Dynamic analysis

Ana Matos
Miguel Correia Pedro Adão

☞ "From dynamic to static and back: Riding the roller coaster of information-flow control research", Andrei Sabelfeld and Alejandro Russo, 2009

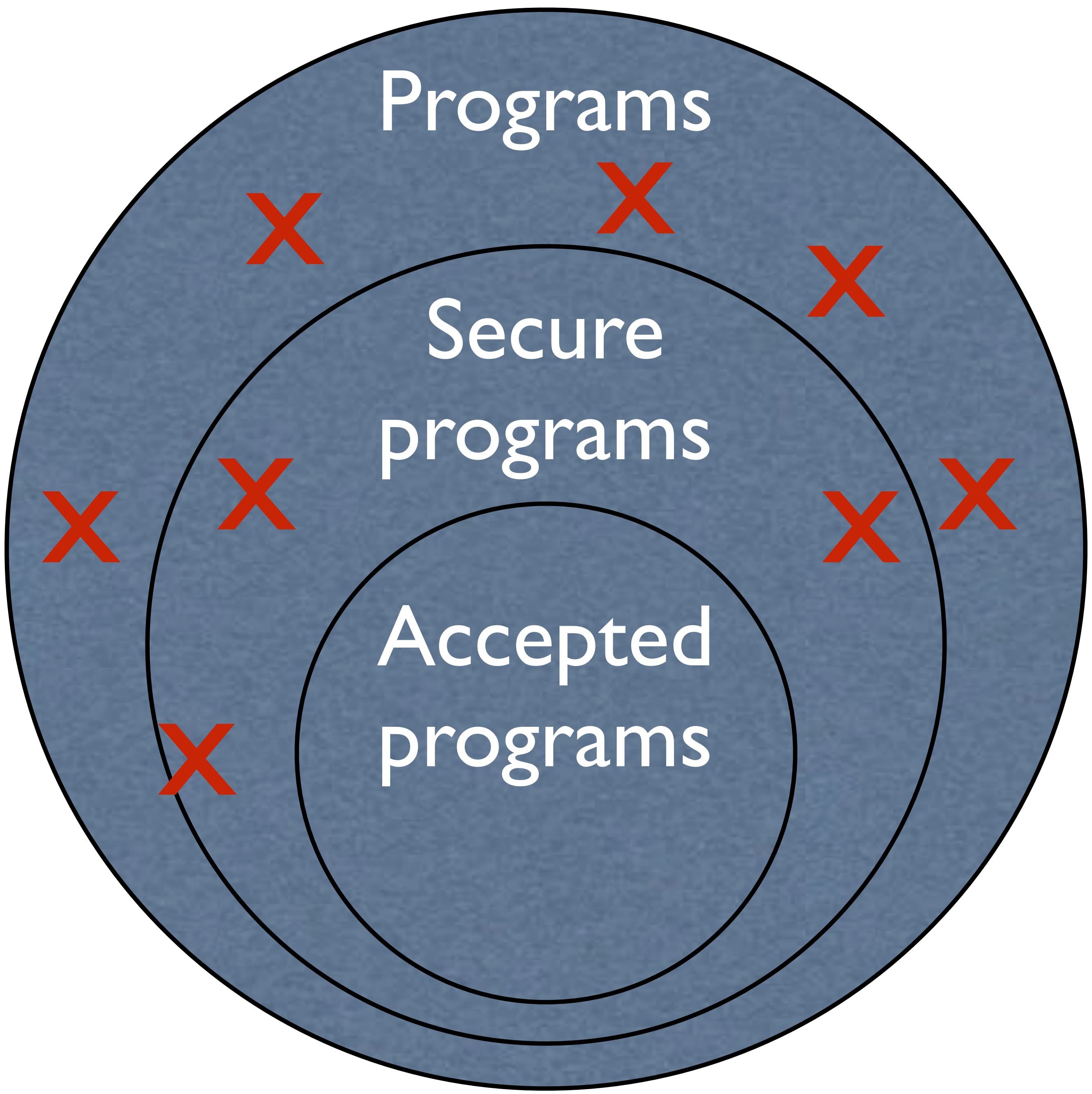
Dynamic analysis

- As **more information** is available **at runtime**, it is possible to use it and accept secure runs of programs that might be otherwise rejected by static analysis.
- Dynamic techniques for tracking information flow are on the rise, driven by the need for permissiveness in today's **dynamic applications**.

Checking vs. Transforming

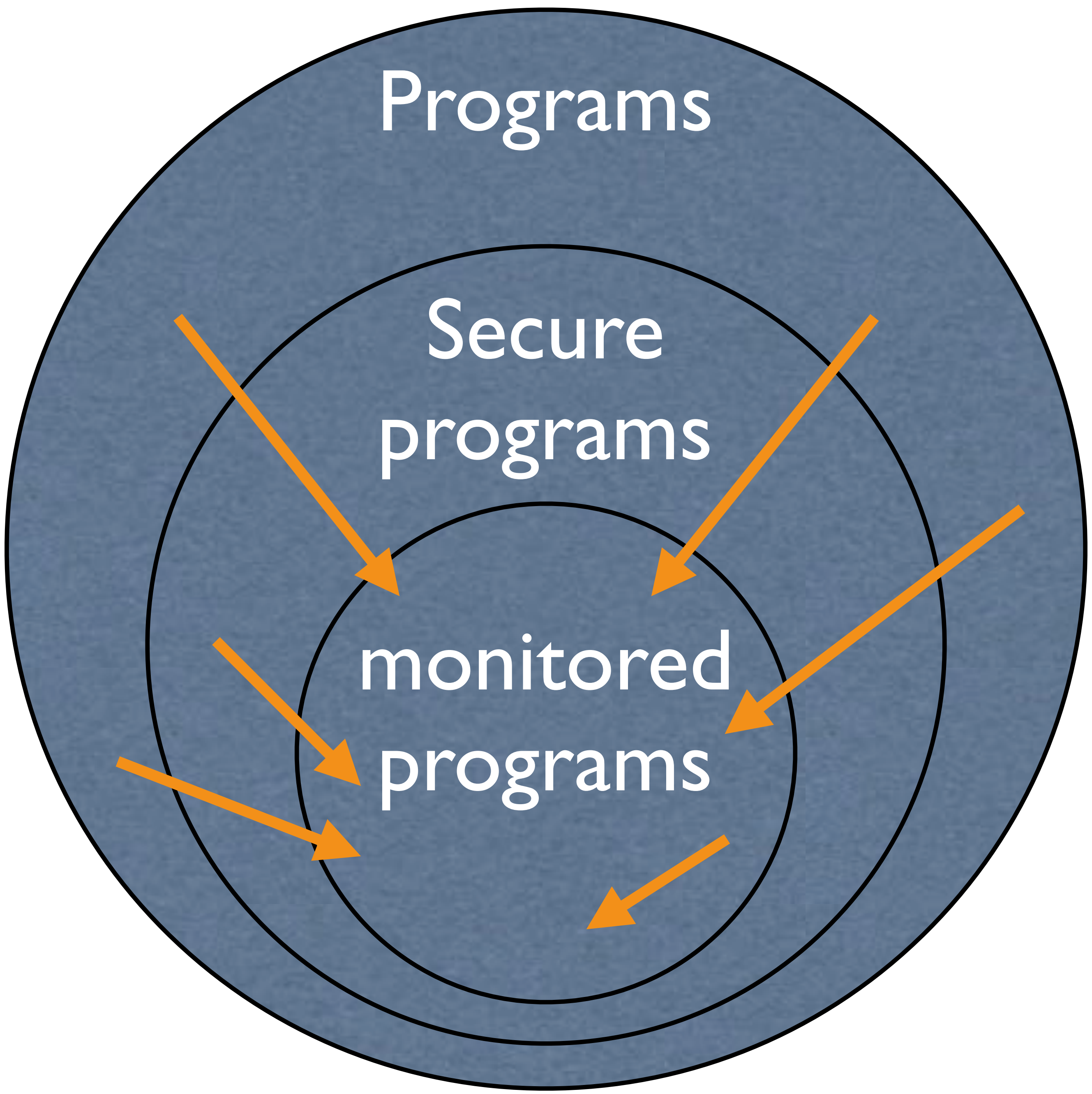
- Our type system *checks* whether programs should be accepted: either accepts or rejects a program. (*)
- Monitors *change* the behaviour of programs, so that the result is acceptable.
- (*) There are type systems that transform programs too.

