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

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

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}.

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

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

control_pickb : control_initial -o {control_terminal b}.
control_stop  : control_initial -o {1}.


init : type = {model_terminal}.

#query * * * 10 init -o {1}.
```

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 $\Sigma_2 \vdash \epsilon_2 : \Delta'' \rightsquigarrow \Delta'$.

By induction on the structure of the trace ϵ.

- Case: $\epsilon = \langle\rangle$. $\epsilon_1 = \epsilon_2 = \langle\rangle$. Done.
- Case: $\epsilon = \{p\} \leftarrow R; \epsilon'$
  By Lemma (below), either

  $$\Delta \vdash_{\Sigma_1} R : A$$

  (where A is a wf type in Δ, Σ_1) or

  $$\Delta \vdash_{\Sigma_2} R : A$$

  (where A is a wf type in Δ, Σ_2).

  - Subcase 1:
  - Subcase 2:

4.1 Lemma