Práctica 2: Semántica del lenguaje EAB

Luis Daniel Aragón Bermúdez César Gustavo Sánchez de la Rosa

Miércoles 19 de septiembre de 2018

Introducción

A continuación se definen los juicios de la semántica dinámica de las EAB.

$$\frac{}{\texttt{bool}[True]\ \texttt{valor}}\ (vtrue) \quad \frac{}{\texttt{bool}[False]\ \texttt{valor}}\ (vfalse) \quad \frac{}{\texttt{num}[n]\ \texttt{valor}}\ (vnum)$$

$$\frac{}{\texttt{bool}[True]\ \texttt{final}}\ (ftrue) \quad \frac{}{\texttt{bool}[False]\ \texttt{final}}\ (ffalse) \quad \frac{}{\texttt{num}[n]\ \texttt{final}}\ (fnum)$$

$$\frac{}{\texttt{add}(\texttt{num}[n], \texttt{num}[m]) \to \texttt{num}[n+m]}\ (eaddf) \quad \frac{t_1 \to t_1'}{\texttt{add}(t_1, t_2) \to \texttt{add}(t_1', t_2)}\ (eaddi)$$

$$\frac{t_2 \to t_2'}{\texttt{add}(\texttt{num}[n], t_2) \to \texttt{add}(\texttt{num}[n], t_2')}\ (eaddd)$$

$$\frac{}{\texttt{prod}(\texttt{num}[n], \texttt{num}[m]) \to \texttt{num}[n+m]}\ (eprodf) \quad \frac{t_1 \to t_1'}{\texttt{prod}(t_1, t_2) \to \texttt{prod}(t_1', t_2)}\ (eprodi)$$

$$\frac{t_2 \to t_2'}{\texttt{prod}(\texttt{num}[n], t_2) \to \texttt{prod}(\texttt{num}[n], t_2')}\ (eprodd)$$

$$\frac{}{\texttt{pred}(\texttt{num}[0]) \to \texttt{num}[0]}\ (tpred0) \quad \frac{}{\texttt{pred}(\texttt{num}[n+1]) \to \texttt{num}[n]}\ (tpreds) \quad \frac{t \to t'}{\texttt{pred}(t) \to \texttt{pred}(t')}\ (tpred)$$

$$\frac{}{\texttt{succ}(\texttt{num}[n]) \to \texttt{num}[n+1]}\ (tsuccn) \quad \frac{t \to t'}{\texttt{succ}(t) \to \texttt{succ}(t')}\ (tsucc)$$

$$\frac{}{\texttt{not}(\texttt{bool}[b]) \to \texttt{bool}[\neg b]}\ (tnotb) \quad \frac{t \to t'}{\texttt{not}(t) \to \texttt{not}(t')}\ (tnot)$$

$$\frac{}{\texttt{and}(\texttt{bool}[b_1], \texttt{bool}[b_2]) \to \texttt{bool}[b_1 \wedge b_2]}\ (eandf) \quad \frac{t_1 \to t_1'}{\texttt{and}(t_1, t_2) \to \texttt{and}(t_1', t_2)}\ (eandi)$$

$$\frac{t_2 \to t_2'}{\texttt{and}(\texttt{bool}[b], t_2) \to \texttt{and}(\texttt{bool}[b], t_2')}\ (eandd)$$

$$\frac{}{\texttt{or}(\texttt{bool}[b_1], \texttt{bool}[b_2]) \to \texttt{bool}[b_1 \vee b_2]}\ (eorf) \quad \frac{t_1 \to t_1'}{\texttt{or}(t_1, t_2) \to \texttt{or}(t_1', t_2)}\ (eori)$$

$$\frac{t_2 \to t_2'}{\texttt{or}(\texttt{bool}[b], t_2) \to \texttt{or}(\texttt{bool}[b], t_2')}\ (eord)$$