Accepting secure programs



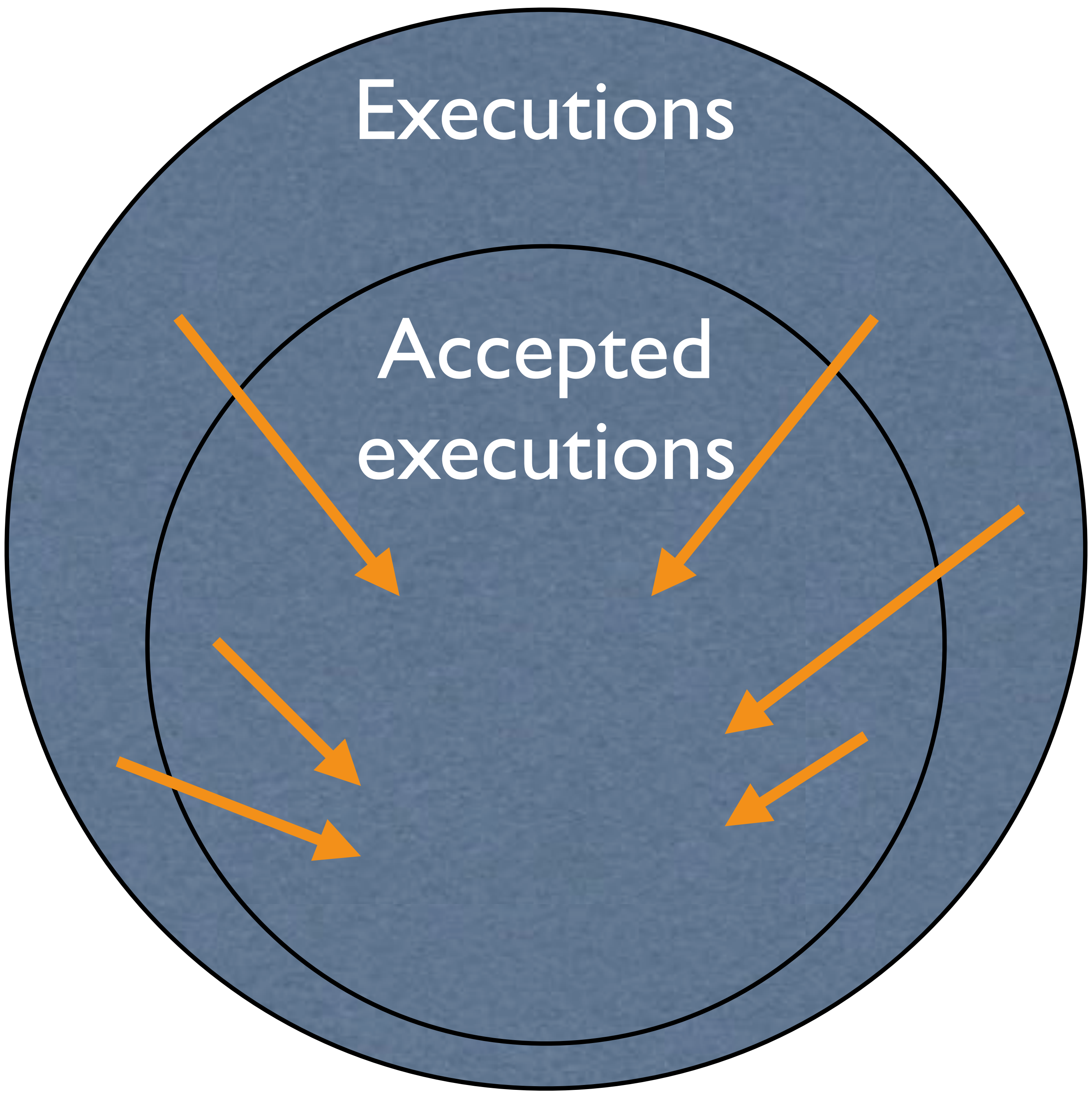
- (here circles are sets of programs)

Transforming into secure



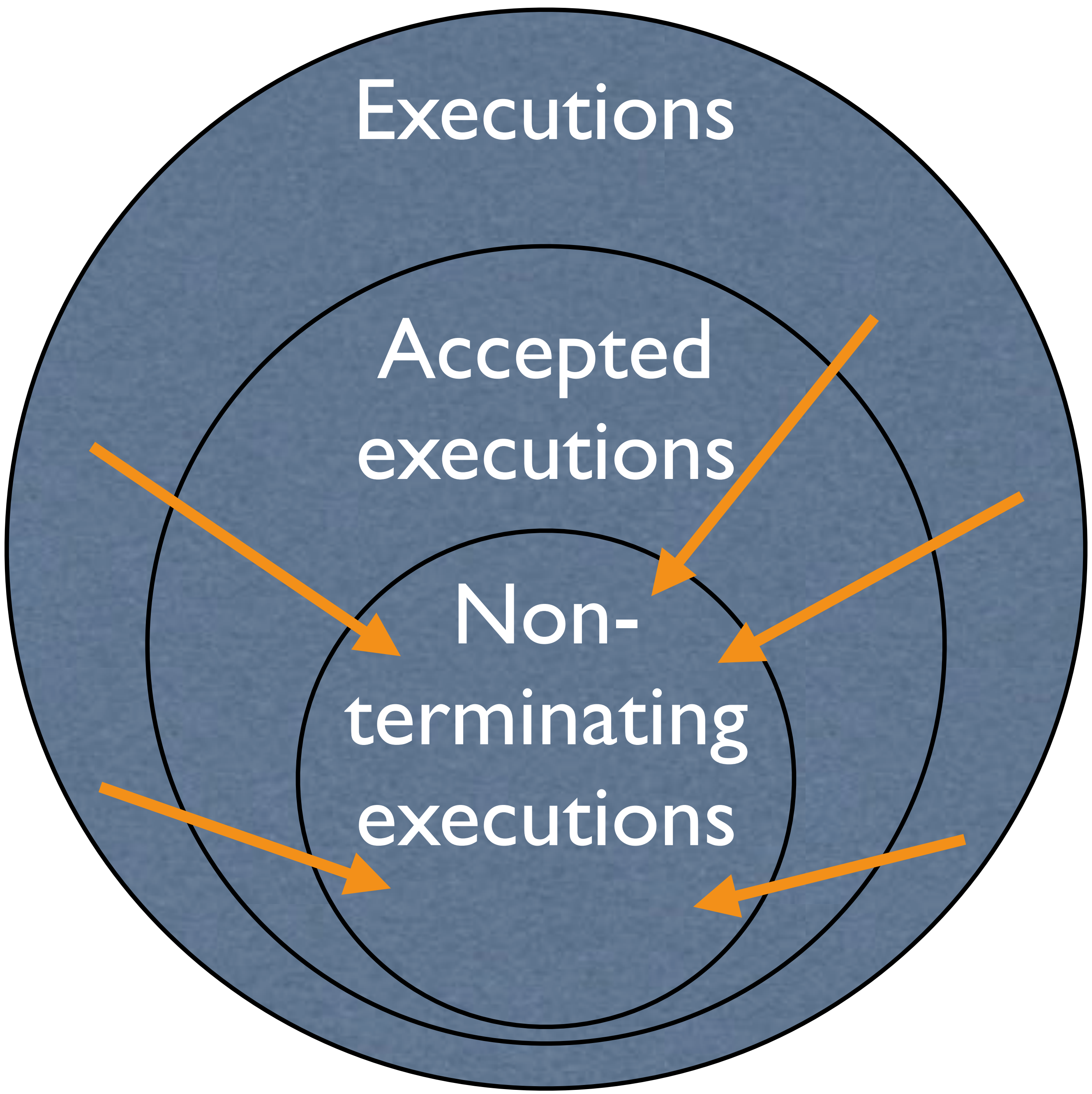
- (here circles are sets of programs)

Transforming into secure



- (here circles are programs = sets of possible executions)

For a Deterministic Input-Output attacker…



- Remember: Non-terminating executions are invisible to a Deterministic Input-Output attacker.
- (here circles are programs = sets of possible executions)

Lock-step monitor: Idea

- To define a monitored semantics for the composition of the program and monitor.
- The monitor can only perform safe executions.
- Synchronize each step of the program and of the monitor.
- Steps that don't synchronise get blocked (don't terminate).

