Webs

Robin Milner, 1 Sep 85

A way of approaching the static semantics of sharing in modules, using congruences to represent sharing of components.

A skeletal language is given, with its operational semantics.

[Stimulated by Bob Harper's draft for the static semantics of modules].

There is a debt to Pebble in the relationship between expressions and declarations.
The LANGUAGE

Functor identifiers `funid`: A denumerable set of identifiers

Labels `lab`: A denumerable set of identifiers.

Expressions `exp`:

```
exp ::= 
  \{ lab_1 \to exp_1, \ldots, lab_n \to exp_n \}  
  path 
  funid exp

path ::= 
  lab_1, \ldots, lab_{n+1}
```

Specifications `spec`:

```
spec ::= 
  tree \[ \text{eqv}_1, \ldots, \text{eqv}_n \] 

tree ::= 
  \{ lab_1 \to \text{tree}_1, \ldots, lab_n \to \text{tree}_n \}  
  \text{eqv} 
  \text{path} = \text{path}'
```

Definitions `def`:

```
def ::= 
  \text{def funid spec} = \text{exp}
```

Declarations `dec`:

```
der ::= 
  \text{dec} \text{ exp} 
```

A Program is a sequence of definitions and declarations.
Trees, Congruences, and Webs

We use a, b, ... to range over Lab, the set of labels, and p, q, ... to range over Lab*. ε is the empty sequence in Lab*. For S ⊆ Lab*, Pref(S) is the prefix closure of S.

A tree is a non-empty prefix-closed subset of Lab*. We use T to range over trees. We define:

- \( \text{Inits}(T) = \{ a | \exists b. ap \in T \} \)
- \( \beta T = \text{Pref}(p) \cup \{ \beta q | q \in T \} \)
- \( T/\beta = \{ q | \beta q \in T \} \quad (p \in T) \)

Clearly, \( \beta T \) and \( T/\beta \) are trees.

A congruence \( C \) on T is an equivalence over T such that

(i) If \( (p, q) \in C \) and \( pa \in T \), then \( qa \in T \) and \( (pa, qa) \in C \) (Closure under right concatenation)

(ii) If \( (p, \epsilon) \in C \) then \( q = \epsilon \) (No cycles).

We define:

- \( \beta C = \text{Id}_{\text{Pref}(p)} \cup \{ (pq_1, pq_2) | (q_1, q_2) \in C \} \)
- \( C/\beta = \{ (q_1, q_2) | (pq_1, pq_2) \in C \} \quad (p, q) \in C \)

If \( C \) is a congruence over \( T \), then clearly \( \beta C \) (resp. \( C/\beta \)) is a congruence over \( \beta T \) (resp. \( T/\beta \)).

A web \( W = (T, C) \) is a tree \( T \) with a congruence \( C \) over \( T \). We define \( (T_2, C_2) \) to be a subweb of \( (T_1, C_1) \), and write \( (T_2, C_2) \preceq (T_1, C_1) \), in case \( T_2 \subseteq T_1 \) and \( C_2 \subseteq C_1 \). (Fewer paths, less "shaving").
If \((T, C)\) is a web, then \((T', C')\) is a web on \((T, C)\) if \(C'\) is a congruence on \(T \cup T'\) (the disjoint union of \(T\) and \(T'\)) and moreover \(C' \cap T = C\).

More exactly, we introduce two special labels \(\text{ext}\) and \(\text{int}\), and define \(T \cup T' = \text{ext}T \cup \text{int}T'\); note that this is a tree (containing \(\epsilon\)) and our requirement may be written \(C' / \text{ext} = C\).

We shall also use

- \(C' / (\text{ext, int}) = \{(p, p') \mid (\text{ext} p, \text{int} p') \in C'\}\) (a relation between \(T\) and \(T'\))
- \((\text{ext, int}) R = \{(\text{ext} p, \text{int} p') \mid (p, p') \in R\}\) \(\subseteq T \cup T'\) (where \(R\) is a relation between \(T\) and \(T'\))

Thus \(C' = \text{ext}(C' / \text{ext})\) \(\subseteq \text{ext}T \times \text{ext}T\)
\(\cup (\text{ext, int})(C' / (\text{ext, int}))\) \(\subseteq \text{ext}T \times \text{int}T'\)
\(\cup (\text{int, ext})(C' / (\text{int, ext}))\) \(\subseteq \text{int}T' \times \text{ext}T\)
\(\cup \text{int}(C' / \text{int})\) \(\subseteq \text{int}T' \times \text{int}T'\)

The idea is that an expression \(\text{exp}\), evaluated on a web \((T, C)\), yields \((T', C')\) which is a web on \((T, C)\). Intuitively, the result tree \(T'\) has parts which may "share" with each other and with parts of \(T\) (the "sharing" being represented by \(C'\)), but the evaluation induces no further sharing among parts of \(T\), since \(C' \cap T = C\).

