Full reduction at full throttle

Extension to the Calculus of Inductive Constructions

Gerco van Heerdt

December 19, 2014

Symbolic CIC

$$
\begin{aligned}
\textbf{Term} \ni t, T, P ::= {} & x \mid t_1\ t_2 \mid v \mid C_i(\vec{t}) \\
& \mid \textbf{case}_{\langle P \rangle}\, t\ \textbf{of}\ (C_i(\vec{x}_i) \to t_i)_{i \in I} \\
& \mid \textbf{fix}_m(f : T := t)
\end{aligned}
$$

$$
\textbf{Val} \ni v ::= \lambda x.t \mid [a\, \vec{v}] \mid C_i(\vec{v})
$$

$$
\begin{aligned}
\textbf{Atom} \ni a ::= {} & \tilde{x} \mid s \mid \Pi x : T_1.T_2 \\
& \mid \textbf{case}_{\langle P \rangle}\, a\ \textbf{of}\ (C_i(\vec{x}_i) \to t_i)_{i \in I} \\
& \mid \textbf{fix}_m(f : T := t)
\end{aligned}
$$

Additional reduction rules

$$\mathbf{case}_{\langle P \rangle}\, C_i(\vec{v})\ \mathbf{of}\ (C_i(\vec{x}_i) \to t_i)_{i \in I} \to_\iota t_i\{\vec{x}_i \leftarrow \vec{v}\}$$

$$\mathbf{case}_{\langle P \rangle}\, [a]\ \mathbf{of}\ (C_i(\vec{x}_i) \to t_i)_{i \in I} \to_\iota [\mathbf{case}_{\langle P \rangle}\, a\ \mathbf{of}\ (C_i(\vec{x}_i) \to t_i)_{i \in I}]$$

$$\mathbf{fix}_m(f : T := t)\, v_1 \ldots v_{m-1}\, C_i(\vec{v}) \to_\iota t\{f \leftarrow \mathbf{fix}_m(f : T := t)\}\, v_1 \ldots v_{m-1}\, C_i(\vec{v})$$

$$\mathbf{fix}_m(f : T := t)\, v_1 \ldots v_{m-1}\, [a] \to_\iota [\mathbf{fix}_m(f : T := t)\, v_1 \ldots v_{m-1}\, a]$$

Readback for pattern matching and fixpoints

$$\mathscr{R}_A(\mathbf{case}_{\langle P \rangle}\, a\ \mathbf{of}\ (C_i(\vec{x}_i) \to t_i)_{i \in I}) =$$
$$\mathbf{case}_{\langle P \rangle}\, \mathscr{R}_A(a)\ \mathbf{of}\ (C_i(\vec{x}_i) \to \mathscr{N}(f(C_i(\overrightarrow{[\tilde{x}_i]}))))_{i \in I}$$
$$\text{where } f = \lambda x.\mathbf{case}_{\langle P \rangle}\, x\ \mathbf{of}\ (C_i(\vec{x}_i) \to t_i)_{i \in I}$$

$$\mathscr{R}_A(\mathbf{fix}_m(f : T := t)) = \mathbf{fix}_m(f : T := \mathscr{N}((\lambda f.t)[\tilde{f}]))$$

Extended value module

```
module type Values = module type
  type head =
    | ...
    | Construct of int * t array
  type atom =
    | Var of var
    | Sort of sort
    | Prod of t * t
    | Match of annot * t * t * (t -> t)
    | Fix of (t -> t) * t * int
  ...
  val mkConstruct : int -> t array -> t
end
```

Compilation scheme

Sorts, products, and constructors

$$
\begin{aligned}
\ulcorner s \urcorner^B &= \texttt{mkAccu (Sort s)} \\
\ulcorner \Pi x : T.U \urcorner^B &= \texttt{mkAccu (Prod(}\ulcorner T \urcorner^B\texttt{, }\ulcorner \lambda x.U \urcorner^B\texttt{))} \\
\ulcorner C_i(\vec{t}) \urcorner^B &= \texttt{mkConstruct i [|}\ulcorner \vec{t} \urcorner^B\texttt{|]}
\end{aligned}
$$

Compilation scheme

Pattern matching

$$
\begin{aligned}
&\ulcorner \texttt{case}_{\langle P \rangle}\, t\ \texttt{of}\ (C_i(\vec{x}_i) \to t_i)_{i \in I} \urcorner^{B}\ = \\
&\quad \texttt{let rec case c =} \\
&\qquad \texttt{match c with} \\
&\qquad \texttt{|}\ \ulcorner C_1(\vec{x}_1) \urcorner^{B \cup \{\vec{x}_1\}}\ \texttt{->}\ \ulcorner \vec{t}_1 \urcorner^{B \cup \{\vec{x}_1\}} \\
&\qquad \texttt{|}\ \ldots \\
&\qquad \texttt{|}\ \ulcorner C_n(\vec{x}_n) \urcorner^{B \cup \{\vec{x}_n\}}\ \texttt{->}\ \ulcorner \vec{t}_n \urcorner^{B \cup \{\vec{x}_n\}} \\
&\qquad \texttt{|}\ \texttt{\_ -> mkAccu (Match (}\tilde{I}\texttt{,c,}\ulcorner P \urcorner^{B}\texttt{,case))} \\
&\quad \texttt{in case}\ \ulcorner t \urcorner^{B}
\end{aligned}
$$

Compilation scheme

Fixpoints

$\ulcorner \mathbf{fix}_m(f : T := t) \urcorner^B$ =

```
  let fnorm f = ⌜t⌝^{B∪{f}} in
  let rec f =
    mkLam (fun x1 -> ... -> mkLam(fun xm ->
      if is_accu xm then
        mkAccu (Fix (fnorm,⌜T⌝^B,m)) x1 ... xm
      else
        fnorm f x1 ... xm
    ) ... )
  in f

let is_accu v = match head v with
| Accu _ -> true
| _ -> false
```

Compilation scheme

Inductive types

The inductive type

```
Inductive I := C1 : T1 | ... | Cn : Tn
```

is translated to

```
type I = Accu_I of t
       | C1 of t * ··· * t
       | ...
       | Cn of t * ··· * t
```

where the signatures match the arity of each constructor.

