Encoding Hoare Logic in Typed Certified Code

Nikolaos S. Papaspyrou
Michalis A. Papakyriakou   Angelos Manousaridis

National Technical University of Athens
School of Electrical and Computer Engineering
Software Engineering Laboratory

{nickie, mpapakyr, amanous}@softlab.ntua.gr

5th Panhellenic Logic Symposium
Athens, July 25-28, 2005

Outline

- Motivation
  - Hoare logic
  - Typed certified code
  - Can we combine the two?
- Our approach
  - The type language
  - The computation language
  - Encoding Hoare logic
  - Problems with Hoare Logic
  - And their Solution
  - Example
- Conclusions

Hoare Logic (i)

- Introduced the strength of formal logic in computer programming
- A tool to:
  - reason about program properties and prove correctness
  - derive programs from their specifications

C. A. R. Hoare, "An axiomatic basis for computer programming", *Communications of the ACM*, vol. 12, no. 10, pp. 576–585, 1969.

Hoare Logic *(ii)*

- Hoare triples represent program specifications

  $\{P\}$ `program` $\{Q\}$

Hoare Logic *(ii)*

- Hoare triples represent program specifications

  $\{P\}$ `program` $\{Q\}$

- Example: greatest common divisor

  $\{n+m>0\}$
  $a := n;\ b := m;$
  $\texttt{while}\ a>0\ \texttt{and}\ b>0\ \texttt{do}$
  $\quad \texttt{if}\ a>b\ \texttt{then}\ a := a\ \texttt{mod}\ b$
  $\qquad\qquad \texttt{else}\ b := b\ \texttt{mod}\ a;$
  $r := a+b$
  $\{r>0\ \wedge\ r\backslash n\ \wedge\ r\backslash m\ \wedge\ (\forall r' \in \mathbb{N}.\ r'\backslash n\ \wedge\ r'\backslash m \Rightarrow r' \leq r)\}$

Typed Certified Code (i)

Methodology:

- a sound formal logic is used
- combined with the programming language
- program specifications are expressed as propositions in this logic
- proofs of these propositions are embedded in programs
  - either explicitly given by the programmer
  - or automatically constructed by the compiler

Typed Certified Code (ii)

Proposed solutions:

- Typed Intermediate Language (TIL); Harper and Morrisett, 1995
- Typed Assembly Language (TAL); Morrisett, Walker, Crary and Glew, 1998
- Proof-Carrying Code (PCC); Necula, 1998
- Foundational Proof-Carrying Code, Appel, 2001
- Shao, Saha, Trifonov and Papaspyrou, 2002, 2005
- Crary and Vanderwaart, 2002

Typed Certified Code *(iii)*

- Example: greatest common divisor
  - `gcd : nat` $\twoheadrightarrow$ `nat` $\twoheadrightarrow$ `nat`

    "A function taking two naturals and returning some natural."

Typed Certified Code *(iii)*

- Example: greatest common divisor
  - $\texttt{gcd} \;:\; \texttt{nat} \twoheadrightarrow \texttt{nat} \twoheadrightarrow \texttt{nat}$

    "A function taking two naturals and returning some natural."

  - $\texttt{gcd} \;:\; \forall n : \mathsf{Nat}. \forall m : \mathsf{Nat}. \forall p^{*} : (n + m > 0).$
    $\texttt{snat}\, n \twoheadrightarrow \texttt{snat}\, m \twoheadrightarrow \texttt{nat}$

    "One of the arguments shall not be zero."

Typed Certified Code (iii)

- Example: greatest common divisor
  - $\texttt{gcd} \; : \; \texttt{nat} \twoheadrightarrow \texttt{nat} \twoheadrightarrow \texttt{nat}$

    "A function taking two naturals and returning some natural."
  - $\texttt{gcd} \; : \; \forall n : \mathsf{Nat}. \forall m : \mathsf{Nat}. \forall p^* : (n + m > 0).$
    $\texttt{snat}\, n \twoheadrightarrow \texttt{snat}\, m \twoheadrightarrow \texttt{nat}$

    "One of the arguments shall not be zero."
- Singleton type $\texttt{snat}\, n$
  A data type whose elements are representations of the single integer value $n : \mathsf{Nat}$

Typed Certified Code (iii)