$$\frac{}{\texttt{lt}(\texttt{num}[n], \texttt{num}[m]) \to \texttt{bool}[n < m]}\ (eltf) \quad \frac{t_1 \to t_1'}{\texttt{lt}(t_1, t_2) \to \texttt{lt}(t_1', t_2)}\ (elti)$$

$$\frac{t_2 \to t_2'}{\texttt{lt}(\texttt{num}[n], t_2) \to \texttt{lt}(\texttt{num}[n], t_2')}\ (eltd)$$

$$\frac{}{\texttt{gt}(\texttt{num}[n], \texttt{num}[m]) \to \texttt{bool}[n > m]}\ (egtf) \quad \frac{t_1 \to t_1'}{\texttt{gt}(t_1, t_2) \to \texttt{gt}(t_1', t_2)}\ (egti)$$

$$\frac{t_2 \to t_2'}{\texttt{gt}(\texttt{num}[n], t_2) \to \texttt{gt}(\texttt{num}[n], t_2')}\ (egtd)$$

$$\frac{}{\texttt{eq}(\texttt{num}[n], \texttt{num}[m]) \to \texttt{bool}[n = m]}\ (eeqf) \quad \frac{t_1 \to t_1'}{\texttt{eq}(t_1, t_2) \to \texttt{eq}(t_1', t_2)}\ (eeqi)$$

$$\frac{t_2 \to t_2'}{\texttt{eq}(\texttt{num}[n], t_2) \to \texttt{eq}(\texttt{num}[n], t_2')}\ (eeqd)$$

$$\frac{}{\texttt{if}(\texttt{bool}[False], t_2, t_3) \to t_3}\ (eiffalse) \quad \frac{}{\texttt{if}(\texttt{bool}[True], t_2, t_3) \to t_2}\ (eiftrue)$$

$$\frac{t_1 \to t_1'}{\texttt{if}(t_1, t_2, t_3) \to \texttt{if}(t_1', t_2, t_3)}\ (eif)$$

$$\frac{v\ \texttt{valor}}{\texttt{let}(v, xe_2) \to e_2[x := v]}\ (eletf) \quad \frac{t_1 \to t_1'}{\texttt{if}(t_1, xe_2) \to \texttt{if}(t_1', xe_2)}\ (eleti)$$

A continuación se definen los juicios de la semántica estática de las EAB.

$$\frac{}{\Gamma \vdash \texttt{bool[True]} : \texttt{Boolean}}\ (ttrue) \quad \frac{}{\Gamma \vdash \texttt{bool[False]} : \texttt{Boolean}}\ (tfalse) \quad \frac{}{\Gamma, x : T \vdash x : T}\ (tvar)$$

$$\frac{}{\Gamma \vdash \texttt{num[n]} : \texttt{Nat}}\ (tnum) \quad \frac{\Gamma \vdash t_1 : \texttt{Nat} \quad \Gamma \vdash t_2 : \texttt{Nat}}{\Gamma \vdash \texttt{add}(t_1, t_2) : \texttt{Nat}}\ (tadd) \quad \frac{\Gamma \vdash t_1 : \texttt{Nat} \quad \Gamma \vdash t_2 : \texttt{Nat}}{\Gamma \vdash \texttt{mul}(t_1, t_2) : \texttt{Nat}}\ (tmul)$$

$$\frac{\Gamma \vdash t : \texttt{Nat}}{\Gamma \vdash \texttt{succ}(t) : \texttt{Nat}}\ (tsucc) \quad \frac{\Gamma \vdash t : \texttt{Nat}}{\Gamma \vdash \texttt{pred}(t) : \texttt{Nat}}\ (tpred) \quad \frac{\Gamma \vdash t_1 : \texttt{Nat} \quad \Gamma \vdash t_2 : \texttt{Nat}}{\Gamma \vdash \texttt{lt}(t_1, t_2) : \texttt{Boolean}}\ (tlt)$$

