Types and type checking

A domain with two types

- Use a sum domain so that we can distinguish values of different type
- Storeable-value = Tr + Nat
- Members: inNat(zero), inTr(true) etc.
- Now type errors can occur. E.g. x + 1 where x has been assigned the value true
- We also need to distinguish between error values and real values:
- Expressible-value = Storeable-value + Errvalue
  where Errvalue = Unit (the domain with only one member)
- Members: inStoreable-value(inNat(zero)), inStoreable-value(inTr(true)), inErrvalue()

Places where type checking occurs

- Expressions:
  - x + 1 where x is bound to true – need to check both halves of a binary expression before carrying out the operation
- Assignment:
  - y := x + 1 where x is bound to true – need to prevent the assignment happening
- Command sequence:
  - y := x + 1; z := 1 where x + 1 is a type error – need to prevent execution of further commands
Type-checking functionals

Expressions:
- check-expr : (Store → Expressible-value) → (Store → Expressible-value)
- Assignment:
  - check-result : (Store → Expressible-value) → (Storeable-value → Store → Post-store) → (Store → Post-store)
- Sequence:
  - check-cmd : (Store → Post-store) → (Store → Post-store) → (Store → Post-store)

Definition of check-expr
- f1 check-expr f2 = λs.cases (f1 s) of
  - isStoreable-value(v) → (f2 v s)
  - isErrvalue() → inErrvalue()
end
- f1 will evaluate the first subexpression
- If the result is a Storeable-value, then the evaluation of the second expression will take place (it may fail also)
- If the first subexpression fails, then the whole expression fails

Definition of check-result
- f check-result g = λs.cases (f s) of
  - isStoreable-value(v) → (g v s)
  - isErrvalue() → inErrvalue()
end
- f will evaluate the expression
- If the result is a Storeable-value, then g will do the assignment
- If the result is an error, then the store s will be tagged as an error store (which prevents further execution)
**Definition of check-cmd**

- \( h_1 \) check-cmd \( h_2 = \)
  \[
    \lambda s. \begin{cases}
      \text{let } s' = (h_1, s) \text{ in} \\
      \text{cases } s' \text{ of} \\
      \quad \text{isOK}(s_i) \rightarrow (h_2, s_i) \\
      \quad \text{isErr}(s_i) \rightarrow s' \\
    \end{cases}
  \]
  \( h_1 \) will execute the first command
  \( \) If the store is OK, then the second command will be executed
  \( \) If the store is an error store, the second command is skipped

---

**Abstract syntax**

- \( P \in \text{Program} \)
- \( C \in \text{Command} \)
- \( E \in \text{Expression} \)
- \( I \in \text{Id} \)
- \( N \in \text{Numeral} \)

- \( P ::= C \)
- \( C ::= C_1;C_2 | I := E | \text{if } E \text{ then } C_1 \text{ else } C_2 | \text{diverge} \)

- Adding true and false to the expressions as constants makes them assignable values, and prompting the type-checking

---

**Semantic domains**

- Domain \( \text{Storeable-value} = Tr + Nat \)
- Domain \( \text{Expressible-value} = \text{Storeable-value} + \text{Errvalue} \)
- Domain \( \text{Store} = \text{Id} \rightarrow \text{Storeable-value} \)
- Domain \( \text{Post-store} = \text{OK} + \text{Err} \)
  where \( \text{OK} \) and \( \text{Err} = \text{Store} \)
Valuation functions: functionality

- **P**: Program $\rightarrow$ Store $\rightarrow$ Post-store
- **C**: Command $\rightarrow$ Store $\rightarrow$ Post-store
- **E**: Expression $\rightarrow$ Store $\rightarrow$ Expressible-value

The initial store will be `newstore`.

$P[C] = \lambda s.C[C]_s$

We need to check that executing $C_1$ does not result in an error store.

$C[C_1;C_1] = C[C_1] \text{ check - cmd } C[C_1]$

Assignment can only succeed if the expression has no type error.

$C[I = E] = E[E] \text{ check - result } (\lambda v.lOk(update[I/v]))$

The second function receives the value from the first function and the same store.
Valuation functions

- Only if the expression has no type error, can we choose between $C_1$ and $C_2$

$$C[\text{if } E \text{ then } C, \text{ else } C_2] = E[E]$$

\[
\text{check\text{-}result} = \lambda x.\lambda x.\text{cases } x \text{ of } t \rightarrow (t \rightarrow C[C_1] \sqcup C[C_2]) \text{ end} \\
\] 

\[
\text{isNat}(n) \rightarrow \text{inNat}(s) \\
\text{end} \\
\]

Valuation functions

- The addition can only go through if both subexpressions yield no type errors

$$E[E_1 + E_2] = E[E_1] \text{ check\text{-}expr}$$

\[
(\text{cases } r \text{ of } \\
\text{inNat}(n) \rightarrow E[E_1] \text{ check\text{-}expr} \\
\text{end}) \\
\]

Valuation functions

- The same with equality (note we cannot test two Booleans)

$$E[E = E_2] = E[E] \text{ check\text{-}expr}$$

\[
(\text{cases } r \text{ of } \\
\text{inNat}(n) \rightarrow \text{inNat}(n) \text{ equals } r) \\
\text{end}) \\
\]
Alternative not using check-expr

\[
\text{cases } \text{of } \begin{cases} 
\text{is () in ()} \\
\text{is ( ) in ()} \\
\text{let } \text{E } \text{in cases } \text{of } \\
\text{inError() inError()} \\
\text{inNat(n) inNat(n)} \\
\text{let } \text{E } \text{in cases } \text{of } \\
\text{inError() inError()} \\
\text{inNat(n) inNat(n)} \\
\end{cases}
\]

