Verification Condition Generation for Conditional Information Flow

Torben Amtoft Anindya Banerjee

Kansas State University

FMSE, November 2, 2007

Objectives

- Specify and implement information flow analysis for sequential OO-programs.
- Integrate with programmer assertions in style of JML
- Precision:
  - flow-sensitive
  - model also conditional flows

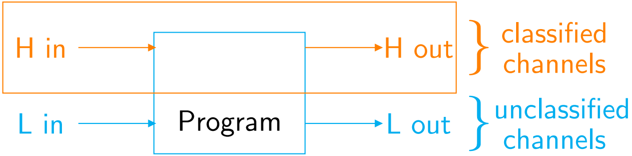
Approach

- Hoare-like assertions, computing (weakest) preconditions
- Object invariants, cf. Boogie methodology (no expensive alias analysis)

Information Flow Regulates Confidentiality

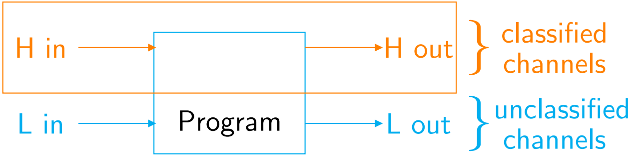
- Data is secret (High) or public/observable (Low).
- Confidentiality: High inputs do not influence Low output channels (End-to-end property)
- Typical analyses based on security types, e.g., (**int**, H), (**com**, L);
  - Flow insensitive [Volpano/Smith/Irvine,Myers,. . .]
  - Flow sensitive [Hunt/Sands]

Noninterference



Noninterference property [Goguen-Meseguer]: For any two runs of program, Low-indistinguishable input states yield Low-indistinguishable output states.

Noninterference



Noninterference property [Goguen-Meseguer]: For any two runs of program, Low-indistinguishable input states yield Low-indistinguishable output states.
Equivalently [Cohen]: L out independent of initial H in.

Logical Approach

Consider (Hoare-style) triple [Amtoft/Banerjee SAS'04]

$$\{x_1 \ltimes, \ldots, x_n \ltimes\}\ P\ \{y_1 \ltimes, \ldots, y_m \ltimes\}$$

Meaning: given any two runs of P:

- If observable inputs $x_1, \ldots, x_n$ agree (precondition)
- Then observable outputs $y_1, \ldots, y_m$ agree in the same two runs (postcondition).

Special case: $x_1, \ldots, x_n$ and $y_1, \ldots y_m$ are the low variables.

Leveraging Standard Assertions


```
if w
then x := 7
else x := 7
```

$\{x \ltimes\}$

Leveraging Standard Assertions

$\{w \ltimes\}$ naive rules need w to be low

```
if w
then x := 7
else x := 7
```

$\{x \ltimes\}$

Leveraging Standard Assertions

$\{\}$ no assumptions about w

`if` w

`then` $x := 7$

`else` $x := 7$

$\{\}$

`assert`$(x = 7)$

$\{x \ltimes\}$

Leveraging Standard Assertions

$\{\}$ **no** assumptions about w

```
if w
then x := 7
else x := 7
```

$\{\}$

assert$(x = 7)$

$\{x \ltimes\}$

Integrate with assertion checker (ESC/Java2, BLAST)

Heap Manipulation in Hoare-Like Logics

Recall rule for variable assignment:

$$\{\phi[E/x]\}\ x := E\ \{\phi\}$$

For field update, we could try

$$\{\phi[E/x.f]\}\ x.f := E\ \{\phi\}$$

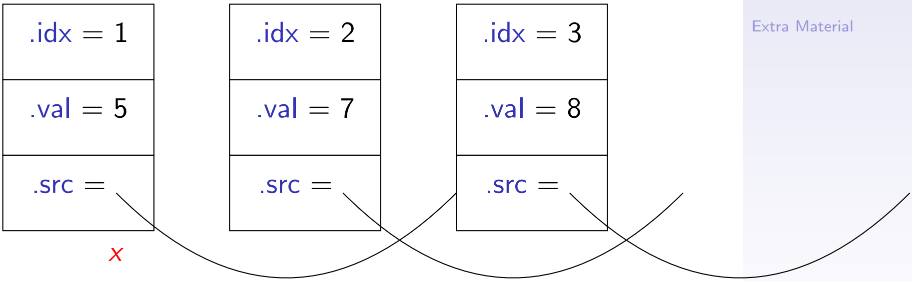
so that for example

$$\{w = 7 \wedge y.f = 5\}\ x.f := w\ \{x.f = 7 \wedge y.f = 5\}$$

but this is incorrect if x and y alias.
(Above is main motivation for separation logic.)

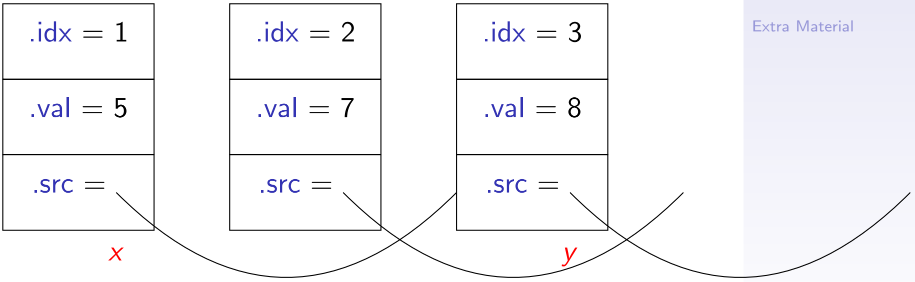
Example Setting

Motivated by an actual program, provided by Rockwell-Collins, used in hardware verification of operational amplifiers.



