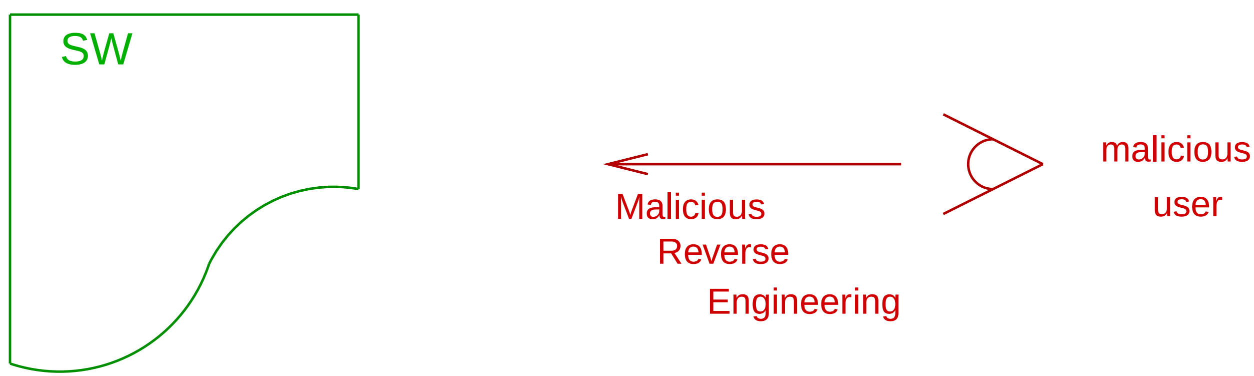
A SEMANTICS-BASED APPROACH TO CODE OBFUSCATION

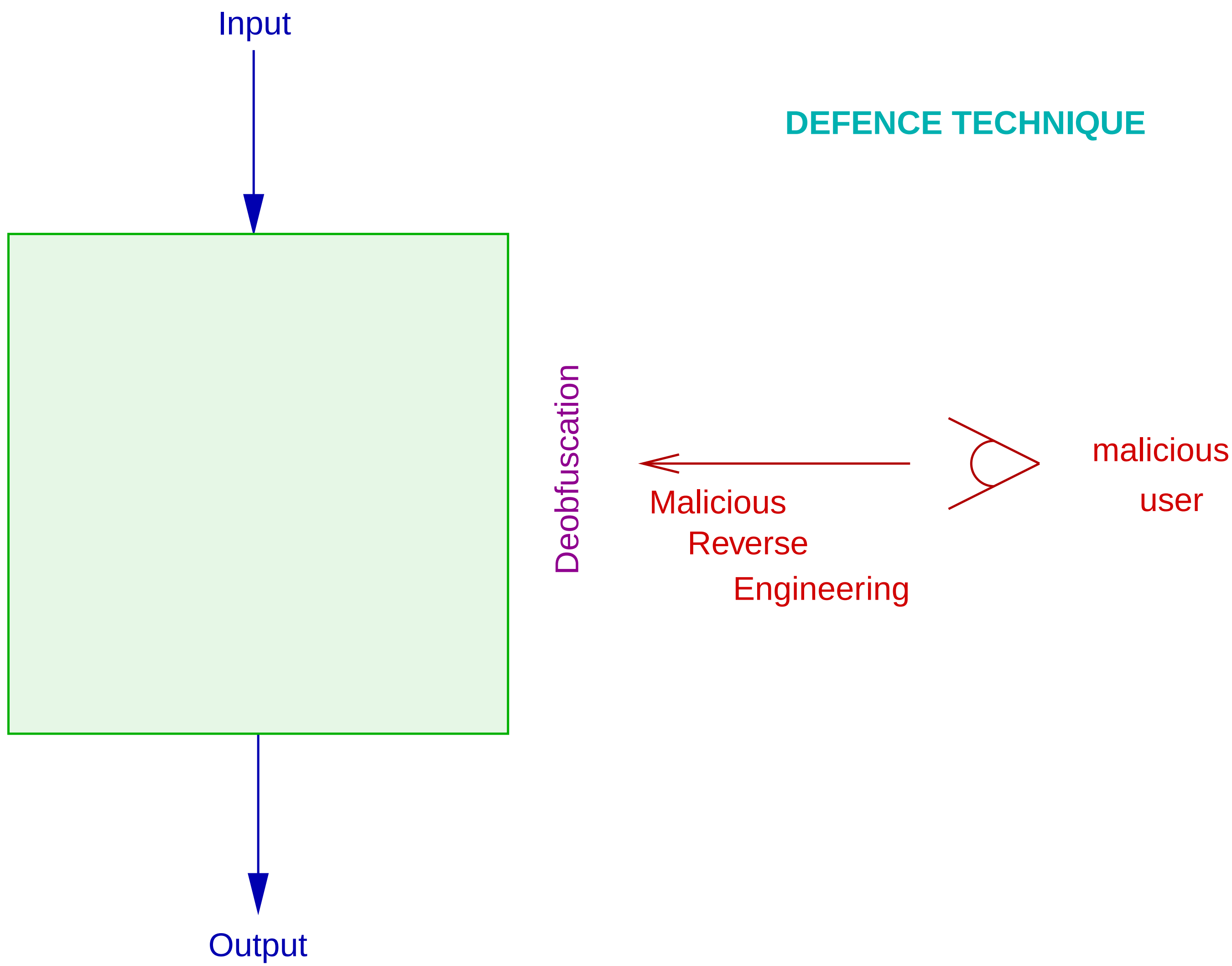
Mila Dalla Preda
University of Verona, Italy

