Automatic Construction of Hoare Proof Trees from Abstract Interpretation Results

Sunae Seo

with Hongseok Yang and Kwangkeun Yi

2003. 8. 18
LiComR workshop 2003

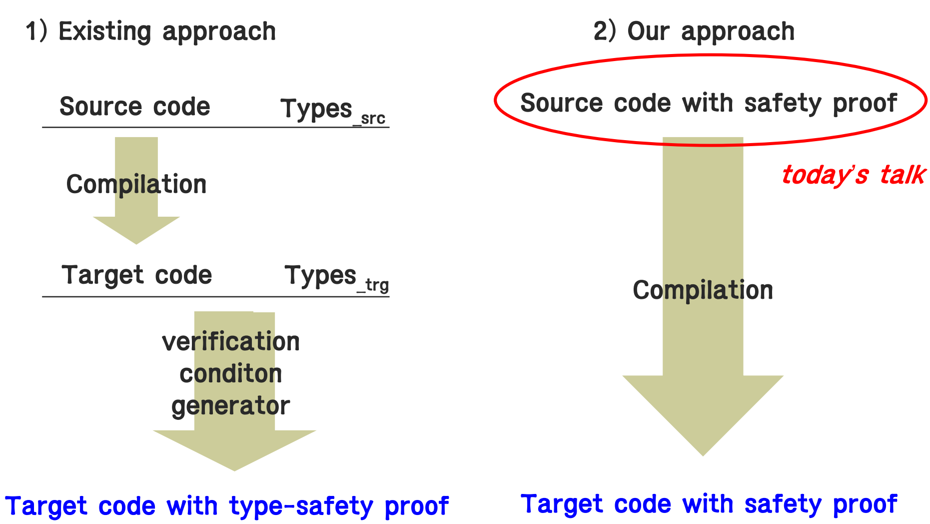
Contents

- Problem
- Outline
- Approach
- Language and preliminary
- Example and algorithm
- Summary
- Future works

Problem

- The safety of mobile code is important
  - Proof-Carrying Code : a safety proof + code
    - A code producer generates a proof.
    - A code consumer checks the proof.
- Limitation of the existing works
  - How to generate program proofs other than types

Outline



Our Approach

- We combine two technologies.
  - Program analysis technique (*abstract interpretation*): to obtain program properties
  - Program logic (*Hoare logic*): to represent and to check proofs of program properties

Contents

Language

Program	P	$::=$	$C;;$	
Commands	C	$::=$	$x := E$	assignment
		$\mid$	$\texttt{if}\ B\ \texttt{then}\ C\ \texttt{else}\ C\ \texttt{fi}$	if-statement
		$\mid$	$\texttt{while}\ B\ \texttt{do}\ C\ \texttt{od}$	while-statement
		$\mid$	$C; C$	sequence
ArithmeticExpressions	E	$::=$	m	integer
		$\mid$	x	program variable
		$\mid$	$E + E$	addition
		$\mid$	$E - E$	subtraction
		$\mid$	$E \times E$	multiplication
BooleanExpressions	B	$::=$	$\texttt{tt}$	true
		$\mid$	$\texttt{ff}$	false
		$\mid$	$B \wedge B$	conjuction
		$\mid$	$B \vee B$	disjunction
		$\mid$	$E = E$	equality
		$\mid$	$E < E$	inequality

Hoare Logic (1/2)

- A specification for program properties
  - A triple with two assertions and a command

$$\{\phi\}\ P\ \{\psi\}$$

Hoare Logic (2/2)

A proof system for verifying Hoare triples

$$\frac{}{\{\psi[E/x]\}\ x := E\ \{\psi\}} \quad \textbf{[HA]}$$

$$\frac{\{\phi \wedge B\}\ C_1\ \{\psi\} \quad \{\phi \wedge \neg B\}\ C_2\ \{\psi\}}{\{\phi\}\ \texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ \texttt{fi}\ \{\psi\}} \quad \textbf{[HI]}$$

✓ $$\frac{\{\psi \wedge B\}\ C\ \{\psi\}}{\{\psi\}\ \texttt{while}\ B\ \texttt{do}\ C\ \texttt{od}\ \{\psi \wedge \neg B\}} \quad \textbf{[HW]}$$

$$\frac{\{\phi\}\ C_1\ \{\eta\} \quad \{\eta\}\ C_2\ \{\psi\}}{\{\phi\}\ C_1; C_2\ \{\psi\}} \quad \textbf{[HS]}$$

✓ $$\frac{\phi' \Rightarrow \phi \quad \{\phi\}\ C\ \{\psi\} \quad \psi \Rightarrow \psi'}{\{\phi'\}\ C\ \{\psi'\}} \quad \textbf{[HC]}$$

Properties from abstract interpretation results

Abstract Interpretation

- A general program analysis framework to compute program properties using an abstract semantic domain and an abstract interpreter.
  - A concrete semantics of a program assigns a set of states to each program point.

$$\langle \wp(\mathbf{Int}), \subseteq \rangle \quad \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \quad \langle \mathbf{D}, \sqsubseteq \rangle$$

$$\langle \wp(\mathbf{Var} \mapsto \mathbf{Int}), \dot{\subseteq} \rangle \quad \underset{\dot{\alpha}}{\overset{\dot{\gamma}}{\leftrightarrows}} \quad \langle \mathbf{Var} \mapsto \mathbf{D}, \dot{\sqsubseteq} \rangle$$