`mkConstruct i` $\vec{v}$ `= Obj.magic` $C_i(\vec{v})$

Example

$$b = \lambda m.\lambda n.\textbf{case}\ m\ \textbf{of}\ (C_1() \to n, C_2(p) \to C_2(add\ p\ n))$$

$\ulcorner \texttt{fix}_1(add := b) \urcorner^{\emptyset}$

```
= let norm_add add = ⌜b⌝{add} in
  let rec add =
    mkLam (fun x1 ->
      if is_accu x1 then
        mkAccu (Fix (norm_add ,1)) x1
      else
        norm_add add x1)
  in add
```

Example

$$b = \lambda m.\lambda n.\textbf{case}\, m\ \textbf{of}\ (C_1() \to n, C_2(p) \to C_2(add\ p\ n))$$

$$\begin{aligned}
&\ulcorner \texttt{fix}_1(add := b) \urcorner^{\emptyset} \\
= \ &\texttt{let norm_add add = } \ulcorner b \urcorner^{\{add\}} \texttt{ in} \\
&\texttt{let rec add =} \\
&\quad \texttt{(fun } x_1 \texttt{ ->} \\
&\qquad \texttt{if is_accu } x_1 \texttt{ then} \\
&\qquad\quad \texttt{mkAccu (Fix (norm_add ,1)) } x_1 \\
&\qquad \texttt{else} \\
&\qquad\quad \texttt{norm_add add } x_1\texttt{)} \\
&\texttt{in add}
\end{aligned}$$

Example

$$b = \textbf{case}\ m\ \textbf{of}\ (C_1() \to n, C_2(p) \to C_2(add\ p\ n))$$

$\ulcorner \lambda m.\lambda n.b \urcorner^{\{add\}}$

```
= mkLam(fun m -> ⌜λn.b⌝^{add,m})
= mkLam(fun m -> mkLam(fun n -> ⌜b⌝^{add,m,n}))
= fun m -> fun n -> ⌜b⌝^{add,m,n}
```

$\ulcorner b \urcorner^{\{add,m,n\}}$

```
= let rec case c =
    match c with
    | ⌜C1()⌝^{add,m,n} -> ⌜n⌝^{add,m,n}
    | ⌜C2(p)⌝^{add,m,n,p} -> ⌜C2(add p n))⌝^{add,m,n,p}
    | _ -> mkAccu (Match (c,case))
  in case ⌜m⌝^{add,m,n}
```

Example

```
   let rec case c =
     match c with
     | ⌜C1()⌝{add,m,n} -> ⌜n⌝{add,m,n}
     | ⌜C2(p)⌝{add,m,n,p} -> ⌜C2(add p n)⌝{add,m,n,p}
     | _ -> mkAccu (Match (c,case))
   in case m
= let rec case c =
     match c with
     | C1() -> n
     | C2(p) -> C2(⌜add p n⌝{add,m,n,p})
     | _ -> mkAccu (Match (c,case))
   in case m
```

Example

```
   let rec case c =
     match c with
     | C1() -> n
     | C2(p) -> C2(⌜add p n⌝{add,m,n,p})
     | _ -> mkAccu (Match (c,case))
   in case m
= let rec case c =
     match c with
     | C1() -> n
     | C2(p) -> C2(add p n)
     | _ -> mkAccu (Match (c,case))
   in case m
```

Example

```
let norm_add add =
  fun m -> fun n ->
    let rec case c =
      match c with
      | C1() -> n
      | C2(p) -> C2(add p n)
      | _ -> mkAccu (Match (c,case))
    in case m
in
let rec add =
  (fun x1 ->
    if is_accu x1 then
      mkAccu (Fix (norm_add,1)) x1
    else
      norm_add add x1)
in add
```

Example

```
let norm_add add m n =
  let rec case c =
    match c with
    | C1() -> n
    | C2(p) -> C2(add p n)
    | _ -> mkAccu (Match (c,case))
  in case m
in
let rec add m =
  if is_accu m then
    mkAccu (Fix (norm_add,1)) m
  else
    norm_add add m
in add
```

Example

Their version

```
let norm_add f m n =
  let rec case c =
    match c with
    | Accu_nat _ -> mk_sw_accu [...]
      (cast_accu m) pred_add (case f n)
    | C1() -> n
    | C2(p) -> C2(f p n)
in
let rec add m n =
  if is_accu m then
    mk_fix_accu [...]
      fixtype_add normtbl_add m n
  else
    case m n
in add
```

Example

Optimization

```
let rec case f n m =
  match m with
  | Accu_nat _ -> mk_sw_accu [...]
    (cast_accu m) pred_add (case f n)
  | C1() -> n
  | C2(p) -> C2(f p n)

let norm_add f m n = case_add f n m

let rec add m n =
  match m with
  | Accu_nat _ -> mk_fix_accu [...]
      fixtype_add normtbl_add m n
  | C1() -> n
  | C2(p) -> C2(f p n)
```