Subordination for linear, forward-chaining logic programs

Chris Martens

July 12, 2013

1 Language

1.1 The logic programming language

Basically lollimon or linear datalog.

Terms and types that classify them

$$\begin{array}{rcl} t & ::= & ... \\ \tau & ::= & ... \end{array}$$

Predicate kinds

$$K ::= \mathsf{rel} \mid \vec{\tau} \to K$$

Synchronous resources

$$S ::= a(t_1, ..., t_n) \mid S \otimes S \mid 1$$

Forward-chaining, asynchronous rules

$$A ::= \Pi \overline{x{:}\tau}.S \multimap \{S\}$$

Signatures, collections of rules

$$\Sigma ::= \cdot \mid \Sigma, t : \tau \mid \Sigma, a : K \mid \Sigma, r : A$$

States (linear contexts)

$$\Delta ::= \cdot \mid \Delta, x : S$$

Programs

$$prog ::= (\Sigma, \Delta_0)$$

1.2 The trace language

The above describes how to populate a signature; this section should describe how proof search is executed and traces are built.

Patterns

$$p ::= \langle x_1, \ldots, x_n \rangle$$

Rule applications

$$\begin{array}{rcl} R & ::= & r\ N \\ N & ::= & () \mid x \mid N \otimes N \end{array}$$

Bindings and traces

$$\begin{array}{rcl} \mathit{bnd} & ::= & \{p\} \leftarrow R \\ \epsilon & ::= & \langle\rangle \mid \mathit{bnd} \mid \epsilon; \epsilon \end{array}$$

Typing judgments

$$\Delta \vdash_\Sigma N : S$$

$$\Delta \vdash_\Sigma R : S$$

$$\Sigma \vdash r : A$$

$$\Sigma \vdash S : \mathsf{prop}$$

$$\frac{}{\cdot \vdash () : 1} \qquad \frac{\Delta_1 \vdash_\Sigma N_1 : S_1 \quad \Delta_2 \vdash_\Sigma N_2 : S_2}{\Delta_1, \Delta_2 \vdash_\Sigma N_1 \otimes N_2 : S_1 \otimes S_2} \qquad \frac{\Sigma \vdash S : \mathsf{prop}}{x{:}S \vdash_\Sigma x : S}$$

$$\frac{\Sigma \vdash r : S \multimap S' \quad \Delta \vdash_\Sigma N : S}{\Delta \vdash_\Sigma r\ N : S'}$$

$$\frac{\Sigma \vdash a : \tau_1 \to \cdots \to \tau_n \to \mathsf{prop} \quad \Sigma \vdash t_1 : \tau_1 \ldots \Sigma \vdash t_n : \tau_n}{\Sigma \vdash a(t_1 \ldots t_n) : \mathsf{prop}}$$

Operational judgment

$$\Delta \vdash_\Sigma \epsilon : \Delta'$$

Pattern binding judgment (note: Δ is an output)

$$\Delta \Vdash p : S$$

Pre- and post-sets of variables in an ϵ:

$$\begin{array}{rcl} \circ(\langle\rangle) & = & \emptyset \\ (\langle\rangle)\circ & = & \emptyset \\ \circ(\{p\} \leftarrow R) & = & \mathsf{fv}(R) \\ (\{p\} \leftarrow R)\circ & = & \mathsf{bv}(p) \\ \circ(\epsilon_1; \epsilon_2) & = & \circ(\epsilon_1) \cup ((\epsilon_1)\circ - \circ(\epsilon_2)) \\ (\epsilon_1; \epsilon_2)\circ & = & (\epsilon_2)\circ \cup (\circ(\epsilon_2) - (\epsilon_1)\circ) \end{array}$$

Concurrent preorder:

$$\epsilon_1 \leq \epsilon_2$$

when

$$(\epsilon_2)\circ \cap \circ(\epsilon_1) = \emptyset$$

Concurrent equality: $\epsilon_1; \epsilon_2 = \epsilon_2; \epsilon_1$ when $\epsilon_1 \leq \epsilon_2$ and $\epsilon_2 \leq \epsilon_1$ (Q: can we say $\epsilon_1 = \epsilon_2$ instead and does this give us the same notion of equality?)

2 Definition of subordination

An atomic resource $a(e)$ is *initial* in Σ if it occurs in Σ, but only to the left of an implication $... \otimes a(e) \otimes ... \multimap A$ ($a(e) \notin A$).

An atom is *intermediate* in Σ if it occurs to the left of the $\multimap$ in some rules and to the right in others.

An atom is *terminal* in Σ if it occurs in Σ and only occurs to the right.

$\Sigma_1 \prec \Sigma_2$ iff

- $\mathsf{intermediates}(\Sigma_1) \cap \Sigma_2 = \emptyset$
- $\mathsf{intermediates}(\Sigma_2) \cap \Sigma_1 = \emptyset$
- $\mathsf{initials}(\Sigma_1) \cap \mathsf{terminals}(\Sigma_2) = \emptyset$

3 Example

Consider the following signature.

```
data : type.
a : data.
b : data.

% model
model_initial : data -> type.
model_terminal : type.
act_on : data -> type. % internal.

model_receive : model_initial X -o {act_on X}.
model_process : act_on X -o {model_terminal}.


% control
control_initial : type.
control_terminal : data -> type.

control_picka : control_initial -o {control_terminal a}.
control_pickb : control_initial -o {control_terminal b}.
```

```
control_stop  : control_initial -o {1}.


% mediating
model-control : model_terminal -o {control_initial}.
control-model : control_terminal X -o {model_initial X}.


init : type = {control_initial}.

#query * * * 10 init -o {model_terminal}.
```

If we consider the signature to be divided by the comments into Σ_m, Σ_c, and Σ_i (for the model signature, control signature, and mediating signature), then $\Sigma_m \prec \Sigma_c$ and $\Sigma_c \prec \Sigma_m$. If we add in just one of the rules from Σ_i, we get an asymmetric subordination relation. For example, let's consider removing `model-control` but keeping `control-model`. This now means that $\Sigma_c \prec \Sigma_m$.

