Semantics for Heaps

Marco Vassena
Philipp Hausmann
Ondrej Pelech

Research questions

- semantics for a minimal language
  - mutable heap
  - (exceptions)
- prove meta-theories

Simple Typed Lambda Calculus

- Very verbose types
- Hard to guarantee totality
- Open research question

Simple Heap Language

```
ref       : ℕ                                -> Term (Ref ty)
new       : Term ty                          -> Term (Ref ty)
!_        : Term (Ref ty)                    -> Term ty
_<-_      : Term (Ref ty) -> Term ty          -> Term ty
try_catch_ : Term ty      -> Term ty          -> Term ty
```

Semantics

- Heap as a list of values, including *error*

$$\textsc{New}\ \frac{t \Downarrow v}{(\textbf{new}\ t) \Downarrow (\mathit{vref}\ m)\ \{\underline{\mathtt{append}}\ H\ v\}}$$

$$\textsc{Ass}\ \frac{\underline{r \in H} \quad t_1 \Downarrow (\mathit{vref}\ r) \quad t_2 \Downarrow v}{(t_1\ \textbf{<-}\ t_2) \Downarrow v\ \{\underline{\mathtt{replace}}\ H\ v\}}$$

$$\textsc{Deref}\ \frac{t \Downarrow (\mathit{vref}\ r)}{(\textbf{!}\ t) \Downarrow (\underline{\mathtt{lookup}}\ r\ H)}$$

$$\textsc{AssOob}\ \frac{\underline{r \notin H} \quad t_1 \Downarrow (\mathit{vref}\ r)}{(t_1\ \textbf{<-}\ t_2) \Downarrow \mathit{verror}}$$

Semantics

- Handling errors

$$\textsc{TryCat}\ \frac{\neg \mathit{verror}\ v \qquad t_1 \Downarrow v}{(\textbf{try}\ t_1\ \textbf{catch}\ t_2) \Downarrow v}$$

$$\textsc{DerefErr}\ \frac{t \Downarrow \mathit{verror}}{(\textbf{!}\ t) \Downarrow \mathit{verror}}$$

$$\textsc{TryCatErr}\ \frac{t_1 \Downarrow \mathit{verror} \qquad t_2 \Downarrow v}{(\textbf{try}\ t_1\ \textbf{catch}\ t_2) \Downarrow v}$$

$$\textsc{AssErr}\ \frac{t_1 \Downarrow \mathit{verror}}{(t_1\ \textbf{<-}\ t_2) \Downarrow \mathit{verror}}$$

Research contribution

- Formal proofs
  - progress
  - preservation
  - soundness
  - completeness
  - determinism
  - type preservation in Heap

Research contribution

- Hoare logic
  - Hoare triples
  - Partial and total interpretation
  - reasoning about the correctness
  - proofs of the principal hoare logic theorems

Further work

- Extend hoare logic
  - statement vs expression
  - evaluation changes the heap
- Complete missing rules
- Make the theorems for hoare logic usable

References

- Types and Programming Languages, Benjamin C. Pierce
- Software Foundations and Programming Languages, Benjamin C. Pierce

Thank you
for your attention

Questions?

Research contribution

- Design and implementation in Agda
  - types
  - terms
  - values
  - heap

The Types, Terms and Values

```
data Type : Set where
  ...
  Boolean :                  Type
  Ref     : (ty : Type) -> Type
```

The Types, Terms and Values

```
data Term : Type -> Set where
  ...
  true          : Term Boolean
  false         : Term Boolean
  error         : ∀ {ty} -> Term ty
  if_then_else_ : ∀ {ty} -> (cond  : Term Boolean)
                         -> (tcase : Term ty)
                         -> (fcase : Term ty)
                         -> Term ty
  new           : ∀ {ty}      -> Term ty                    -> Term (Ref ty)
  !_            : ∀ {ty}      -> Term (Ref ty)              -> Term ty
  _<-_          : ∀ {ty}      -> Term (Ref ty) -> Term ty   -> Term ty
  ref           : ∀ {ty}      -> ℕ                          -> Term (Ref ty)
  try_catch_    : ∀ {ty}      -> Term ty       -> Term ty   -> Term ty
```

The Types, Terms and Values

```
data Value : Type -> Set where
  ...
  vtrue vfalse :                  Value Boolean
  vref         : ∀ {ty} -> ℕ -> Value (Ref ty)
  verror       : ∀ {ty}      -> Value ty
```

Heap semantics

```
data Heap : ℕ -> Set where…
```

- list of `Values`, including `verror`

Heap semantics

```
E-New    : BStep t v          → BStep {H2 = append H2 v} (new t) (vref m)

E-Deref  : BStep t (vref r) → BStep                    (! t)   (lookup r
H2)

E-Ass    : (rep : Elem H3 r ty) →
           BStep t1 (vref r)    →
           BStep t2 v           →
           BStep {H2 = replace H3 rep v } (t1 <- t2) v

E-TryCat : ¬ (isVError v)            →
           BStep t1                v →
           BStep (try t1 catch t2) v
```

Exception

- raised when ref out of bounds
- `error` from `Term`
- `verror` from `Value` may be stored in Heap
- `try_catch_` from `Term`