- Example: greatest common divisor
  - $\mathtt{gcd} \ : \ \mathtt{nat} \twoheadrightarrow \mathtt{nat} \twoheadrightarrow \mathtt{nat}$

    "A function taking two naturals and returning some natural."
  - $\mathtt{gcd} \ : \ \forall n : \mathsf{Nat}. \forall m : \mathsf{Nat}. \forall p^* : (n + m > 0).$
    $\mathtt{snat}\, n \twoheadrightarrow \mathtt{snat}\, m \twoheadrightarrow \mathtt{nat}$

    "One of the arguments shall not be zero."
- Singleton type $\mathtt{snat}\, n$
  A data type whose elements are representations of the single integer value $n : \mathsf{Nat}$
- (Syntactic sugar)
  $\mathtt{nat} \ \equiv \ \exists r : \mathsf{Nat}. \mathtt{snat}\, r$

Typed Certified Code (iv)

- Example *(continued)*
  - $\texttt{gcd} \; : \; \forall n{:}\mathsf{Nat}.\,\forall m{:}\mathsf{Nat}.\,\forall p^*{:}(n+m>0).$

$$\texttt{snat}\,n \twoheadrightarrow \texttt{snat}\,m \twoheadrightarrow$$
$$\exists r{:}\mathsf{Nat}.$$
$$\exists q^*{:}(r>0).$$
$$\texttt{snat}\,r$$

"The result shall not be zero."

Typed Certified Code (iv)

- Example *(continued)*
  - $\texttt{gcd} : \forall n{:}\mathsf{Nat}.\forall m{:}\mathsf{Nat}.\forall p^*{:}(n+m>0).$
    $\texttt{snat}\,n \twoheadrightarrow \texttt{snat}\,m \twoheadrightarrow$
    $\exists r{:}\mathsf{Nat}.$
    $\exists q^*{:}(r>0).$
    $\texttt{snat}\,r$

    "The result shall not be zero."
  - $\texttt{gcd} : \forall n{:}\mathsf{Nat}.\forall m{:}\mathsf{Nat}.\forall p^*{:}(n+m>0).$
    $\texttt{snat}\,n \twoheadrightarrow \texttt{snat}\,m \twoheadrightarrow$
    $\exists r{:}\mathsf{Nat}.$
    $\exists q^*{:}(r>0 \wedge r \backslash n \wedge r \backslash m).$
    $\texttt{snat}\,r$

    "The result shall not be zero and shall divide both arguments."

Typed Certified Code *(V)*

- Example *(continued)*
  - gcd : $\forall n : \mathsf{Nat}. \forall m : \mathsf{Nat}. \forall p^* : (n + m > 0).$
    $\mathtt{snat}\, n \twoheadrightarrow \mathtt{snat}\, m \twoheadrightarrow$
    $\exists r : \mathsf{Nat}.$
    $\exists q_1^* : (r > 0 \wedge r \backslash n \wedge r \backslash m).$
    $\exists q_2^* : (\Pi r' : \mathsf{Nat}. r' \backslash n \wedge r' \backslash m \rightarrow r' \leq r).$
    $\mathtt{snat}\, r$

    "The result shall be the greatest common divisor of the two arguments."

Can we combine the two?

- Hoare Logic
  - \+ long studied, large body of scientific knowledge
  - \+ simple axioms and rules
  - \+ works with variables and destructive update
  - – does not work well with (higher-order) functions
  - – proofs of specifications cannot be automatically verified

Can we combine the two?

- Hoare Logic
  - \+ long studied, large body of scientific knowledge
  - \+ simple axioms and rules
  - \+ works with variables and destructive update
  - – does not work well with (higher-order) functions
  - – proofs of specifications cannot be automatically verified
- Typed Certified Code
  - – relatively new approach
  - – highly complex type system
  - – does not work well with variables and destructive update
  - \+ works well with (higher-order) functions
  - \+ proofs of specifications can be automatically verified

Overview of the Type Language

- A variation of the Calculus of Inductive Constructions
- Incorporates higher-order predicate logic
- Complete grammar:

$$A, B ::= \mathsf{Set} \mid \mathsf{Type} \mid \mathsf{Ext} \mid X \mid \Pi X{:}A.\,B \mid \lambda X{:}A.\,B \mid A\,B$$
$$\mid \mathsf{Ind}(X{:}A)\{\vec{A}\} \mid \mathsf{Constr}(n, A) \mid \mathsf{Elim}[A'](A{:}B\,\vec{B})\{\vec{A}\}$$

$$A \rightarrow B \equiv \Pi X_{new}{:}A.\,B$$

Papaspyrou, Vytiniotis and Koutavas, "Logic-Enhanced Type Systems", *PLS 4*, 2003.