19 December 2006, RE-TRUST meeting

Code Obfuscation



Code Obfuscation



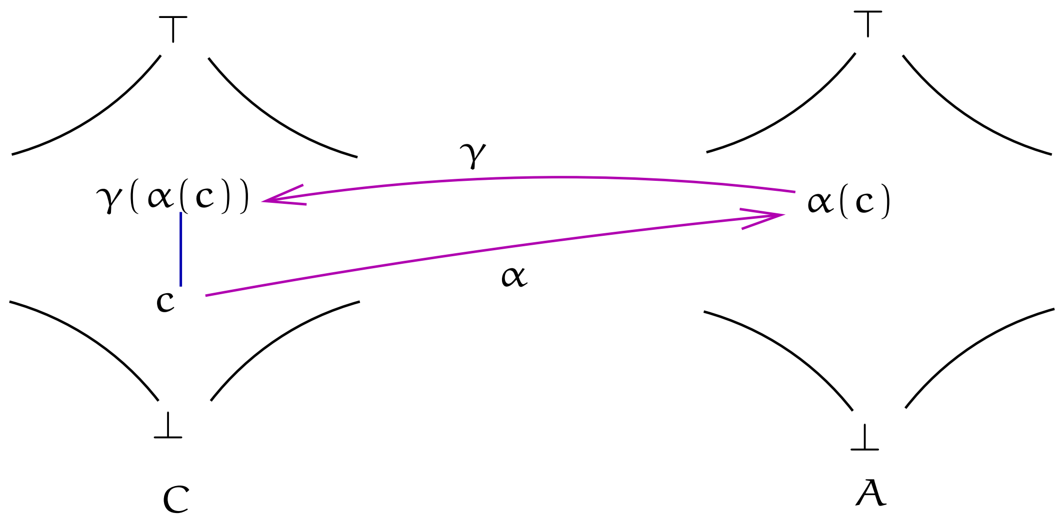
Motivation

- Drawback of Code Obfuscation: weak theoretical basis
  $\Rightarrow$ difficult to formally study and certify effectiveness

Goal: investigate the semantic effects of code obfuscation
in order to provide a formal definition of code obfuscation
based on program semantics and abstract interpretation

Abstract Interpretation

Design approximate semantics of programs [Cousot & Cousot '77, '79].



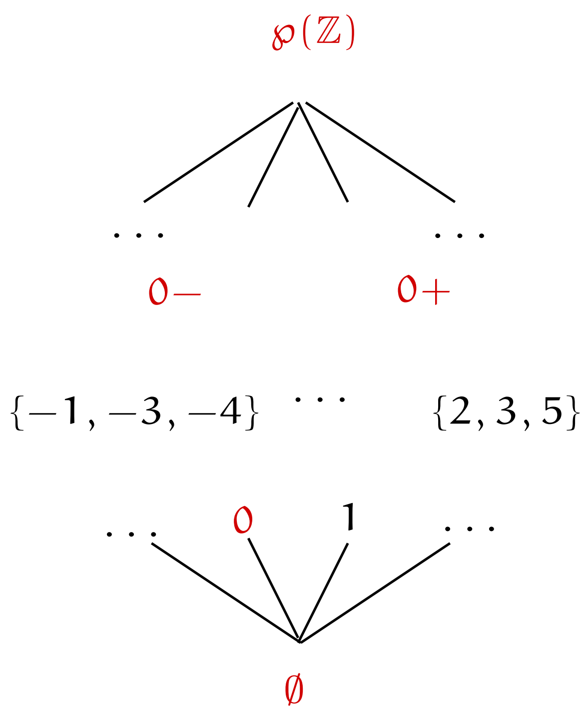
Galois Connection: $\langle C, \alpha, \gamma, A\rangle$, A and C are complete lattices.

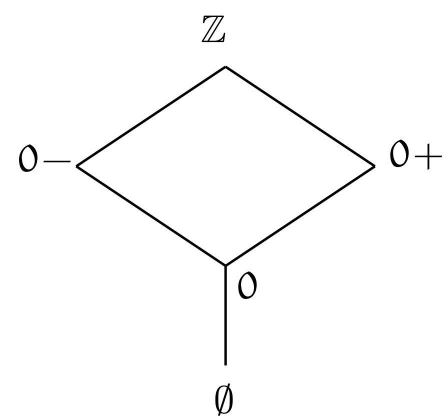
$\langle \mathit{Abs}(C), \sqsubseteq\rangle$ set of all possible abstract domains,