An Example (Interval Analysis)

$\{x : [-1, 2], y : [0, 4]\}$

```
if x + y = 0
then
```

$\{x : [-1, 2], y : [0, 4]\}$

```
   x := x + y
```

$\{x : [-1, 6], y : [0, 4]\}$

```
else
```

$\{x : [-1, 2], y : [0, 4]\}$

```
   x := 0
```

$\{x : [0, 0], y : [0, 4]\}$

```
fi
```

$\{x : [-1, 6], y : [0, 4]\}$

Simple Outline of our Work

Abstract interpretation result

ifS

$\{x \mapsto [-1, 2], y \mapsto [0, 4]\}$

$\texttt{if x + y = 0 then x := x + y else x := 0 fi}$

$\{x \mapsto [-1, 6], y \mapsto [0, 4]\}$

Hoare specification

$$\{\mathsf{tr}_{state}(\{ \begin{array}{l} x \mapsto [-1, 2], \\ y \mapsto [0, 4] \end{array} \})\} \ \texttt{ifS} \ \{\mathsf{tr}_{state}(\{ \begin{array}{l} x \mapsto [-1, 6], \\ y \mapsto [0, 4] \end{array} \})\}$$

Hoare proof tree

$$\frac{\triangledown}{\{(-1 \le x \le 2) \wedge (0 \le y \le 4)\} \ \texttt{ifS} \ \{(-1 \le x \le 1) \wedge (0 \le y \le 4)\}}$$

Generic Abstract Interpretation

$$
\begin{array}{lcl}
\multicolumn{3}{l}{\mathit{Commands} \ : \ \hat{State} \to \hat{State}} \\
[\![x := E]\!]\ \sigma & = & \sigma[x \mapsto ([\![E]\!]\ \sigma)] \\
[\![\texttt{if}\ B\ \texttt{then}\ C_1\ \texttt{else}\ C_2\ \texttt{fi}]\!]\ \sigma & = & ([\![C_1]\!]\ \sigma) \sqcup ([\![C_2]\!]\ \sigma) \\
[\![\texttt{while}\ B\ \texttt{do}\ C\ \texttt{od}]\!]\ \sigma & = & \mathit{lfp}\ \lambda X.\ \sigma \sqcup ([\![C]\!]\ X) \\
[\![C_1; C_2]\!]\ \sigma & = & [\![C_2]\!]\ ([\![C_1]\!]\ \sigma)
\end{array}
$$

$$
\begin{array}{lcl}
\multicolumn{3}{l}{\mathit{ArithmeticExpression} \ : \ \hat{State} \to \hat{Value}} \\
[\![m]\!]\ \sigma & = & \alpha(\{m\}) \\
[\![x]\!]\ \sigma & = & \sigma(x) \\
[\![E_1 + E_2]\!]\ \sigma & = & ([\![E_1]\!]\ \sigma)\ \hat{+}\ ([\![E_2]\!]\ \sigma) \\
[\![E_1 - E_2]\!]\ \sigma & = & ([\![E_1]\!]\ \sigma)\ \hat{-}\ ([\![E_2]\!]\ \sigma) \\
[\![E_1 \times E_2]\!]\ \sigma & = & ([\![E_1]\!]\ \sigma)\ \hat{\times}\ ([\![E_2]\!]\ \sigma)
\end{array}
$$

$$
\begin{array}{lcllcl}
\alpha_{interval}(\{1\}) & = & [1,1] & \gamma_{interval}([1,1]) & = & \{1\} \\
\alpha_{interval}(\{3,5,7\}) & = & [3,7] & \gamma_{interval}([3,7]) & = & \{3,4,5,6,7\} \\
\alpha_{even\_odd}(\{3,5,7\}) & = & \mathbf{odd} & \gamma_{even\_odd}(\mathbf{odd}) & = & \{\cdots, -3, -1, 1, 3, \cdots\}
\end{array}
$$

Safety Requirement of Abstract Operators

- Soundness of abstract operators

$$
\begin{array}{lcl}
p\ \hat{+}\ q & \sqsupseteq & \alpha(\{m + n \in \mathbb{Z} \mid m \in \gamma(p),\ n \in \gamma(q)\}) \\
p\ \hat{-}\ q & \sqsupseteq & \alpha(\{m - n \in \mathbb{Z} \mid m \in \gamma(p),\ n \in \gamma(q)\}) \\
p\ \hat{\times}\ q & \sqsupseteq & \alpha(\{m \times n \in \mathbb{Z} \mid m \in \gamma(p),\ n \in \gamma(q)\})
\end{array}
$$

For example,

$$[1,3]\hat{+}_{interval}[4,5] \ \sqsupseteq \ \alpha_{interval}(\{m+n | m \in \{1,2,3\} \wedge n \in \{4,5\}\})$$

Examples (Sound Abstract Operators)

- Even-odd analysis

$$
\begin{array}{rcl@{\qquad\qquad}rcl}
e \hat{+} e & \triangleq & e & e \hat{\times} e & \triangleq & e \\
e \hat{+} o & \triangleq & o & e \hat{\times} o & \triangleq & e \\
o \hat{+} o & \triangleq & e & o \hat{\times} o & \triangleq & o \\
o \hat{+} e & \triangleq & o & o \hat{\times} e & \triangleq & e
\end{array}
$$

