Colloquium *Logic in Computer Science*, Munich, November 20, 1998

Designing a Java Formalization in HOL

David von Oheimb[1], TU München www.in.tum.de/~oheimb/

---

[1]supported by DFG Project *Bali*

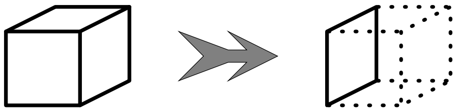
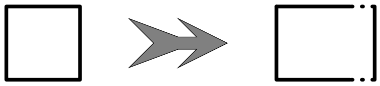
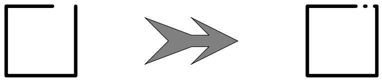
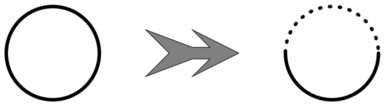
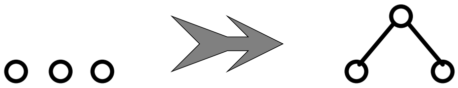
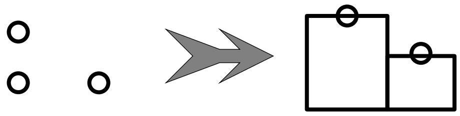
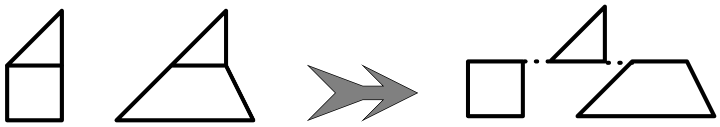
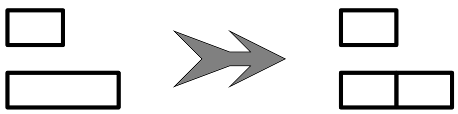
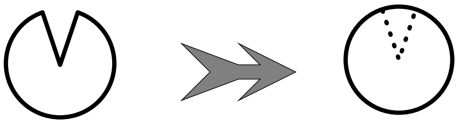
Design Goals

- faithfulness to the official specification
- ease of validation
- readability and simplicity
- minimal redundancy
- maintainability and extendibility
- adequacy for theorem proving

Basic Decisions: use ...

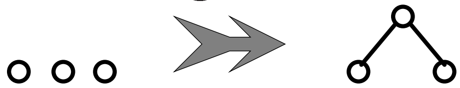
- Isabelle/HOL (expressive, typed, flexible)
- deep embedding (meta properties!)
- (operational) evaluation semantics

Techniques

- shaping
  - restrictions 
  - gentle extensions 
  - filling gaps 
  - underspecification 
- structuring
  - hierarchies 
  - hiding 
- abstraction
  - factoring out 
  - reduction 
  - unification 

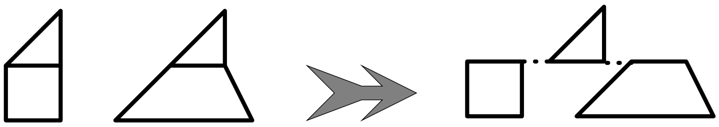
Declarations

"A program consists of classes and interfaces"



types $prog = idecl\ list \times cdecl\ list$

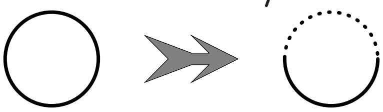
"A declaration maps names to contents"



types $(\alpha, \beta)\ table = \alpha \Rightarrow \beta\ option$

consts table_of :: $(\alpha \times \beta)\ list \Rightarrow (\alpha, \beta)\ table$

"A name / an identifier is ..."



types $tnam$

datatype $tname =$ `Object` | TName $tnam$ | ...

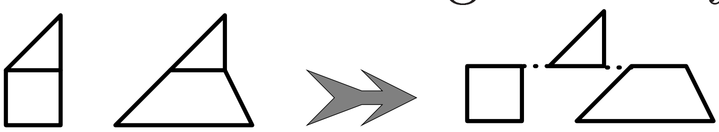
Well-formedness

"A compiler-time error occurs if ..."

consts wf_prog :: $prog \Rightarrow bool$

defs $\quad$ wf_prog $\Gamma \stackrel{\text{def}}{=} \forall i{\in}\mathsf{set}(\mathsf{fst}\ \Gamma).\ \mathsf{wf\_idecl}\ \Gamma\ i \wedge$
$\mathsf{unique}\ (\mathsf{fst}\ \Gamma) \wedge \ldots$

"The method argument types must exist, ..."

