1 Grammars

In the following, D is a data constructor, f a function symbol and we consider them as strings. n represents an integer.

1.1 λ-lifted haskell subset

$$\begin{array}{rcll}
u, e & := & x \mid f \mid e\ e \mid D(e, \ldots, e) \mid \texttt{BAD} \mid n & (\mathit{Expression}) \\
p & := & \Delta, p \mid T, p \mid \epsilon & (\mathit{Program}) \\
\Delta & := & d \mid \mathit{transp}(d, c) \mid \mathit{opaque}(d, c) & (\mathit{Defintion\ and\ contract}) \\
d & := & f\ x_1 \ldots x_n = e \mid f\ x_1 \ldots x_n = \text{ case } e \text{ of } [(\mathit{pat}_i, e_i)] & (\mathit{Definition}) \\
T & := & \text{data } x\ \tau_1 \ldots \tau_n = D_1; D_2; \ldots; D_n & (\mathit{Data\ type\ definition}) \\
\mathit{pat} & := & D(x_1, \ldots, x_n) & (\mathit{Pattern})
\end{array}$$

1.2 FOL

$$\begin{array}{rcll}
t & := & x \mid \text{app}(t_1, t_2) \mid D(t, \ldots, t) \mid f \mid n \mid \texttt{BAD} \mid \texttt{UNR} \mid \text{CF}(t) & (\mathit{Term}) \\
\phi & := & \forall x.\phi \mid \phi \to \phi \mid \neg\phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \mathit{true} \mid t = t \mid \text{CF}(t) & (\mathit{Formula})
\end{array}$$

$\text{CF}(t)$ holds iff t satisfies `Ok`.

1.3 Contracts

$$\begin{array}{rcl}
c & := & x : c_1 \to c_2 \\
 & \mid & (c_1, c_2) \\
 & \mid & \{x \mid e\} \\
 & \mid & \texttt{Any}
\end{array}$$

Semantics of contract satisfaction:

$$\begin{array}{rcl}
e \in \{x \mid p\} & \iff & e \text{ diverges or } (e \text{ is crash-free and } p[e/x] \not\to^{\star} \{\texttt{BAD}, \texttt{UNR}\}) \\
e \in x : t_1 \to t_2 & \iff & \forall e_1 \in t_1, (e\ e_1) \in t_2[e_1/x] \\
e \in (t_1, t_2) & \iff & e \text{ diverges or } (e \to^{\star} (e_1, e_2) \text{ and } e_1 \in t_1, e_2 \in t_2) \\
e \in \texttt{Any} & \iff & \mathit{True}
\end{array}$$

2 Translation

We define several translations: $\mathcal{E}[\![]\!], \mathcal{D}[\![]\!], \mathcal{T}[\![]\!], \mathcal{S}[\![]\!], [\![]\!]$.

$$\begin{array}{rcl}
\mathcal{E}[\![]\!] & :: & \mathit{Expression} \to \mathit{Term} \\
\mathcal{D}[\![]\!] & :: & \mathit{Definition} \to \mathit{FOF} \\
\mathcal{T}[\![]\!] & :: & \mathit{Data\ type} \to \mathit{FOF} \\
\mathcal{S}[\![]\!] & :: & \mathit{Expression} \to \mathit{Contract} \to \mathit{FOF} \\
[\![]\!] & :: & \mathit{Definition} \to \mathit{Contract} \to \mathit{FOF}
\end{array}$$

2.1 $\mathcal{E}[\![]\!]$

$\mathcal{E}[\![e]\!]$ is a term.

$$\begin{aligned}
\mathcal{E}[\![x]\!] &= x && (1)\\
\mathcal{E}[\![f]\!] &= f && (2)\\
\mathcal{E}[\![e_1\ e_2]\!] &= app(e_1, e_2) && (3)\\
\mathcal{E}[\![D(e_1, \ldots, e_n)]\!] &= D(\mathcal{E}[\![e_1]\!], \ldots, \mathcal{E}[\![e_n]\!] && (4)\\
\mathcal{E}[\![\mathtt{BAD}]\!] &= \mathtt{BAD} && (5)\\
\mathcal{E}[\![n]\!] &= n && (6)
\end{aligned}$$

2.2 $\mathcal{D}[\![]\!]$

$\mathcal{D}[\![d]\!]$ is a first-order formula.

$$\begin{aligned}
\mathcal{D}[\![f\ x_1 \ldots x_n = e]\!] &= \forall x_1 \ldots x_n . \mathcal{E}[\![f\ x_1 \ldots x_n]\!] = \mathcal{E}[\![e]\!] && (8)\\
\mathcal{D}[\![f\ x_1 \ldots x_n = \text{case } e \text{ of } [D_i\ \overline{z} \mapsto e_i]]\!] &= \forall x_1 \ldots x_n\ (\bigwedge_i (\forall \overline{z}\ \mathcal{E}[\![e]\!] = \mathcal{E}[\![D_i\ \overline{z}]\!] \rightarrow \mathcal{E}[\![f\ x_1 \ldots x_n]\!] = \mathcal{E}[\![e_i]\!]) && (9)\\
& \wedge \mathcal{E}[\![e]\!] = \mathtt{BAD} \rightarrow \mathcal{E}[\![f\ x_1 \ldots x_n]\!] = \mathtt{BAD}) && (10)\\
& \wedge (\mathcal{E}[\![f\ x_1 \ldots x_n]\!] = \mathtt{UNR} \bigvee_i (\mathtt{HD}(e) = D_i) \vee e = \mathtt{BAD}) && (11)
\end{aligned}$$

Can't we just put a XOR instead of OR in the last line?