- Interval analysis

$$
\begin{array}{rcl}
[n_0, m_0] \hat{+} [n_1, m_1] & \triangleq & [n_0 + n_1, m_0 + m_1] \\
[n_0, m_0] \hat{\times} [n_1, m_1] & \triangleq & [\mathsf{min}\ X, \mathsf{max}\ X] \\
& & (\text{where } X = \{n_0 \times n_1, n_0 \times m_1, m_0 \times n_1, m_0 \times m_1\})
\end{array}
$$

Claim

- We construct Hoare proof tree when we have
  - soundness proof of abstract operators
  - translation function tr

Safety Requirement of Abstract Operators in Proof Tree

- Soundness proof of abstract operators

$$[\mathbf{AP}] \quad \exists \nabla : \frac{\nabla}{(\mathsf{tr}(p, E_1) \wedge \mathsf{tr}(q, E_2)) \Rightarrow \mathsf{tr}(p \hat{+} q, E_1 + E_2)}$$

$$[\mathbf{AM}] \quad \exists \nabla : \frac{\nabla}{(\mathsf{tr}(p, E_1) \wedge \mathsf{tr}(q, E_2)) \Rightarrow \mathsf{tr}(p \hat{-} q, E_1 - E_2)}$$

$$[\mathbf{AT}] \quad \exists \nabla : \frac{\nabla}{(\mathsf{tr}(p, E_1) \wedge \mathsf{tr}(q, E_2)) \Rightarrow \mathsf{tr}(p \hat{\times} q, E_1 \times E_2)}$$

Translation function tr(*d*,*E*) (1/2)

$$\mathsf{tr} \; : \; \hat{Value} \times Expressions \rightarrow Formulas$$

- It returns a formula indicating "the value of E is included in the meaning of d."
- For example,

$$\mathsf{tr}_{interval}([1,3], E) \;=\; 1 \leq E \leq 3$$
$$\mathsf{tr}_{even\_odd}(\mathtt{Odd}, E) \;=\; \exists n : \; E = 2n + 1$$

An Example

$\{x : [-1, 2], y : [0, 4]\}$
```
if x + y = 0
then
```
$\qquad\{x : [-1, 2], y : [0, 4]\}$
```
    x := x + y
```
$\qquad\{x : [-1, 6], y : [0, 4]\}$
```
else
```
$\qquad\{x : [-1, 2], y : [0, 4]\}$
```
    x := 0
```
$\qquad\{x : [0, 0], y : [0, 4]\}$
```
fi
```
$\{x : [-1, 6], y : [0, 4]\}$

[AD1] if $d_1 \sqsubseteq d_2$, $\dfrac{\nabla}{\mathsf{tr}(d_1, E) \Rightarrow \mathsf{tr}(d_2, E)}$

[AP] $\dfrac{\nabla}{(\mathsf{tr}(p, E_1) \wedge \mathsf{tr}(q, E_2)) \;\Rightarrow\; \mathsf{tr}(p \hat{+} q, E_1 + E_2)}$

[AD1]

$\vdots$

$$\overline{\mathsf{tr}_{state}(\sigma_1) \Rightarrow \mathsf{tr}_{state}(\sigma_1 \sqcup \sigma_2)}$$

[AP]

$\vdots$

$$\overline{\mathsf{tr}_{state}(\sigma) \wedge (x + y = 0) \Rightarrow \mathsf{tr}_{state}(\sigma)} \qquad \overline{\{\mathsf{tr}_{state}(\sigma)\}\ \mathtt{x := x + y}\ \{\mathsf{tr}_{state}([\![x := x + y]\!]\sigma)\}} \qquad \overline{\mathsf{tr}_{state}([\![x := x + y]\!]\sigma) \Rightarrow \psi}$$

$$\dfrac{\overline{\{\mathsf{tr}_{state}(\sigma) \wedge (x + y = 0)\}\ \mathtt{x := x + y}\ \{\psi\}} \qquad \{\mathsf{tr}_{state}(\sigma) \wedge (x + y \neq 0)\}\ \mathtt{x := 0}\ \{\psi\}}{\{\mathsf{tr}_{state}(\sigma)\}\ \mathtt{ifS}\{\psi\}}$$

- $\mathtt{ifS} \triangleq \mathtt{if\ x + y = 0\ then\ x := x + y\ else\ x := 0\ fi}$
- $\mathsf{tr}_{state}(\sigma) \triangleq \bigwedge_{x \in domain(\sigma)} \mathsf{tr}(\sigma(x), x)$
- $\psi \triangleq \mathsf{tr}_{state}(([\![\mathtt{x := x + y}]\!]\sigma) \sqcup (([\![\mathtt{x := 0}]\!]\sigma))$

$$\dfrac{\dfrac{\dfrac{\vdots}{\begin{array}{l}(-1 \le x \le 2) \wedge (0 \le y \le 4)\\ \wedge (x+y=0)\\ \Rightarrow (-1 \le x \le 2) \wedge (0 \le y \le 4)\end{array}} \quad \dfrac{\vdots}{\left\{ \begin{array}{l}(-1 \le x \le 2)\\ \wedge(0 \le y \le 4)\end{array} \right\} \; \mathtt{x := x + y} \; \left\{ \begin{array}{l}(-1 \le x \le 6)\\ \wedge(0 \le y \le 4)\end{array} \right\}} \quad \dfrac{\vdots}{\begin{array}{l}(-1 \le x \le 6) \wedge (0 \le y \le 4)\\ \Rightarrow (-1 \le x \le 6) \wedge (0 \le y \le 4)\end{array}}}{\{(-1 \le x \le 2) \wedge (0 \le y \le 4) \wedge (x+y=0)\} \; \mathtt{x := x + y} \; \{(-1 \le x \le 6) \wedge (0 \le y \le 4)\}}}{\{(-1 \le x \le 2) \wedge (0 \le y \le 4)\} \; \mathtt{ifS} \; \{(-1 \le x \le 6) \wedge (0 \le y \le 4)\}}$$