We will need

- We want to be able to talk about each individual step performed by programs and monitors. We will need:
  - To define a small-step labeled semantics for the programs ($\Rightarrow^{\alpha}$)
  - To define a small-step labeled semantics for the monitor ($\Rightarrow_{m}^{\alpha}$)

Let's see this in two parts:
1st - basic small-step semantics
2nd - labeled small-step semantics

Topics

- Accepting vs. Transforming - mechanisms
- A monitor for Information flow analysis (WHILE)
  - Small-step semantics for WHILE
  - Labelled transitions
  - Lock-step information flow monitor

Small-step semantics
(small-step transition system)

- Configurations:
  - intermediate $<$Statement S, state $\rho$$>$
  - terminal ρ
- Transitions: $<S, \rho> \Rightarrow \Upsilon$ where Υ is either $<S', \rho'>$ or ρ'
- Rules: $$\frac{<S_1, \rho_1> \Rightarrow \Upsilon_1 \ \ldots <S_n, \rho_n> \Rightarrow \Upsilon_n}{<S, \rho> \Rightarrow \Upsilon_n} \quad \text{if } \ldots.$$

Small-step transitions

Axioms - do not depend on any hypothesis in order to give the final result of the step

- Skip: $<\text{skip}, \rho> \Rightarrow \rho$
- Assignment: $<x := a, \rho> \Rightarrow \rho[x \mapsto \mathcal{A}[\![a]\!]_\rho]$

the update of state ρ is defined as:

$$\begin{cases} (\rho[y \mapsto c])(x) = c, & \text{if } x = y \\ \rho(x), & \text{otherwise} \end{cases}$$

These programs execute fully in a single step.

Rules - the final result of the step below the line, depends on the hypothesis above the line

- Sequential composition:

When S_1 starting on ρ terminates and produces ρ'...

$$\frac{\langle S_1, \rho\rangle \Rightarrow \rho'}{\langle S_1;S_2, \rho\rangle \Rightarrow \langle S_2, \rho'\rangle}$$

... then the entire sequential composition starting on ρ leads to S_2 and produces ρ'.

When S_1 starting on ρ leads to S_1' and produces ρ'...

$$\frac{\langle S_1, \rho\rangle \Rightarrow \langle S_1', \rho'\rangle}{\langle S_1;S_2, \rho\rangle \Rightarrow \langle S_1';S_2, \rho'\rangle}$$

... then the entire sequential composition $S_1;S_2$ starting on ρ leads to $S_1';S_2$ and produces ρ'.

- Conditional test:

When t evaluates to true...

$$\langle \text{if } t \text{ then } S_1 \text{ else } S_2, \rho\rangle \Rightarrow \langle S_1, \rho\rangle \quad \text{if } \mathcal{B}[\![t]\!]_\rho = \text{true}$$

... then the step starting on ρ leads to the first branch and changes nothing.

When t evaluates to false…

$$\langle \text{if } t \text{ then } S_1 \text{ else } S_2, \rho\rangle \Rightarrow \langle S_2, \rho\rangle \quad \text{if } \mathcal{B}[\![t]\!]_\rho = \text{false}$$

... then the step starting on ρ leads to the second branch and changes nothing.

When t evaluates to true...

- While loop:

$$<\text{while } t \text{ do } S, \rho> \Rightarrow <S; \text{while } t \text{ do } S, \rho> \quad \text{if } B[\![t]\!]_\rho = \text{true}$$

... the step leads to the evaluation of the body, and then followed by the entire cycle, and changes nothing.

$$<\text{while } t \text{ do } S, \rho> \Rightarrow \rho \quad \text{if } B[\![t]\!]_\rho = \text{false}$$

When t evaluates to false the cycle changes nothing.

(All) Small-step rules

Assignment: $< x{:=}a, \rho > \Rightarrow \rho[x \mapsto A[\![a]\!]]_\rho$

Skip: $< skip, \rho > \Rightarrow \rho$

Sequential comp.:

$$\frac{<S_1, \rho> \Rightarrow \rho'}{< S_1;S_2, \rho> \Rightarrow <S_2, \rho'>} \qquad \frac{<S_1, \rho> \Rightarrow <S_1', \rho' >}{< S_1;S_2, \rho> \Rightarrow <S_1';S_2, \rho'>}$$

Conditional test: $<\text{if } t \text{ then } S_1 \text{ else } S_2, \rho> \Rightarrow <S_1, \rho>$ if $B[\![t]\!]_\rho = \text{true}$

$<\text{if } t \text{ then } S_1 \text{ else } S_2, \rho> \Rightarrow <S_2, \rho>$ if $B[\![t]\!]_\rho = \text{false}$

While loop: $<\text{while } t \text{ do } S, \rho> \Rightarrow <S;\text{while } t \text{ do } S , \rho>$ if $B[\![t]\!]_\rho = \text{true}$

$<\text{while } t \text{ do } S, \rho> \Rightarrow \rho$ if $B[\![t]\!]_\rho = \text{false}$

Example - Evaluation

- Evaluate $(z{:=}x\ ;\ x{:=}y)\ ;\ y{:=}z$, starting from a state ρ_0 that maps all variables except x and y to 0, and has $\rho_0(x) = 5$ and $\rho_0(y) = 7$.

$$\langle (z{:=}x\ ;\ x{:=}y)\ ;\ y{:=}z,\ \rho_0 \rangle \Rightarrow \langle x{:=}y\ ;\ y{:=}z,\ \rho_0[z\mapsto 5] \rangle$$

$$\Rightarrow \langle y{:=}z,\ \rho_0[z\mapsto 5, x\mapsto 7] \rangle$$

$$\Rightarrow \rho_0[z\mapsto 5, x\mapsto 7, y\mapsto 5]$$

derivation sequence

Example - Evaluation

- Eva ... state ρ_0
  tha ...
  and ...

$$\frac{\langle\ z:=x,\ \rho_0\rangle \Rightarrow\ \rho_0[z\mapsto 5]}{\underline{\langle (z:=x\ ;\ x:=y),\ \rho_0\rangle \Rightarrow \langle x:=y,\ \rho_0[z\mapsto 5]\rangle}}$$

$$\langle (z:=x\ ;\ x:=y)\ ;\ y:=z,\ \rho_0\rangle \Rightarrow\ \langle x:=y\ ;\ y:=z,\ \rho_0[z\mapsto 5]\rangle$$

$$\Rightarrow \langle y:=z,\ \rho_0[z\mapsto 5, x\mapsto 7]\rangle$$

$$\Rightarrow\ \rho_0[z\mapsto 5, x\mapsto 7, y\mapsto 5]$$

derivation sequence

Example - Evaluation

- Evaluate (z:=x ; x:=y) ; y:=z, starting from a state ρ_0 that maps a … and has ρ_0(…

$$\frac{\langle x:=y, \rho_0[z \mapsto 5] \rangle \Rightarrow \rho_0[z \mapsto 5, x \mapsto 7]}{}$$

$$\langle (z:=x \; ; x:=y) \; ; y:=z, \rho_0 \rangle \Rightarrow \langle x:=y \; ; y:=z, \rho_0[z \mapsto 5] \rangle$$

$$\Rightarrow \langle y:=z, \rho_0[z \mapsto 5, x \mapsto 7] \rangle$$

$$\Rightarrow \rho_0[z \mapsto 5, x \mapsto 7, y \mapsto 5]$$

derivation sequence

Evaluation - derivation sequence

To evaluate statement S, starting from an initial state ρ, a derivation sequence can be constructed:

1. Try to find an axiom or rule whose left side matches <S, ρ>, whose side conditions are satisfied, and for which you can build a derivation tree showing that it can be applied.

- If it is an axiom - determine the final state and terminate.
- If it is a rule - determine the intermediate configuration that is the right side of the rule, and try to find the derivation sequence starting form it.