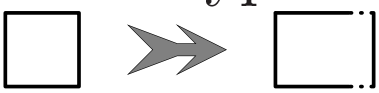


defs wf_mhead $\Gamma\ C\ m \stackrel{\text{def}}{=} \ldots$

wf_idecl $\Gamma\ i \stackrel{\text{def}}{=} \ldots\ \mathsf{wf\_mhead}\ \Gamma\ s\ mh \wedge \ldots$

wf_cdecl $\Gamma\ c \stackrel{\text{def}}{=} \ldots\ \mathsf{wf\_mhead}\ \ldots$

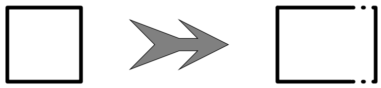
"An overriding method must have the same return type as the overridden one"



wf_cdecl $\Gamma\ c \stackrel{\text{def}}{=} \ldots\ \Gamma \vdash \mathsf{mrt}\ m' \preceq \mathsf{mrt}\ m \ldots$

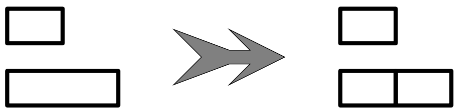
Terms

"certain expressions can be used as statements"



datatype $\mathit{stmt} = \ldots \mid \mathsf{Expr}\ \mathit{expr} \mid \ldots$

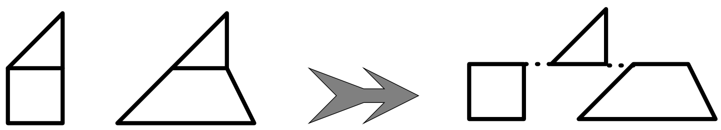
"several variants of conditional statements"



datatype $\mathit{stmt} = \texttt{if}\ (\mathit{expr})\ \mathit{stmt}\ \texttt{else}\ \mathit{stmt} \mid \ldots$

mixfix constructors!

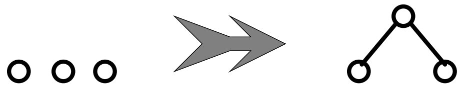
"A method invocation using `super` …"



datatype $\mathit{expr} = \ldots \mid \mathit{expr}\,\texttt{.}\,\mathit{mname}(\ldots)$

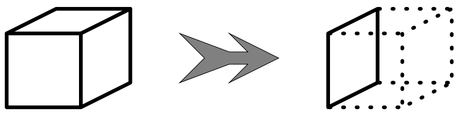
Types

"There are two kinds of types . . ."



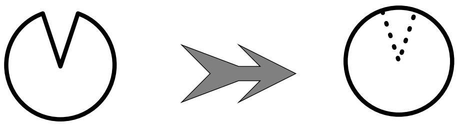
datatype $ty = \mathsf{PrimT}\ prim\_ty \mid \mathsf{RefT}\ ref\_ty$

"The primitive types are . . ."



datatype $prim\_ty = \texttt{boolean} \mid \texttt{int}$

"A method has a return type or it is `void`"



types $mhead = modi \times ename\ list \times ty$
datatype $prim\_ty = \texttt{void} \mid \ldots$
datatype $val = \mathsf{Unit} \mid \ldots$

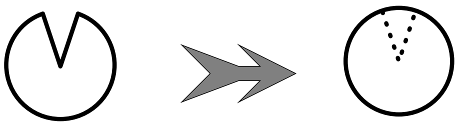
Well-typedness

"A while loop is well-typed if ..."

inductive wt_stmt

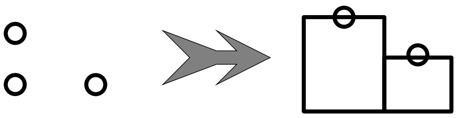
$$\frac{E \vdash e{::}\mathsf{PrimT}\ \texttt{boolean} \qquad E \vdash c{::}\diamond}{E \vdash \texttt{while}(e)\ \ c{::}\diamond}$$