- $\mathtt{ifS} \triangleq \mathtt{if\ x + y = 0\ then\ x := x + y\ else\ x := 0\ fi}$
- $\mathsf{tr}_{state}(\sigma) = (-1 \le x \le 2) \wedge (0 \le y \le 4)$
- $\psi = (-1 \le x \le 6) \wedge (0 \le y \le 4)$
- $\mathsf{tr}_{state}([\![x := x + y]\!]\sigma) = (-1 \le x \le 6) \wedge (0 \le y \le 4)$

Generic AI (full)

$$\begin{array}{l}
\mathit{Commands} \;:\; \hat{State} \to \hat{State}\\
\quad [\![\mathtt{if}\ B\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2\ \mathtt{fi}]\!]\ \sigma \;=\; ([\![C_1]\!]\ ([\![B]\!]\ \sigma)) \sqcup ([\![C_2]\!]\ ([\![\neg B]\!]\ \sigma))\\
\quad [\![\mathtt{while}\ B\ \mathtt{do}\ C\ \mathtt{od}]\!]\ \sigma \;=\; [\![\neg B]\!]\ (\mathit{lfp}\ \lambda X.\ \sigma \sqcup ([\![C]\!]([\![B]\!]\ X)))
\end{array}$$

$$\begin{array}{lcl}
\multicolumn{3}{l}{\mathit{BooleanExpressions} \;:\; \hat{State} \to \hat{State}}\\
\quad [\![\mathtt{tt}]\!]\ \sigma & = & \sigma\\
\quad [\![\mathtt{ff}]\!]\ \sigma & = & \forall x \in \text{variables in the program} : x \mapsto \bot\\
\quad [\![B_1 \wedge B_2]\!]\ \sigma & = & ([\![B_1]\!]\ \sigma) \sqcap ([\![B_2]\!]\ \sigma)\\
\quad [\![B_1 \vee B_2]\!]\ \sigma & = & ([\![B_1]\!]\ \sigma) \sqcup ([\![B_2]\!]\ \sigma)\\
\quad [\![E_1 = E_2]\!]\ \sigma & = & ([\![E_1]\!]_b\ \sigma\ (\hat{=}_1([\![E_1]\!]\ \sigma, [\![E_2]\!]\ \sigma))) \sqcap ([\![E_2]\!]_b\ \sigma\ (\hat{=}_2([\![E_1]\!]\ \sigma, [\![E_2]\!]\ \sigma)))\\
\quad [\![E_1 < E_2]\!]\ \sigma & = & ([\![E_1]\!]_b\ \sigma\ (\hat{<}_1([\![E_1]\!]\ \sigma, [\![E_2]\!]\ \sigma))) \sqcap ([\![E_2]\!]_b\ \sigma\ (\hat{<}_2([\![E_1]\!]\ \sigma, [\![E_2]\!]\ \sigma)))
\end{array}$$

$$\begin{array}{l}
\mathit{Backward\ Expressions}\\
\qquad : \hat{State} \times \hat{Value} \to \hat{State}
\end{array}$$

Second Example

$\{x : [1,4], y : [2,5]\}$
```
if x = y + 1
then
```
$\quad \{x : [3,4], y : [2,3]\}$
```
   x := x + y
```
$\quad \{x : [5,7], y : [2,3]\}$
```
else
```
$\quad \{x : [1,4], y : [2,5]\}$
```
   x := x + 1
```
$\quad \{x : [2,5], y : [2,5]\}$
```
fi
```
$\{x : [2,7], y : [2,5]\}$

Proof tree for the example

$$\left\{ \begin{array}{r} 3 \le x \le 4 \\ \wedge\ 2 \le y \le 3 \end{array} \right\} x := x + y \left\{ \begin{array}{r} 5 \le x \le 7 \\ \wedge\ 2 \le y \le 3 \end{array} \right\} \qquad \left\{ \begin{array}{r} 1 \le x \le 4 \\ \wedge\ 2 \le y \le 3 \end{array} \right\} x := x + 1 \left\{ \begin{array}{r} 2 \le x \le 5 \\ \wedge\ 2 \le y \le 5 \end{array} \right\}$$

$$\begin{array}{rcl} 1 \le x \le 4 \ \wedge\ 2 \le y \le 5 \ \wedge\ x = y + 1 & \Rightarrow & 3 \le x \le 4 \ \wedge\ 2 \le y \le 3 \\ 1 \le x \le 4 \ \wedge\ 2 \le y \le 5 \ \wedge\ x \ne y + 1 & \Rightarrow & 1 \le x \le 4 \ \wedge\ 2 \le y \le 5 \\ 5 \le x \le 7 \ \wedge\ 2 \le y \le 3 & \Rightarrow & 2 \le x \le 7 \ \wedge\ 2 \le y \le 5 \\ 2 \le x \le 5 \ \wedge\ 2 \le y \le 5 & \Rightarrow & 2 \le x \le 7 \ \wedge\ 2 \le y \le 5. \end{array}$$