Example 1 $\rho(x_H)=\text{true}$

Program execution:

$< (\text{if } x_H \text{ then } y_L := 1 \text{ else } x_H := 0); z_L := 1, \rho >$

$\Rightarrow < y_L := 1; z_L := 1, \rho >$

$\Rightarrow < z_L := 1, \rho[y_L \mapsto 1] >$

$\Rightarrow \rho[y_L \mapsto 1, z_L \mapsto 1]$

- What information would be useful to pass to a monitor in order for it to decide about the acceptance of this executions?

Example 2 $\rho(x_H)=\text{false}$

Program execution:

$< (\text{if } x_H \text{ then } y_L := 1 \text{ else } x_H := 0); z_L := 1, \rho >$

$\Rightarrow < x_H := 0; z_L := 1, \rho >$

$\Rightarrow < z_L := 1, \rho[x_H \mapsto 0] >$

$\Rightarrow \rho[x_H \mapsto 0, z_L \mapsto 1]$

- What information would be useful to pass to a monitor in order for it to decide about the acceptance of this executions?

Example 3

Program execution:

$< (\text{if } x_H \neq x_H \text{ then } y_L := 1 \text{ else } x_H := 0); z_L := 1, \rho >$

$\Rightarrow < x_H := 0; z_L := 1, \rho >$

$\Rightarrow < z_L := 1, \rho[x_H \mapsto 0] >$

$\Rightarrow \rho[x_H \mapsto 0, z_L \mapsto 1]$

- What information would be useful to pass to a monitor in order for it to decide about the acceptance of this executions?

Labels

- Assignment:
  - What is the level of the variable that is assigned?
  - What is the level of the variables used in the expression
- Conditions / loops:
  - What is the level of the variables that are tested?
  - When are the regions being exited?

Labeled small-step semantics (small-step transition system)

- Configurations:
  - intermediate $\langle \text{Statement } S, \text{state } \rho \rangle$
  - terminal ρ
- Transitions: $\langle S, \rho \rangle \Rightarrow^{label} \Upsilon$ where Υ is either $\langle S', \rho' \rangle$ or ρ'
- Rules: $\dfrac{\langle S_1, \rho_1 \rangle \Rightarrow^{label1} \Upsilon_1 \ \ldots \langle S_n, \rho_n \rangle \Rightarrow^{labeln} \Upsilon_n}{\langle S, \rho \rangle \Rightarrow^{label} \Upsilon_n}$ if ....

Labelled Small-step transitions

Axioms - do not depend on any hypothesis in order to give the final result of the step

- Skip: $< \text{skip}, \rho > \Rightarrow^{\text{nop}} \rho$
- Assignment: $< x:=a, \rho > \Rightarrow^{(x,a)} \rho[x \mapsto \mathcal{A}[\![a]\!]_\rho]$

the update of state ρ is defined as:

$$\begin{cases} (\rho[y \mapsto c])(x) = c, & \text{if } x=y \\ \rho(x), & \text{otherwise} \end{cases}$$

- End: $< \text{end}, \rho > \Rightarrow^{f} \rho$

The runtime instruction 'end' is only for sending a message to the monitor.

When program S_1 makes a step with a label...

- Sequential composition:

$$\frac{\langle S_1, \rho\rangle \Rightarrow^{\alpha} \rho'}{\langle S_1;S_2, \rho\rangle \Rightarrow^{\alpha} \langle S_2, \rho'\rangle}$$

$$\frac{\langle S_1, \rho\rangle \Rightarrow^{\alpha} \langle S_1', \rho'\rangle}{\langle S_1;S_2, \rho\rangle \Rightarrow^{\alpha} \langle S_1';S_2, \rho'\rangle}$$

... then the step made by the entire sequential composition has the same label.

- Conditional test:

After branching, the junction point is signaled with 'end'. For the monitor to use.

$$\langle \text{if } t \text{ then } S_1 \text{ else } S_2, \rho\rangle \Rightarrow^{b(t)} \langle S_1 ;\text{end}, \rho\rangle \quad \text{if } B[\![t]\!]_\rho = \text{true}$$

$$\langle \text{if } t \text{ then } S_1 \text{ else } S_2, \rho\rangle \Rightarrow^{b(t)} \langle S_2 ;\text{end}, \rho\rangle \quad \text{if } B[\![t]\!]_\rho = \text{false}$$

Indicates to the monitor that the program is entering a region guarded by t

When t evaluates to true...

- While loop:

$\langle \text{while } t \text{ do } S, \rho\rangle \Rightarrow^{b(t)} \langle S; \text{end}; \text{while } t \text{ do } S, \rho\rangle$ if $B[\![t]\!]_{\rho}=\text{true}$

... the step leads to the evaluation of the body, followed by 'end', and then followed by the entire cycle.

$\langle \text{while } t \text{ do } S, \rho\rangle \Rightarrow^{b(t)} \langle \text{end}, \rho\rangle$ if $B[\![t]\!]_{\rho}=\text{false}$

When t evaluates to false the cycle leads to 'end'.

All rules

- Skip: $\langle \text{skip}, \rho \rangle \Rightarrow^{nop} \rho$
- Assignment: $\langle x{:=}a, \rho \rangle \Rightarrow^{(x,a)} \rho[x \mapsto \mathcal{A}[\![a]\!]_\rho]$
- End: $\langle \text{end}, \rho \rangle \Rightarrow^{f} \rho$
- Sequential composition:

$$\frac{\langle S_1, \rho \rangle \Rightarrow^{\alpha} \langle S_1', \rho' \rangle}{\langle S_1;S_2, \rho \rangle \Rightarrow^{\alpha} \langle S_1';S_2, \rho' \rangle} \qquad \frac{\langle S_1, \rho \rangle \Rightarrow^{\alpha} \rho'}{\langle S_1;S_2, \rho \rangle \Rightarrow^{\alpha} \langle S_2, \rho' \rangle}$$

- Conditional test:

$\langle \text{if } t \text{ then } S_1 \text{ else } S_2, \rho \rangle \Rightarrow^{b(t)} \langle S_1;\text{end}, \rho \rangle$ if $\mathcal{B}[\![t]\!]_\rho = \text{true}$

$\langle \text{if } t \text{ then } S_1 \text{ else } S_2, \rho \rangle \Rightarrow^{b(t)} \langle S_2;\text{end}, \rho \rangle$ if $\mathcal{B}[\![t]\!]_\rho = \text{false}$

- While loop:

$\langle \text{while } t \text{ do } S, \rho \rangle \Rightarrow^{b(t)} \langle S;\text{end};\text{while } t \text{ do } S, \rho \rangle$ if $\mathcal{B}[\![t]\!]_\rho = \text{true}$

$\langle \text{while } t \text{ do } S, \rho \rangle \Rightarrow^{b(t)} \langle \text{end}, \rho \rangle$ if $\mathcal{B}[\![t]\!]_\rho = \text{false}$

Example 1 $\rho(x_H)=\text{true}$

Program execution:

$< (\text{if } x_H \text{ then } y_L := 1 \text{ else } x_H := 0); z_L := 1, \rho >$

$\Rightarrow^{b(xH)} < y_L := 1; \text{end} ; z_L := 1, \rho >$

$\Rightarrow^{(yL,1)} < \text{end} ; z_L := 1, \rho[y_L \mapsto 1] >$

$\Rightarrow^{f} < z_L := 1, \rho[y_L \mapsto 1] >$

$\Rightarrow^{(zL,1)} \rho[y_L \mapsto 1, z_L \mapsto 1]$

- What data structure would be appropriate for the monitor to keep track of the relevant levels?

Example 2 $\rho(x_H)=\text{false}$

Program execution:

$< (\text{if } x_H \text{ then } yL := 1 \text{ else } x_H := 0); z_L := 1, \rho >$

$\Rightarrow^{b(xH)} < x_H := 0; \text{end} ; z_L := 1, \rho >$

$\Rightarrow^{(xH,0)} < \text{end} ; z_L := 1, \rho[x_H \mapsto 0] >$

$\Rightarrow^{f} < z_L := 1, \rho[x_H \mapsto 0] >$

$\Rightarrow^{(zL,1)} \rho[x_H \mapsto 0, z_L \mapsto 1]$

- What data structure would be appropriate for the monitor to keep track of the relevant levels?