$$\frac{\Gamma \vdash t_1 : \texttt{Nat} \quad \Gamma \vdash t_2 : \texttt{Nat}}{\Gamma \vdash \texttt{gt}(t_1, t_2) : \texttt{Boolean}}\ (tgt) \quad \frac{\Gamma \vdash t_1 : \texttt{Nat} \quad \Gamma \vdash t_2 : \texttt{Nat}}{\Gamma \vdash \texttt{eq}(t_1, t_2) : \texttt{Boolean}}\ (teq) \quad \frac{\Gamma \vdash t : \texttt{Boolean}}{\Gamma \vdash \texttt{not}(t) : \texttt{Boolean}}\ (tnot)$$

$$\frac{\Gamma \vdash t_1 : \texttt{Boolean} \quad \Gamma \vdash t_2 : \texttt{Boolean}}{\Gamma \vdash \texttt{and}(t_1, t_2) : \texttt{Boolean}}\ (tand) \quad \frac{\Gamma \vdash t_1 : \texttt{Boolean} \quad \Gamma \vdash t_2 : \texttt{Boolean}}{\Gamma \vdash \texttt{or}(t_1, t_2) : \texttt{Boolean}}\ (tor)$$

$$\frac{\Gamma \vdash t_1 : \texttt{Boolean} \quad \Gamma \vdash t_2 : T \quad \Gamma \vdash t_3 : T}{\Gamma \vdash \texttt{if}(t_1, t_2, t_3) : T}\ (tif) \quad \frac{\Gamma \vdash t_1 : \texttt{Boolean} \quad \Gamma \vdash t_2 : \texttt{Boolean}}{\Gamma \vdash \texttt{or}(t_1, t_2) : \texttt{Boolean}}\ (tor)$$

$$\frac{\Gamma \vdash e_1 : T \quad \Gamma, x : T \vdash e_2 : S}{\Gamma \vdash \texttt{let}(e_1, xe_2) : S}\ (tlet)$$

Ejecución

Utilizando cabal de Cabal Development Team (2018) es sencillo cargar los ejemplos usando el siguiente comando:

```
$ cabal repl
```

Y una vez en la REPL, puede correr cada uno de los ejemplos o todos usando el `main`:

```
*Examples> main
```