$$\frac{\begin{array}{c} \vdots \\ \left\{ \begin{array}{r} 1 \le x \le 4 \\ \wedge\ 2 \le y \le 5 \\ \wedge\ x = y + 1 \end{array} \right\} x := x + y \left\{ \begin{array}{r} 2 \le x \le 7 \\ \wedge\ 2 \le y \le 5 \end{array} \right\} \end{array} \qquad \begin{array}{c} \vdots \\ \left\{ \begin{array}{r} 1 \le x \le 4 \\ \wedge\ 2 \le y \le 5 \\ \wedge\ x \ne y + 1 \end{array} \right\} x := x + 1 \left\{ \begin{array}{r} 2 \le x \le 7 \\ \wedge\ 2 \le y \le 5 \end{array} \right\} \end{array}}{\left\{ \begin{array}{r} 1 \le x \le 4 \\ \wedge\ 2 \le y \le 5 \end{array} \right\} \texttt{if}\ x = y + 1\ \texttt{then}\ x := x + y\ \texttt{else}\ x := x + 1\ \texttt{fi} \left\{ \begin{array}{r} 2 \le x \le 7 \\ \wedge\ 2 \le y \le 5 \end{array} \right\}}$$

Algorithm

Algorithm

- **Proof construction** $\mathcal{T}([s]C[s'])$ **for an annotated program**

$$\frac{\vdots}{\{\mathsf{trst}(\hat{s})\}\ C\ \{\mathsf{trst}(\hat{s}')\}}$$

- **Proof construction** $\mathcal{E}\ (s, E)$ **for an arithmetic expression** E

$$\frac{\vdots}{\mathsf{trst}(\hat{s}) \Rightarrow \mathsf{tr}([\![E]\!]\hat{s}, E)}$$

- **Proof construction** $\mathcal{E}_b\ (s, a, E)$ **for a backward arithmetic expression** E

$$\frac{\vdots}{\mathsf{trst}(\hat{s}) \wedge \mathsf{tr}(a, E) \Rightarrow \mathsf{trst}([\![E]\!]_b \hat{s}\ a)}$$

- **Proof construction** $\mathcal{B}_b\ (s, B)$ **for a boolean expression** B

$$\frac{\vdots}{\mathsf{trst}(\hat{s}) \wedge B \Rightarrow \mathsf{trst}([\![B]\!]_b \hat{s})}$$

Proof Construction $\mathcal{T}([s]C[s'])$ (part)

Case $A \equiv [\hat{s}]x := E[\hat{s}']$

$$\dfrac{\dfrac{\dfrac{}{\mathsf{trst}(\hat{s}) \Rightarrow (\bigwedge_{y \in \mathsf{Vars}-\{x\}} \mathsf{tr}(\hat{s}'(y), y))} \quad \mathcal{E}(\hat{s}, E)}{\mathsf{trst}(\hat{s}) \Rightarrow \mathsf{trst}(\hat{s}')[E/x]} \qquad \dfrac{}{\{\mathsf{trst}(\hat{s}')[E/x]\}\ x := E\ \{\mathsf{trst}(\hat{s}')\}}}{\{\mathsf{trst}(\hat{s})\}\ x := E\ \{\mathsf{trst}(\hat{s}')\}}$$

Case $A \equiv [\hat{s}]\texttt{if}\ B\ \texttt{then}\ [\hat{s}_1]R_1[\hat{s}_1']\ \texttt{else}\ [\hat{s}_2]R_2[\hat{s}_2']\ \texttt{fi}[\hat{s}']$

$$\dfrac{\dfrac{\mathcal{B}_b(\hat{s}, B) \quad \mathcal{T}([\hat{s}_1]R_1[\hat{s}_1']) \quad \mathsf{monSt}(\hat{s}_1', \hat{s}_1' \sqcup \hat{s}_2')}{\{\mathsf{trst}(\hat{s}) \wedge B\}\ \overline{R_1}\ \{\mathsf{trst}(\hat{s}')\}} \quad \dfrac{\mathcal{B}_b(\hat{s}, \neg B) \quad \mathcal{T}([\hat{s}_2]R_2[\hat{s}_2']) \quad \mathsf{monSt}(\hat{s}_2', \hat{s}_1' \sqcup \hat{s}_2')}{\{\mathsf{trst}(\hat{s}) \wedge \neg B\}\ \overline{R_2}\ \{\mathsf{trst}(\hat{s}')\}}}{\{\mathsf{trst}(\hat{s})\}\ \texttt{if}\ B\ \texttt{then}\ \overline{R_1}\ \texttt{else}\ \overline{R_2}\ \texttt{fi}\ \{\mathsf{trst}(\hat{s}')\}}$$

Complexity

- **Proposition**
  - For a command A of size n, the tree $\mathcal{T}(A)$ has $O(n^2 + n \times |Vars|)$ nodes

Summary

- Hoare proof construction method from program analysis results (for use in safe anonymous computing of mobile environment).
  - fully automatic
  - general
    - insensitive to the various abstract properties derived from a single concrete semantics.
    - room for trade-off between trusted base size and proof size.
  - not completely general
    - our method is tightly coupled with the source language and the concrete semantics.
    - for different concrete semantics we must change the proof construction method and the trusted base.