$A_1 \sqsubseteq A_2$ if A_1 is more concrete than A_2

Example

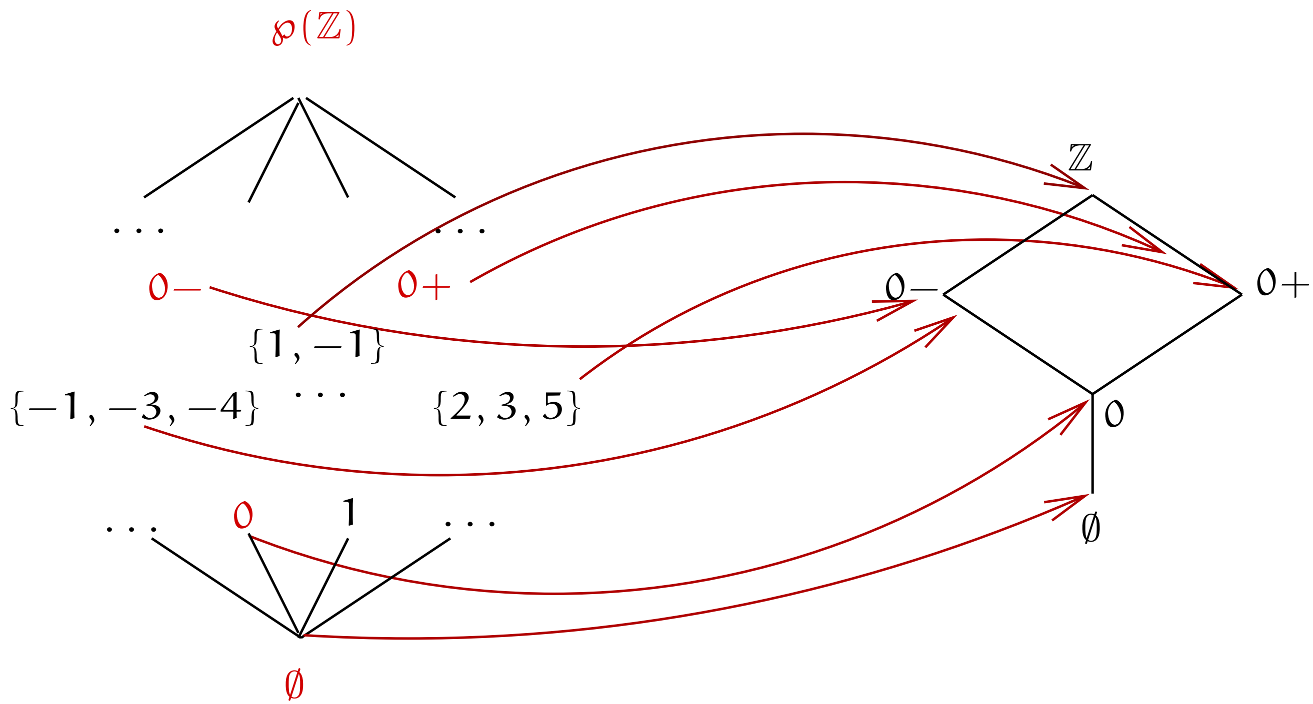
$\mathcal{S}ign$ is an abstraction of $\wp(\mathbb{Z})$:



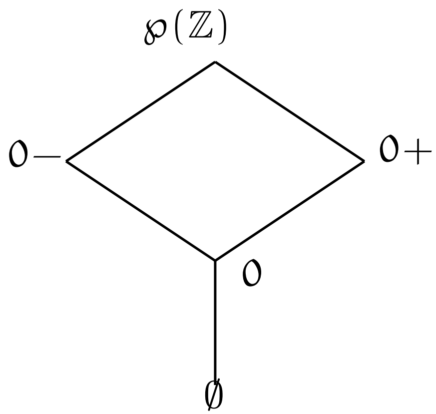


Example

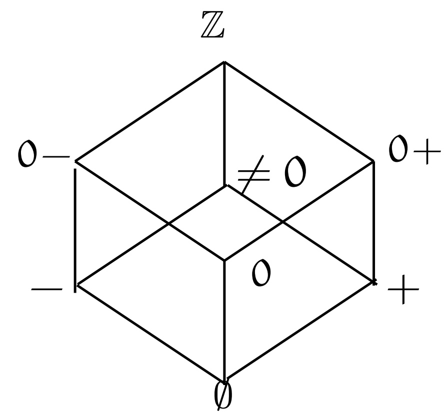
Sign is an abstraction of $\wp(\mathbb{Z})$:



Compare Abstractions



Sign

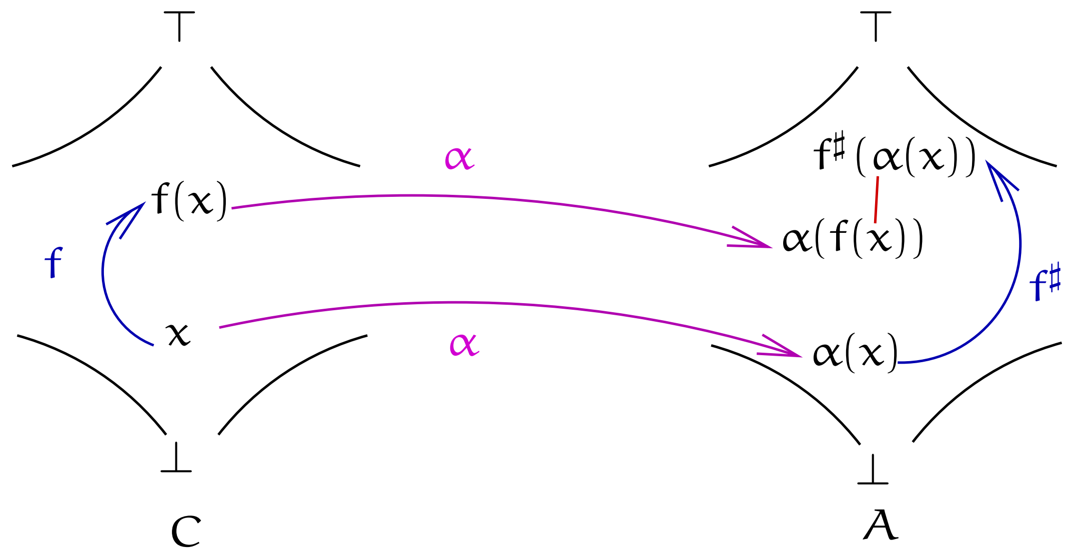


Sign^+

- *Sign* is more abstract than Sign^+: $\mathit{Sign}^+ \sqsubseteq \mathit{Sign}$

