The Abstract *Term* Type

Eli Barzilay

`eli@cs.cornell.edu`

March 4, 2002

Quick Review

- Quoted terms are the same as normal Nuprl terms, with a "quotedness level" tag.
- Main point: they are still binding terms.
- For example, the quotation of $\lambda x.\ x + 1$ is $\underline{\lambda(}x\underline{.}\ x\underline{{}+1)}$
- In other words: these are *abstract* terms — we don't get concrete variable names.
- Until now, too much hand-waving.

Motivation

- Reflection by exposing internal functionality
  - Using concrete terms would be $\underline{\lambda}(`x', t)$
  - Different syntax from actual Nuprl terms
  - Leads to problems (reimplementation, efficiency)
- Using the same terms mean that quotations of binding operators are themselves binding.
- An additional motivation: expose a higher level interface.

Defining *Term*

- Need a general *Term* type.

  (*Term*$_A$ for all types A — too restrictive)

- How do we determine if something is in *Term*?

  Intuition:

$$\vdash \underline{\mathrm{opid}}(\overline{v_1}.b_1; \overline{v_2}.b_2; \ldots; \overline{v_n}.b_n) \in \textit{Term}$$

$$\begin{aligned}
\text{if } \; & \overline{v_1} : \textit{Term} \vdash b_1 \in \textit{Term} \\
& \overline{v_2} : \textit{Term} \vdash b_2 \in \textit{Term} \\
& \vdots \\
& \overline{v_n} : \textit{Term} \vdash b_n \in \textit{Term}
\end{aligned}$$

The Problem

- Problem:
  $\vdash \underline{\lambda(}x\underline{.\ }\text{if } x = \underline{0} \text{ then } \underline{1} \text{ else } \underline{2)} \in \textit{Term}$
- Compare this to:
  1. $\underline{\lambda(}x\underline{.\ \text{if }}x\underline{\ = 0 \text{ then } 1 \text{ else } 2)}$
  2. $\underline{\lambda(}x\underline{.\ }\text{if } x = \underline{0} \text{ then } \underline{1} \text{ else } \underline{1)}$
  3. $\lambda x.\ \text{if } x = \underline{0} \text{ then } \underline{1} \text{ else } \underline{2}$
- The bound variable should not be used as a value, just as a "template hole".

Analogies

- This situation is similar to:
  - Universe expressions in Stuart's thesis: "computationally inert".
  - Bindings in Scheme high-level macros:

```
(define-syntax foo
  (syntax-rules ()
    ((foo x body ...)  (let ((x 123)) body ...))))
```

    (everything except for template variables is implicitly quoted.)

Conclusion

- Conclusion: variables bound by quoted terms are different than normal bindings: their cannot be used.
- There is no way to define an appropriate rule inside the system, must give a new primitive rule.

Solution: `is_subst`

- `is_subst`: a predicate that comes with a primitive rule.
- It is used to "scan" a term, making sure that it is a valid quotation.
- Two cases:
  - $H \vdash \mathtt{is\_subst}(x_1, x_2, \ldots, x_n.\ x_i)$
  - $H \vdash \mathtt{is\_subst}(\overline{x}.\ \underline{\mathsf{opid}}(\overline{y_1}.\ b_1; \ldots; \overline{y_n}.\ b_n))$

    where <u>opid</u> is some quoted opid and

    $H \vdash \mathtt{is\_subst}(\overline{x}, \overline{y_1}.\ b_1)$

    $H \vdash \mathtt{is\_subst}(\overline{x}, \overline{y_2}.\ b_2)$

    $\vdots$

    $H \vdash \mathtt{is\_subst}(\overline{x}, \overline{y_n}.\ b_n)$

Term Definition

- Define the *Term* type as the intuitive meaning, using the standard known type of concrete terms:

$$\textit{Term} \equiv \textit{CTerm}//\alpha$$

- *Term* values are built using *shifted* operators.
- Using $\texttt{is\_subst}_n$ which is the `is_subst` predicate for functions of arity n, we have the following example:

$$\underline{\lambda} : \{f : \textit{Term} \to \textit{Term} \mid \texttt{is\_subst}_1(f)\} \to \textit{Term}$$

- Also needed: *Var* and *CVar*, atomic variable types.

Miscellaneous Operations

$[\,\cdot\,]_\alpha$ the α-equivalence class ('unpick').

$\downarrow\cdot$ the inverse ('pick') (uses lexicographic order).

$\cdot[\cdot/\cdot]$ standard concrete substitution.

$\cdot[\![\cdot/\cdot]\!]$ substitution for *Terms*: $b[\![\overline{x}/\overline{v}]\!] = [\,\downarrow b[\downarrow\overline{x}/\downarrow\overline{v}]\,]_\alpha$.

$\mathrm{newcvar}(\cdot)$ get a fresh *CVar*.