Shao, Trifonov, Saha and Papaspyrou, "A Type System for Certified Binaries", *ACM TOPLAS*, vol. 27, no. 1, pp. 1-45, 2005.

The Computation Language (i)

- The simple imperative language WHILE

$$
\begin{array}{lcl}
x : \text{Var} & & \textit{variables} \\
e : \text{Expr} & ::= & n \mid b \mid x \mid \diamond e \mid e \star e \\
c : \text{Comm} & ::= & \texttt{skip} \mid x := e \mid c ; c \\
 & \mid & \texttt{if } e \texttt{ then } c \texttt{ else } c \\
 & \mid & \texttt{while } e \texttt{ do } c \\
\diamond : \text{UnOp} & ::= & - \mid \neg \\
\star : \text{BinOp} & ::= & + \mid - \mid * \mid \texttt{div} \mid \texttt{mod} \\
 & \mid & = \mid \neq \mid < \mid > \mid \leq \mid \geq \\
 & \mid & \texttt{and} \mid \texttt{or}
\end{array}
$$

The Computation Language (ii)

Typing

- Types $\tau : \Omega ::= \mathsf{int} \mid \mathsf{bool}$
- Type environments $\Gamma : \mathsf{Env} \; = \; \mathsf{Var} \to \Omega$
- Typing of expressions $\Gamma \vdash e : \tau$
- Typing of commands $\Gamma \vdash c$

The Computation Language (ii)

Typing

- Types $\tau : \Omega ::= \mathsf{int} \mid \mathsf{bool}$
- Type environments $\Gamma : \mathsf{Env} \; = \; \mathsf{Var} \to \Omega$
- Typing of expressions $\Gamma \vdash e : \tau$
- Typing of commands $\Gamma \vdash c$

Semantics

- Meaning of types $[\![\mathsf{int}]\!] = \mathsf{Int},\ [\![\mathsf{bool}]\!] = \mathsf{Bool}$
- Stores $s : \mathsf{Store}\,\Gamma \; = \; \Pi x{:}\mathsf{Var}.\,[\![\Gamma x]\!]$
- Meaning of expressions $[\![e]\!]\,s \Downarrow v$
- Meaning of commands $[\![c]\!]\,s \Downarrow s'$

Encoding Hoare Logic (i)

- Predicates $P, Q, R : \mathsf{Pred}\,\Gamma \;=\; \mathsf{Store}\,\Gamma \longrightarrow \mathsf{Set}$

Encoding Hoare Logic *(i)*

- Predicates $P, Q, R : \mathsf{Pred}\,\Gamma \;=\; \mathsf{Store}\,\Gamma \to \mathsf{Set}$
- Specification of commands $\{P\}\ c\ \{Q\}$
  $\Gamma \vdash c \qquad P, Q : \mathsf{Pred}\,\Gamma$
- $\{P\}\ c\ \{Q\}$ is valid if for all $s : \mathsf{Store}\,\Gamma$,
  if $P\,s$ and $[\![c]\!]\,s \Downarrow s'$, for some $s' : \mathsf{Store}\,\Gamma$,
  then $Q\,s'$

Encoding Hoare Logic (i)

- Predicates $P, Q, R : \mathsf{Pred}\,\Gamma \;=\; \mathsf{Store}\,\Gamma \rightarrow \mathsf{Set}$
- Specification of commands $\{P\}\ c\ \{Q\}$
  $\Gamma \vdash c \qquad P, Q : \mathsf{Pred}\,\Gamma$
- $\{P\}\ c\ \{Q\}$ is valid if for all $s : \mathsf{Store}\,\Gamma$,
  if $P\,s$ and $[\![c]\!]\,s \Downarrow s'$, for some $s' : \mathsf{Store}\,\Gamma$,
  then $Q\,s'$
- Specification of expressions $\{P\}\ e\ \{F\}$
  $\Gamma \vdash e : \tau \qquad P : \mathsf{Pred}\,\Gamma \qquad F : [\![\tau]\!] \rightarrow \mathsf{Pred}\,\Gamma$
- $\{P\}\ e\ \{F\}$ is valid if for all $s : \mathsf{Store}\,\Gamma$,
  if $P\,s$ and $[\![e]\!]\,s \Downarrow v$ for some $v : [\![\tau]\!]$,
  then $F\,v\,s$