Example 3

Program execution:

$< (\text{if } x_H \neq x_H \text{ then } y_L := 1 \text{ else } x_H := 0); z_L := 1, \rho >$

$\Rightarrow^{b(xH \neq xH)} < x_H := 0; \text{end} \; ; z_L := 1, \rho >$

$\Rightarrow^{(xH,0)} < \text{end} \; ; z_L := 1, \rho[x_H \mapsto 0] >$

$\Rightarrow^{f} < z_L := 1, \rho[x_H \mapsto 0] >$

$\Rightarrow^{(zL,1)} \rho[x_H \mapsto 0, z_L \mapsto 1]$

- What data structure would be appropriate for the monitor to keep track of the relevant levels?

Monitor transitions

- The monitor either accepts an event generated by the program or blocks it by getting stuck.
- The monitor operates on stacks of security levels. (The empty stack is represented by ε.) Transitions are labeled.

$$st \Rightarrow_m^{\alpha} st'$$

stacks of
security levels

Monitor semantics

- nop: $st \Rightarrow_m^{nop} st$

Always accepted without changes in the monitor state.

- branching: $st \Rightarrow_m^{b(a)} lev(a) :: st$
- end: $l : st \Rightarrow_m^{f} st$
- assignment: $$\frac{lev(a) \sqsubseteq \Gamma(x) \quad lev(st) \sqsubseteq \Gamma(x)}{st \Rightarrow_m^{(x,a)} st}$$

lev(a) = $\sqcup$ levels of the variables in a

lev(st) = $\sqcup$ levels in st

Monitor semantics

- nop: $st \Rightarrow_m^{nop} st$
- **branching:** $st \Rightarrow_m^{b(a)} lev(a) :: st$
- end: $l : st \Rightarrow_m^{f} st$
- assignment: $$\frac{lev(a) \sqsubseteq \Gamma(x) \quad lev(st) \sqsubseteq \Gamma(x)}{st \Rightarrow_m^{(x,a)} st}$$

Pushes the security level of the expression onto the stack.

$lev(a) = \bigsqcup$ levels of the variables in a

$lev(st) = \bigsqcup$ levels in st

Monitor semantics

- nop: $st \Rightarrow_m^{nop} st$
- branching: $st \Rightarrow_m^{b(a)} lev(a)$:
- end: $l :: st \Rightarrow_m^{f} st$
- assignment: $\dfrac{lev(a) \sqsubseteq \Gamma(x)}{st \Rightarrow_m^{(x,}}$

When the conditional or while terminates (junction point has been reached), its guard level is popped off the stack.

lev(a) = ⊔ levels of the variables in a

lev(st) = ⊔ levels in st

Monitor semantics

If the security level of the written variable is at least as high as that of the read expression...

... and as that of the highest security level in the stack...

... then the event is accepted without changes to the monitor state.

st

lev(a) : :st

- end: Γ : st ⇒ st

- assignment: $$\frac{lev(a)\sqsubseteq\Gamma(x) \quad lev(st)\sqsubseteq\Gamma(x)}{st \Rightarrow_m^{(x,a)} st}$$

lev(a) = ⊔ levels of the variables in a

lev(st) = ⊔ levels in st

Monitor semantics (all rules)

- nop: $st \Rightarrow_m^{nop} st$
- branching: $st \Rightarrow_m^{b(a)} lev(a) :: st$
- end: $l : st \Rightarrow_m^{f} st$
- assignment: $$\frac{lev(a) \sqsubseteq \Gamma(x) \quad lev(st) \sqsubseteq \Gamma(x)}{st \Rightarrow_m^{(x,a)} st}$$

$lev(a) = \bigsqcup$ levels of the variables in a

$lev(st) = \bigsqcup$ levels in st

Example 1 $\rho(x_H)=\text{true}$

Program execution:

$< (\text{if } x_H \text{ then } y_L := 1 \text{ else } x_H := 0); z_L := 1, \rho >$

$\Rightarrow^{b(xH)} < y_L := 1; \text{end} ; z_L := 1, \rho >$

$\Rightarrow^{(yL,1)} < \text{end} ; z_L := 1, \rho[y_L \mapsto 1] >$

$\Rightarrow^{f} < z_L := 1, \rho[y_L \mapsto 1] >$

$\Rightarrow^{(zL,1)} \rho[y_L \mapsto 1, z_L \mapsto 1]$

Monitor execution:

ε

$\Rightarrow_m^{b(xH)} H::\varepsilon$

$\Rightarrow_m^{(yL,1)}$ ✗

Example 2 $\rho(x_H)$=false

Program execution:

$< (\text{if } x_H \text{ then } y_L := 1 \text{ else } x_H := 0); z_L := 1, \rho >$

$\Rightarrow^{b(xH)} < x_H := 0; \text{end} ; z_L := 1, \rho >$

$\Rightarrow^{(xH,0)} < \text{end} ; z_L := 1, \rho[x_H \mapsto 0] >$

$\Rightarrow^{f} < z_L := 1, \rho[x_H \mapsto 0] >$

$\Rightarrow^{(zL,1)} \rho[x_H \mapsto 0, z_L \mapsto 1]$

Monitor execution:

ε

$\Rightarrow_m^{b(xH)} H::\varepsilon$

$\Rightarrow_m^{(xH,0)} H::\varepsilon$

$\Rightarrow_m^{f} \varepsilon$

$\Rightarrow_m^{(zL,1)} \varepsilon$

Example 3

Program execution:

$<$ (if $x_H \neq x_H$ then y_L := 1 else x_H := 0); z_L := 1, ρ $>$

$\Rightarrow^{b(xH \neq xH)} <$ x_H := 0; end ; z_L := 1, ρ $>$

$\Rightarrow^{(xH,0)} <$ end ; z_L := 1, $\rho[x_H \mapsto 0]$ $>$

$\Rightarrow^{f} <$ z_L := 1, $\rho[x_H \mapsto 0]$ $>$

$\Rightarrow^{(zL,1)}$ $\rho[x_H \mapsto 0, z_L \mapsto 1]$

Monitor execution:

ε

$\Rightarrow_m^{b(xH \neq xH)}$ $H::\varepsilon$

$\Rightarrow_m^{(xH,0)}$ $H::\varepsilon$

$\Rightarrow_m^{f}$ ε

$\Rightarrow_m^{(zL,1)}$ ε

Lock-step monitored semantics

- Idea: To compose ($|_m$) the execution of the program and of the monitor

Labelled transition between program configurations

Labelled transition between monitor (m) configurations

$$\frac{cfg \Rightarrow^{\alpha} cfg' \qquad cfgm \Rightarrow_m^{\alpha} cfgm'}{< cfg\ |_m\ cfgm > \Rightarrow < cfg'\ |_m\ cfgm' >}$$

Transition between monitored program configurations

Lock-step monitored semantics

- Idea: To compose ($|_m$) the execution of the program and of the monitor

If the program can make a step with label α...

... and the monitor m can also make a step with label α...

$$\frac{cfg \Rightarrow^{\alpha} cfg' \qquad cfgm \Rightarrow_m^{\alpha} cfgm'}{< cfg \mid_m cfgm > \Rightarrow^{\alpha} < cfg' \mid_m cfgm' >}$$

Then the monitored program can make that step

Lock-step monitored semantics

- Idea: To compose ($|_m$) the execution of the program and of the monitor m

If the program can make a step with label α...

... and the monitor cannot make the corresponding transition...

$$\frac{cfg \Rightarrow^{\alpha} cfg' \qquad cfgm \Rightarrow_m^{\alpha} XXXXXX}{< cfg \;|_m\; cfgm > \Rightarrow^{\alpha} XXXXXX}$$

... then the monitored program cannot make the corresponding step either.

