The Qualification Principle

Programming Languages CS442

David Toman

School of Computer Science
University of Waterloo

The Qualification Principle

Semantically meaningful phrases allow local definitions.

Syntax:

$$U ::= \ldots \mid \textbf{begin}\, D\, \textbf{in}\, U\, \textbf{end}$$

Type rule:

$$\frac{\pi_1 \vdash D : \pi_2 dec \qquad \pi_1 \uplus \pi_2 \vdash U : \theta}{\pi_1 \vdash \textbf{begin}\, D\, \textbf{in}\, U\, \textbf{end} : \theta}$$

$\Rightarrow$ supersedes $P ::= D\, \textbf{in}\, C$ syntax

Command Block

- Typing rule

$$\frac{\pi_1 \vdash D : \pi_2 dec \qquad \pi_1 \uplus \pi_2 \vdash C : comm}{\pi_1 \vdash \textbf{begin}\, D\, \textbf{in}\, C\, \textbf{end} : comm}$$

- Semantics

$$[\![\pi \vdash \textbf{begin}\, D\, \textbf{in}\, C\, \textbf{end} : comm]\!] e\ s = free(s'')$$
$$where\ (e', s') = [\![\pi \vdash D : \pi' dec]\!] e\ s$$
$$and\ s'' = [\![\pi \uplus \pi' \vdash C : comm]\!] e \uplus e'\ s'$$

. . . is the use of the "free" safe???

Scope

Idea

Scope is determined by eager/lazy resolution of identifiers.

. . . i.e., when do you get to look at the environment!

Static (at definition time; works w/type rules)

$$\begin{aligned} [\![\textbf{define}\, I = U]\!]e\ s &= \{I = [\![U]\!]e\} \\ [\![\textbf{invoke}\, I]\!]e\ s &= f\ s \ \textit{for}\ \ (I, f) \in e \end{aligned}$$

Dynamic (at reference/run time)

$$\begin{aligned} [\![\textbf{define}\, I = U]\!]e\ s &= \{I = [\![U]\!]\} \\ [\![\textbf{invoke}\, I]\!]e\ s &= f\ e\ s \ \textit{for}\ \ (I, f) \in e \end{aligned}$$

Note: eager/lazy is w.r.t. *environments* not the *store*!

Extent

Idea

Can local variables (i.e. locations!) escape their scope?

- **Command Blocks:**

  $\Rightarrow$ no problem (why?)

- **Declaration Blocks:**

$$\begin{array}{l}\textbf{module}\, N = \textbf{begin}\, \textbf{var}\, A : \textbf{newint} \\ \quad \textbf{in}\{\textbf{proc}\, \mathit{init} = A := 0; \\ \qquad \textbf{proc}\, \mathit{succ} = A := @A + 1; \\ \qquad \textbf{fun}\, \mathit{val} = @A\} \\ \quad \textbf{end}\end{array}$$

!!! problem $\Rightarrow$ deallocation of variables $\neq$ block structure

Type Structure Blocks

Idea

Local declarations = private data of an object.

- **class** C = **begin var** A : **newint**
  in record
  proc *init* = $A := 0$; **proc** *succ* = $A :=$ @$A + 1$;
  fun *val* = @A
  end end
- different from *modules*!
- can be extended:

  class $C2$ = **begin var** N : C
  in record
  proc *init* = N.*init*; . . . ;
  proc *succ*2 = N.*succ*; N.*succ*
  end end

Inheritance

Idea

*Let's avoid the annoying "**proc** init = **call** N.init".*

$\Rightarrow$ *inheritance: allows (parts of) parent class to be visible.*

- **class** $C2 =$ **inherits** C **with**
  begin ... **in record**
  proc $succ2 = succ; succ$
  end end
- Which parts of C are visible to $C2$?
- What happens if we redefine an identifier?
- Can we inherit from several classes?

Dynamic Scope and Virtual Methods

Idea

Can we re-define identifiers used in parent's definitions?

$\Rightarrow$ *virtual methods*

- **class** $C3 =$ **inherits** $C2$ **with**
  record
  proc $succ = \ldots$
  end end
- In **var** $x : C3$: which $succ$ does $x.succ2$ use?
- Not statically-scoped! Solutions:
  1. a syntactic (lazy) "copy rule"?
  2. a "self" parameter?
- Naive redefinition (e.g., change of type) $\sim$ runtime error.

Subclasses and Subtypes

Inheritance seems to induce “subclass” relationship.

Idea

(elements of) subclasses in place of (elements of) super classes.

with the intuition that $\theta_1 \leq \theta_2$ if θ_1 can be used in place of θ_2.

- subsets or coercion rules and coherence?

  $\Rightarrow$ semantics of coercion $[\![\theta \leq \theta']\!]$?
- complex type tags?

  records: $\{l_j : \theta_j\}_{j=1}^{m+n} \leq \{l_j : \theta'_j\}_{j=1}^{m}$ if $n \geq 0$ and $\theta_j \leq \theta'_j$

  functions: $\theta_1 \to \theta'_1 \leq \theta_2 \to \theta'_2$ if $\theta_2 \leq \theta_1$ and $\theta'_1 \leq \theta'_2$

Summary

Idea

Qualification allows Abstractions to be local.

- allows encapsulation ala OO languages

  $\Rightarrow$ but virtual methods mess up (static) typing.
- (qualified) abstraction and parametrization reduce to substitution (for lazy abstractions/parameters).
- subclasses and subtypes *aren't* the same.
- Questions:
  1. how does block structure impact *continuations*?
  2. how does block structure impact *recursive abstractions*?