Encoding Hoare Logic (ii)

- Hoare logic axioms and inference rules

$$\{P\}\ \texttt{skip}\ \{P\} \qquad \frac{\{P\}\ e\ \{\lambda v.\lambda s.Q\,s\{x \mapsto v\}\}}{\{P\}\ x := e\ \{Q\}}$$

$$\frac{\{P\}\ c_1\ \{R\} \quad \{R\}\ c_2\ \{Q\}}{\{P\}\ c_1;c_2\ \{Q\}}$$

$$\frac{\{P\}\ e\ \{F\} \quad \{F\,\mathsf{true}\}\ c_1\ \{Q\} \quad \{F\,\mathsf{false}\}\ c_2\ \{Q\}}{\{P\}\ \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \{Q\}}$$

$$\frac{\{P\}\ e\ \{F\} \quad \{F\,\mathsf{true}\}\ c\ \{P\}}{\{P\}\ \texttt{while}\ e\ \texttt{do}\ c\ \{F\,\mathsf{false}\}}$$

Encoding Hoare Logic *(ii)*

- Hoare logic axioms and inference rules

$$\{P\}\ \texttt{skip}\ \{P\} \qquad \frac{\{P\}\ e\ \{\lambda v. \lambda s. Q s\{x \mapsto v\}\}}{\{P\}\ x := e\ \{Q\}}$$

$$\frac{\{P\}\ c_1\ \{R\} \quad \{R\}\ c_2\ \{Q\}}{\{P\}\ c_1; c_2\ \{Q\}}$$

$$\frac{\{P\}\ e\ \{F\} \quad \{F\ \mathsf{true}\}\ c_1\ \{Q\} \quad \{F\ \mathsf{false}\}\ c_2\ \{Q\}}{\{P\}\ \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2\ \{Q\}}$$

$$\frac{\{P\}\ e\ \{F\} \quad \{F\ \mathsf{true}\}\ c\ \{P\}}{\{P\}\ \texttt{while}\ e\ \texttt{do}\ c\ \{F\ \mathsf{false}\}}$$

- Consequence rules

$$\frac{P \Rightarrow P' \quad \{P'\}\ c\ \{Q\}}{\{P\}\ c\ \{Q\}} \qquad \frac{\{P\}\ c\ \{Q'\} \quad Q' \Rightarrow Q}{\{P\}\ c\ \{Q\}}$$

Problems with Hoare Logic

Proof of specifications is undecidable!

- e.g. can we find the unknown R?

$$\frac{\{P\}\ c_1\ \{R\} \quad \{R\}\ c_2\ \{Q\}}{\{P\}\ c_1;c_2\ \{Q\}}$$

Problems with Hoare Logic

Proof of specifications is undecidable!

- e.g. can we find the unknown R?

$$\frac{\{P\}\ c_1\ \{R\} \quad \{R\}\ c_2\ \{Q\}}{\{P\}\ c_1; c_2\ \{Q\}}$$

- Weakest preconditions (Dijkstra 1976)

  $R \;=\; wp[c_2](Q)$

Problems with Hoare Logic

Proof of specifications is undecidable!

- e.g. can we find the unknown R?

$$\frac{\{P\}\ c_1\ \{R\} \quad \{R\}\ c_2\ \{Q\}}{\{P\}\ c_1;c_2\ \{Q\}}$$

- Weakest preconditions (Dijkstra 1976)

$$R = wp[c_2](Q)$$

- but what about F and P'?

$$\frac{\{P\}\ e\ \{F\} \quad \{F\ \mathsf{true}\}\ c\ \{P\}}{\{P\}\ \mathtt{while}\ e\ \mathtt{do}\ c\ \{F\ \mathsf{false}\}} \qquad \frac{P \Rightarrow P' \quad \{P'\}\ c\ \{Q\}}{\{P\}\ c\ \{Q\}}$$

Problems with Hoare Logic

Proof of specifications is undecidable!