Example 1 $\rho(x_H)=\text{true}$

Program execution:

$<(\text{if } x_H \text{ then } y_L := 1 \text{ else } x_H := 0); z_L := 1, \rho >$

$\Rightarrow^{b(xH)} < y_L := 1; \text{end}\, ; z_L := 1, \rho >$

$\Rightarrow^{(yL,1)} < \text{end}\, ; z_L := 1, \rho[y_L \mapsto 1] >$

$\Rightarrow^{f} < z_L := 1, \rho[y_L \mapsto 1] >$

$\Rightarrow^{(zL,1)} \rho[y_L \mapsto 1, z_L \mapsto 1]$

Monitor execution:

ε

$\Rightarrow_m^{b(xH)} H::\varepsilon$

$\Rightarrow_m^{(yL,1)}$

Monitored execution:

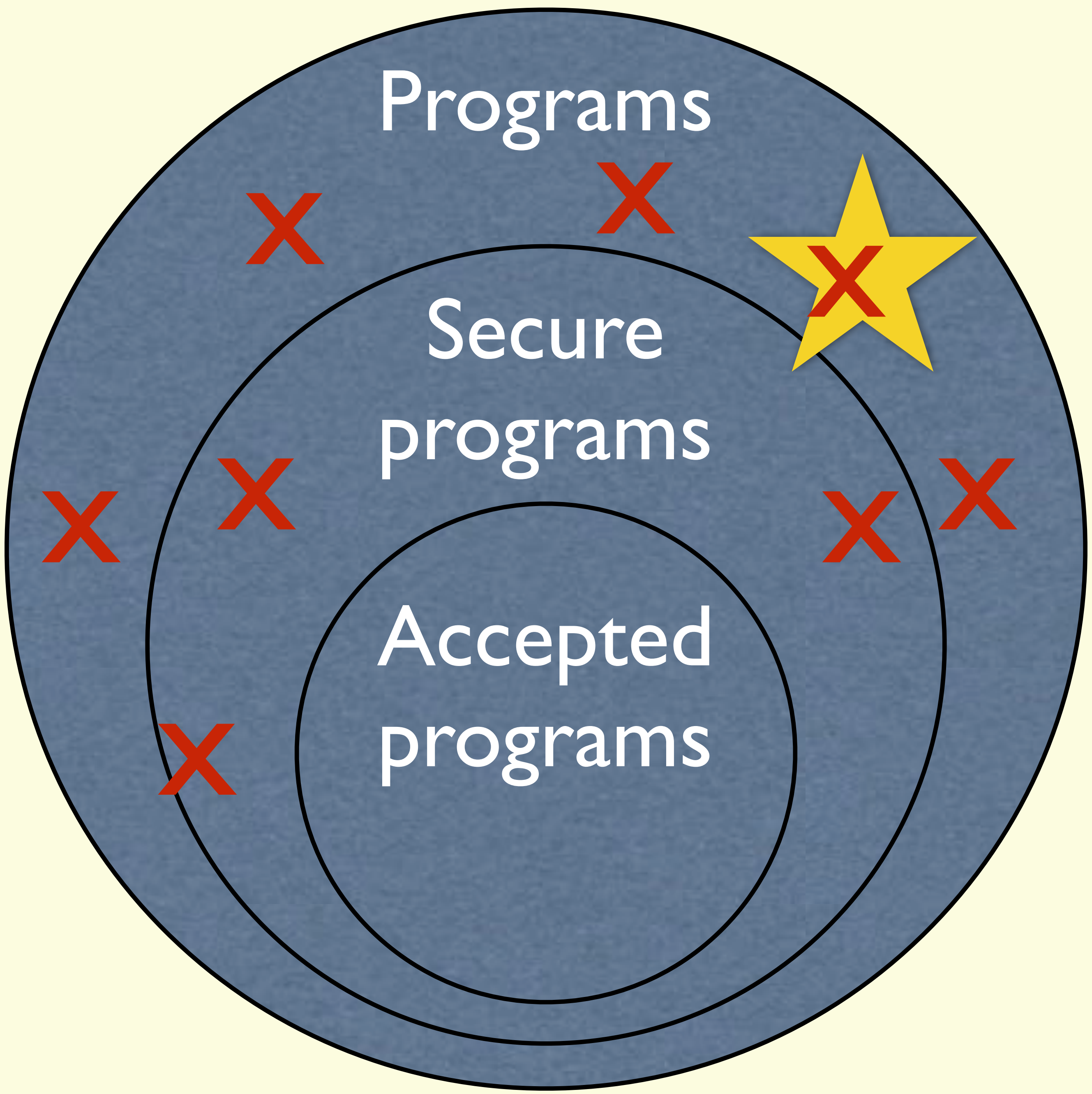
$< <(\text{if } x_H \text{ then } y_L := 1 \text{ else } x_H := 0); z_L := 1,\rho > |_m\, \varepsilon >$

$\Rightarrow^{b(xH)} < < y_L := 1; \text{end}\, ; z_L := 1,\rho > |_m\, H::\varepsilon >$

$\Rightarrow$

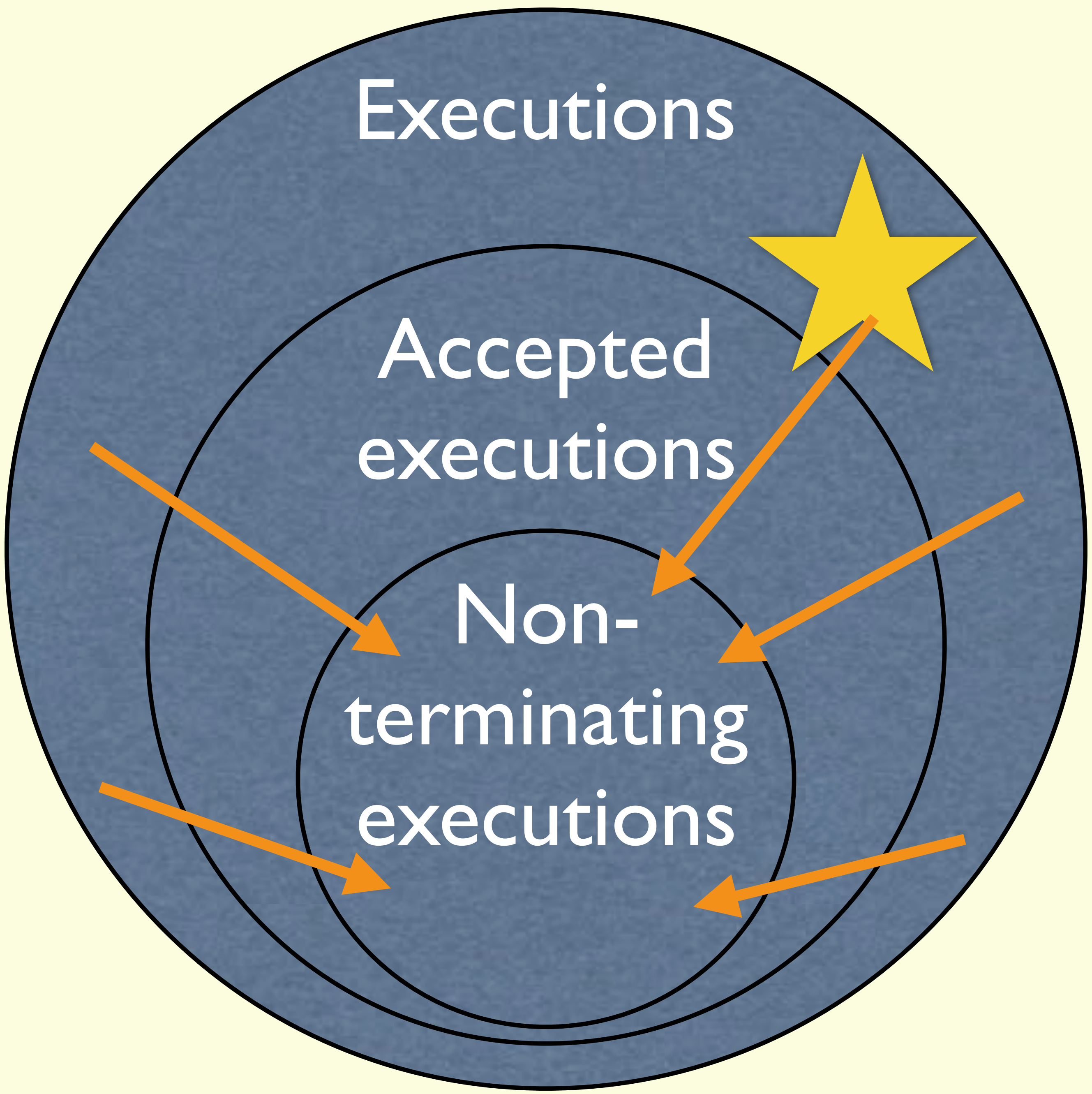
This execution does not terminate.