Things to Complete

- Currently, we are working on the implementation emitting Isabelle code.
- To study practicality of the current work
  - Finding interesting analysis example to apply our method
  - Estimating practicality factors like proof tree size
- To extend the language syntax with arrays and pointers

Backup Slides

Related Works

- PCC (Proof-Carrying Code) [Ne97]
  - Gaining safety certainty on mobile code
  - Focusing on proof checking
- Foundational PCC [App01]
  - Compensating PCC by reducing the trusted-bases
  - Limited for types

References

Ne97 George C. Necula. Proof-Carrying Code. In *Proceedings of The ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pages,* 106-119, 1997.

App01 Andrew W. Appel. Foundational proof-carrying code. In *16th Annual IEEE Symposium on Logic in Computer Science,* June 2001.

HST+02 Nadeem Hamid, Zhong Shao, Valery Trifonov, Stefan Monier, and Zhaoshong Ni. A syntactic approach to foundational proof-carrying code. In *17th Annual IEEE Symposium on Logic in Computer Science,* June 2002.

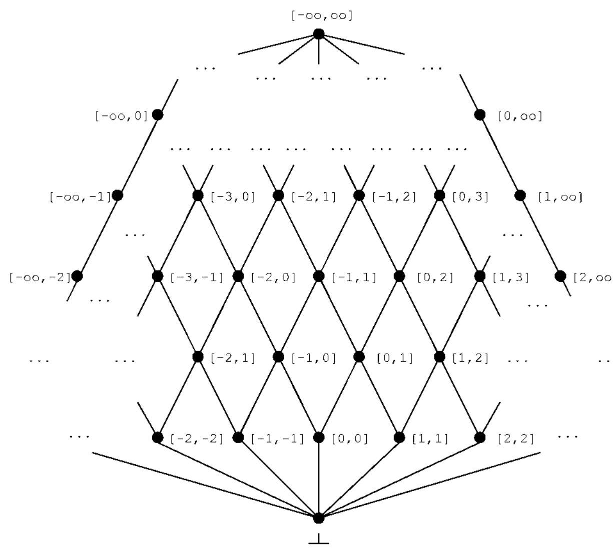
Boolean Rewriting

$$
\begin{array}{rcl}
\overline{\texttt{tt}} & \triangleq & \texttt{tt} \\
\overline{\texttt{ff}} & \triangleq & \texttt{ff} \\
\overline{E_1 < E_2} & \triangleq & E_1 < E_2 \\
\overline{E_1 \leq E_2} & \triangleq & (E_1 < E_2) \vee (E_1 = E_2) \\
\overline{E_1 = E_2} & \triangleq & E_1 = E_2 \\
\overline{E_1 <> E_2} & \triangleq & (E_1 < E_2) \vee (E_1 > E_2) \\
\overline{E_1 > E_2} & \triangleq & E_1 > E_2 \\
\overline{E_1 \geq E_2} & \triangleq & (E_1 = E_2) \vee (E_1 > E_2) \\
\overline{B_1 \vee B_2} & \triangleq & B_1 \vee B_2 \\
\overline{B_1 \wedge B_2} & \triangleq & B_1 \wedge B_2
\end{array}
\qquad
\begin{array}{rcl}
\overline{\neg\texttt{tt}} & \triangleq & \texttt{ff} \\
\overline{\neg\texttt{ff}} & \triangleq & \texttt{tt} \\
\overline{\neg(E_1 < E_2)} & \triangleq & \overline{E_1 \geq E_2} \\
\overline{\neg(E_1 \leq E_2)} & \triangleq & \overline{E_1 > E_2} \\
\overline{\neg(E_1 = E_2)} & \triangleq & \overline{E_1 <> E_2} \\
\overline{\neg(E_1 <> E_2)} & \triangleq & \overline{E_1 = E_2} \\
\overline{\neg(E_1 > E_2)} & \triangleq & \overline{E_1 \leq E_2} \\
\overline{\neg(E_1 \geq E_2)} & \triangleq & \overline{E_1 < E_2} \\
\overline{\neg(B_1 \vee B_2)} & \triangleq & \overline{\neg B_1} \wedge \overline{\neg B_2} \\
\overline{\neg(B_1 \wedge B_2)} & \triangleq & \overline{\neg B_1} \vee \overline{\neg B_2} \\
\overline{\neg(\neg B)} & \triangleq & \overline{B}
\end{array}
$$