$\mathrm{newvar}(\cdot)$ the same for *Term*: $\mathrm{newvar}(x) = [\,\mathrm{newcvar}(\downarrow x)\,]_\alpha$.

$\mathrm{newcvar}_n(\cdot)$, $\mathrm{newvar}_n(\cdot)$
extensions that return a number of distinct fresh variables.

These are extended to structures in the obvious way.
$\mathrm{newcvar}(\cdot)$ and $\mathrm{newvar}(\cdot)$ are also extended to functions by plugging $\underline{0}$.

Defining `is_subst`

A function f from *Term*s to *Term* is a substitution function iff it has a matching substitution.

- First, using *CTerm*s:

$$\texttt{is\_subst}(f) \equiv \exists b : \textit{CTerm}.\ \exists\overline{v} : \textit{CVar}.\ \forall\overline{t} : \textit{CTerm}.\ f([\ \overline{t}\ ]_\alpha) = [\ b[\overline{t}/v]\ ]_\alpha$$

- Using *Term*s for the function's arguments:

$$\texttt{is\_subst}(f) \equiv \exists b : \textit{CTerm}.\ \exists\overline{v} : \textit{CVar}.\ \forall\overline{r} : \textit{Term}.\ f(\overline{r}) = [\ b[\downarrow\overline{r}/v]\ ]_\alpha$$

- Finally, using only *Term*s:

$$\texttt{is\_subst}(f) \equiv \exists b_a : \textit{Term}.\ \exists\overline{v_a} : \textit{Var}.\ \forall\overline{t_a} : \textit{Term}.\ f(\overline{t_a}) = b_a[\![\overline{t_a}/\overline{v_a}]\!]$$

Related Facts

Proving that all three definitions are equivalent leads to several useful facts:

- $x = [\,\downarrow x\,]_\alpha$
- $x =_\alpha \downarrow [\,x\,]_\alpha$
- $x = \downarrow [\,x\,]_\alpha$, if $x \in$ *CVar* $(x \in \textit{CVar} \Rightarrow \downarrow [\,x\,]_\alpha = \downarrow \{x\} = x)$
- $\forall \overline{x_1}, \overline{x_2} \in$ *CTerm*. $\overline{x_1} =_\alpha \overline{x_2} \Rightarrow b[\overline{x_1}/\overline{v}] =_\alpha b[\overline{x_2}/\overline{v}]$
- $\forall b_1, b_2, \overline{t} \in$ *CTerm*, $\overline{v} \in$ *CVar*. $b_1 =_\alpha b_2 \Rightarrow b_1[\overline{t}/\overline{v}] =_\alpha b_2[\overline{t}/\overline{v}]$

Shifted Operators

- A shifted operator, <u>opid</u>, takes substitution functions of some arities and return a *Term* value.
- The obvious way to do this is:
  - use the functions and fresh variable names to construct a *CTerm*,
  - produce the *Term* from this result using $[\,\cdot\,]_\alpha$.
- The construction could be as in this example:

$$\underline{\lambda}(f) = [\ \langle `\lambda', [\langle[\downarrow\mathrm{newvar}(f)], \downarrow f(\mathrm{newvar}(f))\rangle]\rangle\ ]_\alpha$$

- A principle held so far: treat substitution as a black box.
- This means that the actual representation is irrelevant, as long as the substitution facts hold.

Shifted Operators

Instead, use abstract types and constructors:

- *OpId*, *BndCTerm*: types for opids and bound subterms.
- $\texttt{mkBCTerm}(\overline{x}, t)$: constructor for *BndCTerm*s.
- $\texttt{mkCTerm}(o,\ [\texttt{mkBCTerm}(\overline{x_1}, t_1), \ldots, \texttt{mkBCTerm}(\overline{x_m}, t_m)])$: constructor for *CTerm*s.
- *SubstFunc*$_n = \{f : \textit{Term} \xrightarrow{n} \textit{Term} \mid \texttt{is\_subst}_n(f)\}$
- $\texttt{mkTerm}(o, [\langle n_1, f_1\rangle, \ldots, \langle n_m, f_m\rangle])$: constructor for *Term*s.

Shifted Operators

Now we can define `mkTerm`:

$$
\begin{array}{l}
\texttt{mkTerm}(o, [\langle n_1, f_1\rangle, \ldots, \langle n_m, f_m\rangle]) = \\
\quad \text{let } \overline{x_1} = \text{newvar}_{n_1}(f_1), \ldots, \overline{x_m} = \text{newvar}_{n_m}(f_m) \text{ in} \\
\qquad \big[\, \texttt{mkCTerm}(o,\ \big[\texttt{mkBCTerm}({\downarrow}\overline{x_1}, {\downarrow}f_1(x_1)), \\
\qquad\qquad\qquad\qquad \ldots, \\
\qquad\qquad\qquad\qquad \texttt{mkBCTerm}({\downarrow}\overline{x_m}, {\downarrow}f_m(x_m))\big]) \,\big]_\alpha
\end{array}
$$

