9 Backtracking and Committed Choice

The system from the previous sections is sound and complete when compared to intuitionistic linear logic with an added lax modality. This can be proved by a cut elimination argument as for other focusing systems.

Now we consider some lower-level aspects of the operational semantics. For backchaining with judgments $\Gamma; \Delta; \Psi \Vdash A$, $\Gamma; \Delta; A \ll P$ and $\Gamma; \Delta \gg S$, we solve subgoals from left to right, try alternatives from first-to-last, and backtrack upon failure. Moreoever, choices of terms t are post-

poned and determined by unification. So the backchaining fragment of LolliMon is fully consistent with pure Prolog and Lolli, that is, logic programming in the asynchronous fragment of linear logic. It is weakly complete which means that failure implies that a proof does not exist, but not all true propositions can be proven due to potentially non-terminating depth-first search.

For forward chaining, as defined by the judgments $\Gamma; \Delta; A \ll S\ lax$ and $\Gamma; \Delta; S' \Vdash S\ lax$ we use committed choice selection of the assumption to focus on. Moreoever, we continue to forward chain until we reach saturation and quiescence before we focus on the lax goal on the right. At present, we would consider it an error if we encounter a free logic variable during forward chaining because the semantics of this has not been satisfactorily settled.

If the program does not use the linear context in an essential way (that is, we are simulating unrestricted forward chaining in the linear setting), then due to the monotonicity of the unrestricted assumptions we still have non-deterministic completeness.

If the program *does* use the linear context then the system is complete in only a very weak sense: if we can always happen to make the right choice we can find a proof if it exists, but even if computation fails a proof might still exist because we do not backtrack.

I believe that this is actually the desired behavior because linear forward chaining was designed to model concurrency. The operational semantics of forward chaining corresponds exactly to the operational semantics of concurrent languages such as CCS or the π-calculus: if a transition is possible it may be taken without any chance for backtracking over this choice. This means programs are correct only if all legal computations behave correctly, such as, for example, avoiding deadlock. A corresponding correctness criteria applies to LolliMon program that use linear forward chaining: we must write the program in such a way that if multiple choices can be made, each one will give a correct answer in the end. The example in the next section illustrates this point.

As a programming language the logical fragment we have identified here is quite rich, since it allows backward chaining, saturating forward chaining, and linear forward chaining to capture many different sorts of operational behavior. We have to guard against viewing it as a general purpose inference engine. For example, it is often easy to write down a concurrent system as a linear LolliMon specification, but LolliMon presently provides little help in analyzing such a specification beyond simulating individual runs. This is an interesting area of further research.

23.10 Checking Bipartiteness on Graphs

The literature contains some examples of LolliMon programs, as does the LolliMon distribution. Many of the examples for forward chaining previously given in these notes such as CKY parsing or unification, can easily be expressed in LolliMon. We give here one more example, checking whether a given graph is bipartite, which illustrates a program that requires multiple saturating phases of forward chaining, and combines linear and nonlinear forward chaining.

A graph is bipartite if there is a division of its nodes into just two sets such that no two nodes in a set are connected to each other. Here is a high-level description of algorithm to check if a graph is bipartite. We use two colors, a and b to represent the two sets. We start with a graph where all nodes are unlabeled.

1. If all nodes are labeled, stop. The graph is bipartite.
2. Otherwise, pick an unlabeled node and color it a.
3. Propagate until no further changes occur to the graph:
   (a) If node u has color a and u is connected to w, color w with b.
   (b) If node u has color b and u is connected to w, color w with a.
4. If there is a node with both colors a and b the graph is not bipartite. Stop.
5. Otherwise, go to Step 1.

In the representation we have the following predicates, together with their intended usage.

$\mathsf{edge}(u, w)$	unrestricted; true if there is an edge from u to w
$\mathsf{unlabeled}(u)$	linear; true of node u if it has not yet been labeled
$\mathsf{label}(u, c)$	unrestricted; true if node u has color c
notbip	linear; true if graph is not bipartite

We start with the database initialized with unrestricted $\mathsf{edge}(u, w)$ and linear $\mathsf{unlabeled}(u)$ facts.

$$\mathsf{notbip} \multimap \exists U.\, !\mathsf{label}(U, \mathsf{a}) \otimes !\mathsf{label}(U, \mathsf{b}) \otimes \top.$$
$$\mathsf{notbip} \multimap \mathsf{unlabeled}(U) \otimes (\mathsf{label}(U, \mathsf{a}) \supset \{\mathsf{notbip}\}).$$

The first rule checks if there is a node with both colors and succeeds if that is the case. The second consumes an unlabeled node U, assigns it color a,

saturates by forward chaining, and then recurses, starting the next iteration of the algorithm. notbip fails if there is no two-colored node and no unlabeled node, in which case the graph is indeed bipartite.

