Subordination for linear, forward-chaining logic programs

Chris Martens

May 2, 2013

1 Language

1.1 The logic programming language

Basically lollimon or linear datalog.

Terms and types that classify them

$$t ::= ...$$

$$\tau ::= ...$$

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

$$prog ::= (\Sigma, \Delta_0) \mid (\Sigma, \Delta_0) \longrightarrow^{?} A$$

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

$$R ::= r(N_1, ..., N_n)$$

Traces (sequences of bindings)

$$\epsilon ::= \langle\rangle \mid \{p\} \leftarrow R; \epsilon'$$

Operational judgment

$$\Sigma \vdash \epsilon : \Delta \rightsquigarrow \Delta'$$

Pre- and post-sets of variables in a binding
Concurrent equality judgment

$$\epsilon_1 = \epsilon_2$$

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 $\Sigma_1, \Sigma_2 \vdash \epsilon : \Delta \rightsquigarrow \Delta'$ and $\Sigma_1 \prec \Sigma_2$, then $\epsilon = \epsilon_1 ; \epsilon_2$ where $\Sigma_1 \vdash \epsilon_1 : \Delta \rightsquigarrow \Delta''$ and $\Sigma_2 \vdash \epsilon_2 : \Delta'' \rightsquigarrow \Delta'$.

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 structure of the trace ϵ.

- Case: $\epsilon = \langle\rangle$. $\epsilon_1 = \epsilon_2 = \langle\rangle$. ✓

- Case: $\epsilon = \{p\} \leftarrow R; \epsilon'$
  By Lemma (below), either
  $$\Delta \vdash_{\Sigma_1} R : A$$
  or
  $$\Delta \vdash_{\Sigma_2} R : A$$
  - 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 $\Sigma_1, \Sigma_2 \vdash p \leftarrow R; \epsilon : \Delta \rightsquigarrow \Delta'$ then either $\Sigma_1; \Delta \vdash R : A$ or $\Sigma_2; \Delta \vdash R : A$.