Верифицированный спецификатор языка S-Graph

Владимир Гладштейн

По мотивам статьи Robert Glück, Andrei V. Klimov: Occam's Razor in Metacomputation: the Notion of a Perfect Process Tree

План

Задача

Инструменты

Н.У.О.

Решение

Задача

Начальные данные: S-Graph

```
Prog ::= [Def]
Def  ::= (DEFINE Fname [Var] Tree)
Tree ::= (LET Var Exp Tree)
       | (CALL Fname [Exp])
       | (IF Cntr Tree Tree) | Exp
Cntr ::= (CONS? Arg Var Var)
       | (EQA? Arg Arg)
Exp  ::= Arg | (CONS Exp Exp)
Arg  ::= Val | Var
Val  ::= (ATOM Atom)
Var  ::= (VAR Name)
```

Начальные данные: Семантика

- `int` :: `Tree` $\rightarrow$ `Env` $\rightarrow$ `Exp`

Начальные данные: Семантика

- `int` :: `Tree` $\rightarrow$ `Env` $\rightarrow$ `Exp`
- `int_Prog` :: `Def` $\rightarrow$ `Prog` $\rightarrow$ [`Exp`] $\rightarrow$ `Exp`

Начальные данные: Семантика

- `int :: Tree → Env → Exp`
- `int_Prog :: Def → Prog → [Exp] → Exp`

Пример:

```
int_Prog  main  (main : p)  [CONS 'A 'B]
```

Хотим

- `dev` :: `Tree` $\rightarrow$ `CEnv` $\rightarrow$ `Tree`
- `spec` :: `Def` $\rightarrow$ [`Exp`] $\rightarrow$ `Def`

Хотим

- $\texttt{dev} :: \texttt{Tree} \to \texttt{CEnv} \to \texttt{Tree}$
- $\texttt{spec} :: \texttt{Def} \to [\texttt{Exp}] \to \texttt{Def}$
- $$\texttt{int\_Prog}\ (\texttt{spec}\ \texttt{main}\ \texttt{e}_1)\ (\texttt{main} : \texttt{p})\ \texttt{e}_2 =$$
  $$\texttt{int\_Prog}\ \texttt{main}\ (\texttt{main} : \texttt{p})\ (\texttt{e}_1 +\!\!+\ \texttt{e}_2)$$
- программа после dev не (намного) сложнее

Инструменты

SSReflect Coq

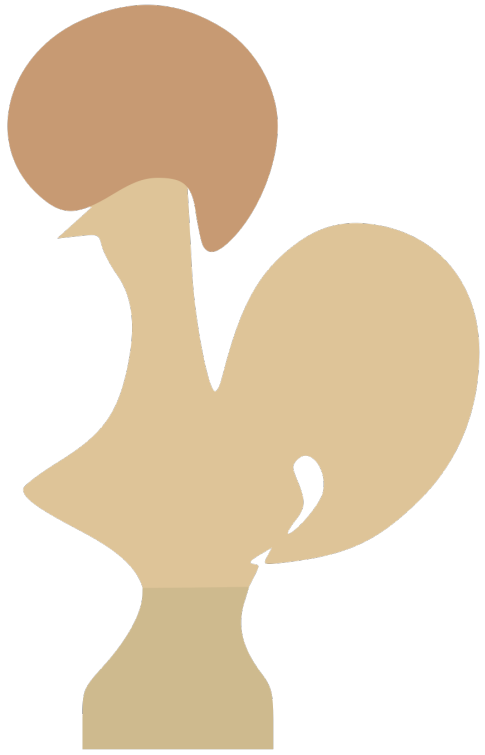

Figure 1: Coq Proof Assistant

H.Y.O.

Меняем язык

Было

```
Cntr ::= (CONS? Arg Var Var)
       | (EQA? Arg Arg)
```

Стало

```
Cntr ::= (CONS? Arg)
       | (EQA? Arg Arg)
Tree += (HT Var Var Var Tree)
```

Меняем `int`

- `int :: Int → Tree → Env → Exp`
- `int_Prog ::`
  `Int → Def → Prog → [Exp] → Exp`

Меняем `dev` : свежие переменные

Было

- `dev` :: `Tree` $\rightarrow$ `CEnv` $\rightarrow$ `Tree`

Надо

- `dev` :: `Var` $\rightarrow$ `Tree` $\rightarrow$ `CEnv` $\rightarrow$ `Tree`

Меняем dev

Было

```
dev (CALL fn as) e =
  dev t (mkEnv vs [x // e | x <- as])
  where (DEFINE _ vs t) = getDef fn
```

Стало

```
dev f k t@(CALL fn as) e = f k t e
```

Решение

caller

```
caller ::
  (Var -> Tree -> Cenv -> Tree) ->
  Var -> Tree -> Cenv -> Tree
caller f k (CALL fn as) (env, rs) =
  f (max k (maxvar t))
    t
    (mkEnv vs [x // env | x <- as], rs)
  where (DEFINE _ vs t) = getDef fn
```

id_call

```
id_call k (CALL g args) c:=
  let (e, _) = c in
    CALL g [x // e | x <- args]
```

correct

Definition (Correct transformation)

Будем говорить, что функция

$$\texttt{f} :: \texttt{Var} \to \texttt{Tree} \to \texttt{Cenv} \to \texttt{Tree}$$

корректа, если, при опредeнных условиях, выполняется

$$\texttt{int n t (comp } \texttt{e}_1 \texttt{ e}_2\texttt{)} \;=$$

$$\texttt{int n (f k t (}\texttt{e}_1\texttt{, rs))}\ \texttt{e}_2$$

где $\texttt{t} = \texttt{CALL fn as}$

Ключевые леммы и опредения

- `correct id_call`
- `correct f` $\rightarrow$ `correct (caller (dev f))`
- `correct (caller (dev id_call))`
- $\forall n$, `correct (devn` n`)`
- `specn :: Int` $\rightarrow$ `Def` $\rightarrow$ `[Exp]` $\rightarrow$ `Def`

Главный Результат

Theorem (Корректость спецификатора)

при опредeнных условиях, для любого $\mathtt{m}$ *выполняется*

$$\mathtt{int\_Prog}\ \mathtt{n}\ \mathtt{main}\ (\mathtt{main} : \mathtt{p})\ (\mathtt{e}_1 +\!\!+\ \mathtt{e}_2) =$$

$$\mathtt{int\_Prog}\ (\mathtt{n}+1)\ (\mathtt{specn}\ \mathtt{m}\ \mathtt{main}\ \mathtt{e}_1)\ (\mathtt{main} : \mathtt{p})\ \mathtt{e}_2$$

Пример: Брюки превращаются

```
Definition eq_str : Def :=
DEFINE eqStr [:: x1; x2] (
  Case x1 Of
  | h1 & t1 -->
      Case x2 Of
      | h2 & t2 -->
          If h1 =? h2 Then
            eqStr $ [:: (t1 : Exp); (t2 : Exp)]
          Else NO
      | ATOM --> NO
  | ATOM --> If (CONS? x2) Then
        NO
      Else If x1 =? x2 Then
        YES
      Else NO).
```

Превращаются...

К сожалению код не исполняемый, но помучавшись можно доказать, что

`specn 2 eq_tree [:: CONS '3 '4] =`

В элегантные шорты

```
DEFINE eqStr [:: x2] (
  Case x2 Of
  | h' & t' -->
      If '3 =? h' Then
        If CONS? t' Then NO
        Else If '4 =? t' Then YES
        Else NO
      Else NO
  | ATOM --> NO).
```