Hoare Logic for NanoJava

David von Oheimb & Tobias Nipkow

Technische Universität München

What is the problem

A sound and complete Hoare logic for Java-like languages

Many dimensions:

- Exceptions
- Side effects in expressions
- Recursive procedures
- Object-orientation
- Concurrency
- ...

First attempts (w/o threads): [DvO 2001], [N 2002]

Our focus

- Side effects in expressions
- Auxiliary variables
- Method calls

Our focus

- Side effects in expressions
- Auxiliary variables
- Method calls

Closely related work:

- Kowaltowski (side effects)
- Kleymann (recursive procedures)
- Poetzsch-Heffter (object orientation)

Machine checked!

Everything formalized & proved in Isabelle/HOL

Machine checked!

Everything formalized & proved in Isabelle/HOL

Why?

Machine checked!

Everything formalized & proved in Isabelle/HOL

Why?

Because Hoare logics are notoriously tricky to get right.

History: recursive procedures

Hoare	71	PC	sound & complete [Olderog 83]
Gorelick	75	PC	sound & complete
Sokołowski	77	TC	incomplete
EUCLID	78	PC	unsound
Gries & Levin	80	PC	incomplete
Apt	81	TC	unsound
America & de Boer	90	TC	sound & complete

History: machine checked proofs

Gordon	89	HOL	sound	(no procs)
Nipkow	96	Isabelle	sound & complete	(no procs)
Kleymann	97–99	LEGO	sound & complete	(procs)

What appears to be the problem with procedures

```
proc  =  if i=0 then skip
         else i := i-1; CALL; i := i+1
```

What appears to be the problem with procedures

```
proc  =  if i=0 then skip
         else i := i-1; CALL; i := i+1
```

$$\frac{\dfrac{\overline{\qquad\qquad}}{\qquad}}{\{i=N\}\ \mathtt{CALL}\ \{i=N\}}$$

What appears to be the problem with procedures

```
proc  =  if i=0 then skip
         else i := i-1; CALL; i := i+1
```

$$\frac{\{i=N\}\,\mathtt{CALL}\,\{i=N\} \vdash \{i=N\}\ \mathtt{if}\ i=0\ \ldots\ \{i=N\}}{\{i=N\}\ \mathtt{CALL}\ \{i=N\}}$$

What appears to be the problem with procedures

```
proc  =  if i=0 then skip
         else i := i-1; CALL; i := i+1
```

$$\dfrac{\dfrac{}{\{i{=}N\}\mathtt{CALL}\{i{=}N\} \vdash \{i{=}N\}\, i{:=}i{-}1;\mathtt{CALL};i{:=}i{+}1\, \{i{=}N\}}}{\dfrac{\{i{=}N\}\mathtt{CALL}\{i{=}N\} \vdash \{i{=}N\}\ \mathtt{if}\ i{=}0\ \ldots\ \{i{=}N\}}{\{i{=}N\}\ \mathtt{CALL}\ \{i{=}N\}}}$$

What appears to be the problem with procedures

```
proc  =  if i=0 then skip
         else i := i-1; CALL; i := i+1
```

$$\cfrac{\cfrac{\cfrac{\texttt{\{i=N\}CALL\{i=N\}} \vdash \texttt{\{i=N-1\} CALL \{i=N-1\}}}{\vdots}}{\texttt{\{i=N\}CALL\{i=N\}} \vdash \texttt{\{i=N\}i:=i-1;CALL;i:=i+1\{i=N\}}}}{\cfrac{\texttt{\{i=N\}CALL\{i=N\}} \vdash \texttt{\{i=N\} if i=0 ... \{i=N\}}}{\texttt{\{i=N\} CALL \{i=N\}}}}$$

What appears to be the problem with procedures

```
proc  =  if i=0 then skip
         else i := i-1; CALL; i := i+1
```

$$\dfrac{\dfrac{\dfrac{\{\mathtt{i{=}N}\}\mathtt{CALL}\{\mathtt{i{=}N}\} \vdash \{\mathtt{i{=}N{-}1}\}\ \mathtt{CALL}\ \{\mathtt{i{=}N{-}1}\}}{\vdots}}{\{\mathtt{i{=}N}\}\mathtt{CALL}\{\mathtt{i{=}N}\} \vdash \{\mathtt{i{=}N}\}\mathtt{i{:=}i{-}1;CALL;i{:=}i{+}1}\{\mathtt{i{=}N}\}}}{\dfrac{\{\mathtt{i{=}N}\}\mathtt{CALL}\{\mathtt{i{=}N}\} \vdash \{\mathtt{i{=}N}\}\ \mathtt{if\ i{=}0\ ...}\ \ \{\mathtt{i{=}N}\}}{\{\mathtt{i{=}N}\}\ \mathtt{CALL}\ \{\mathtt{i{=}N}\}}}$$

Need to adapt the values of auxiliary variables!