- e.g. can we find the unknown R?

$$\frac{\{P\}\ c_1\ \{R\} \quad \{R\}\ c_2\ \{Q\}}{\{P\}\ c_1;c_2\ \{Q\}}$$

- Weakest preconditions (Dijkstra 1976)

$$R \;=\; wp[c_2](Q)$$

- but what about F and P'?

$$\frac{\{P\}\ e\ \{F\} \quad \{F\,\mathsf{true}\}\ c\ \{P\}}{\{P\}\ \mathtt{while}\ e\ \mathtt{do}\ c\ \{F\,\mathsf{false}\}} \qquad \frac{P \Rightarrow P' \quad \{P'\}\ c\ \{Q\}}{\{P\}\ c\ \{Q\}}$$

- Solution: annotate programs with proof hints!

And their Solution (i)

Annotated computation language WHILE

$$e : \mathsf{AExpr}_\Gamma \quad ::= \quad \ldots \mid \texttt{assert}\,[p : P \Rightarrow Q], e$$

$$c : \mathsf{AComm}_\Gamma \quad ::= \quad \ldots \mid \texttt{inv}\,[P]\ \texttt{while}\ e\ \texttt{do}\ c \mid \texttt{assert}\,[p : P \Rightarrow Q]$$

And their Solution (i)

Annotated computation language WHILE

$$e : \mathsf{AExpr}_\Gamma \quad ::= \quad \ldots \mid \texttt{assert}\,[p : P \Rightarrow Q], e$$

$$c : \mathsf{AComm}_\Gamma \quad ::= \quad \ldots \mid \texttt{inv}\,[P]\;\texttt{while}\;e\;\texttt{do}\;c \mid \texttt{assert}\,[p : P \Rightarrow Q]$$

Proof hints:

- All loops have explicit invariants

And their Solution *(i)*

Annotated computation language WHILE

$$e : \mathsf{AExpr}_\Gamma \quad ::= \quad \ldots \mid \mathtt{assert}\,[p : P \Rightarrow Q], e$$

$$c : \mathsf{AComm}_\Gamma \quad ::= \quad \ldots \mid \mathtt{inv}\,[P]\;\mathtt{while}\;e\;\mathtt{do}\;c \mid \mathtt{assert}\,[p : P \Rightarrow Q]$$

Proof hints:

- All loops have explicit invariants
- All uses of the consequence rules are replaced by assert constructs, which
  - provide the implication involved $P \Rightarrow Q$
  - provide a proof of this implication p

$$p : \Pi s : \mathsf{Store}\,\Gamma.\, P\,s \rightarrow Q\,s$$

And their Solution (ii)

- Revised axioms and inference rules

$\{P\}\ \texttt{assert}\,[p : P \Rightarrow Q]\ \{Q\}$

$$\frac{\{P\}\ e\ \{F\} \quad \{F\ \mathsf{true}\}\ c\ \{P\}}{\{P\}\ \texttt{inv}\,[P]\ \texttt{while}\ e\ \texttt{do}\ c\ \{F\ \mathsf{false}\}}$$

And their Solution (ii)

- Revised axioms and inference rules

$$\{P\}\ \texttt{assert}\,[p : P \Rightarrow Q]\ \{Q\} \qquad \frac{\{P\}\ e\ \{F\} \quad \{F\,\mathsf{true}\}\ c\ \{P\}}{\{P\}\ \texttt{inv}\,[P]\ \texttt{while}\ e\ \texttt{do}\ c\ \{F\,\mathsf{false}\}}$$

- Well definedness of weakest preconditions

if	$c = \texttt{assert}\,[p : P \Rightarrow Q]$
then	$wp[c](Q') = P$
provided	$Q \Leftrightarrow Q'$

And their Solution (ii)

- Revised axioms and inference rules

  $\{P\}\ \texttt{assert}\,[p : P \Rightarrow Q]\ \{Q\}$ $\qquad \dfrac{\{P\}\ e\ \{F\} \quad \{F\ \mathsf{true}\}\ c\ \{P\}}{\{P\}\ \texttt{inv}\,[P]\ \texttt{while}\ e\ \texttt{do}\ c\ \{F\ \mathsf{false}\}}$

- Well definedness of weakest preconditions

  if $\quad c = \texttt{assert}\,[p : P \Rightarrow Q]$
  then $\quad wp[c](Q') = P$
  provided $\quad Q \Leftrightarrow Q'$