4 Theorem statement and proof

If $\Delta \vdash_{\Sigma_1,\Sigma_2} \epsilon : \Delta'$ and $\Sigma_1 \prec \Sigma_2$, then $\epsilon = \epsilon_1;\epsilon_2$ where $\Delta \vdash_{\Sigma_1} \epsilon_1 : \Delta''$ and $\Delta'' \vdash_{\Sigma_2} \epsilon_2 : \Delta'$.

In other words, any trace run in a signature that can be divided into a subordination relation has a prefix that can be run in only the subordinate signature, and does not need to consider rules from the later signature. In our example above, this means that as long as we haven't produced a terminal atom from the control (`control_terminal` being the only one), then we only need to consider rules from Σ_c.

Proof: By induction on the linearized structure of the trace ϵ.

- Case: $\epsilon = \langle\rangle$. $\epsilon_1 = \epsilon_2 = \langle\rangle$. ✓
- Case: $\epsilon = \{p\} \leftarrow R; \epsilon'$

  By typing rule, $\Delta = \Delta_1, \Delta_2$ where

  $$\frac{\Delta_1 \vdash_{\Sigma_1,\Sigma_2} R : S \quad \Delta_1' \Vdash p : S}{\Delta_1, \Delta_2 \vdash_{\Sigma_1,\Sigma_2} \{p\} \leftarrow R : \Delta_1', \Delta_2}$$

  By Lemma (below), either

  $$\Delta_1 \vdash_{\Sigma_1} R : S$$

  or

  $$\Delta_1 \vdash_{\Sigma_2} R : S$$

– Subcase 1:

$$\Sigma_1 \vdash \{p\} \leftarrow R : \Delta^o, fv(R) \rightsquigarrow \Delta^o, bv(p)$$

(and $\epsilon' : \Delta^o, bv(p) \rightsquigarrow \Delta'$).
By IH, $\epsilon' = \epsilon_1'; \epsilon_2'$.
Let ϵ_1 be $\{p\} \leftarrow R; \epsilon_1'$ and ϵ_2 be ϵ_2'. $\epsilon = \epsilon_1; \epsilon_2$ as needed.

– Subcase 2: $\{p\} \leftarrow R$ is wf in Σ_2
By IH, $\epsilon = \{p\} \leftarrow R; \epsilon_1'; \epsilon_2'$.
By definition of subordination, $bv(p)$'s types are intermediates or terminals of Σ_2 (because, for $R = r(e)$, initials can't appear on the right of the $\multimap$ in r's type).
Since ϵ_1' is well-formed in Σ_1 (also from IH), $bv(p) \notin \epsilon_1'$.
Thus we can rewrite

$$\{p\} \leftarrow R; \epsilon_1'; \epsilon_2'$$

as

$$\epsilon_1'; \{p\} \leftarrow R; \epsilon_2'$$

So letting ϵ_1 be ϵ_1' and ϵ_2 be $\{p\} \leftarrow R; \epsilon_2'$ gives us $\epsilon = \epsilon_1; \epsilon_2$ as needed.

4.1 Lemma

If $\Sigma_1 \prec \Sigma_2$ and $\Delta \vdash_{\Sigma_1,\Sigma_2} R : S$ then either $\Delta \vdash_{\Sigma_1} R : S$ or $\Delta \vdash_{\Sigma_2} R : S$.

Typing rule ($R = r\ N$): (n.b. this is more complicated w/instantiation at different LF terms, but ignoring for now)

$$\frac{\Sigma_1, \Sigma_2 \vdash r : S \multimap S' \quad \Delta \vdash_{\Sigma_1,\Sigma_2} N : S}{\Delta \vdash_{\Sigma_1,\Sigma_2} r\ N : S'}$$

Left premise: a rule can only live in one part of the signature, so either $\Sigma_1 \vdash r : S \multimap S'$ or $\Sigma_2 \vdash r : S \multimap S'$.

In either case, we need to show $\Delta \vdash_{\Sigma_i} N : S$.

Subcase: $N = x, \Delta = x{:}S$

$$\frac{\Sigma_i \vdash S : \mathsf{prop}}{x : S \vdash x : S}$$

It remains to show we can derive $\Sigma_i \vdash S : \mathsf{prop}$.

If we can show that Σ_i is a well-formed signature on its own then we are done: $\Sigma_i \vdash r : S \multimap S'$ and Σ_i ok (XXX define this judgment) mean that $\Sigma_i \vdash S \multimap S' : \mathsf{prop}$, so by inversion $\Sigma_i \vdash S : \mathsf{prop}$.

XXX stuck here: in the case where $i = 1$, we should be able to do this by the definition of subordination, except that the given definition says nothing about type declarations. This is the spot where I suspect we need to have a separate "LF signature" and "transition rule signature", where the LF signature has its own (standard) notion of subordination that will factor into this proof.