Example Setting

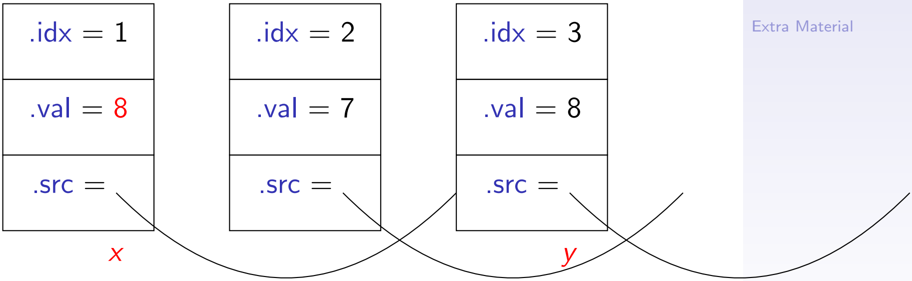
$y := x.src;$



Example Setting

$y := x.src;$

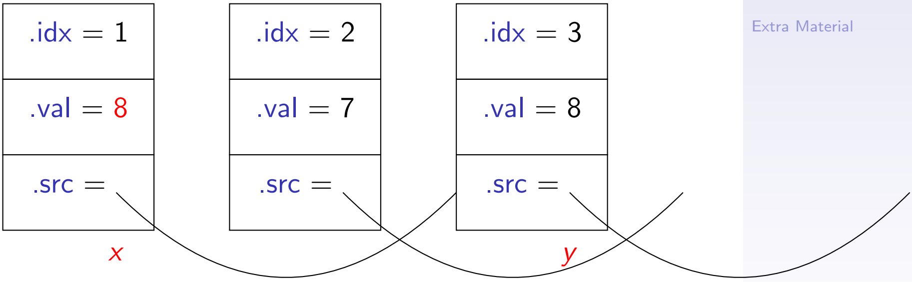
$t := y.val; \quad x.val := t$



Example Setting

$y := x.src;$
$t := y.val; \quad x.val := t$
$result := x.val$

$result = 8$



Object Flow Invariant

Overall policy: odd elements should be public

$$y := x.src; \quad i := x.idx$$

$$t := y.val; \quad x.val := t$$
$$result := x.val$$
$$\{odd(i) \Rightarrow result \ltimes\}$$

Object Flow Invariant

Overall policy: odd elements should be public

$$y := x.src; \quad i := x.idx$$

$$t := y.val; \quad x.val := t$$
$$result := x.val$$

$\{odd(i) \Rightarrow result \ltimes\}$

Object Flow Invariant (holds for object in steady state)

$$\{odd(o.idx) \Rightarrow o.val \ltimes\}$$
$$\{odd(o.idx) \Rightarrow o.src \ltimes\}$$

Object Flow Invariant

Overall policy: odd elements should be public

$$y := x.src; \quad i := x.idx$$
$$\texttt{assert}(odd(i) \rightarrow odd(y.idx))$$
$$t := y.val; \quad x.val := t$$
$$result := x.val$$
$$\{odd(i) \Rightarrow result \ltimes\}$$

Object Flow Invariant (holds for object in steady state)

$$\{odd(o.idx) \Rightarrow o.val \ltimes\}$$
$$\{odd(o.idx) \Rightarrow o.src \ltimes\}$$

Intuition: to update odd elements, only use odd elements

Scoping

- All objects are manipulated within scopes.
- Each scope must maintain the object invariant (cf. **pack/unpack** in Boogie)
- Then aliasing issue become irrelevant.

Scoping

- All objects are manipulated within scopes.
- Each scope must maintain the object invariant (cf. **pack/unpack** in Boogie)
- Then aliasing issue become irrelevant.

$y := x.src;$
$i := x.idx$

$\mathtt{open}\ x\ \{$
$\quad y := .src;\ i := .idx\ \}$

Scoping

- All objects are manipulated within scopes.
- Each scope must maintain the object invariant (cf. **pack/unpack** in Boogie)
- Then aliasing issue become irrelevant.

$y := x.src;$
$i := x.idx$
$\texttt{assert}(odd(i))$
$\quad \rightarrow odd(y.idx))$
$t := y.val;$

$\texttt{open } x \ \{$
$\quad y := .src; \ i := .idx \ \}$
$\texttt{open } y \ \{$
$\quad \texttt{assert}(odd(i) \rightarrow odd(.idx));$
$\quad t := .val \ \}$

Scoping

- All objects are manipulated within scopes.
- Each scope must maintain the object invariant (cf. **pack/unpack** in Boogie)
- Then aliasing issue become irrelevant.

$y := x.src;$
$i := x.idx$
$\texttt{assert}(odd(i))$
$\quad \rightarrow odd(y.idx))$
$t := y.val;$
$x.val := t$
$result := x.val$

$\texttt{open}\ x\ \{$
$\quad y := .src;\ \ i := .idx\ \}$
$\texttt{open}\ y\ \{$
$\quad \texttt{assert}(odd(i) \rightarrow odd(.idx));$
$\quad t := .val\ \}$
$\texttt{open}\ x\ \{$
$\quad .val := t; \quad result := .val\ \}$

Scoping

- All objects are manipulated within scopes.
- Each scope must maintain the object invariant (cf. **pack/unpack** in Boogie)
- Then aliasing issue become irrelevant.

$$
\begin{array}{l}
y := x.src; \\
i := x.idx \\
\texttt{assert}(odd(i)) \\
\quad \rightarrow odd(y.idx)) \\
t := y.val; \\
x.val := t \\
result := x.val
\end{array}
$$

$$
\begin{array}{l}
\texttt{open}\ x\ \{ \\
\quad y := .src;\ \ i := .idx\ \} \\
\texttt{open}\ y\ \{ \\
\quad \texttt{assert}(odd(i) \rightarrow odd(.idx)); \\
\quad t := .val\ \} \\
\texttt{open}\ x\ \{ \\
\quad \texttt{assert}(.idx = i) \\
\quad .val := t; \quad result := .val\ \}
\end{array}
$$

Propagating Assertions

`open` x

`assert` $.idx = i$

$.val := t$

$result := .val$

`close` x

Propagating Assertions

Object invariant:
$odd(.idx) \Rightarrow .val \ltimes, \; odd(.idx) \Rightarrow .src \ltimes$

$\texttt{open}\ x$

$\texttt{assert}\ .idx = i$

$.val := t$

$result := .val$

$\texttt{close}\ x$
$odd(i) \Rightarrow result \ltimes$

Propagating Assertions

Object invariant:

$odd(.idx) \Rightarrow .val \ltimes, \; odd(.idx) \Rightarrow .src \ltimes$

open x

assert $.idx = i$

$.val := t$

$result := .val$

$odd(i) \Rightarrow result \ltimes, \; odd(.idx) \Rightarrow .val \ltimes, \; odd(.idx) \Rightarrow .src \ltimes$

close x

$odd(i) \Rightarrow result \ltimes$

Propagating Assertions

Object invariant:

$odd(.idx) \Rightarrow .val \ltimes,\ odd(.idx) \Rightarrow .src \ltimes$

open x

assert $.idx = i$

$.val := t$

$odd(i) \Rightarrow .val \ltimes,\ odd(.idx) \Rightarrow .val \ltimes,\ odd(.idx) \Rightarrow .src \ltimes$

$result := .val$

$odd(i) \Rightarrow result \ltimes,\ odd(.idx) \Rightarrow .val \ltimes,\ odd(.idx) \Rightarrow .src \ltimes$

close x

$odd(i) \Rightarrow result \ltimes$

Propagating Assertions

Object invariant:

$odd(.idx) \Rightarrow .val \ltimes,\ odd(.idx) \Rightarrow .src \ltimes$

open x

assert $.idx = i$

$odd(i) \Rightarrow t \ltimes,\ odd(.idx) \Rightarrow t \ltimes,\ odd(.idx) \Rightarrow .src \ltimes$

$.val := t$

$odd(i) \Rightarrow .val \ltimes,\ odd(.idx) \Rightarrow .val \ltimes,\ odd(.idx) \Rightarrow .src \ltimes$

$result := .val$

$odd(i) \Rightarrow result \ltimes,\ odd(.idx) \Rightarrow .val \ltimes,\ odd(.idx) \Rightarrow .src \ltimes$

close x

$odd(i) \Rightarrow result \ltimes$

Propagating Assertions

Object invariant:

$odd(.idx) \Rightarrow .val \ltimes,\ odd(.idx) \Rightarrow .src \ltimes$

$\quad \texttt{open}\ x$

$odd(i) \wedge .idx = i \Rightarrow t \ltimes,\ odd(.idx) \wedge .idx = i \Rightarrow .src \ltimes$

$\qquad \texttt{assert}\ .idx = i$

$odd(i) \Rightarrow t \ltimes,\ odd(.idx) \Rightarrow t \ltimes,\ odd(.idx) \Rightarrow .src \ltimes$

$\qquad .val := t$

$odd(i) \Rightarrow .val \ltimes,\ odd(.idx) \Rightarrow .val \ltimes,\ odd(.idx) \Rightarrow .src \ltimes$

$\qquad result := .val$

$odd(i) \Rightarrow result \ltimes,\ odd(.idx) \Rightarrow .val \ltimes,\ odd(.idx) \Rightarrow .src \ltimes$

$\quad \texttt{close}\ x$

$odd(i) \Rightarrow result \ltimes$

Propagating Assertions

Object invariant:

$odd(.idx) \Rightarrow .val \ltimes,\ odd(.idx) \Rightarrow .src \ltimes$

$true \Rightarrow x \ltimes,\ odd(i) \Rightarrow t \ltimes$

open x

$odd(i) \wedge .idx = i \Rightarrow t \ltimes,\ odd(.idx) \wedge .idx = i \Rightarrow .src \ltimes$

assert $.idx = i$

$odd(i) \Rightarrow t \ltimes,\ odd(.idx) \Rightarrow t \ltimes,\ odd(.idx) \Rightarrow .src \ltimes$

$.val := t$

$odd(i) \Rightarrow .val \ltimes,\ odd(.idx) \Rightarrow .val \ltimes,\ odd(.idx) \Rightarrow .src \ltimes$

$result := .val$

$odd(i) \Rightarrow result \ltimes,\ odd(.idx) \Rightarrow .val \ltimes,\ odd(.idx) \Rightarrow .src \ltimes$

close x

$odd(i) \Rightarrow result \ltimes$

Propagating Assertions

Object invariant:

$odd(.idx) \Rightarrow .val \ltimes, \; odd(.idx) \Rightarrow .src \ltimes$

$true \Rightarrow x \ltimes$

$\quad\quad \ldots$

$true \Rightarrow x \ltimes, \; odd(i) \Rightarrow t \ltimes$

$\quad$ open x

$odd(i) \wedge .idx = i \Rightarrow t \ltimes, \; odd(.idx) \wedge .idx = i \Rightarrow .src \ltimes$

$\quad\quad$ assert $.idx = i$

$odd(i) \Rightarrow t \ltimes, \; odd(.idx) \Rightarrow t \ltimes, \; odd(.idx) \Rightarrow .src \ltimes$

$\quad\quad .val := t$

$odd(i) \Rightarrow .val \ltimes, \; odd(.idx) \Rightarrow .val \ltimes, \; odd(.idx) \Rightarrow .src \ltimes$

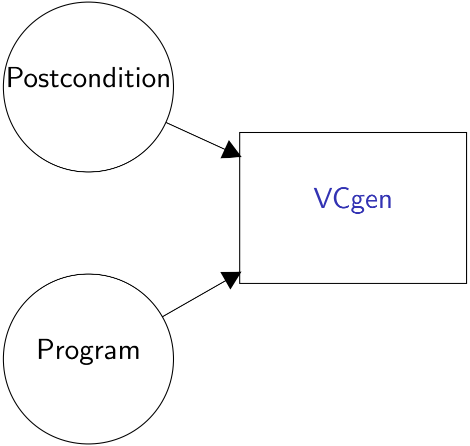
$\quad\quad result := .val$

$odd(i) \Rightarrow result \ltimes, \; odd(.idx) \Rightarrow .val \ltimes, \; odd(.idx) \Rightarrow .src \ltimes$

