Engineering Hoare-Logic based Program Verification in K

Andrei Arusoaie

Faculty of Computer Science, Iași

SYNASC 2013

October 30, 2013
1 Motivation

2 K

3 Methodology

4 Conclusions
Motivation

- K- (remember Grigore Roșu’s talk)
- Framework for defining operational semantics of programming languages (PL)
Motivation

- $\mathbb{K}$ - (remember Grigore Roșu’s talk)
- Framework for defining operational semantics of programming languages (PL)
- We are interested in using the $\mathbb{K}$ semantics for program verification
Motivation

- K- (remember Grigore Roşu’s talk)
- Framework for defining operational semantics of programming languages (PL)
- We are interested in using the K semantics for program verification
- Reachability Logic (see MatchC approach)
Motivation

- K- (remember Grigore Roșu’s talk)
- Framework for defining operational semantics of programming languages (PL)
- We are interested in using the K semantics for program verification
- Reachability Logic (see MatchC approach)
- What about Hoare Logic?
Motivation

- What should we do (in practice) to create a Hoare-like verifier using the \( K \) semantics of a PL?
Problem

Given \( S \), the \( \mathbb{K} \) semantics of a programming language \( \mathcal{L} \)
Problem

- Given $S$, the $\mathbb{K}$ semantics of a programming language $\mathcal{L}$
- Provide a methodology, which applied to the semantics of $L$ transforms the $\mathbb{K}$ tool into a Hoare-like verification tool
K- remember Grigore Roșu’s talk

- K- framework for defining operational semantics of programming languages
- Imperative: IMP, SIMPLE (untyped, typed), C
- Object Oriented: KOOL (untyped, typed), Java, Python
- Assembly: SSRISC, Verilog
- Functional: Scheme, Haskell, OCaml
Defining languages in $\mathbb{K}$

1. Define $\mathcal{L}$’s syntax (BNF)
Defining languages in $\mathcal{K}$

1. Define $\mathcal{L}$’s syntax (BNF)
2. Define $\mathcal{L}$’s semantics
   1. configuration - nested structure of cells
   2. $k$ - computations cell
   3. $\mathcal{K}$ rules - rewriting rules
IMP syntax

- IMP statements

**SYNTAX**  \[ Stmt ::= \{ \} | \{ Stmt \} \\
| \text{Id} = \text{AExp} ; [\text{strict}(2)] \\
| \text{if} (\text{BExp}) \text{Block else Block} [\text{strict}(1)] \\
| \text{while} (\text{BExp}) \text{Block} \\
| Stmt \ Stmt \]
IMP syntax

- IMP statements

**SYNTAX** \[ Stmt ::= \emptyset \mid \{Stmt\} \mid Id = AExp ; \text{[strict(2)]} \mid \text{if}(BExp)Block\text{ else } Block \text{[strict(1)]} \mid \text{while}(BExp)Block \mid StmtStmt \]**

- Sample program SUM:

```plaintext
S = 0; i = 1;
while(i <= n) {
    S = S + i;
    i = i + 1;
}
```
IMP configuration:

Concrete configuration:

\[ S = 0; \quad i = 1; \quad \text{while}(i \leq n) \]
\[ \{ \quad S = S + i; \quad i = i + 1; \quad \} \]

- \( k \) \mapsto 0
- \( i \) \mapsto 0
- \( n \) \mapsto 10

Andrei Arusoaie (UAIC)
IMP configuration:

Concrete configuration:

\[
\begin{align*}
S &= 0; \
i &= 1; \\
\text{while}(i \leq n)\{ \quad S &= S + i; \quad i = i + 1; \}
\end{align*}
\]

\[
\begin{array}{c}
S \mapsto 0 \\
i \mapsto 0 \\
n \mapsto 10
\end{array}
\]
IMP semantics

Assignment:

\[ X = I; \]

\[ X \rightarrow I \]

Concrete configuration:

\[ S = 0; i = 1; \text{while}(i \leq n) \{
S = S + i; i = i + 1;
\}\]

\[ \text{state} \]
IMP semantics

Assignment:

\[ X = I ; \]

\[ X \mapsto I \]

Concrete configuration:

\[ S=0; i=1; \text{while}(i \leq n) \{ S=S+i; i=i+1; \} \]

\[ S \mapsto 0 \]

\[ i \mapsto 0 \]

\[ n \mapsto 10 \]
IMP semantics

Assignment:

Concrete configuration:

Andrei Arusoaie (UAIC) Engineering Hoare-Logic based Program Verification October 30, 2013 11 / 25
IMP semantics

Assignment:

\[ X = I ; \]

Concrete configuration:

\[ i=1; \text{while}(i \leq n)\{ S=S+i; i=i+1; \} \]

\[ S \mapsto 0 \quad i \mapsto 0 \quad n \mapsto 10 \]
IMP semantics

Assignment:

\[
X = I ; \\
\text{\textbullet}_K
\]

Concrete configuration:

\[
i = 1; \text{ while}(i \leq n) \{ \ S = S + i; \ i = i + 1; \} \]

\[
S \leftrightarrow 0 \ i \leftrightarrow 0 \ n \leftrightarrow 10
\]
IMP semantics

Assignment:

\[ X = l ; \]

\[ \cdot_K \]

Concrete configuration:

\[ \text{while}(i \leq n) \{ \ S = S + i ; \ i = i + 1 ; \} \]

\[ S \mapsto 0 \quad i \mapsto 1 \quad n \mapsto 10 \]
Creating Hoare Logic-based verification tools in $\mathbb{K}$

- In practice, a Hoare Logic verifier is language dependent
In practice, a Hoare Logic verifier is language dependent:

- deductive rules depend on the language syntactical constructs
- syntax of the logical assertions
- ...

...
Creating Hoare Logic-based verification tools in $K$

- In practice, a Hoare Logic verifier is language dependent
  - deductive rules depend on the language syntactical constructs
  - syntax of the logical assertions
  - ...

- We have to deal with
Creating Hoare Logic-based verification tools in $\mathbb{K}$

- In practice, a Hoare Logic verifier is language dependent
  - deductive rules depend on the language syntactical constructs
  - syntax of the logical assertions
  - ...

- We have to deal with
  - language dependency issues (syntax & semantics)
Creating Hoare Logic-based verification tools in $\mathbb{K}$

- In practice, a Hoare Logic verifier is language dependent
  - deductive rules depend on the language syntactical constructs
  - syntax of the logical assertions
  - ...

- We have to deal with
  - language dependency issues (syntax & semantics)
  - support for verification of Hoare triples in $\mathbb{K}$ (transform Hoare triples into Reachability formulas)
Hoare Logic vs. Reachability logic - example

- \[ \text{SUM} = \text{"S=0; i=1; while(i<=n)\{ S=S+i; i=i+1;\}"} \]
Hoare Logic vs. Reachability logic - example

- SUM = “S=0; i=1; while(i<=n){ S=S+i; i=i+1;}”

- Hoare triple:

\[ \{ n \geq 0 \} \text{SUM} \{ S = n \times (n + 1)/2 \} \]
Hoare Logic vs. Reachability logic - example

- $SUM = "S=0; \ i=1; \ while(i<=n)\{ \ S=S+i; \ i=i+1;\}"$

- Hoare triple:

\[
\{n \geq 0\} \quad SUM \quad \{S = n \cdot (n + 1)/2\}
\]

- Reachability rule:

\[
\begin{align*}
&k \\
&\text{SUM} \quad S \leftrightarrow s, \ n \leftrightarrow n, \ i \leftrightarrow i \\
&\quad \land n \geq 0 \Rightarrow \\
&k \\
&\text{state} \quad S \leftrightarrow s', \ n \leftrightarrow n', \ i \leftrightarrow i' \\
&\quad \land s' = n \cdot (n + 1)/2
\end{align*}
\]
Hoare Logic vs. Reachability logic - example

- $\text{SUM} = \text{"S=0; i=1; while(i<=n)\{ S=S+i; i=i+1; \}"}$

- Hoare triple:

  $$\{n \geq 0\} \text{SUM} \{S = n \times (n + 1)/2\}$$

- Reachability rule:

  $k$
  \[
  \begin{array}{c}
  \text{SUM} \\
  S \leftrightarrow s, n \leftrightarrow n, i \leftrightarrow i \\
  \land n \geq 0 \Rightarrow
  \end{array}
  \]

  $k$
  \[
  \begin{array}{c}
  . \\
  S \leftrightarrow s', n \leftrightarrow n', i \leftrightarrow i' \\
  \land s' = n \times (n + 1)/2
  \end{array}
  \]