Accepting secure programs



- (circles are sets of programs)

In our case…



- Non-terminating executions are invisible to a Deterministic Input-Output attacker.
- (circles are programs = sets of possible executions)

Example 2 $\rho(x_H)=\mathit{false}$

Monitored execution:

$< <(\text{if } x_H \text{ then } yL := 1 \text{ else } x_H := 0); z_L := 1, \rho > |_m\ \varepsilon >$

$\Rightarrow^{b(xH)} < <x_H := 0; \text{end}\ ; z_L := 1, \rho > |_m\ H::\varepsilon >$

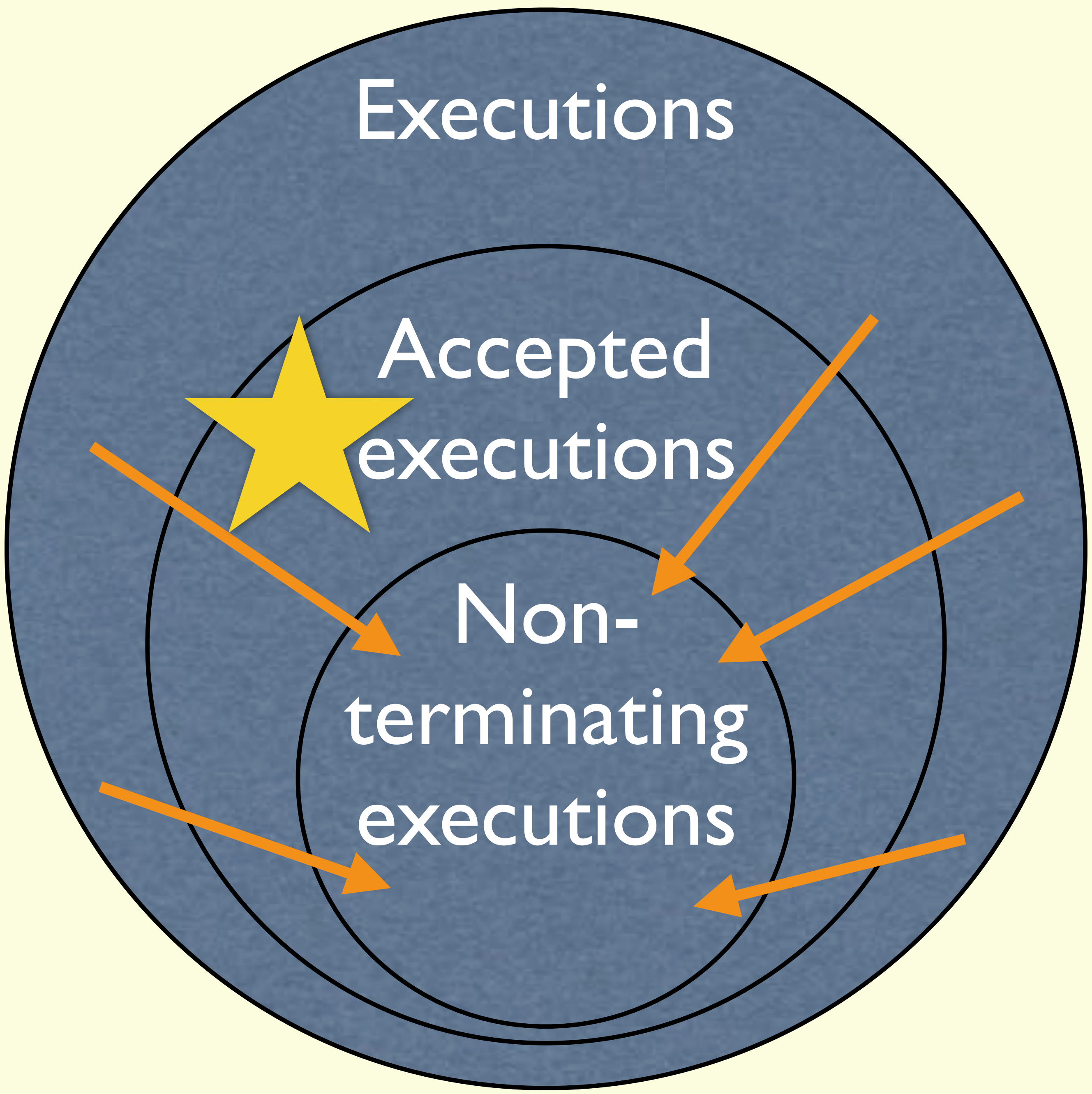
$\Rightarrow^{(xH,0)} < <\text{end}\ ; z_L := 1, \rho[x_H \mapsto 0] > |_m\ H::\varepsilon >$

$\Rightarrow^{f} < <z_L := 1, \rho[x_H \mapsto 0] > |_m\ \varepsilon >$

$\Rightarrow^{(zL,1)} < \rho[x_H \mapsto 0, z_L \mapsto 1] \ |_m\ \varepsilon >$

This execution terminates.

In our case...



- Non-terminating executions are invisible to a Deterministic Input-Output attacker.
- (circles are programs = sets of possible executions)

Example 3

Monitored execution:

$< <(\text{if } x_H \neq x_H \text{ then } yL := 1 \text{ else } x_H := 0); z_L := 1, \rho > |_m\ \varepsilon >$

$\Rightarrow^{b(xH\neq xH)} < <x_H := 0; \text{end} ; z_L := 1, \rho > |_m\ H::\varepsilon >$

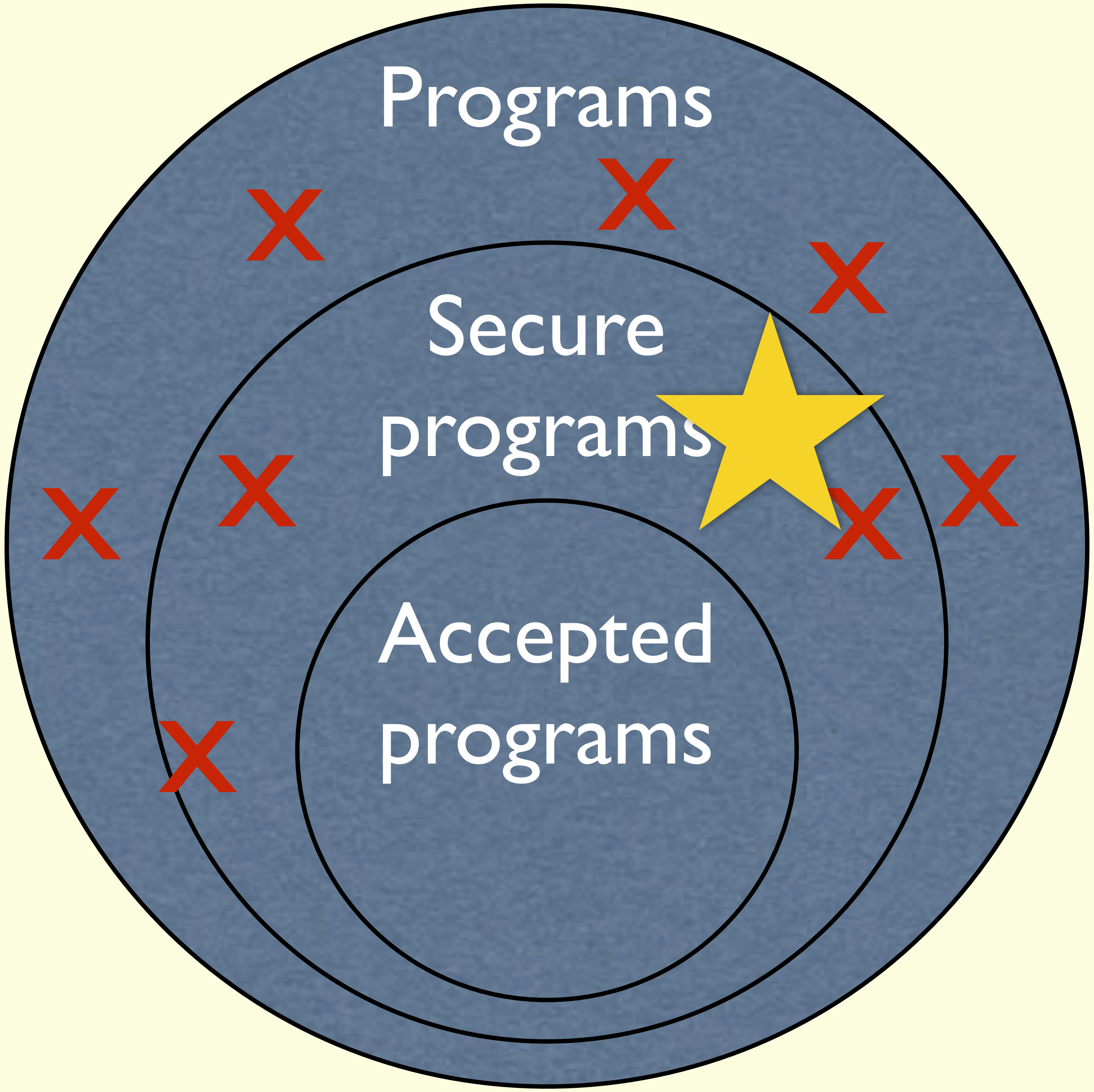
$\Rightarrow^{(xH,0)} < <\text{end} ; z_L := 1, \rho[x_H \mapsto 0] > |_m\ H::\varepsilon >$