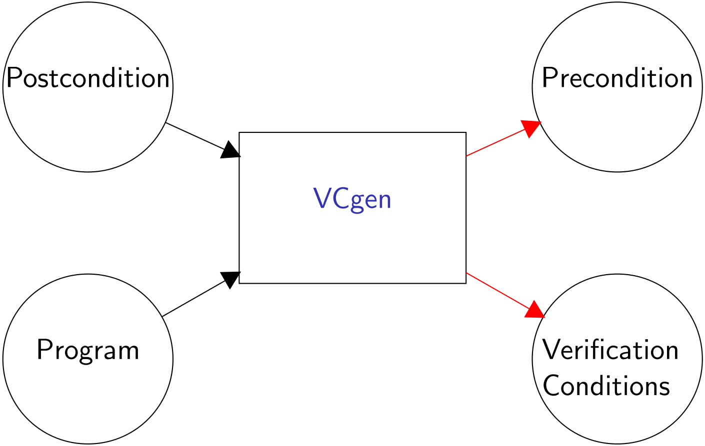
$\quad$ close x

$odd(i) \Rightarrow result \ltimes$

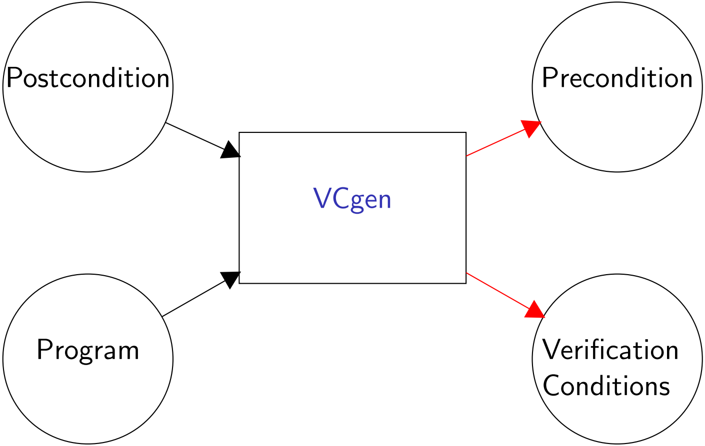
The Algorithm VCgen



The Algorithm VCgen

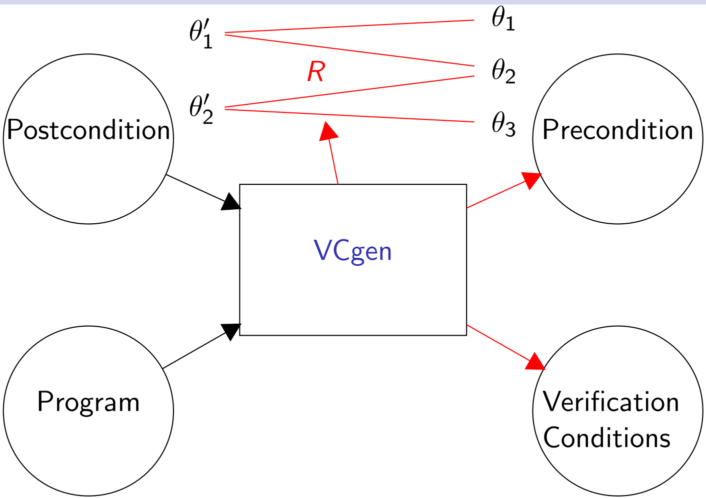


The Algorithm VCgen



Always terminates, given loop & object invariants, but VC may fail (if invariants not strong enough)

The Algorithm VCgen



Always terminates, given loop & object invariants, but VC may fail (if invariants not strong enough)

Syntax & Semantics

$$
\begin{array}{lll}
\mathit{RS} & ::= & \texttt{skip} \\
 & \mid & \texttt{assert}(\phi) \\
 & \mid & \mathit{RS}\ ;\ \mathit{RS} \\
 & \mid & \texttt{if}\ B\ \texttt{then}\ \mathit{RS} \\
 & & \quad \texttt{else}\ \mathit{RS} \\
 & \mid & \texttt{while}\ B\ \texttt{do}\ \mathit{RS} \\
 & \mid & x := A \\
 & \mid & .f := A
\end{array}
$$

$$
\begin{array}{lll}
\mathit{TS} & ::= & \texttt{skip} \\
 & \mid & \texttt{assert}(\phi) \\
 & \mid & \mathit{TS}\ ;\ \mathit{TS} \\
 & \mid & \texttt{if}\ B\ \texttt{then}\ \mathit{TS}\ \mathit{TS} \\
 & & \quad \texttt{else}\ \mathit{TS} \\
 & \mid & \texttt{while}\ B\ \texttt{do}\ \mathit{TS} \\
 & \mid & x := A \\
 & & \\
 & \mid & \texttt{new}\ x\ \mathit{RS} \\
 & \mid & \texttt{open}\ x\ \mathit{RS}
\end{array}
$$

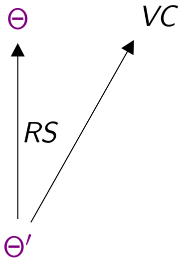
Syntax & Semantics

$$
\begin{array}{rcl}
RS & ::= & \texttt{skip} \\
& \mid & \texttt{assert}(\phi) \\
& \mid & RS\ ;\ RS \\
& \mid & \texttt{if}\ B\ \texttt{then}\ RS \\
& & \quad \texttt{else}\ RS \\
& \mid & \texttt{while}\ B\ \texttt{do}\ RS \\
& \mid & x := A \\
& \mid & .f := A
\end{array}
\qquad
\begin{array}{rcl}
TS & ::= & \texttt{skip} \\
& \mid & \texttt{assert}(\phi) \\
& \mid & TS\ ;\ TS \\
& \mid & \texttt{if}\ B\ \texttt{then}\ TS\ TS \\
& & \quad \texttt{else}\ TS \\
& \mid & \texttt{while}\ B\ \texttt{do}\ TS \\
& \mid & x := A \\
& & \\
& \mid & \texttt{new}\ x\ RS \\
& \mid & \texttt{open}\ x\ RS
\end{array}
$$

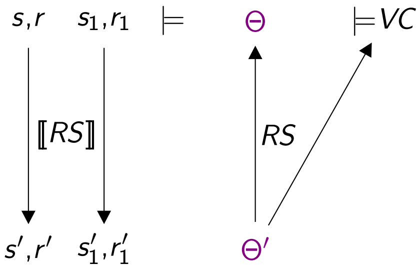
$$s,r\ [\![RS]\!]\ s',r' \qquad\qquad s,h\ [\![TS]\!]\ s',h'$$