Soundness

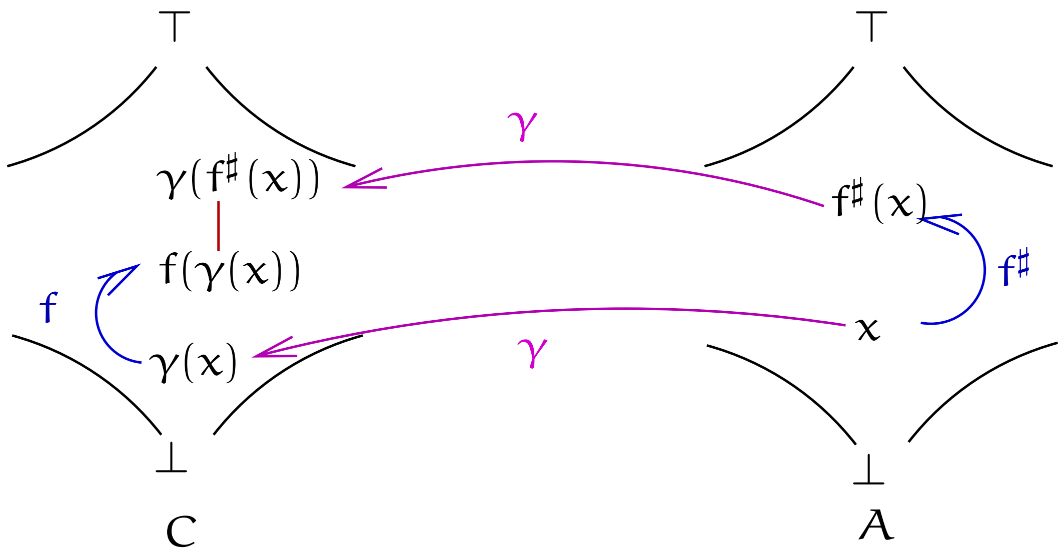
Let $\langle A, \alpha, \gamma, C \rangle$ be a GC, $f : C \to C$ and $f^\sharp : A \to A$, then:



Soundness: $\alpha \circ f(x) \leq_A f^\sharp \circ \alpha(x)$

Soundness

Let $\langle A, \alpha, \gamma, C \rangle$ be a GC, $f : C \longrightarrow C$ and $f^\sharp : A \longrightarrow A$, then:



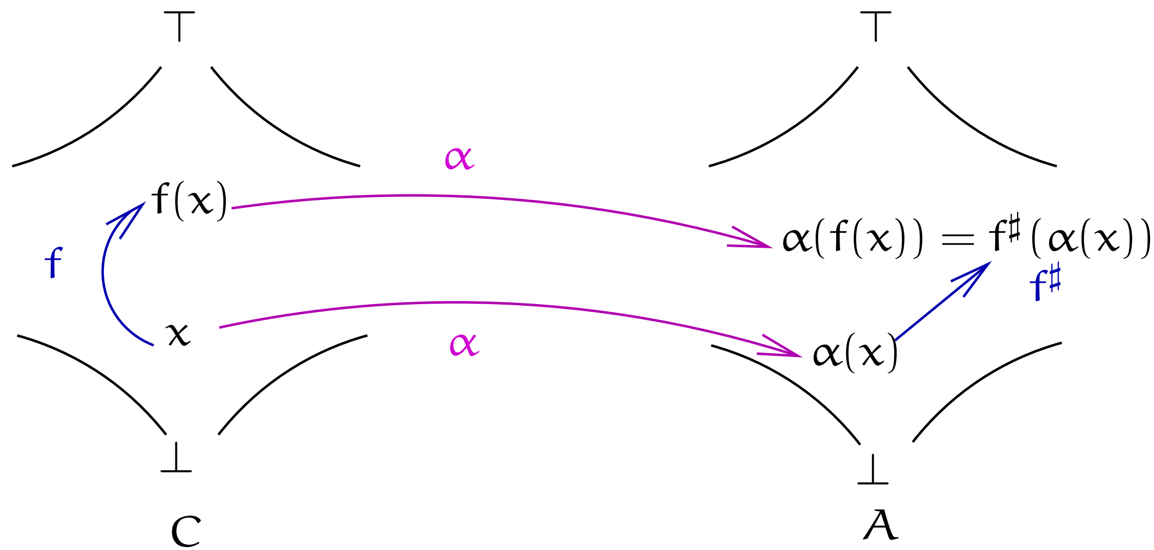
Soundness: $\alpha \circ f(x) \leq_A f^\sharp \circ \alpha(x)$

Soundness: $f \circ \gamma(x) \leq_C \gamma \circ f(x)$

Best Correct Approximation: $\alpha \circ f \circ \gamma$

Backward-Completeness

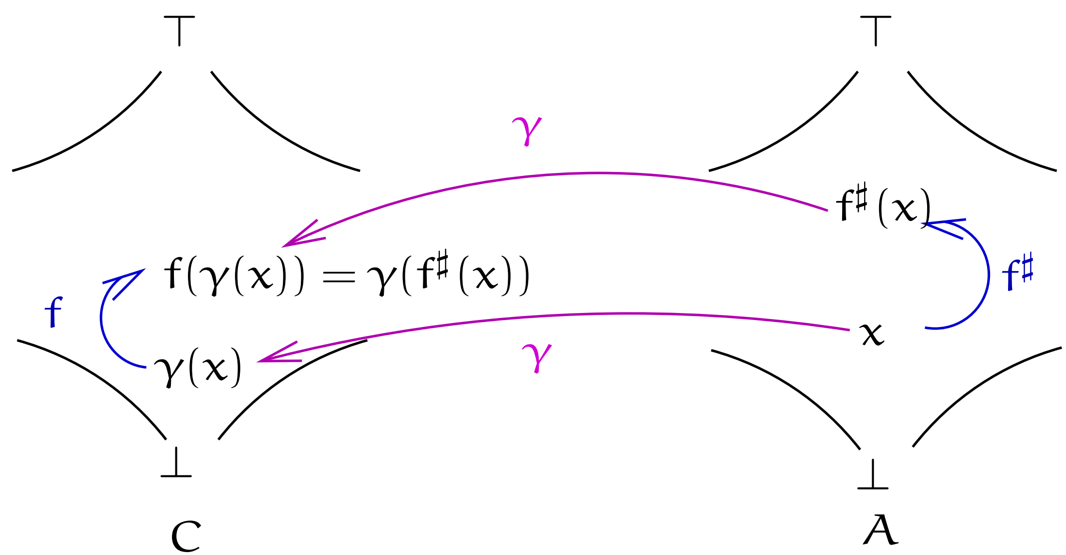
Let $\langle A, \alpha, \gamma, C \rangle$ be a GC, $f : C \to C$ and $f^\sharp : A \to A$, then:



$\mathcal{B}$-Completeness: $\alpha \circ f(x) = f^\sharp \circ \alpha(x)$

Forward-Completeness

Let $\langle A, \alpha, \gamma, C \rangle$ be a GC, $f : C \to C$ and $f^\sharp : A \to A$, then:

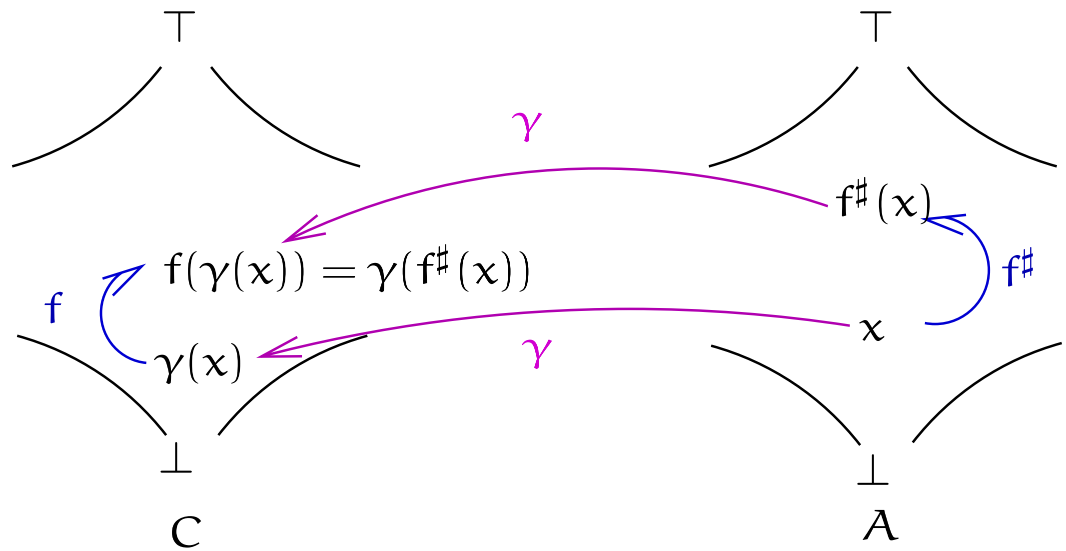


$\mathcal{B}$-Completeness: $\alpha \circ f(x) = f^\sharp \circ \alpha(x)$

$\mathcal{F}$-Completeness: $f \circ \gamma(x) = \gamma \circ f^\sharp(x)$

Forward-Completeness

Let $\langle A, \alpha, \gamma, C \rangle$ be a GC, $f : C \longrightarrow C$ and $f^\sharp : A \longrightarrow A$, then:



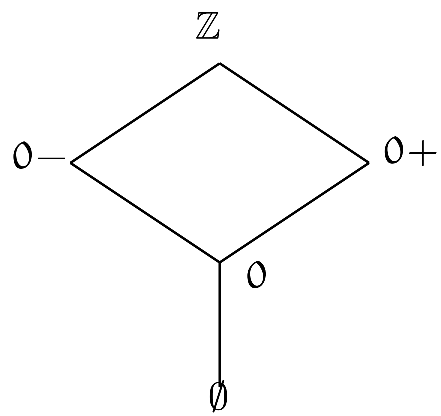
$\mathcal{B}$-Completeness: $\alpha \circ f(x) = f^\sharp \circ \alpha(x)$

$\mathcal{F}$-Completeness: $f \circ \gamma(x) = \gamma \circ f^\sharp(x)$

How to get complete domains? Completeness domain refinement

[Giacobazzi et al. 2000]

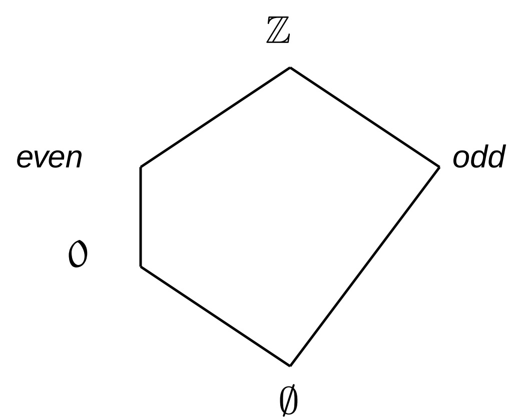
Example



Sign

- $\mathit{Sign}(10) \oplus \mathit{Sign}(-5) = 0+ \ \oplus 0- = \mathbb{Z}$
- $\mathit{Sign}(10 + (-5)) = \mathit{Sign}(5) = 0+$
- *Sign* is not $\mathcal{B}$-complete for addition

Example



Parity

- $\textit{Parity}(10) \oplus \textit{Parity}(-5) = \textit{even} \oplus \textit{odd} = \textit{odd}$
- $\textit{Parity}(10 + (-5)) = \textit{Parity}(5) = \textit{odd}$
- *Parity* is $\mathcal{B}$-complete for addition