As an example, we may draw a web \((T, C)\) as a dag with labelled arcs, as follows:
\[ T = \{ \varepsilon, a, b, ba, bc \} \]
\[ C = \text{Id}_T \cup \{(a, ba)\} \]

and evaluation may produce \((T', C')\) on \((T, C)\), shown thus:

\[ T' = \{ \varepsilon, a, ac, ac, c, cd, e \} \]
\[ C' = \{(a, e)\} \cup \text{ext } C \]
\[ \cup \text{int } (\text{Id}_{T'}, \{(ac, cd)\}) \]
\[ \cup \{(b, a), (ba, aa), (bc, ac)\} \]
\[ \cup \{(a, b), (aa, ba), (b, bc)\} \]

Clearly we can represent congruences more briefly if we define the closure:

- \(\text{Cl } R = \) the smallest congruence containing \(R\)

For example, in the above we have \(C = \text{Cl } \{(a, ba)\}\), and
\(C' = \text{Cl } \{(\text{ext } C \cup \text{int } \{(ac, cd)\}) \cup \{(b, a)\}\}\)

We shall call any web of the form \((T \cup T', C')\) a double web; if \(R\) is clearly a web \((T', C')\) on \((T, C)\)

where \(C = C'/\text{ext}\).

*Strictly, the closure is relative to a particular tree \(T\). For \(R \in \text{Txt}T\), one should write \(\text{Cl}_T R\); in our usage \(T\) will be clear from context.*
3. Composition of double webs

Clearly if \( W' = (T', C') \) is a web on \( W \), then \( (T', C'/\text{int}) \) is a web, which we denote by \( W'/\text{int} \).

Given a web \( W' \) on \( W \), and a web \( W'' \) on \( W'/\text{int} \), there is a natural way of composing them to form a web \( W'' \) on \( W \).

\[
\left( (T'', C'') \right) \text{ on } (T', C'/\text{int}) \quad \text{with} \quad (T', C') \text{ on } (T, C) \\
\text{yields} \quad (T'', C'') \text{ on } (T, C)
\]

In fact \( T'' = T'' \), and \( C'' \) (a congruence on \( TUT'' \)) is the result of extending \( C'' \) and \( C' \) to \( TUTUT'' \), taking the closure congruence, and restricting it to \( TUT'' \). We shall denote the resulting \( W'' = (T'', C'') \) by

\[ \text{Compose } (W'', W', W) \]

— the formal definition is a little tedious, but not difficult.
4. Operations on webs and double webs

Each form of expression (construction, selection, abbreviation) will be understood semantically as an operation (on webs or double webs) yielding a double web.

1. Selection. Let $W = (T, C)$ be a web, and $\phi \in T$. Then select $(\overline{\phi}_* W)$, the selection of $\phi$ from $W$, is the web $(T', C')$ on $W$ where $T' = T/\phi$ and $C' = \text{cl}(\text{ext} C \cup \text{int}(C/\phi))$

\[ \cup \{(\text{ext} p q, \text{int} q) \mid p q \in T_3^2\} \]

Example 1. $\phi = b$, and $(T, C) = a \triangle b \bullet c$. Then

\[ (T \cup T', C') = \]

It is simple (but necessary) to prove that the selection of $\phi$ from $W$ is indeed a web on $W$.

2. Construction. Let $W = (T, C)$ be a web, let $W_i = (T_i', C_i')$ be webs on $(T, C)$, $1 \leq i \leq n$, and let $a_i$ $(1 \leq i \leq n)$ be distinct labels. Write $\overline{a} = (a_1, \ldots, a_n)$ and $\overline{W} = (W_1, \ldots, W_n)$. Then $\text{Construction}(\overline{W}, \overline{a}, W)$, the construction of $\overline{W}$ by $\overline{a}$ from $W$, is the web $(T', C')$ on $W$, where $T' = a_1 T_1' \cup \ldots \cup a_n T_n'$ and
\[ C' = C \cap \left( \text{ext}^+ C \cup \text{int}(U_{i=1}^{n} (C_i' \cap \text{int})) \right) \]
\[ \cup \left\{ \text{ext}^+ \rho, \text{int} a_i, \rho \right\} \mid (\text{ext}^+ \rho, \text{int} a_i) \in C_i, 1 \leq i \leq n \right\} \]

Again, it is necessary to prove that a construction from \( W \) is indeed a web on \( W \).