s: store, r: object state (maps fields to values), h: heap

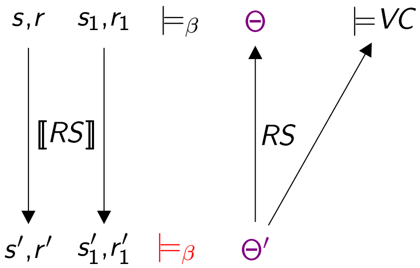
Correctness Properties



Correctness Properties

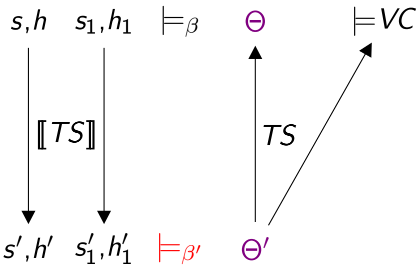


Correctness Properties



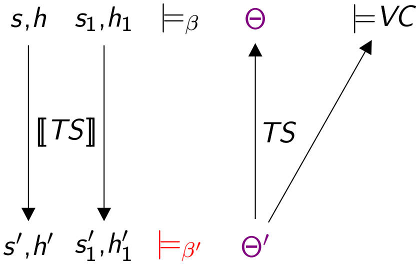
- For RS, the heap stays the same

Correctness Properties



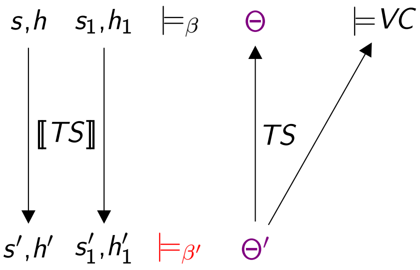
- For *RS*, the heap stays the same
- For *TS*, the heap may be augmented

Correctness Properties



- For *RS*, the heap stays the same
- For *TS*, the heap may be augmented
- This is termination insensitive

Correctness Properties



- For *RS*, the heap stays the same
- For *TS*, the heap may be augmented
- This is termination insensitive
- Auxiliary lemmas tell about *R* component
  - write confinement, aka "* property"

Assignments

Θ

R $\quad x := y + z$

Θ' $\quad x > 7 \Rightarrow w \ltimes \quad w > 5 \Rightarrow x \ltimes$

$[VC]\{\Theta\}\ (R) \Longleftarrow x := A\ \{\Theta'\}$
iff

Assignments

Θ $y + z > 7 \Rightarrow w \ltimes$ $w > 5 \Rightarrow (y + z) \ltimes$

R ↑ $x := y + z$ ↑

Θ' $x > 7 \Rightarrow w \ltimes$ $w > 5 \Rightarrow x \ltimes$

$$[VC]\{\Theta\}\ (R) \Longleftarrow x := A\ \{\Theta'\}$$
$$\text{iff} \quad R = \{(\phi[A/x] \Rightarrow E[A/x] \ltimes, \quad \phi \Rightarrow E \ltimes) \mid (\phi \Rightarrow E \ltimes) \in \Theta'$$

and $\Theta = dom(R)$ and $VC = \emptyset$

Assignments

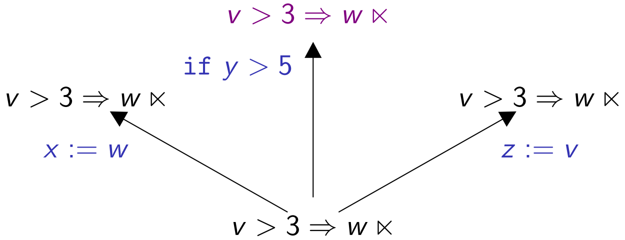
Θ $\quad y + z > 7 \Rightarrow w \ltimes \qquad w > 5 \Rightarrow (y + z) \ltimes$

R $\qquad \uparrow \qquad x := y + z \qquad \uparrow$

Θ' $\quad x > 7 \Rightarrow w \ltimes \qquad w > 5 \Rightarrow x \ltimes$

$$
\begin{aligned}
&[VC]\{\Theta\}\ (R) \Longleftarrow x := A\ \{\Theta'\} \\
&\quad \text{iff} \quad R = \{(\phi[A/x] \Rightarrow E[A/x] \ltimes, \gamma, \phi \Rightarrow E \ltimes) \\
&\qquad\qquad\qquad \mid (\phi \Rightarrow E \ltimes) \in \Theta', \\
&\qquad\qquad\qquad\quad \gamma = m \text{ iff } x \in fv(E)) \\
&\quad \text{and } \Theta = dom(R) \text{ and } VC = \emptyset
\end{aligned}
$$

Conditionals



$$[VC]\{\Theta\}\ (R) \Longleftarrow \texttt{if}\ B\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \{\Theta'\}$$

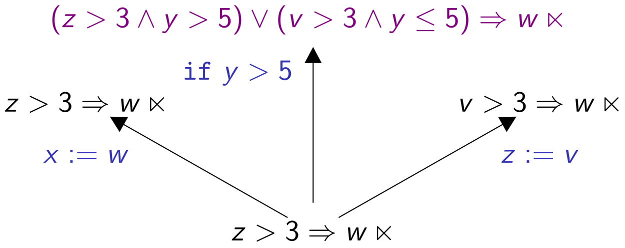
iff $[VC_1]\{\Theta_1\}\ (R_1) \Longleftarrow S_1\ \{\Theta'\}$

and $[VC_2]\{\Theta_2\}\ (R_2) \Longleftarrow S_2\ \{\Theta'\}$ and $VC = VC_1 \cup VC_2$

and $R = R_0 \cup R_1' \cup R_2' \cup R_0'$ and $\Theta = dom(R)$

where $R_0 = \{(\phi \Rightarrow E \ltimes, u, \theta') \mid (\phi \Rightarrow E \ltimes, u, \theta') \in R_1, (\phi \Rightarrow E \ltimes, u, \theta') \in R_2$

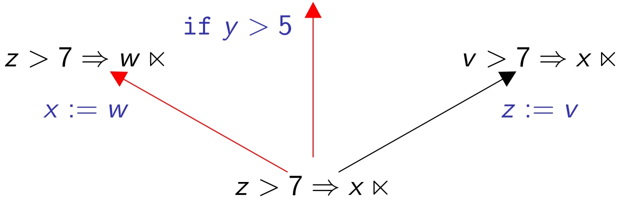
Conditionals



$$
\begin{array}{l}
[VC]\{\Theta\}\ (R) \Longleftarrow \texttt{if}\ B\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \{\Theta'\} \\
\quad \text{iff} \quad [VC_1]\{\Theta_1\}\ (R_1) \Longleftarrow S_1\ \{\Theta'\} \\
\quad \text{and}\ [VC_2]\{\Theta_2\}\ (R_2) \Longleftarrow S_2\ \{\Theta'\}\ \text{ and }\ VC = VC_1 \cup VC_2 \\
\quad \text{and}\ R = R_0 \cup R_1' \cup R_2' \cup R_0'\ \text{ and }\ \Theta = dom(R) \\
\quad \text{where}\ R_0 = \{(((\phi_1 \wedge B) \vee (\phi_2 \wedge \neg B)) \Rightarrow E \ltimes, u, \theta') \\
\qquad\qquad\qquad \mid\ (\phi_1 \Rightarrow E \ltimes, u, \theta') \in R_1, \\
\qquad\qquad\qquad\quad\ (\phi_2 \Rightarrow E \ltimes, u, \theta') \in R_2
\end{array}
$$