Program Transformation

[Cousot & Cousot POPL'02]

$$\begin{tikzcd}[column sep=huge, row sep=huge]
\begin{array}{c}\text{Subject}\\ \text{program } P\end{array} \arrow[rr, "\text{Syntactic}", "\text{transformation } \mathbb{T}"'] \arrow[d, shift left, "S"] & & \begin{array}{c}\text{Transformed}\\ \text{program } \mathbb{T}[\![P]\!]\end{array} \arrow[d, shift left, "S"] \\
\begin{array}{c}\text{Subject program}\\ \text{semantics } S[\![P]\!]\end{array} \arrow[u, shift left, "p"] \arrow[rr, "\text{Semantic}", "\text{transformation } t"'] & & \begin{array}{c}\text{Transformed program}\\ \text{semantics } t[S[\![P]\!]] \sqsubseteq S[\![\mathbb{T}[\![P]\!]]\!]\end{array} \arrow[u, shift left, "p"]
\end{tikzcd}$$

Syntactic transformation: $\mathbb{T} = p \circ t \circ S$

Code Obfuscation

[C. Collberg et al. '97, '98]

$\mathbb{T} : \mathbb{P} \rightarrow \mathbb{P}$ is a code obfuscation if:

- $\mathbb{T}$ is potent, i.e. $\mathbb{T}[\![\mathrm{P}]\!]$ is more complex than P
- $\mathbb{T}$ preserves the observational behaviour of programs, i.e. the input-output behaviour (denotational semantics *DenSem*)

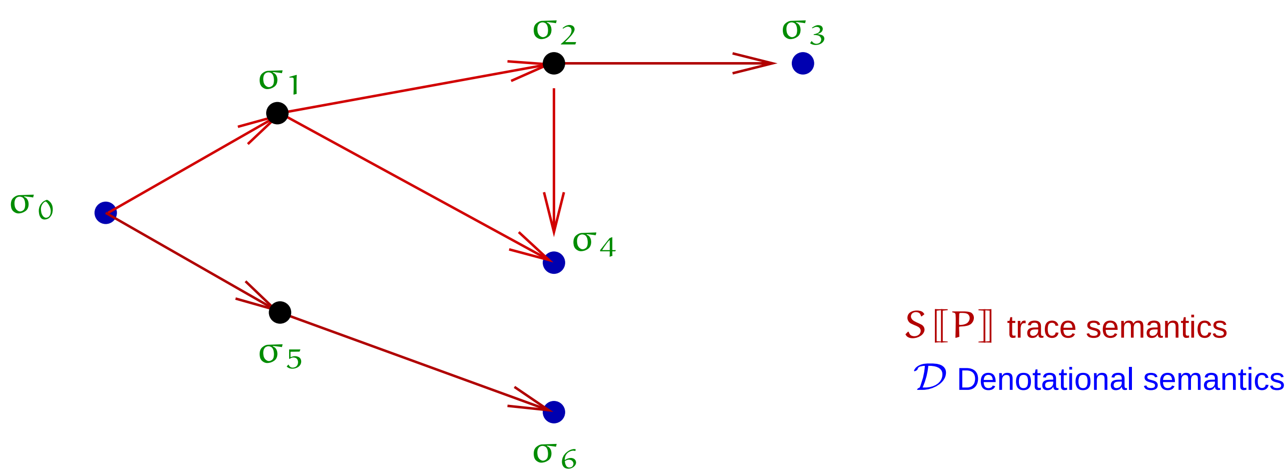
Code Obfuscation

[C. Collberg et al. ’97, ’98]

$\mathbb{T} : \mathbb{P} \rightarrow \mathbb{P}$ is a code obfuscation if:

- $\mathbb{T}$ is potent, i.e. $\mathbb{T}[\![\mathrm{P}]\!]$ is more complex than P
- $\mathbb{T}$ preserves the observational behaviour of programs, i.e. the input-output behaviour (denotational semantics *DenSem*)

AI describes the relation among semantics at different levels of abstraction: Hierarchy of semantics defined by [Cousot ’00]:



Code Obfuscation and Program Semantics

[M. Dalla Preda and R. Giacobazzi ICALP'05]

$\mathbb{T} : \mathbb{P} \to \mathbb{P}$ is potent if there is a property $\varphi \in \mathit{Abs}(\mathit{Sem})$ such that:

$$\varphi(S[\![P]\!]) \neq \varphi(S[\![\mathbb{T}[\![P]\!]]\!])$$

Code Obfuscation and Program Semantics

[M. Dalla Preda and R. Giacobazzi ICALP'05]

$\mathbb{T} : \mathbb{P} \to \mathbb{P}$ is potent if there is a property $\varphi \in \mathit{Abs}(\mathit{Sem})$ such that:

$$\varphi(S[\![P]\!]) \neq \varphi(S[\![\mathbb{T}[\![P]\!]]\!])$$

The most concrete property preserved by $\mathbb{T}$ is:

$$\delta_{\mathbb{T}} = \sqcap \left\{ \varphi \in \mathit{Abs}(\mathit{Sem}) \;\middle|\; \varphi(S[\![P]\!]) = \varphi(S[\![\mathbb{T}[\![P]\!]]\!]) \right\}$$