"There are no literals for references"



$$\frac{\mathsf{typeof}\ (\lambda a.\ \mathsf{None})\ x = \mathsf{Some}\ \ T}{E \vdash \mathsf{Lit}\ \ x{::}\,T}$$

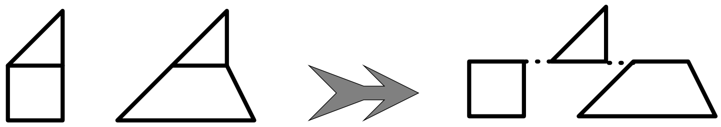
"The most specific method is selected"



$$\begin{aligned}\mathsf{max\_spec}\ \Gamma\ \ T\ sig \overset{\mathsf{def}}{=} \{m\ \in & \mathsf{appl\_methds}\ \Gamma\ \ T\ sig.\\ & (\forall m' {\in} \mathsf{appl\_methds}\ \Gamma\ \ T\ sig.\\ \mathsf{more\_spec}\ \Gamma\ \ m'\ \ m & \longrightarrow m' = m)\}\end{aligned}$$

State

"Both class instances and arrays are objects"

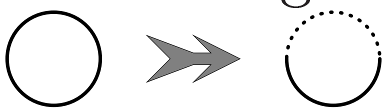


types $\quad oref = loc + tname$

$\quad\quad obj \;=\; obj\_tag \times (vn,\ val)\ table$

$\quad\quad st \;=\; (oref,\ obj)\ table \times locals$

"Allocating an object may fail"



$$\mathsf{new\_Addr} \;::\; state \Rightarrow (loc \times xcpt\ option)\ option$$

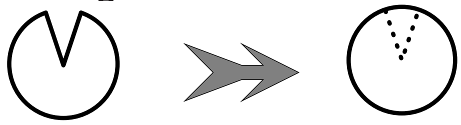
$$\mathsf{new\_Addr}\ \sigma \overset{\mathsf{def}}{=} \varepsilon y.\ y = \mathsf{None} \wedge \text{'}no\ free\ location\text{'} \\ \vee\ \ldots$$

Exceptions not as terms, but as part of the state:

$state = xcpt\ option \times st$

Semantics

"Expression evaluation may throw exceptions"



$prog \vdash state \; -expr \triangleright val \rightarrow \; state$

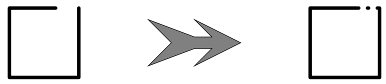
"Exceptions are propagated unless caught"

$$\frac{}{\Gamma \vdash (\mathsf{Some}\ xc{,}\sigma) \; -c\rightarrow \; (\mathsf{Some}\ xc{,}\sigma)}$$

$$\frac{}{\Gamma \vdash \mathsf{Norm}\ \sigma \; -\mathsf{Skip}\rightarrow \; \mathsf{Norm}\ \sigma}$$

$$\frac{\Gamma \vdash \mathsf{Norm}\ \sigma_0 \; -c_1 \rightarrow \; \sigma_1 \quad \Gamma \vdash \sigma_1 \; -c_2\rightarrow \; \sigma_2}{\Gamma \vdash \mathsf{Norm}\ \sigma_0 \; -c_1;\; c_2\rightarrow \; \sigma_2}$$

"A `throw` statement throws an exception"



$$\frac{\Gamma \vdash \mathsf{Norm}\ \sigma_0 \; -e \triangleright a' \rightarrow \; (x_1{,}\sigma_1) \quad x_1{}'{=}\ldots \; \mathsf{np}\ a' \; \ldots}{\Gamma \vdash \mathsf{Norm}\ \sigma_0 \; -\mathtt{throw}\ e\rightarrow \; (x_1{}'{,}\sigma_1)}$$

Experience

- Isabelle/HOL is good formalization tool
- rapid prototyping facility would be helpful
- proofs give feedback for the formalization
- iterated improvements pay

Achievements

- precise language specification
- proof of language properties
- a solid basis for ...
  - completing the formalization
  - modelling language extentions
  - compiler verification
  - program verification