Conditionals

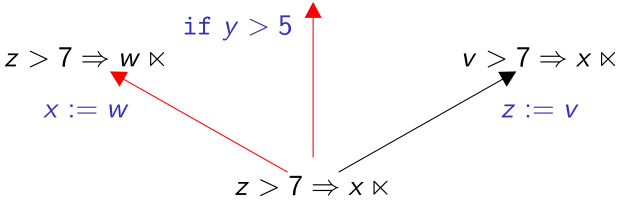
$z > 7 \wedge y > 5 \Rightarrow w \ltimes$



$$
\begin{array}{l}
[VC]\{\Theta\}\ (R) \Longleftarrow \texttt{if}\ B\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \{\Theta'\} \\
\quad \text{iff} \quad [VC_1]\{\Theta_1\}\ (R_1) \Longleftarrow S_1\ \{\Theta'\} \\
\quad \text{and}\ [VC_2]\{\Theta_2\}\ (R_2) \Longleftarrow S_2\ \{\Theta'\}\ \text{ and } VC = VC_1 \cup VC_2 \\
\quad \text{and}\ R = R_0 \cup R'_1 \cup R'_2 \cup R'_0 \text{ and } \Theta = dom(R) \\
\quad \text{where } R'_1 = \{((\phi_1 \wedge B) \Rightarrow E_1 \ltimes, m, \theta') \\
\qquad\qquad\qquad \mid\ (\phi_1 \Rightarrow E_1 \ltimes, \gamma, \theta') \in R_1, \\
\qquad\qquad\qquad\quad \gamma = m \text{ or } (\_, m, \theta') \in R_2
\end{array}
$$

Conditionals

$z > 7 \wedge y > 5 \Rightarrow w \ltimes \qquad v > 7 \wedge y \leq 5 \Rightarrow x \ltimes$



$$[VC]\{\Theta\}\ (R) \Longleftarrow \texttt{if}\ B\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \{\Theta'\}$$

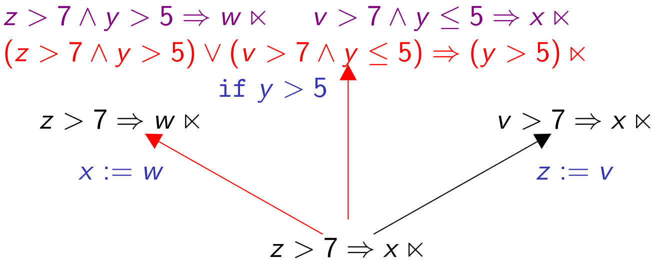
iff $[VC_1]\{\Theta_1\}\ (R_1) \Longleftarrow S_1\ \{\Theta'\}$

and $[VC_2]\{\Theta_2\}\ (R_2) \Longleftarrow S_2\ \{\Theta'\}$ and $VC = VC_1 \cup VC_2$

and $R = R_0 \cup R_1' \cup R_2' \cup R_0'$ and $\Theta = dom(R)$

where $R_2' = \{((\phi_2 \wedge \neg B) \Rightarrow E_2 \ltimes, m, \theta') \mid (\phi_2 \Rightarrow E_2 \ltimes, \gamma, \theta') \in R_2, \gamma = m \text{ or } (\_, m, \theta') \in R_1$

Conditionals



$[VC]\{\Theta\}\ (R) \Longleftarrow \texttt{if}\ B\ \texttt{then}\ S_1\ \texttt{else}\ S_2\ \{\Theta'\}$

iff $[VC_1]\{\Theta_1\}\ (R_1) \Longleftarrow S_1\ \{\Theta'\}$

and $[VC_2]\{\Theta_2\}\ (R_2) \Longleftarrow S_2\ \{\Theta'\}$ and $VC = VC_1 \cup VC_2$

and $R = R_0 \cup R_1' \cup R_2' \cup R_0'$ and $\Theta = dom(R)$

where
$$R_0' = \{(((\phi_1 \wedge B) \vee (\phi_2 \wedge \neg B)) \Rightarrow B \ltimes, m, \theta') \mid (\phi_1 \Rightarrow E_1 \ltimes, \gamma_1, \theta') \in R_1, (\phi_2 \Rightarrow E_2 \ltimes, \gamma_2, \theta') \in R_2, \gamma_1 = m \text{ or } \gamma_2 = m$$

Simplification

Needs to apply rules like

$$\begin{aligned}
\phi \Rightarrow (x + w) \ltimes &\quad\Longrightarrow\quad \phi \Rightarrow x \ltimes,\ \phi \Rightarrow w \ \ltimes \\
(x = 1 \wedge x \neq 1) \Rightarrow w \ltimes &\quad\Longrightarrow\quad \mathbf{T} \Rightarrow 0 \ \ltimes \\
x = 1 \Rightarrow x \ltimes &\quad\Longrightarrow\quad \mathbf{T} \Rightarrow 0 \ltimes
\end{aligned}$$

Ongoing Work

- implementation (Jonathan Hoag)
- compute loop invariants
- interprocedural (given method summaries)
- array manipulation
- language for conditional information flow (extending SPARK)
- conditional information flow for state chart language

Previous Work

Logic can be augmented by points-to assertions $x \rightsquigarrow L$ [Amtoft/Bandhakavi/Banerjee POPL'06]

- If also $y \rightsquigarrow L_1$ with L and L_1 disjoint, then x and y cannot alias.
- semantically sound
- sound intraprocedural algorithm

Previous Work

Logic can be augmented by points-to assertions $x \rightsquigarrow L$ [Amtoft/Bandhakavi/Banerjee POPL'06]

- If also $y \rightsquigarrow L_1$ with L and L_1 disjoint, then x and y cannot alias.
- semantically sound
- sound intraprocedural algorithm

Problems:

- algorithm needs to be told the shape of the heap (might be overcome by Indus)
- does not easily integrate with programmer assertions
- logic does not capture conditional information flows

Related work

Self-composition

$\{x \ltimes\}$
$\quad P$
$\{w \ltimes\}$

is equivalent to (using primes for fresh copies)

$\{x = x'\}$
$\quad P;$
$\quad P'$
$\{w = w'\}$

which can in principle be checked by tool for standard safety analysis.