Code Obfuscation and Program Semantics

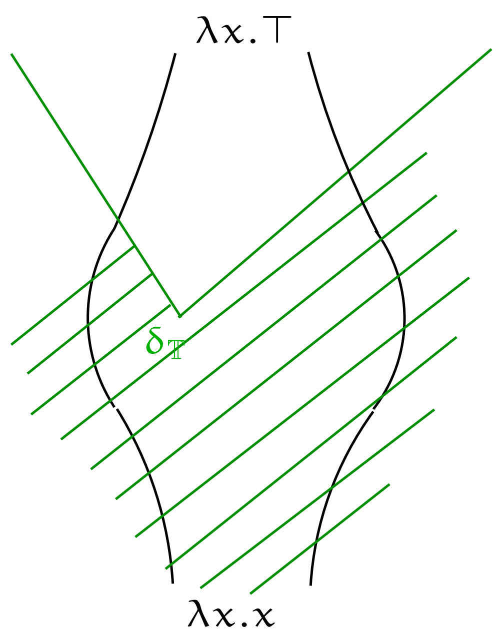
[M. Dalla Preda and R. Giacobazzi ICALP'05]

$\mathbb{T} : \mathbb{P} \to \mathbb{P}$ is potent if there is a property $\varphi \in \mathit{Abs}(\mathit{Sem})$ such that:

$$\varphi(S[\![P]\!]) \neq \varphi(S[\![\mathbb{T}[\![P]\!]]\!])$$

The most concrete property preserved by $\mathbb{T}$ is:

$$\delta_{\mathbb{T}} = \sqcap \left\{ \varphi \in \mathit{Abs}(\mathit{Sem}) \;\middle|\; \varphi(S[\![P]\!]) = \varphi(S[\![\mathbb{T}[\![P]\!]]\!]) \right\}$$

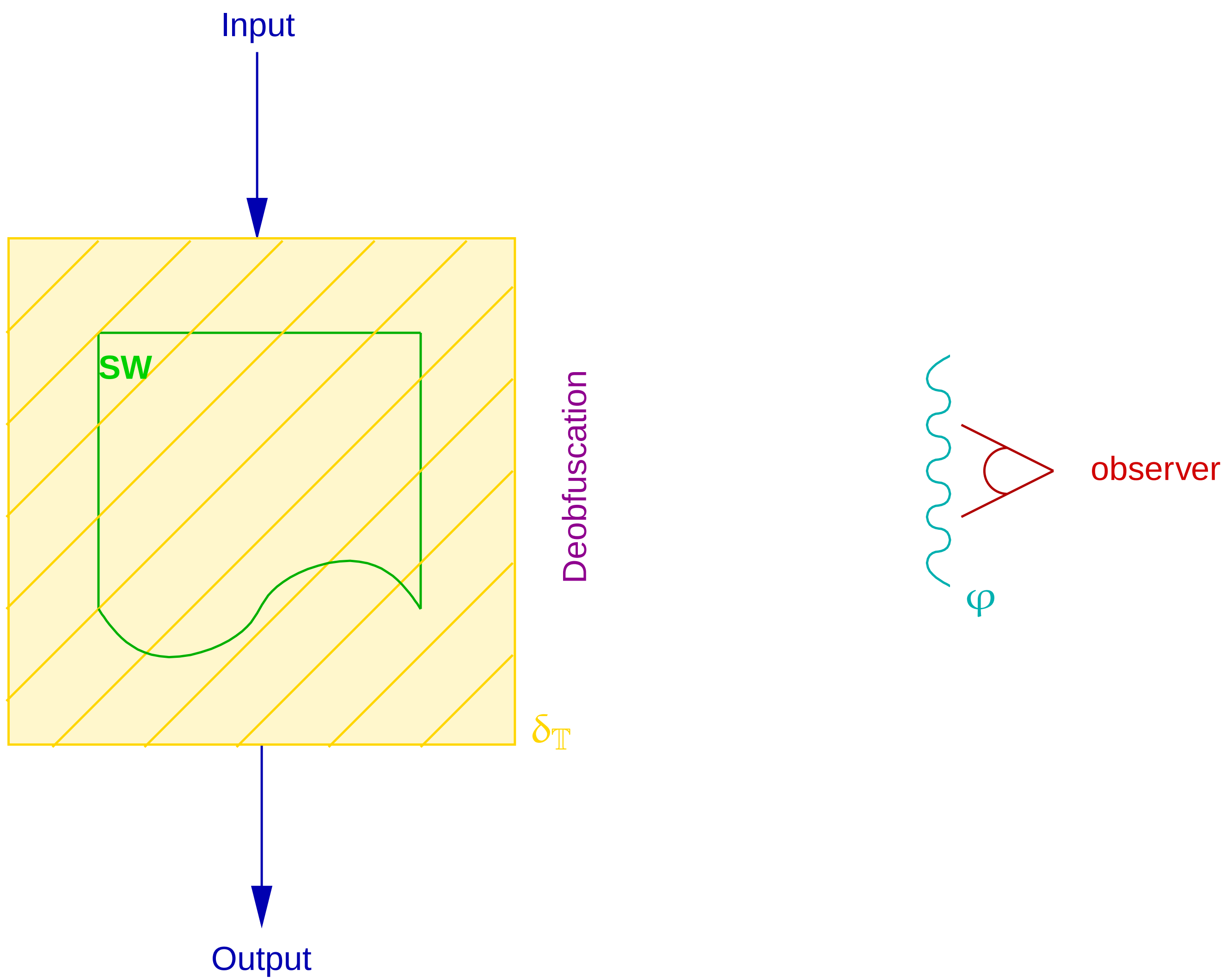


$$O_{\delta_{\mathbb{T}}} = \left\{ \varphi \;\middle|\; \varphi \ominus (\varphi \sqcup \delta_{\mathbb{T}}) \neq \top \right\}$$

$$O_{\delta_{\mathbb{T}}} = \left\{ \varphi \;\middle|\; \delta_{\mathbb{T}} \not\sqsubseteq \varphi \right\}$$

Semantic Code Obfuscation

[M. Dalla Preda and R. Giacobazzi ICALP'05]



