Enforcing Abstraction Boundaries

Gul Agha
CS 322, Fall 2006
Enforcing Type Abstractions

Are these type correct?

(- #\a #\b)
(- #\a 1)

Representation independence of the data type:
• Correct in the implementation
• Incorrect in the abstraction (client side)
Defining Abstraction Boundaries

Define a new type \textit{tid} (represented by type \textit{t}) with interface procedures \textit{p1, p2, ...}

\begin{verbatim}
lettype tid = t
    \textit{t1 \ p1 (t11 \ x11, ..., t1 \ xn_1)} = e1
    \textit{t2 \ p2 (t21 \ x21, ..., t2 \ xn_2)} = e2
    ...
\end{verbatim}

\textit{in body}
Example: Non-negative integers

```
lettype myint = int
  myint zero ( ) = 1
  myint succ (myint x) = add1 (x)
  myint pred (myint x) = sub1 (x)
  bool iszero? (myint x) =
    zero? (sub1 (x))
in  body
```
Finite Functions Interface

lettype ff = (int \rightarrow int)

ff zero-ff () = ...

ff extend-f (int k, int val, ff old-ff) = ...

int apply-ff (ff f, int k) = ...

in

body
Finite Function Representation

\[
\begin{align*}
\text{ff zero-ff} (\ ) &= \text{proc (int } k \text{) } 0 \\
\text{ff extend-f} (\text{int } k, \text{ int } \text{val, ff old-ff}) &= \text{proc (int } k1) \\
&\quad \text{if zero? } (- (k1 \ k)) \\
&\quad \text{then val} \\
&\quad \text{else (apply-ff old-ff } \text{k1)} \\
\text{int apply-ff} (\text{ff f, int } k) &= (f \ k)
\end{align*}
\]
Sample Application

lettype ff = (int \to\ int)
    ....
in
    let ff1= (extend-ff 1 11
             (extend-ff 2 22
             (zero-ff)))
in (apply-ff ff1 2)
Sample Application?

lettype ff = (int → int)
  ....
in
  let ff1 = (extend-ff 1 11
      (extend-ff 2 22
           (zero-ff)))
in (ff1 2)
Abstract Syntax for Type Definitions

lettype-exp
  (type-name texp
   result-texps proc-names
   arg-texpss idss bodies
   lettype-body)

tid-type-exp (id)
Extending the Type Environment

Bind

• *ordinary identifiers to types*
• *type identifiers to types*

Type identifiers are added one at a time. Extending the type environment allows the same name to be used for ordinary and type identifiers!
(define-datatype
type-environment type-environment?
(empty-tenv-record)
(extended-tenv-record
  (syms (list-of symbol?))
  (vals (list-of type?))
  (tenv type-environment?))
(typedef-record
  (name symbol?)
  (definition type?)
  (tenv type-environment?)))
(define apply-tenv
  (lambda (tenv sym)
    (cases type-environment tenv
      (empty-tenv-record ( ) error…)
      (extended-tenv-record (syms vals tenv)
        (let ((pos (list-find-position sym syms)))
          (if (number? pos)
              (list-ref vals pos)
              (apply-tenv tenv sym)))))))
  (typedef-record (name type tenv)
    (apply-tenv tenv sym))))
(define expand-type-expression
  (lambda (texp tenv)
    (cases type-exp texp
      (tid-type-exp (id)
        (find-typedef tenv id))
      (int-type-exp () (atomic-type 'int))
      (bool-type-exp () (atomic-type 'bool))
      (proc-type-exp (…) …))))
Expanding Type Expression II

\[ (\text{proc-type-exp} \quad (\text{arg-texps result-texp}) \quad \text{(proc-type} \quad (\text{expand-type-expressions} \quad \text{arg-texps tenv}) \quad (\text{expand-type-expression} \quad \text{result-texp tenv})))))))) \]
Expanding Type Expressions

(define expand-type-expressions
  (lambda (texps tenv)
    (map
      (lambda (texp)
        (expand-type-expression
          (expand-type-expression
            texps))
      texps)))
(define type-of-proc-exp
  (lambda (texps ids body tenv)
    (let ((arg-types
           (expand-type-expressions
            texps tenv)))
      (let ((result-type
             (type-of-expression body
              (extend-tenv
               ids arg-types tenv))))
        (proc-type arg-types result-type))))
Typing Type Definitions

- Check procedure declarations in an environment
  - type identifiers are bound their representation.
- Check body in an environment
  - type identifier is bound to new atomic type.
Two Typing Environments

- Type environment for the implementation
  \( tenv\text{-implementation} = [tid = t] tenv \)
- Type environment for the client
  \( tenv\text{-implementation} = \]
  \( [tid = <\text{fresh atomic type}>] tenv \)
Type Environments for Procedures

- Type environment for representation (procedures)

\[ tenv_i = [x_1^1 = t_i^1, x_2^2 = t_i^2, \ldots] \]

\[ [p_1 = (t_{11} \times t_{12} \times \ldots \Rightarrow t_1)^\wedge, \]

\[ p_2 = (t_{21} \times t_{22} \times \ldots \Rightarrow t_2)^\wedge, \]

\[ \ldots] tenv-implementation \]

\( t^\wedge \) means expansion of the type expression in tenv-implementation
Type Environment for the Body

- Type environment in the body of lettype

\[
\text{tenv-body} = \\
[p1 = (t11 \ast t12 \ast \ldots \Rightarrow t1)^+] \\
p2 = (t21 \ast t22 \ast \ldots \Rightarrow t2)^+] \\
\ldots \\
] \text{tenv-client}
\]

\(t^+\) denotes the expansion of the type expression \(t\) in \(\text{tenv-client}\)