Solutions

- A suite of adaption rules (Apt, America & de Boer, ...)

Solutions

- A suite of adaption rules (Apt, America & de Boer, . . .)
- A generalized consequence rule (Kleymann, . . .)

Solutions

- A suite of adaption rules (Apt, America & de Boer, . . .)
- A generalized consequence rule (Kleymann, . . .)
- Use VDM — does not need auxiliary variables!

Structure of talk

- Motivation ($\surd$)
- NanoJava
- Hoare logic
- Semantics, soundness and completeness
- Conclusions

NanoJava: Commands and expressions

c	::=	`skip`	
	|	c;c	
	|	`if` e `then` c `else` c	
	|	`while` e `do` c	
	|	x = e	parameter/local variable
	|	e.f = e	field

NanoJava: Commands and expressions

c	::=	`skip`	
	|	c;c	
	|	`if` e `then` c `else` c	
	|	`while` e `do` c	
	|	x = e	parameter/local variable
	|	e.f = e	field
e	::=	`new` C	
	|	(C)e	cast
	|	x	
	|	e.f	
	|	e_C.m(e)	method call

Turing completeness

- No types except references. Not even booleans.

Turing completeness

- No types except references. Not even booleans.
- `null` simulates `false`, all other refs `true` (Lisp).

Turing completeness

- No types except references. Not even booleans.
- `null` simulates `false`, all other refs `true` (Lisp).
- All vars are initialized to `null`.

Turing completeness

- No types except references. Not even booleans.
- `null` simulates `false`, all other refs `true` (Lisp).
- All vars are initialized to `null`.
- Simulating $n \in I\!N$: linked list of n objects.

Structure of talk

- ...
- Hoare logic
- ...

Assertions: The extensional approach

Assertions are predicates on the program state:

$$\mathit{assertion} = \mathit{state} \longrightarrow \mathit{bool}$$

Assertions: The extensional approach

Assertions are predicates on the program state:

$$\textit{assertion} = \textit{state} \rightarrow \textit{bool}$$

Not syntax, semantics!

Not `{x != Y}` but $\{\lambda s.\ lookup(s, x) \neq Y\}$

Format of judgements

set of Hoare triples $\vdash$ *set of Hoare triples*

Format of judgements

set of Hoare triples $\vdash$ *set of Hoare triples*

- Why assumptions: to deal with recursion
- Why sets of triples: mutual recursion

Format of judgements

set of Hoare triples $\vdash$ *set of Hoare triples*

- Why assumptions: to deal with recursion
- Why sets of triples: mutual recursion

Conversion between sets and singletons:

$$\frac{\forall \{P\}s\{Q\} \in C.\ A \vdash \{P\}s\{Q\}}{A \vdash C} \qquad \frac{A \vdash C \quad \{P\}s\{Q\} \in C}{A \vdash \{P\}s\{Q\}}$$

How to deal with...

- Side effects in functions/methods
- Auxiliary variables
- Method calls

Side effects

Idea: program transformation

$$f(g(y)) \quad \rightsquigarrow \quad v := g(y); f(v)$$

Side effects

Idea: program transformation

$$f(g(y)) \quad \leadsto \quad v := g(y); f(v)$$

Better: integration into rule format:

$$\textit{tripels} \vdash \textit{e-triple}$$

$$\begin{array}{rcl} \textit{e-triple} & = & \{\textit{assertion}\}e\{\textit{v-assertion}\} \\ \textit{v-assertion} & = & \textit{value} \rightarrow \textit{assertion} \end{array}$$

Side effects

Idea: program transformation

$$f(g(y)) \quad \leadsto \quad v := g(y); f(v)$$

Better: integration into rule format:

$$\textit{tripels} \vdash \textit{e-triple}$$

$$\begin{aligned} \textit{e-triple} \quad &= \quad \{\textit{assertion}\}e\{\textit{v-assertion}\} \\ \textit{v-assertion} \quad &= \quad \textit{value} \rightarrow \textit{assertion} \end{aligned}$$