Example 2 \( W = (T, C) = \)

\[ (T \cup T'_1, C'_1) = \]

\[ (T \cup T'_2, C'_2) = \]

Then construct \((W_1, W_2), (a_1, a_2), W) = (T', C')\), where \((T' \cup T', C') = \)

Before considering the meaning of application, it may help to see how the two examples may be declared in our language.

\[ \text{dec } a = \{ b \rightarrow \{ a \} \} \]
\[ \text{dec } b = \{ a \rightarrow a, c \rightarrow \{ b \} \} \]

Now the "environment" in the web \( W = (T, C) \) of Example 1. Then, the expressions

\[ b \quad \text{and} \quad \{ b \rightarrow b, c, c \rightarrow \{ a \} \} \]

evaluate to the webs \( W_1 \) and \( W_2 \) on \( W \), of Example 2, while the expression

\[ \{ a_1 \rightarrow b, a_2 \rightarrow \{ b \rightarrow b, c, c \rightarrow \{ a \} \} \}

evaluates to the web \( (T', C') \) of Example 2.
3. Application. The meaning of a functor identifier \( F \) defined to be a dense web, namely \( W''_F \) on \( W_F' \), where \( W_F' \) is the web denoted by its specification and \( W''_F \) the result of evaluating its expression on \( W_F' \). Consider an example:

\[
W_F' \quad \sigma \quad W''_F \quad \text{on} \quad W_F'
\]

Using this example, we show what the result should be from applying this dense web to an argument web \( W' \geq W_F' \):

<table>
<thead>
<tr>
<th>Argument</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>( W' \quad a \quad \sigma \quad c \quad g )</td>
<td>( W'' \quad \sigma \quad W' \quad a \quad \sigma \quad d \quad e )</td>
</tr>
</tbody>
</table>

We can see that, because \( W' \) has more paths and more sharing than \( W_F' \), so \( W'' \) has more paths and more sharing than \( W''_F \).

Proposition. Given a web \( W''_F \) on \( W_F' \) and a web \( W' \geq W_F' \), there is a least web \( W'' \) on \( W' \) such that \( W'' \geq W''_F \).

We define \( \text{Extend} \left( W''_F, W_F', W' \right) \) to be this web \( W'' \) on \( W' \).

Now in general we want, roughly speaking, to apply the (meaning of the) functor identifier \( F \) to a web \( W' \) on \( W \) (i.e., the result of evaluating the argument expression on \( W \)). The full result should be a web \( W'' \) on \( W \).
Finally, therefore, we define the application of a web \( W'_F \) on \( W'_F \) to a web \( W' \) on \( W \), provided that \( W'_F \leq W'/\text{int} \), as follows:

\[
\text{Apply} \left( W''_F, W'_F, W', W \right) = \text{Compose} \left( \text{Extend} \left( W''_F, W'_F, W'/\text{int} \right), W', W \right)
\]

4) **Absorption.** The final operation is to do with declarations, not expressions. The declared expression yields a web \( W' = (T', C') \) on the "environment" web \( W = (T, C) \), and we wish to produce a new "environment" \( W_{\text{new}} = (T_{\text{new}}, C_{\text{new}}) \) in which \( T' \) overrides \( T \) (in a precise sense) to form \( T_{\text{new}} \).

If \( A \in \text{Lab} \), define in general:

- \( T^A = \{ \varepsilon \} \cup \{ a \varepsilon \in T \mid a \in A \} \)
- \( C^A = \{ (\varepsilon, \varepsilon) \} \cup \{ (a \varepsilon, b \varepsilon) \in C \mid a, b \in A \} \)

Then we define

\[
\text{Absorb} \left( (T, C), (T', C') \right) = (T_{\text{new}}, C_{\text{new}})
\]

where

\[
T_{\text{new}} = T^A \cup \text{ext} - \text{init}(T') \cup T'
\]

\[
C_{\text{new}} = C^A \cup C'/\text{int} \cup \{ (a \varepsilon, b \varepsilon) \in C'/\text{ext} \mid a \in \text{init}(T') \}
\]

**Example**

\( (T', C') \) on \( (T, C) \):

\[
\begin{array}{c}
\text{ext} \quad \text{int} \\
\begin{array}{c}
\leftarrow \\
\rightarrow \\
\end{array}
\end{array}
\]

\( \text{Absorb} \left( (T, C), (T', C') \right) \):

\[
\begin{array}{c}
\text{ext} \quad \text{int} \\
\begin{array}{c}
\leftarrow \\
\rightarrow \\
\end{array}
\end{array}
\]
5. Semantics of the Language

We imagine that a "program" in our language is a sequence of definitions mixed with declarations. Each declaration modifies the environment web, and thus affects the meaning of subsequent declarations, since they can refer to it using selection. However, the environment web has no effect upon the meaning of a definition, since all selections within a definition are taken to refer to the (virtual) environment web denoted by the definition's specification.