2.3 $\mathcal{S}[\![]\!]$

$\mathcal{S}[\![e \in c]\!]$ is a first-order formula.

$$\begin{aligned}
\mathcal{S}[\![e \in \mathtt{Any}]\!] &= true && (12)\\
\mathcal{S}[\![e \in \{x \mid u\}]\!] &= e = \mathtt{UNR} \vee (\mathrm{CF}(\mathcal{E}[\![e]\!]) \wedge \mathcal{E}[\![u[e/x]]\!] \neq \mathtt{BAD} \wedge \mathcal{E}[\![u[e/x]]\!] \neq False) && (13)\\
\mathcal{S}[\![e \in x : c_1 \rightarrow c_2]\!] &= \forall x_1 . \mathcal{S}[\![x_1 \in c_1]\!] \rightarrow \mathcal{S}[\![e\ x_1 \in c_2[x_1/x]]\!] && (14)
\end{aligned}$$

$False$ is a data constructor here.

Remark: we follow the semantics of the POPL paper but it's a bit restrictive. e.g. in equation 13 we could use the alternate semantics (namely B1 in the POPL paper) :

$$\mathcal{S}[\![e \in \{x \mid u\}]\!] = e = \mathtt{UNR} \vee (\mathcal{E}[\![u[e/x]]\!] \neq \mathtt{BAD} \wedge \mathcal{E}[\![u[e/x]]\!] \neq False)$$

2.4 $[\![]\!]$

It's the final translation, which takes a function definition and its contract and returns a first-order formula (14)

$$\begin{aligned}
[\![opaque(f\ x_1 \ldots x_n = e, c)]\!] &= \mathcal{D}[\![f\ x_1 \ldots x_n = e[f_p/f]]\!] \wedge \mathcal{S}[\![f \in c]\!] \wedge \mathcal{S}[\![f_p \in c]\!] && (16)\\
[\![transp(f\ x_1 \ldots x_n = e, c)]\!] &= \mathcal{D}[\![f\ x_1 \ldots x_n = e[f_p/f]]\!] \wedge \mathcal{S}[\![f \in c]\!] \wedge \mathcal{S}[\![f_p \in c]\!] && (17)\\
[\![f\ x_1 \ldots x_n = e]\!] &= [\![opaque(f\ x_1 \ldots x_n) = e, Ok \rightarrow \cdots \rightarrow Ok)]\!] && (18)
\end{aligned}$$

We'd like a typical contract-checking session to go like this:

1. Start with an empty theory T.

2. Let $f\ x_1 \dots x_n = e \in c$ be an opaque function definition to check wrt contract c. Check (with equinox) the consistency of the theory $T' = T \cup [\![f\ x_1 \dots x_n = e \in c]\!]$

3. If T' is consistent then let $T = \mathcal{S}[\![f \in c]\!] \cup T$ and go to 2. with the next function definition; otherwise give a counter-example and ask the user for refinement of the contracts and/or lemmas(?)

3 Datatypes

$\mathcal{T}[\![T]\!]$ is a set of first-order formulae. $\mathcal{T}[\![\text{data } T = D_1; \dots ; D_n]\!] = S_1 \cup S_2 \cup S_3$.

First, for each D_i we introduce selectors sel_{k,D_i}, which are basically projections of $D_i(x_1, \dots, x_n)$ on its k-th composant, so that we can express that constructors are injective :

$$S_1 := \{\forall x_1, \dots, x_n. \bigwedge_{1 \le i \le n} sel_{i,D_i}(D_i(x_1, \dots, x_n)) = x_i \mid 1 \le i\ n\}$$

For each pair of different constructors D_i, D_j, we state that they can never map to the same value:

$$S_2 := \{\forall x_1, \dots, x_n\ \forall y_1, \dots, y_k. D_i(x_1, \dots, x_n) \neq D_j(y_1, \dots, y_k) \mid 1 \le i < j \le n\}$$

Finally, we have to give crash-freeness conditions for each D_i:

$$S_3 := \{\forall x_1, \dots, x_n.(\mathrm{CF}(x_1) \wedge \dots \wedge \mathrm{CF}(x_n) \leftrightarrow \mathrm{CF}(D_i(x_1, \dots, x_n))) \mid 1 \le i \le n\}$$