Intuition: extend state space with new result variable

Auxiliary variables

Adaption of auxiliary variables via generalized consequence rule:

$$\frac{\forall z.\ A \vdash \{P'\,z\}c\{Q'\,z\}}{A \vdash \{P\}c\{Q\}} \quad \forall s\ t.(\forall z.\ P'\,z\,s \longrightarrow Q'\,z\,t) \longrightarrow (P\,s \longrightarrow Q\,t)$$

Method calls: virtual methods

$\texttt{meth}(C, m)$: **virtual method** (Poetzsch-Heffter)
Parameter passing:

$$\frac{\vdash \{P\}e\{Q\} \quad \vdash \{Q\}e'\{R\} \quad \vdash \{R'\}\texttt{meth}(C, m)\{S\}}{\vdash \{P\}e_C.m(e')\{S\}}$$

Verifying all implementations $\texttt{impl}(C, m)$:

$$\frac{\forall D < C.\ \{P\}\texttt{impl}(D, m)\{Q\}}{\{P\}\texttt{meth}(C, m)\{Q\}}$$

Method calls: recursion

The naive way:

$$\frac{\{P_1\}\mathtt{impl}(D_1, m_1)\{Q_1\}, \cdots \vdash \{P_1\}body(D_1, m_1)\{Q_1\}, \ldots}{\vdash \{P_1\}\mathtt{impl}(D_1, m_1)\{Q_1\}, \ldots}$$

Method calls: recursion

The naive way:

$$\frac{\{P_1\}\texttt{impl}(D_1, m_1)\{Q_1\}, \dots \vdash \{P_1\}body(D_1, m_1)\{Q_1\}, \dots}{\vdash \{P_1\}\texttt{impl}(D_1, m_1)\{Q_1\}, \dots}$$

With auxiliary variables (z):

$$\frac{\bigcup_{z}\{\{P_1\, z\}\texttt{impl}(D_1, m_1)\{Q_1\, z\}, \dots\} \vdash \{P_1\, z\}body(D_1, m_1)\{Q_1\, z\}, \dots}{\vdash \{P_1\, z\}\texttt{impl}(D_1, m_1)\{Q_1\, z\}, \dots}$$

Trick: use $\bigcup_{z}$ instead of $\forall z$.

Soundness & Completeness

- Operational semantics: $s \;{-c\!\!\rightarrow}\; t$ (and $s \;{-c\triangleright v\!\!\rightarrow}\; t$)

Soundness & Completeness

- Operational semantics: $s\ {-c\rightarrow}\ t$ (and $s\ {-c\triangleright v\rightarrow}\ t$)
- Validity: $\models \{P\}c\{Q\} \ \equiv\ \forall s, t.\ s\ {-c\rightarrow}\ t \longrightarrow P\,s \longrightarrow Q\,t$

Soundness & Completeness

- Operational semantics: $s \ -c\!\rightarrow\ t$ (and $s \ -c\triangleright v\!\rightarrow\ t$)
- Validity: $\models \{P\}c\{Q\} \ \equiv\ \forall s,t.\ s\ -c\!\rightarrow\ t \longrightarrow P\,s \longrightarrow Q\,t$
- Soundness: $\vdash \{P\}c\{Q\} \quad\longrightarrow\quad \models \{P\}c\{Q\}$
  easy

Soundness & Completeness

- Operational semantics: $s \; -c\rightarrow \; t$ (and $s \; -c \triangleright v \rightarrow \; t$)
- Validity: $\models \{P\}c\{Q\} \;\equiv\; \forall s, t.\; s \; -c\rightarrow \; t \longrightarrow P\, s \longrightarrow Q\, t$
- Soundness: $\vdash \{P\}c\{Q\} \quad\longrightarrow\quad \models \{P\}c\{Q\}$
  easy
- Completeness: $\models \{P\}c\{Q\} \quad\longrightarrow\quad \vdash \{P\}c\{Q\}$
  follows standard proof

Review

- NanoJava:
  core language for studying object-orientation

Review

- NanoJava:
  core language for studying object-orientation
- Compact Hoare logic with various slick features

Review

- NanoJava:
  core language for studying object-orientation
- Compact Hoare logic with various slick features
- Machine checked proofs very feasible
  (completeness: 100 lines)

Review

- NanoJava:
  core language for studying object-orientation
- Compact Hoare logic with various slick features
- Machine checked proofs very feasible
  (completeness: 100 lines)
- Machine checked proofs ideal for incremental developments

Conclusion

No programming language should come
w/o a machine checked formalization!