$\Rightarrow^{f} < <z_L := 1, \rho[x_H \mapsto 0] > |_m\ \varepsilon >$

$\Rightarrow^{(zL,1)} < \rho[x_H \mapsto 0, z_L \mapsto 1] |_m\ \varepsilon >$

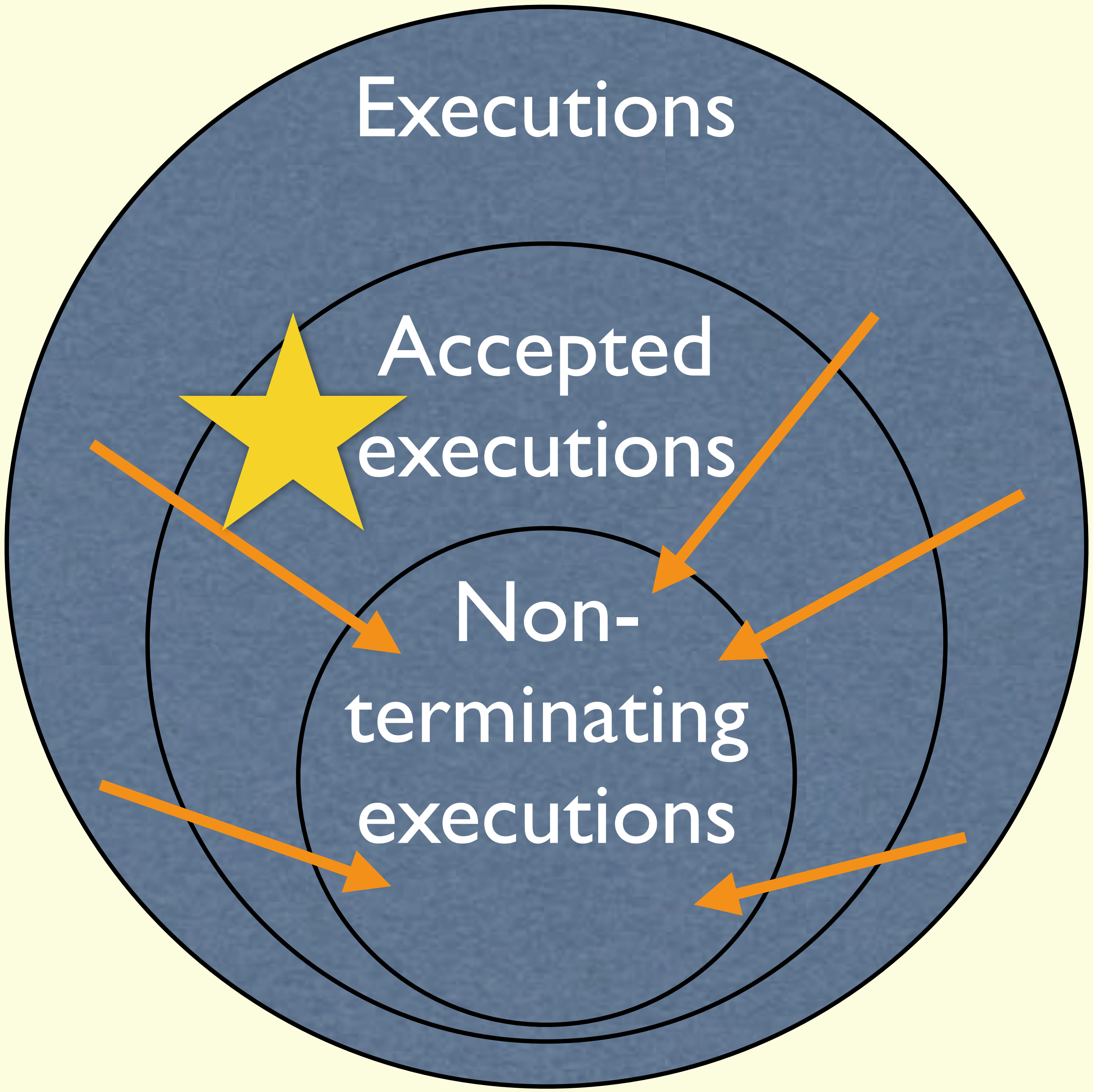
All executions terminate.

Accepting secure programs



- (circles are sets of programs)

In our case…



- Non-terminating executions are invisible to a Deterministic Input-Output attacker.
- (circles are programs = sets of possible executions)

Big-step semantics

- Configurations:
  - initial: <Program, memory>
  - final: memory
- Transitions: $\langle P, \rho \rangle \rightarrow_m \rho'$ where
  - ρ - memory at program start
  - ρ' - memory at program termination

We've seen that for the purpose of defining the Noninterference property it is useful to have a big-step transition notation.
We can define it in terms of the small-step one.

- $\Rightarrow^*$ is the reflexive and transitive closure of $\Rightarrow$

Initial configuration:
S - program
ρ - initial memory
ε - empty monitor stack

Final configuration:
ρ' - final memory
st - left-over stack

$$\frac{\langle \langle S,\rho \rangle \,|_m\, \varepsilon \rangle \Rightarrow^* \langle \rho' \,|_m\, \mathrm{st} \rangle}{\langle S, \rho \rangle \rightarrow_m \rho'}$$

The monitor is at least as permissive

All runs of typable programs are accepted by the monitor.

- Theorem:
  If $\Gamma \vdash S : \tau \text{ cmd}$ and $\langle S, \rho \rangle \to \rho'$ then $\langle S, \rho \rangle \to_m \rho'$.

The monitor is more permissive I (accepts more programs)

(using the studied type system...)

There are non-typable programs whose runs (all of them) are accepted by the monitor.

- Proposition:
  There exists S such that $\Gamma \not\vdash S : \tau$ cmd and for all ρ such that $\langle S,\rho \rangle \rightarrow \rho'$ we also have $\langle S,\rho \rangle \rightarrow_m \rho'$.

The monitor is more permissive II (accepts more executions)

(using the studied type system...)

There are non-typable programs whose runs (only some of them) are accepted by the monitor.

- Proposition:
  There exists S such that $\Gamma \nvdash S : \tau$ cmd and for some ρ_1 such that $<S,\rho_1> \rightarrow \rho_1'$ we also have $<S,\rho_1> \rightarrow_m \rho_1'$, and for some ρ_2 such that $<S,\rho_2>$ diverges.

The monitor is sound (w.r.t. Deterministic Input-Output NI)

- Theorem:
  For every program S, security level L and memories ρ_1 and ρ_2 such that $\rho_1 \sim_L \rho_2$, we have that
  $\langle S,\rho_1 \rangle \rightarrow_m \rho_1'$ and $\langle S,\rho_2 \rangle \rightarrow_m \rho_2'$
  implies $\rho_1' \sim_L \rho_2'$.

For any monitored programs, its set of executions is secure.

Conclusions

- We have formally defined a monitor that enforces Noninterference.
- The monitor prevents implicit leaks by transforming them into non-terminating computations.
- The monitor is more precise than the type system we studied, since it accepts some executions of programs that are statically rejected.
- Note however that there exist type systems that are more permissive than all sound monitors.