An Example Domain

Interval domain

$\wp(\mathbf{Int}) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}}$



An AI Example

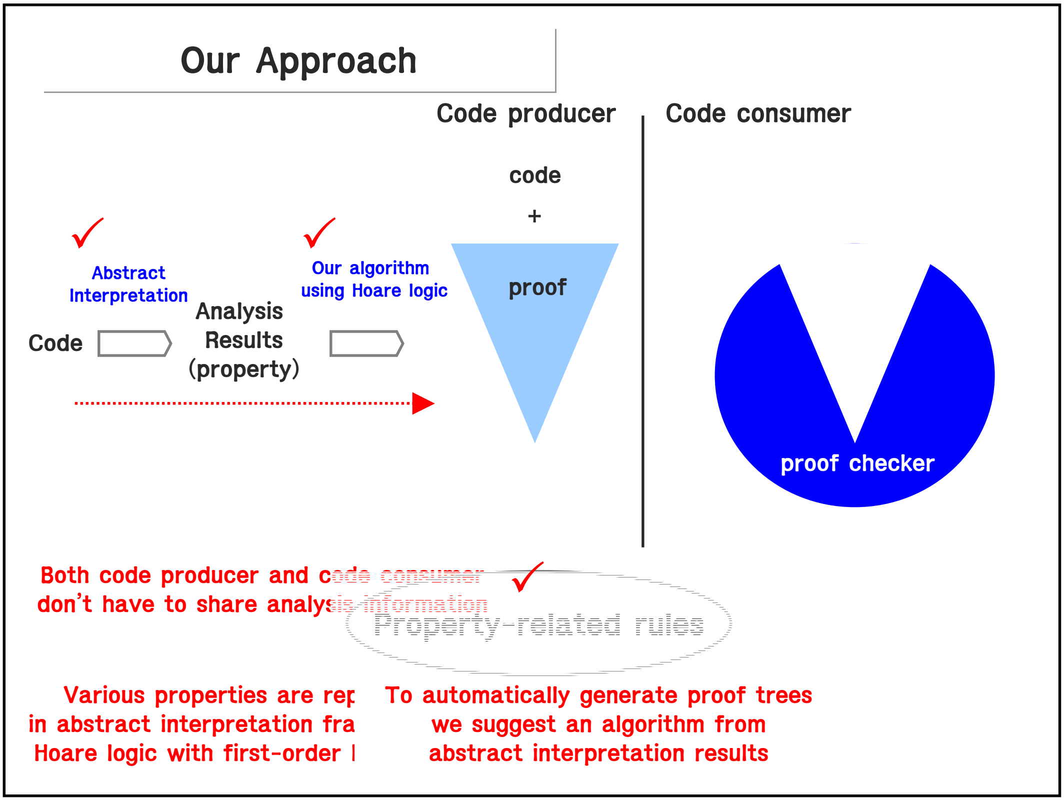
프로그램

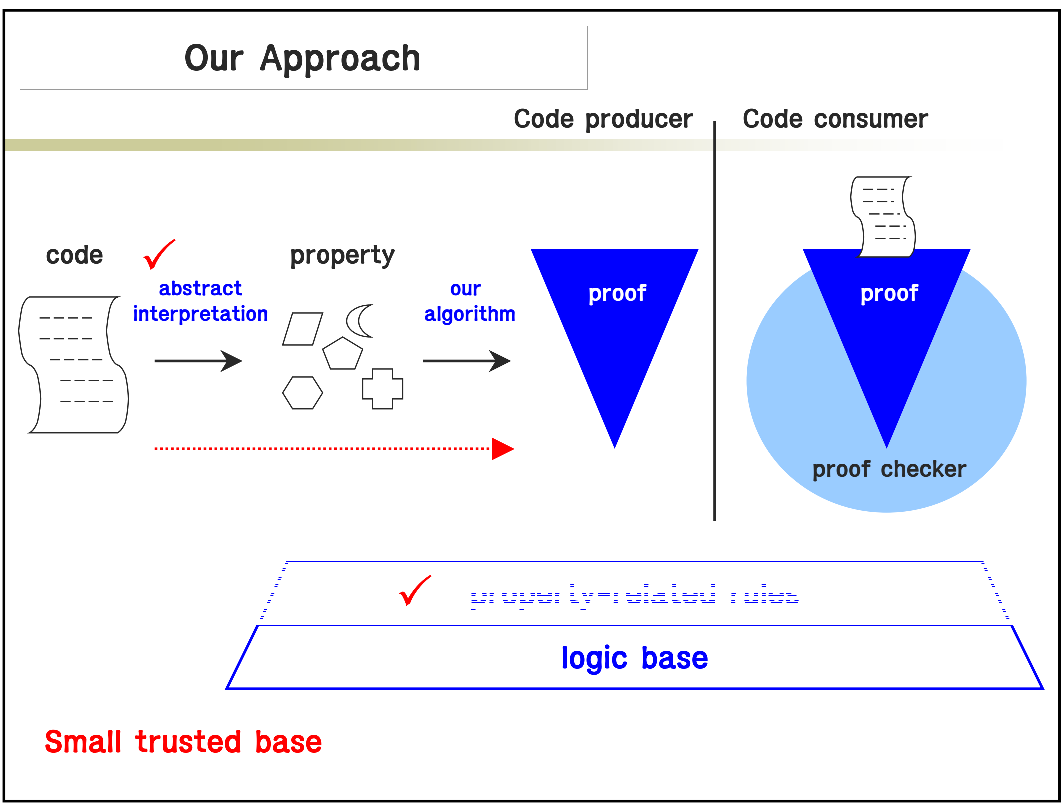
```
i:=0; x:=0
while i<3 do
    if i=0
      then x:=1
      else x:=4
   fi;
   i:=i+2
od
```

환경

$\hat{E} : \{i \mapsto \mathsf{ui}, x \mapsto \mathsf{ui}\}$
$\hat{E} : \{i \mapsto [0,0], x \mapsto [0,0]\}$
$\hat{E} : \{i \mapsto [0,2], x \mapsto [0,4]\}$

$\hat{E} : \{i \mapsto [0,0], x \mapsto [1,1]\}$
$\hat{E} : \{i \mapsto [1,2], x \mapsto [4,4]\}$
$\hat{E} : \{i \mapsto [0,2], x \mapsto [1,4]\}$
$\hat{E} : \{i \mapsto [2,4], x \mapsto [1,4]\}$
$\hat{E} : \{i \mapsto [3,4], x \mapsto [0,4]\}$