A continuación se muestran los ejemplos disponibles.

```
------------------------------
---------- EJEMPLOS ----------
------------------------------

eval1Ej1 :: Exp
eval1Ej1 = eval1 (Add (I 1) (I 2))
-- >> num[3]

eval1Ej2 :: Exp
eval1Ej2 = eval1 (Mul (I 4) (I 0))
-- >> num[0]

eval1Ej3 :: Exp
eval1Ej3 = eval1 (Let "y" (I 10) (Mul (V "y") (I 2)))
-- >> mul(num[10], num[2])

eval1Ej4 :: Exp
eval1Ej4 = eval1 (Let "y" (Succ (I 0)) (Lt (V "y") (I 0)))
-- >> let(num[1], y.lt(var[y], num[0]))

eval1Ej5 :: Exp
eval1Ej5 = eval1 (Let "y" (Add (I 2) (I 3)) (Mul (V "y") (I 0)))
-- >> let(num[5], y.mul(var[y], num[0]))

evalsEj1 :: Exp
evalsEj1 = evals (Mul (Mul (I 10) (I 2)) (B False))
-- >> mul(num[20], bool[False])

evalsEj2 :: Exp
evalsEj2 = evals (Let "y" (Succ (I 0)) (Lt (V "y") (I 0)))
-- >> bool[False]

evalsEj3 :: Exp
evalsEj3 =  evals (Let "y" (Pred (I 11)) (Lt (V "y") (I 10)))
-- >> bool[False]

evalsEj4 :: Exp
evalsEj4 =  evals (Let "y" (Mul (I 10) (I 10))
                (Eq (V "y") (I 100)))
-- >> bool[True]

evalsEj5 :: Exp
evalsEj5 =  evals (Let "y" (And (B True) (B True))
                (Or (V "y") (B False)))
-- >> bool[True]

evalEj1 :: Exp
```

```haskell
evalEj1 = eval (Add (Mul (I 1) (I 6)) (I 9))
-- >> num[15]

evalEj2 :: Exp
evalEj2 =  eval (Succ (Mul (I 1) (I 6)))
-- >> num[7]

evalEj3 :: Exp
evalEj3 = eval (Pred (B True))
-- >> *** Exception: [Pred] requires a number ...

evalEj4 :: Exp
evalEj4 = eval (And (Eq (I 2) (I 2)) (Eq (Add (I 1) (I 1)) (I 2) ))
-- >> bool[True]

evalEj5 :: Exp
evalEj5 = eval (And (Eq (Mul (I 2) (I 4)) (Pred (I 9)))
  (Or (Eq (Add (I 4) (I 4)) (I 0) ) (B False)))
-- >> bool[False]

vtEj1 :: Bool
vtEj1 = vt [("x", Boolean)] (And (Eq (Mul (I 2) (I 4)) (Pred (I 9)))
  (Or (Eq (Add (I 4) (I 4)) (I 0) ) (V "x"))) Boolean
-- >> True

vtEj2 :: Bool
vtEj2 = vt [("x", Boolean)] (And (Eq (Mul (I 2) (I 4)) (Pred (I 9)))
  (Or (Eq (Add (I 4) (I 4)) (I 0) ) (V "x"))) Nat
-- >> False

vtEj3 :: Bool
vtEj3 = vt [("y", Boolean)] (And (And (B True) (B True))
  (Or (V "y") (B False))) Boolean
-- >> True

vtEj4 :: Bool
vtEj4 = vt [] (Let "x" (Add (I 1) (I 2))
  (Eq (Mul (Add (V "x") (I 5)) (I 0)) (Add (V "x") (I 2)))) Boolean
-- >> True

vtEj5 :: Bool
vtEj5 = vt [("x", Nat)] (Let "y" (Add (I 1) (I 2))
  (Eq (Mul (Add (V "x") (I 5)) (I 0)) (Add (V "x") (I 2)))) Boolean
-- >> True
```

Implementación

Se incluye una implementación de la semántica de las Expresiones Aritmético Booleanas (**EAB**) cuya tipo de Haskell puede verse a continuación.

```haskell
data Exp = V Id | I Int | B Bool | Add Exp Exp
| Mul Exp Exp | Succ Exp | Pred Exp | Not Exp
| And Exp Exp | Or Exp Exp | Lt Exp Exp | Gt Exp Exp
| Eq Exp Exp | If Exp Exp Exp | Let Id Exp Exp deriving (Eq)
```

Además, se implementaron dos semánticas de las EAB.

Semántica dinámica

Se implementaron las siguientes funciones principales:

- **eval1**. Devuelve la transición e' tal que `eval1` $e = e' \iff e \to e'$.
- **evals**. Dado e, devuelve e' si existe e' tal que $e \to^* e'$ y e' está bloqueado.
- **eval**. Devuelve la evaluación de un programa tal que `eval` $e = e' \iff e \to^* e'$ y e' es un valor. En caso de que e' no sea un valor muestra un mensaje de error.

La parte más importante fue detectar los estados bloqueados en `eval1` usando el tipo `Maybe`. Después fue fácil implementar `evals` en términos de `eval1` y `eval` en términos de `evals`. Para `eval` además, nos ayudamos del tipo `Either` para guardar nuestros mensajes de error.

Semántica estática

Se implementó la siguiente función:

- **vt**. Verifica el tipado de un programa tal que `vt` Γ `e` T = `True` $\iff \Gamma \vdash$ `e` $: T$.

Bibliografía

Cabal Development Team. 2018. «Cabal: A framework for packaging Haskell software». http://hackage.haskell.org/package/Cabal.