- Grigore Roșu and Andrei Ștefănescu.
  From hoare logic to matching logic reachability.
  FM’12
Verification of Hoare triples using symbolic execution

- Intuition: transform Hoare triples into reachability rules and verify them using symbolic execution
Verification of Hoare triples using symbolic execution

- Intuition: transform Hoare triples into reachability rules and verify them using symbolic execution
- If $\varphi \Rightarrow \varphi'$, where $\varphi = \pi \land \psi$, then
Intuition: transform Hoare triples into reachability rules and verify them using symbolic execution

If $\varphi \Rightarrow \varphi'$, where $\varphi = \pi \land \psi$, then

- assume $\varphi$: $\pi$ the initial configuration, $\psi$ the initial path condition
Verification of Hoare triples using symbolic execution

- Intuition: transform Hoare triples into reachability rules and verify them using symbolic execution
- If $\varphi \Rightarrow \varphi'$, where $\varphi = \pi \land \psi$, then
  - assume $\varphi$: $\pi$ the initial configuration, $\psi$ the initial path condition
  - perform symbolic execution of $\varphi$, and obtain $\{\varphi''\}$
Verification of Hoare triples using symbolic execution

- Intuition: transform Hoare triples into reachability rules and verify them using symbolic execution
- If $\varphi \Rightarrow \varphi'$, where $\varphi = \pi \land \psi$, then
  - assume $\varphi$: $\pi$ the initial configuration, $\psi$ the initial path condition
  - perform symbolic execution of $\varphi$, and obtain $\{\varphi''\}$
  - check if for all $\varphi''$, $\varphi''$ implies $\varphi'$
Methodology

Steps:

- Add syntax and semantics for Hoare-like annotations
- Define `assume` & `implies` operations for checking reachability rules
Methodology

- **Steps:**
  - Add syntax and semantics for Hoare-like annotations
Methodology

Steps:
- Add syntax and semantics for Hoare-like annotations
- Define `assume` & `implies` operations for checking reachability rules
Add syntax for Hoare-like annotations

pre: \( 0 \leq n \)
post: \( 2 \times S = n \times (n + 1) \)

\[
\begin{align*}
S &= 0; \\
i &= 1; \\
\text{while } (i \leq n) \\
\text{inv: } 2\times S &= i \times (i - 1) \text{ and } i \leq n + 1 \\
\{ \\
& \quad S = S + i; \\
& \quad i = i + 1; \\
\}
\end{align*}
\]
Semantics of the new constructs

- Pre/post conditions

\[
\text{assume (} \begin{array}{c} k \\ Stmt \end{array} \sigma \land \psi) \implies (\begin{array}{c} k \\ . \end{array} \sigma' \land \psi')
\]

\[
\text{pre: } \psi \text{ post: } \psi'
\]

Stmt
Semantics of the new constructs

- **while loop**

\[
\begin{align*}
\text{while}(B)\text{inv:}\psi \text{ Stmt } & \sim K \\
\text{assume}(\text{Stmt} & \land B \land \psi) \land \text{implies}(\text{Stmt} & \land \psi) \\
\text{implies}(\cdot & \land \psi) \land \text{assume}(K & \land \neg B \land \psi)
\end{align*}
\]
Define assume

- **assume semantics:**

\[
\text{assume (} \quad \begin{array}{c}
\text{Stmt} \\
\text{k}
\end{array} \quad E \quad \text{\textstate} \quad \land \; \psi \\
\hline
\text{Stmt}
\end{array} \quad \begin{array}{c}
\text{state} \\
\quad \overline{E}
\end{array} \quad \land \; \overline{\psi}\]

Andrei Arusoaie (UAIC)
Define implies

- implies semantics:

\[
\text{implies}(k, \text{state } E' \land \psi') \quad \text{when} \quad (\text{state } E'' \land \psi'') \rightarrow (\text{state } E' \land \psi')
\]
Conclusions

- Methodology to create a Hoare Logic verifier in K

Future work: eliminate \texttt{assume} and \texttt{implies}
Conclusions

- Methodology to create a Hoare Logic verifier in \( K \)
- Prove reachability rules generated from Hoare triples, using symbolic execution
Conclusions

- Methodology to create a Hoare Logic verifier in $\mathbb{K}$
- Prove reachability rules generated from Hoare triples, using symbolic execution
- Future work: eliminate assume and implies
Thank you!

The $\mathbf{K}$ framework webpage:

http://kframework.org/