Our Approach
Code producer
Code consumer
code
+
proof
Abstract Interpretation
Our algorithm using Hoare logic
Code
Analysis Results (property)
proof checker
Both code producer and code consumer
don't have to share analysis information
Property-related rules
Various properties are rep
in abstract interpretation fra
Hoare logic with first-order l
To automatically generate proof trees
we suggest an algorithm from
abstract interpretation results


Our Approach
Code producer
Code consumer
code
abstract interpretation
property
our algorithm
proof
proof
proof checker
property-related rules
logic base
Small trusted base

$$[\![E_1 + E_2]\!]\sigma \;=\; ([\![E_1]\!]\sigma)\hat{+}([\![E_2]\!]\sigma)$$

$$\dfrac{\vdots}{\mathsf{tr}_{state}(\sigma) \Rightarrow (\mathsf{tr}([\![x]\!]\sigma, x)) \wedge (\mathsf{tr}([\![y]\!]\sigma, y))} \qquad \dfrac{[\mathbf{AP}]}{(\mathsf{tr}([\![x]\!]\sigma, x)) \wedge (\mathsf{tr}([\![y]\!]\sigma, y)) \Rightarrow \mathsf{tr}(([\![x]\!]\sigma)\hat{+}([\![y]\!]\sigma), x + y)}$$

$$\dfrac{\dfrac{}{\begin{array}{l}\mathsf{tr}_{state}(\sigma) \\ \Rightarrow \mathsf{tr}_{state}(\sigma\backslash_x) \wedge \mathsf{tr}([\![x+y]\!]\sigma, x+y)\end{array}} \qquad \dfrac{[\mathrm{IIA}]}{\left\{\begin{array}{l}\mathsf{tr}_{state}(\sigma\backslash_x) \\ \wedge\mathsf{tr}([\![x+y]\!]\sigma, x+y)\end{array}\right\} \mathtt{x} := \mathtt{x} + \mathtt{y} \left\{\begin{array}{l}\mathsf{tr}_{state}(\sigma\backslash_x) \\ \wedge\mathsf{tr}([\![x+y]\!]\sigma, x)\end{array}\right\}}}{\{\mathsf{tr}_{state}(\sigma)\}\ \mathtt{x} := \mathtt{x} + \mathtt{y}\{\mathsf{tr}_{state}(\sigma\backslash_x) \wedge \mathsf{tr}([\![x+y]\!]\sigma, x)\}}$$

- $\mathtt{ifS} \triangleq \mathtt{if\ x + y = 0\ then\ x := x + y\ else\ x := 0\ fi}$
- $\mathsf{tr}_{state}(\sigma) \triangleq \bigwedge_{x \in domain(\sigma)} \mathsf{tr}(\sigma(x), x)$

Translation function tr(*d*,*E*) (2/2)

- **Required properties**

  γ's property

  - **Monotonicity :** **if** $a \sqsubseteq a'$, **then** $\mathrm{tr}(a,E) \Rightarrow \mathrm{tr}(a',E)$
  - **Meet Preservation :** $\mathrm{tr}(a \sqcap a', E) = \mathrm{tr}(a,E) \wedge \mathrm{tr}(a', E)$
  - **Strictness :** $\mathrm{tr}(\perp, E) = \mathrm{ff}$
  - **Constants Preservation :** $\mathrm{tr}(\alpha(\{n\}), n)$ **holds**
  - **Closedness :** $\mathrm{Free}\ (\mathrm{tr}(a, E)) = \mathrm{Free}\ (E)$
  - **Commutativity :** $\mathrm{tr}(a, E)[E'/x] = \mathrm{tr}(a, E[E'/x])$

  a hole with E

Proof Construction $\mathcal{T}([s]C[s'])$ (cont.)

Case $A \equiv [\hat{s}]([\mathsf{inv}\ \hat{s}_0]\texttt{while}\ B\ \mathsf{do}\ [\hat{s}_1]R_1[\hat{s}'_1]\ \mathsf{od})[\hat{s}']$

$$\dfrac{\mathsf{monSt}(\hat{s},\hat{s}_0) \qquad \dfrac{\dfrac{\mathcal{B}_b(\hat{s}_0, B) \qquad \mathcal{T}([\hat{s}_1]R_1[\hat{s}'_1]) \qquad \mathsf{monSt}(\hat{s}'_1,\hat{s}_0)}{\{\mathsf{trst}(\hat{s}_0) \wedge B\}\ \overline{R_1}\ \{\mathsf{trst}(\hat{s}_0)\}}}{\{\mathsf{trst}(\hat{s}_0)\}\ \texttt{while}\ B\ \mathsf{do}\ \overline{R_1}\ \mathsf{od}\ \{\mathsf{trst}(\hat{s}_0) \wedge \neg B\}} \qquad \mathcal{B}_b(\hat{s}_0, \neg B)}{\{\mathsf{trst}(\hat{s})\}\ \texttt{while}\ B\ \mathsf{do}\ \overline{R_1}\ \mathsf{od}\ \{\mathsf{trst}(\hat{s}')\}}$$

Case $A \equiv [\hat{s}]A_1; A_2[\hat{s}']$

$$\dfrac{\mathcal{T}(A_1) \qquad \mathcal{T}(A_2)}{\{\mathsf{trst}(\hat{s})\}\ \overline{A_1}; \overline{A_2}\ \{\mathsf{trst}(\hat{s}')\}}$$