Related work

Self-composition

$\{x \ltimes\}$
$\quad P$
$\{w \ltimes\}$

is equivalent to (using primes for fresh copies)

$\{x = x'\}$
$\quad P;$
$\quad P'$
$\{w = w'\}$

which can in principle be checked by tool for standard safety analysis.

- For good results, need to combine with security static analysis [Terauchi/Aiken, SAS'05]
- To deal with heap manipulation, more machinery is needed [Naumann, ESORICS'06]

Future Work

- Integrate with automatic safety analysis tool
- compute object invariants
- compute method summaries
- compilation from structured code to state chart, also compiling information flow assertions
- applications to declassification
- concurrency

Language for Conditional Information Flow

SPARK has annotations

`derives` w_1 `from` x_1,x_2

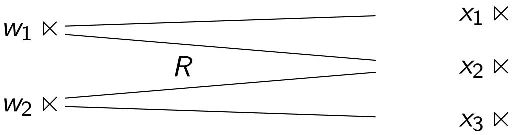
`derives` w_2 `from` x_2,x_3

Language for Conditional Information Flow

SPARK has annotations

`derives` w_1 `from` x_1,x_2

`derives` w_2 `from` x_2,x_3

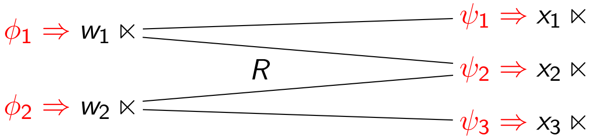


Language for Conditional Information Flow

SPARK has annotations

derives w_1 from x_1,x_2 when ϕ_1

derives w_2 from x_2,x_3 when ϕ_2



Assertions

$$[VC]\{\Theta\}\ (R) \Longleftarrow \texttt{assert}(\phi_0)\ \{\Theta'\}$$
$$\text{iff} \quad R = \{((\phi \wedge \phi_0) \Rightarrow E \ltimes, u, \phi \Rightarrow E \ltimes) \mid (\phi \Rightarrow E \ltimes) \in \Theta'$$
$$\text{and } \Theta = dom(R) \text{ and } VC = \emptyset$$

Assertions

$$[VC]\{\Theta\}\ (R) \Longleftarrow \texttt{assert}(\phi_0)\ \{\Theta'\}$$

$$\text{iff} \quad R = \{((\phi \wedge \phi_0) \Rightarrow E \ltimes, u, \phi \Rightarrow E \ltimes) \mid (\phi \Rightarrow E \ltimes) \in \Theta'$$

$$\text{and } \Theta = dom(R) \text{ and } VC = \emptyset$$

Θ $\quad y < 5 \wedge y > 7 \Rightarrow z \ltimes \quad\quad w > 5 \wedge y > 7 \Rightarrow x \ltimes$

R $\quad \uparrow \quad \texttt{assert}(y > 7) \quad \uparrow$

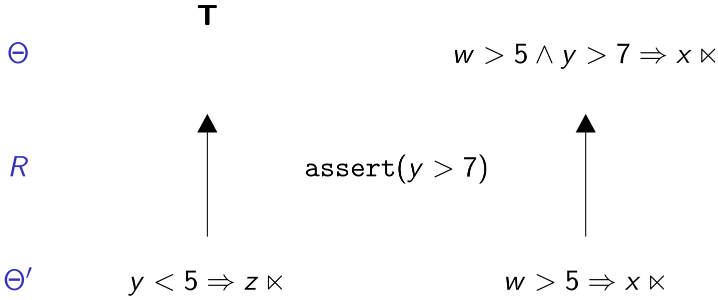
Θ' $\quad y < 5 \Rightarrow z \ltimes \quad\quad w > 5 \Rightarrow x \ltimes$

Assertions

$$[VC]\{\Theta\}\ (R) \Longleftarrow \texttt{assert}(\phi_0)\ \{\Theta'\}$$

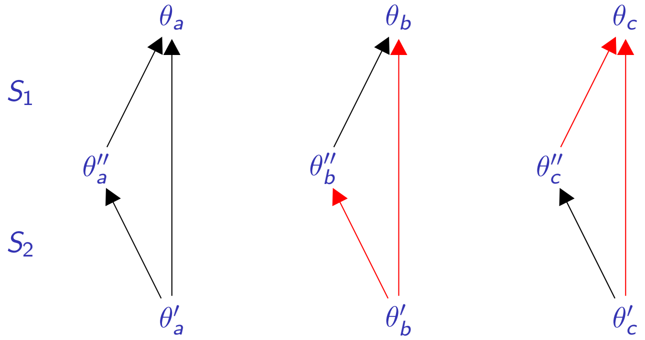
$$\text{iff} \quad R = \{((\phi \wedge \phi_0) \Rightarrow E \ltimes, u, \phi \Rightarrow E \ltimes) \mid (\phi \Rightarrow E \ltimes) \in \Theta'$$

$$\text{and } \Theta = dom(R) \text{ and } VC = \emptyset$$



Sequential Composition

$$\{\Theta\}\ (R) \Longleftarrow S_1\ ;\ S_2\ \{\Theta'\}$$
$$\text{iff}\quad \{\Theta''\}\ (R_2) \Longleftarrow S_2\ \{\Theta'\}\ \text{ and } \{\Theta\}\ (R_1) \Longleftarrow S_1\ \{\Theta''\}$$
$$\text{and}\ \ R = \{(\theta, \gamma, \theta') \mid \exists \theta'', \gamma_1, \gamma_2 : (\theta, \gamma_1, \theta'') \in R_1,\ (\theta'', \gamma_2, \theta') \in R_2$$
$$\text{where } \gamma = m \text{ iff } \gamma_1 = m \text{ or } \gamma_2 = m$$



Procedure Calls

Procedure p modifies $X = \{x\}$
Precondition for $x \ltimes$: $\{y > 0 \Rightarrow z \ltimes,\ y \leq 0 \Rightarrow w \ltimes,\ y \ltimes\}$
derives x from z when $y > 0$, from w when $y \leq 0$