- The relation $P \Leftrightarrow Q$ defines a decidable notion of predicate equivalence
  - We use $\alpha\beta\eta\iota$-equality in CIC
  - A weaker notion of equivalence would result in fewer explicit assertions

And their Solution (iii)

Theorem

Annotations preserve typing, semantics and logic

And their Solution (iii)

Theorem

Annotations preserve typing, semantics and logic

Theorem

Weakest preconditions are correct and exact

1. *If* $wp[c](Q)$ *is defined*
   then $\{wp[c](Q)\}\ c\ \{Q\}$ *is derivable.*
2. *If* $\{P\}\ c\ \{Q\}$ *is derivable*
   then $wp[c](Q)$ *is defined and* $P \Leftrightarrow wp[c](Q)$

And their Solution (iii)

Theorem

Annotations preserve typing, semantics and logic

Theorem

Weakest preconditions are correct and exact

1. *If* $wp[c](Q)$ *is defined*
   then $\{wp[c](Q)\}\ c\ \{Q\}$ *is derivable.*
2. *If* $\{P\}\ c\ \{Q\}$ *is derivable*
   then $wp[c](Q)$ *is defined and* $P \Leftrightarrow wp[c](Q)$

Theorem

Proof of specifications is decidable

Example

- Find the integer part of $\log_2(n)$

$\{\lambda s.\, s\, n \geq 1 \wedge s\, n = X\}$

```
m := 0;
while n > 1 do
   n := n div 2;
   m := m + 1
```

$\{\lambda s.\, 2^{s\, m} \leq X < 2^{s\, m+1}\}$

Example

- Find the integer part of $\log_2(n)$

  $\{\lambda s. s n \geq 1 \wedge s n = X\}$
  $m := 0;$
  $\texttt{while}\ n > 1\ \texttt{do}$
  $\quad n := n\ \texttt{div}\ 2;$
  $\quad m := m + 1$
  $\{\lambda s. 2^{s m} \leq X < 2^{s m+1}\}$

- Loop invariant

  $\lambda s. X / 2^{s m} = s n \ \wedge\ s n \geq 1 \ \wedge\ s m \geq 0$

Example

- Find the integer part of $\log_2(n)$

  $\{\lambda s. sn \geq 1 \wedge sn = X\}$
  $m := 0;$
  `while` $n > 1$ `do`
  $\quad n := n$ `div` $2;$
  $\quad m := m + 1$
  $\{\lambda s. 2^{sm} \leq X < 2^{sm+1}\}$

- Loop invariant

  $\lambda s. X/2^{sm} = sn \wedge sn \geq 1 \wedge sm \geq 0$

- 4 assertions, of which the hardest to prove is

  $(\lambda s. X/2^{sm} = sn \wedge sn \geq 1 \wedge sm \geq 0 \wedge sn \leq 1) \Rightarrow$
  $\quad (\lambda s. 2^{sm} \leq X < 2^{sm+1})$

Conclusions *(i)*

Our contribution

- Hoare logic and type certified code combined
- Certified programs can be represented in a high-level imperative language with proof hints as annotations
- Specifications are expressed as Hoare triples
- Proof checking is decidable and efficient
- The annotated language is consistent to the original in terms of typing, operational semantics and validity of specifications

Conclusions *(ii)*

Related work

- Hamid and Shao (2004): low-level typed assembly programs, predefined safety policy
- Franssen and de Swart (2004): similar, many-sorted first-order logic, differs in expressiveness and non-foundational character

Conclusions (ii)

Related work

- Hamid and Shao (2004): low-level typed assembly programs, predefined safety policy
- Franssen and de Swart (2004): similar, many-sorted first-order logic, differs in expressiveness and non-foundational character

Future work

- Fewer explicit assertions: weaken the $P \Leftrightarrow Q$ equivalence relation
  - amounts to theorem proving

Conclusions *(ii)*

Related work

- Hamid and Shao (2004): low-level typed assembly programs, predefined safety policy
- Franssen and de Swart (2004): similar, many-sorted first-order logic, differs in expressiveness and non-foundational character

Future work

- Fewer explicit assertions: weaken the $P \Leftrightarrow Q$ equivalence relation
  - amounts to theorem proving
- Exploit more of Hoare logic's benefits in typed certified code
  - variables and destructive update
  - pointers and dynamic variables