Valuation functions

* Not can only be applied to a Boolean value

\[
E[\cdot E] = E[E] \text{ check-expr}
\]

\[
(\lambda x. \lambda s. \text{cases } \text{of } \\
\text{Tr(t) inTr(\text{Not } t)} \\
\text{Nat(n) inNat()} \\
\text{end})
\]

Valuation functions

* The rest are straightforward (apart from the tags!)

\[
E[\lambda x. s] = \lambda x. \text{inStoreable - value(access[1][x])} \\
E[N] = \lambda x. \text{inStoreable - value(inNat(N[N]))} \\
E[true] = \lambda x. \text{inStoreable - value(inTr(true))} \\
E[false] = \lambda x. \text{inStoreable - value(inTr(false))}
\]
Example derivation

- The program has a type error:

```
  x := 1;
  x := true;
  if x + 1 = 0 then
    diverge
  else
    x := 0
```

Cannot add true to 1

Example continued

- Peel off the first command

```
  C[x := 1]
  = check-cmd
  \{ C[x := true; if x + 1 = 0 then diverge else x := 0] \}
```

- expand check-cmd

```
  let z = C[x := 1](newstore) in
  cases z of
  isOK() \rightarrow C[x := true; if x + 1 = 0 then diverge else x := 0](s)
  isErr() \rightarrow z
end
```

Example continued

- Working on the assignment

```
  C[x := 1](newstore)
  = (E[1].check_result (λv.λs.inOK(update[v], s)))(newstore)
```

- Which is

```
cases E[1].newstore of
  isStoreable - value(v) \rightarrow inOK(update[v], newstore)
  isErrvalue() \rightarrow inErr(newstore)
end
```
Example continued

- The new store is
  \[ \text{inOK}(\text{update}[x] \to \text{inNat}(\text{one}))\text{newstore} \]
- Which is
  \[ \text{inOK}(\text{in}[x] \to \text{inNat}(\text{one})) \to \text{newstore} \]
- Call this \( s_1 \)

Example continued

- The store is OK, so we can execute the next command
  \[
  \begin{align*}
  \text{let } s' = \text{in}[x = \text{true}](s) \text{ in} \\
  \text{cases } s' \text{ of} \\
  \quad \text{inOK}(s) \to \text{in}[x + 1 = 0 \text{ then } \text{diverge} \text{ else } x = 0](s) \\
  \quad \text{inErr}(s) \to s'
  \end{align*}
  \]

Example continued

- The same things happens for \( x := \text{true} \) to give the store \( s_2 \)
  \[ \text{inOK}(\text{in}[x] \to \text{inTr}(\text{true}))(\text{in}[x] \to \text{inNat}(\text{one}))\text{newstore}) \]
- Which reduces by extensionality of functions to
  \[ \text{inOK}(\text{in}[x] \to \text{inTr}(\text{true})) \to \text{newstore} \]
**Example continued**

- Now the conditional

\[
C[x + 1 = 0 \text{ then } \text{diverge} \text{ else } x = 0](s_2)
\]

\[
= (E[x + 1] = 0) \text{ check - result } (\lambda \lambda \lambda \text{cases } \alpha) \text{ of }
\]

\[
is\text{Tr}(t) \rightarrow (t \rightarrow C[x + 1 = 0])x
\]

\[
is\text{Nat}(n) \rightarrow \text{Nat}(s) \text{ end}) (s_2)
\]

- We need to evaluate the test expression in store \(s_2\)

---

**Example continued**

- This is:

\[
E[x + 1 - 0], s_2
\]

\[
(\text{check - expr})
\]

\[
(\text{is cases } \nu) \text{ of }
\]

\[
is\text{Tr}(t) \rightarrow \text{is lnNat}\nu(t)
\]

\[
is\text{Nat}(n) \rightarrow E(0) \text{ check - expr}
\]

\[
is\nu \lambda \text{cases } \lambda^\prime \text{ of }
\]

\[
is\text{Tr}(t') \rightarrow \text{lnNat}\nu(t')
\]

\[
is\text{Nat}(n') \rightarrow \text{lnStore}\nu - \text{NatNat}(\text{lnNat}(\text{lnNat}(n) + n'))
\]

- \text{end}(E(x))

---

**Example continued**

- This relies on evaluation of \(x + 1\):

\[
E[x + 1], s_2
\]

\[
(\text{check - expr})
\]

\[
(\text{is cases } \nu) \text{ of }
\]

\[
is\text{Tr}(t) \rightarrow \lambda \text{lnNat}\nu(t)
\]

\[
is\text{Nat}(n) \rightarrow E(1) \text{ check - expr}
\]

\[
is\nu \lambda \text{cases } \lambda^\prime \text{ of }
\]

\[
is\text{Tr}(t') \rightarrow \text{lnNat}\nu(t')
\]

\[
is\text{Nat}(n') \rightarrow \text{lnStore}\nu - \text{NatNat}(\text{lnNat}(\text{lnNat}(n) + n'))
\]

- \text{end}(E(x))
Example continued

- Which expands to

```
cases E[x]s, of
  inStorable-value(r) ->
    cases r of
      inTr(r) -> inErr(value)
      inNat(r) -> E[r] check-over
    end
  end
```

Example continued

- The evaluation of x gives

```
inStorable-value(access[x]s)
```

- Which is

```
inStorable-value(inTr(true))
```

- So the removal of the Storable-value tag reveals the Tr tag, which gives an Errorvalue

Example concluded

- The Errorvalue propagates up to the check-result in the valuation function for the conditional

- This returns an error store inErr(s2), which is the denotation for the whole program

- If there were extra commands after the conditional, they would not be executed