$\{\Theta\}\ (R) \Longleftarrow$ **call** $p\ \{\Theta'\}$
iff $\quad R = R_u \cup R_0 \cup R_m$ and $\Theta = dom(R)$

,

Procedure Calls

Procedure p modifies $X = \{x\}$
Precondition for $x \ltimes$: $\{y > 0 \Rightarrow z \ltimes,\ y \leq 0 \Rightarrow w \ltimes,\ y \ltimes\}$
derives x from z when $y > 0$, from w when $y \leq 0$

$\{\Theta\}\ (R) \Longleftarrow$ **call** $p\ \{\Theta'\}$
iff $\quad R = R_u \cup R_0 \cup R_m$ and $\Theta = dom(R)$
where $R_u = \{(rm^{+}_{X}(\phi) \Rightarrow E \ltimes, u, \phi \Rightarrow E \ltimes)$
$\mid (\phi \Rightarrow E \ltimes) \in \Theta' \wedge fv(E) \cap X = \emptyset\}$

,

$y > 0 \Rightarrow w \ltimes$

$\uparrow$ **call** p

$x > 0 \wedge y > 0 \Rightarrow w \ltimes$

Procedure Calls

Procedure p modifies $X = \{x\}$

Precondition for $x \ltimes$: $\{y > 0 \Rightarrow z \ltimes,\ y \leq 0 \Rightarrow w \ltimes,\ y \ltimes\}$

derives x from z when $y > 0$, from w when $y \leq 0$

$\{\Theta\}\ (R) \Longleftarrow$ **call** $p\ \{\Theta'\}$

iff $\quad R = R_u \cup R_0 \cup R_m$ and $\Theta = dom(R)$

where $R_0 = \{(rm^{+}_{X}(\phi) \Rightarrow v \ltimes, m, \phi \Rightarrow E \ltimes)$

$\mid (\phi \Rightarrow E \ltimes) \in \Theta',\ v \in fv(E) \setminus X \subset fv(E)\}$

$w > 0 \Rightarrow v \ltimes,$

$y > 0 \Rightarrow w \ltimes$

call p

$x > 0 \wedge y > 0 \Rightarrow w \ltimes$

$w > 0 \Rightarrow (x + v) \ltimes$

Procedure Calls

Procedure p modifies $X = \{x\}$

Precondition for $x \ltimes$: $\{y > 0 \Rightarrow z \ltimes,\ y \leq 0 \Rightarrow w \ltimes,\ y \ltimes\}$

derives x from z when $y > 0$, from w when $y \leq 0$

$$\{\Theta\}\ (R) \Longleftarrow \textbf{call}\ p\ \{\Theta'\}$$

iff $R = R_u \cup R_0 \cup R_m$ and $\Theta = dom(R)$

where $R_m = \{(rm_X^+(\phi) \wedge \phi_x \Rightarrow E_x \ltimes, m, \phi \Rightarrow E \ltimes) \mid (\phi \Rightarrow E \ltimes) \in \Theta', x \in fv(E) \cap X\}$

$y > 0 \Rightarrow w \ltimes$

$w > 0 \Rightarrow v \ltimes$, $w > 0 \Rightarrow y \ltimes$

$w > 0 \wedge y > 0 \Rightarrow z \ltimes$

$w > 0 \wedge y \leq 0 \Rightarrow w \ltimes$

call p

$x > 0 \wedge y > 0 \Rightarrow w \ltimes$

$w > 0 \Rightarrow (x + v) \ltimes$

While Loops

Need iteration

```
while i < 7 do
    if odd(i)
    then r := r + v; v := v + h
    else v := x;
    i := i + 1
```

$r \ltimes$

While Loops

Need iteration

```
while i < 7 do
    if odd(i)
    then r := r + v; v := v + h
    else v := x;
    i := i + 1
```

$r \ltimes$

	$\Rightarrow$	$\ltimes$
F		h
F		i
T		r
F		v
F		x

While Loops

Need iteration

```
while i < 7 do
    if odd(i)
    then r := r + v; v := v + h
    else v := x;
    i := i + 1
```

$r \ltimes$

		$\Rightarrow$	$\ltimes$
F	**F**		h
F	**T**		i
T	**T**		r
F	$odd(i)$		v
F	**F**		x

While Loops

Need iteration

```
while i < 7 do
    if odd(i)
    then r := r + v; v := v + h
    else v := x;
    i := i + 1
```

$r \ltimes$

			$\Rightarrow$	$\ltimes$
F	**F**	**F**		h
F	**T**	**T**		i
T	**T**	**T**		r
F	$odd(i)$	$odd(i)$		v
F	**F**	$\neg odd(i)$		x

While Loops

Need iteration to reach fixed point

```
while i < 7 do
    if odd(i)
    then r := r + v; v := v + h
    else v := x;
    i := i + 1
```

$r \ltimes$

				$\Rightarrow$	$\ltimes$
F	**F**	**F**	**F**		h
F	**T**	**T**	**T**		i
T	**T**	**T**	**T**		r
F	$odd(i)$	$odd(i)$	$odd(i)$		v
F	**F**	$\neg odd(i)$	**T**		x

While Loops, continued

```
while x > 5 do
    x := x - 1;
    y := y + 1
```

While Loops, continued

```
while x > 5 do
   x := x - 1;
   y := y + 1
```

	$\Rightarrow$	$\ltimes$
T		x
F		y
$y > 25$		z

While Loops, continued


```
while x > 5 do
    x := x - 1;
    y := y + 1
```

		⇒	⋉
T	**T**		x
F	**F**		y
$y > 25$	$y > 24$		z

While Loops, continued

Iteration may not terminate.

```
while x > 5 do
    x := x - 1;
    y := y + 1
```

			$\Rightarrow$	$\ltimes$
T	**T**	**T**		x
F	**F**	**F**		y
$y > 25$	$y > 24$	$y > 23$		z

While Loops, continued

Iteration may not terminate. Use widening.

```
while x > 5 do
   x := x - 1;
   y := y + 1
```

				$\Rightarrow$	$\ltimes$
T	**T**	**T**	**T**		x
F	**F**	**F**	**F**		y
$y > 25$	$y > 24$	$y > 23$	**T**		z