A shifted operator is a *Term* constructor which uses `mkTerm` with some fixed operator name and arity list. For example, '$\underline{\lambda}$' and '$\underline{\Sigma}$' are defined as:

$$\underline{\lambda}(f) = \texttt{mkTerm}(\text{`}\lambda\text{'}, [\langle 1, f\rangle]), \quad \underline{\Sigma}(f, g) = \texttt{mkTerm}(\text{`}\Sigma\text{'}, [\langle 0, f\rangle, \langle 1, g\rangle])$$

More Facts

- $\forall t, \overline{x_1}, \overline{x_2} \in$ *CTerm*, $\overline{v_1}, \overline{v_2}, \overline{u} \in$ *CVar*.

  $\mathrm{len}(\overline{x_1}) = \mathrm{len}(\overline{v_1}) = \mathrm{len}(\overline{u})\ \&\ \mathrm{len}(\overline{x_2}) = \mathrm{len}(\overline{v_2})\ \&$

  $\overline{u}$ are distinct, and they do not occur in $t, \overline{x_2}$

  $\Rightarrow\ t[\overline{x_1}, \overline{x_2}/\overline{v_1}, \overline{v_2}] =_\alpha t[\overline{u}, \overline{x_2}/\overline{v_1}, \overline{v_2}][\overline{x_1}/\overline{u}]$

- $\forall t, \overline{x_1}, \overline{x_2} \in$ *Term*, $\overline{v_1}, \overline{v_2}, \overline{u} \in$ *Var*.

  $\mathrm{len}(\overline{x_1}) = \mathrm{len}(\overline{v_1}) = \mathrm{len}(\overline{u})\ \&\ \mathrm{len}(\overline{x_2}) = \mathrm{len}(\overline{v_2})\ \&$

  $\overline{u} = [\ \overline{w}\ ]_\alpha$ for some $\overline{w} \in$ *CVar*, where $\overline{w}$ are distinct, and not in $\downarrow t, \downarrow\overline{x_2}$

  $\Rightarrow\ t[\![\overline{x_1}, \overline{x_2}/\overline{v_1}, \overline{v_2}]\!] = t[\![\overline{u}, \overline{x_2}/\overline{v_1}, \overline{v_2}]\!][\![\overline{x_1}/\overline{u}]\!]$

- If none of $\overline{\overline{x}}$ occur free in $\overline{r}, \overline{v}$ then:

  $\mathtt{mkCTerm}(o,\ [\mathtt{mkBCTerm}(\overline{x_1}, t_1), \ldots, \mathtt{mkBCTerm}(\overline{x_m}, t_m)])[\overline{r}/\overline{v}]$

  $=_\alpha \mathtt{mkCTerm}(o,$

  $[\mathtt{mkBCTerm}(\overline{x_1}, t_1[\overline{r}/\overline{v}]), \ldots, \mathtt{mkBCTerm}(\overline{x_m}, t_m[\overline{r}/\overline{v}])])$

Back to the Rule

- Flashback #1:
  - $H \vdash \texttt{is\_subst}(x_1, x_2, \ldots, x_n.\ x_i)$
  - $H \vdash \texttt{is\_subst}(\overline{x}.\ \underline{\mathsf{opid}}(\overline{y_1}.\ b_1; \ldots; \overline{y_n}.\ b_n))$

    where $\underline{\mathsf{opid}}$ is some quoted opid

    $H \vdash \texttt{is\_subst}(\overline{x}, \overline{y_1}.\ b_1)$

    $H \vdash \texttt{is\_subst}(\overline{x}, \overline{y_2}.\ b_2)$

    $\vdots$

    $H \vdash \texttt{is\_subst}(\overline{x}, \overline{y_n}.\ b_n)$

- Flashback #2:
  - $\texttt{is\_subst}(f) \equiv$

    $\exists b_a : \textit{Term}.\ \exists \overline{v_a} : \textit{Var}.\ \forall \overline{t_a} : \textit{Term}.\ f(\overline{t_a}) = b_a[\![\overline{t_a}/\overline{v_a}]\!]$

Fun

The first case is the simple one:

$\forall n, i : \mathbb{N}^+ .\ i \leq n \ \Rightarrow\ \texttt{is\_subst}_n(\pi_n^i)$

Choose distinct $v_1, \ldots, v_n$ variables, and let

$$b = \pi_n^i(v_1, \ldots, v_n) = v_i$$

Now, showing that:

$\forall t_1, \ldots, t_n$: *Term*.

$$\pi_n^i(t_1, \ldots, t_n) = v_i[\![t_1, \ldots, t_n / v_1, \ldots, v_n]\!]$$

is trivial.

The Real Fun

The *real* fun will take to long to fit on a slide. . .