The set of function environments is

\[(f_e) \quad \text{Funs} = \quad \text{Funid} \quad \text{FIN} \rightarrow \text{Webs} \]

The semantics is an inference system over sentences of the following forms:

\[ f, W \vdash \text{exp} \Rightarrow W' \quad (W' \text{ a web over } W) \]

\[ \vdash \text{spec} \Rightarrow W \]

\[ \vdash \text{tree} \Rightarrow T \]

\[ F \vdash \text{def} \Rightarrow F_{\text{new}} \]

\[ f, W \vdash \text{dec} \Rightarrow W_{\text{new}} \]
\[ \mathcal{F}, W \vdash \text{exp} \Rightarrow W' \]

**Construction:**

\[ \mathcal{F}, W \vdash \text{exp}_i \Rightarrow W_i \quad (1 \leq i \leq n) \]

\[ \mathcal{F}, W \vdash \text{lat} \rightarrow \text{exp}_1, \ldots, \text{lat} \rightarrow \text{exp}_n \Rightarrow \text{Construct} \left( (W_1, \ldots, W_n), (\text{lat}_1, \ldots, \text{lat}_n), W \right) \] (\text{lat}, \text{distinct})

**Selection:**

\[ \mathcal{F}, W \vdash \text{path} \Rightarrow \text{Select} \left( \text{path}, W \right) \]

\[ W = (T, c), \text{ path } \in T \]

**Application:**

\[ \mathcal{F}, W \vdash \text{exp} \Rightarrow W' \]

\[ \mathcal{F}, W \vdash \text{funid} \text{ exp} \Rightarrow \text{Apply} \left( W''_{\text{funid}}, W'_{\text{funid}}, W', W \right) \]

\[ (\mathcal{F}(\text{funid}) = (W''_{\text{funid}}, W'_{\text{funid}}); W'_{\text{funid}} \leq W/\text{int}) \]

\[ \vdash \text{tree} \Rightarrow T \]

\[ \vdash \text{tree}_i \Rightarrow T_i \quad (1 \leq i \leq n) \]

\[ \vdash \text{lat}_1 \rightarrow \text{tree}_1, \ldots, \text{lat}_n \rightarrow \text{tree}_n \Rightarrow \text{lat}_1 T_1 u \ldots u \text{lat}_n T_n \] (\text{lat}_i \text{ distinct})

\[ \vdash \text{spec} \Rightarrow W \]

\[ \vdash \text{tree} \Rightarrow T \]

\[ \vdash \text{tree} \left[ p_i = p'_i, \ldots, p_n = p'_n \right] \Rightarrow (T, c) \quad (p_i, p'_i \in T) \]

(\text{provided } C = CE \left\{ (p_i, p'_i), \ldots, (p_n, p'_n) \right\} \text{ exists})

\[ \vdash \]

* It may fail to exist, due to either of the conditions (i),(ii) on a congruence.
\[ \vdash \text{def} \Rightarrow \mathcal{F}_{new} \]

\[ \begin{align*}
\vdash & \text{spec} \Rightarrow W \\
\mathcal{F}, W \vdash & \text{exp} \Rightarrow W' \\
\vdash & \text{def funid spec = exp} \Rightarrow \mathcal{F} \left[ \text{funid} \mapsto (W', W) \right]
\end{align*} \]

\[ \begin{align*}
\vdash & \text{dec} \Rightarrow W_{new} \\
\mathcal{F}, W \vdash & \text{exp} \Rightarrow W' \\
\mathcal{F}, W \vdash & \text{dec exp} \Rightarrow \text{Absorb} (W, W')
\end{align*} \]
6 Remarks

The correspondence between this language and MacLane modules
has not been detailed, but notice that his signatures correspond
to our specifications. I find it tidy to allow every "signature"
to have a sharing clause, so that the formal parameter to a
functor is just a "signature". Also, signature identifiers
don't exist here — but they can easily be introduced, and
then evaluating a specification from component specifications would
entail a little bit of congruence closure.

The main point of this whole exercise is to give a
skeletal framework within which questions about sharing —
and signature matching — can be clearly studied; also, I wanted
to be sure that Bob Harper's Timestamps are not essential
in the static semantics of modules. (The question arises whether
the machinery of congruences, though mathematically precise, is a
reasonable price to pay for dispensing with Timestamps).

One pertinent question to study is the effect of adding
explicit "signature" constraints; in particular, what is the meaning of
def fund spec = exp; spec'
where the body of the definition must now match spec'?