The first two forward chaining rules are straightforward.

$$!\mathsf{label}(U, \mathsf{a}) \otimes !\mathsf{edge}(U, W) \multimap \{!\mathsf{label}(U, \mathsf{b})\}.$$
$$!\mathsf{label}(U, \mathsf{b}) \otimes !\mathsf{edge}(U, W) \multimap \{!\mathsf{label}(U, \mathsf{a})\}.$$

The third rule contains the interaction between linear and unrestricted assumptions: if a previously unlabeled node has been labeled, remove the assumption that it is unlabeled.

$$!\mathsf{label}(U, C) \otimes \mathsf{unlabeled}(U) \multimap \{\mathbf{1}\}.$$

Finally, a rule to take the symmetric closure of the edge relation, needed if we do not want to assume the relation is symmetric to start with.

$$!\mathsf{edge}(U, W) \multimap \{!\mathsf{edge}(W, U)\}.$$

Syntactically, $\mathsf{edge}(u, w)$, $\mathsf{label}(u, c)$, and notbip must be asynchronous atoms. Atoms $\mathsf{unlabeled}(u)$ could be either synchronous or asynchronous, and both posibilities execute correctly. A synchronous interpretation is most natural. There is a version of this program where label is also linear, in which case it would also be most naturally interpreted synchronously (see Exercise 23.4).

23.11 Historical Notes

The presentation of the lax modality in the form presented here originates with [3]; see that paper for further references on modal logic and monads.

LolliMon[1] [2] is a relatively recent development. The language supported by the implementation and described in the paper is slightly different from what we discussed here. In particular, all atoms are asynchronous, and nested implications in forward chaining are processed in reverse order to what we show here. Moreover, the implementation does not fully support $\oplus$ and $\mathbf{0}$, but its term language is a simply-typed λ-calculus with prefix polymorphism solved by pattern unification. One aspect of this is discussed in the next lecture.

An example applying LolliMon in computer security is given by Polakow and Skalka [4].

[1] Distribution available at `http://www.cs.cmu.edu/~fp/lollimon/`

LolliMon originated in the work on the Concurrent Logical Framework (CLF) [6, 1]. CLF is a type theory for formalizing deductive systems supporting state (through linearity) and concurrency (through a lax modality). Its original formulation lacks disjunction ($\oplus$) and falsehood ($\mathbf{0}$), but permits dependent types and a rich term language. The most recent and most complete work on the foundation of CLF is by Watkins [5].

23.12 Exercises

Exercise 23.1 *Extend the LolliMon language from this lecture, adding equality* $\mathbf{s} \doteq \mathbf{t}$ *and* $S \otimes! A$*, both of which are synchronous as goals and asynchronous as assumptions.*

Exercise 23.2 *Some compound connectives in LolliMon are redundant and could be eliminated. For example,* $A_1 \supset A_2$ *is equivalently expressed by* $(!A_1) \multimap A_2$*. Consider which connectives are redundant in this sense in the language.*

Exercise 23.3 *We have hinted in a prior lecture on resource management that, though logically equivalent, certain connectives may still be important for operational reasons. For example,* $S \otimes! A$ *and* $S \otimes (!A)$ *have different resource management behavior. Reconsider the redundant operators you identified in response to the Exercise 23.2 and determine which ones also have identical resource management properties and which do not.*

Exercise 23.4 *Rewrite the program to check whether graphs are bipartite, making the* label *predicate linear.*

On this program, consider the assignment where both unlabeled *and* label *are synchronous and the one where both are asynchronous. Explain differences in operational behavior, if any. Discuss which is more natural or potentially more efficient.*

23.13 References

[1] Iliano Cervesato, Frank Pfenning, David Walker, and Kevin Watkins. A concurrent logical framework II: Examples and applications. Technical Report CMU-CS-02-102, Department of Computer Science, Carnegie Mellon University, 2002. Revised May 2003.

[2] 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 Declar-*

ative Programming (PPDP'05), pages 35–46, Lisbon, Portugal, July 2005. ACM Press.

[3] Frank Pfenning and Rowan Davies. A judgmental reconstruction of modal logic. *Mathematical Structures in Computer Science*, 11:511–540, 2001. Notes to an invited talk at the *Workshop on Intuitionistic Modal Logics and Applications* (IMLA'99), Trento, Italy, July 1999.

[4] Jeff Polakow and Christian Skalka. Specifying distributed trust management in LolliMon. In S.Zdancewic and V.R.Sreedhar, editors, *Proceedings of the Workshop on Programming Languages and Security*, Ottawa, Canada, June 2006. ACM.

[5] Kevin Watkins. CLF: A logical framework for concurrent systems. Thesis Proposal, May 2003.

[6] Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. A concurrent logical framework I: Judgments and properties. Technical Report CMU-CS-02-101, Department of Computer Science, Carnegie Mellon University, 2002. Revised May 2003.