Semantic Code Obfuscation

[M. Dalla Preda and R. Giacobazzi ICALP'05]

Semantics-based Definition

$\mathbb{T} : \mathbb{P} \to \mathbb{P}$ is a $\delta-$obfuscator if:

$\delta_{\mathbb{T}} = \delta$ and $O_{\delta} \neq \varnothing$

Semantic Code Obfuscation

[M. Dalla Preda and R. Giacobazzi ICALP'05]

Semantics-based Definition

$\mathbb{T} : \mathbb{P} \longrightarrow \mathbb{P}$ is a $\delta-$obfuscator if:

$\delta_{\mathbb{T}} = \delta$ and $O_\delta \neq \varnothing$

- EXAMPLE: $X \rightsquigarrow 2X$ obfuscates the parity and preserves the sign of X

Semantic Code Obfuscation

[M. Dalla Preda and R. Giacobazzi ICALP'05]

Semantics-based Definition

$\mathbb{T} : \mathbb{P} \longrightarrow \mathbb{P}$ is a $\delta-$obfuscator if:

$\delta_{\mathbb{T}} = \delta$ and $\mathrm{O}_{\delta} \neq \varnothing$

- EXAMPLE: $X \rightsquigarrow 2X$ obfuscates the parity and preserves the sign of X
- Collberg's Obfuscators $= \left\{ \delta - \text{obfuscators} \,\middle|\, \delta \sqsubseteq \textit{DenSem} \right\}$

Semantic Code Obfuscation

[M. Dalla Preda and R. Giacobazzi ICALP'05]

Semantics-based Definition

$\mathbb{T} : \mathbb{P} \to \mathbb{P}$ is a $\delta-$obfuscator if:
$\delta_{\mathbb{T}} = \delta$ and $O_\delta \neq \varnothing$

- EXAMPLE: $X \rightsquigarrow 2X$ obfuscates the parity and preserves the sign of X
- Collberg's Obfuscators $= \left\{ \delta - \text{obfuscators} \,\middle|\, \delta \sqsubseteq \textit{DenSem} \right\}$
- Compare obfuscators w.r.t. potency: $\mathbb{T}_1$ is more potent than $\mathbb{T}_2$ iff $\delta_2 \sqsubseteq \delta_1$

Semantic Code Obfuscation

[M. Dalla Preda and R. Giacobazzi ICALP'05]

Semantics-based Definition

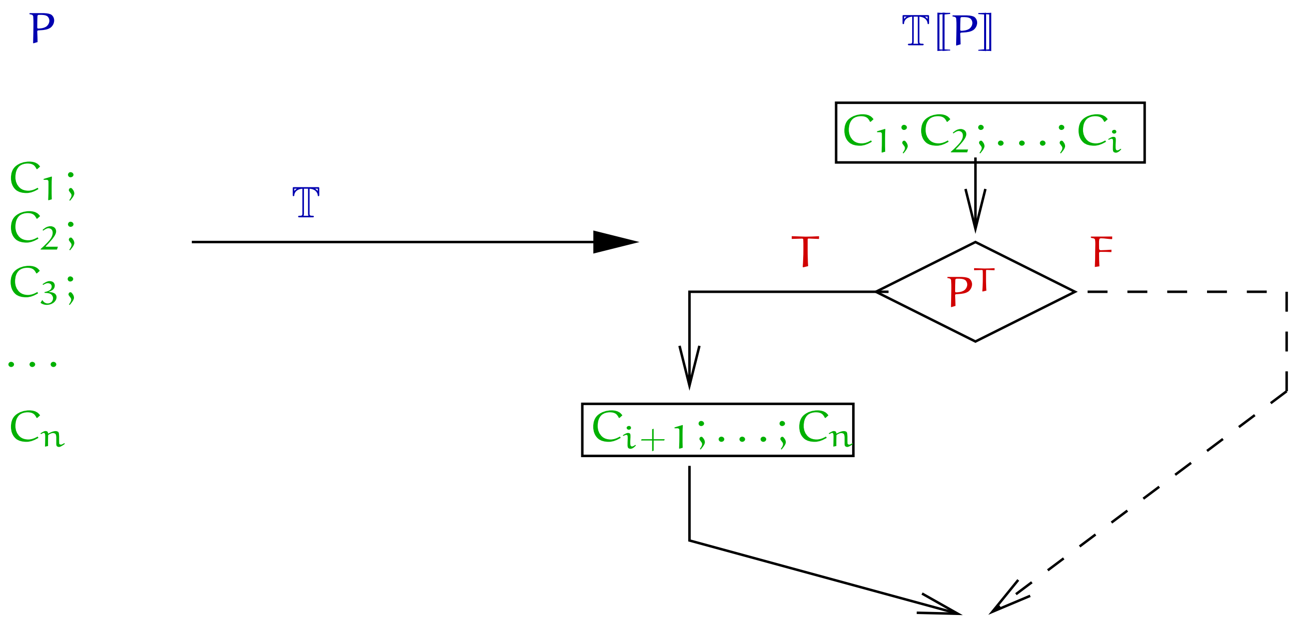
$\mathbb{T} : \mathbb{P} \to \mathbb{P}$ is a $\delta-$obfuscator if:

$\delta_{\mathbb{T}} = \delta$ and $O_{\delta} \neq \varnothing$

- EXAMPLE: $X \rightsquigarrow 2X$ obfuscates the parity and preserves the sign of X
- Collberg's Obfuscators $= \left\{ \delta - \text{obfuscators} \,\middle|\, \delta \sqsubseteq \textit{DenSem} \right\}$
- Compare obfuscators w.r.t. potency: $\mathbb{T}_1$ is more potent than $\mathbb{T}_2$ iff $\delta_2 \sqsubseteq \delta_1$
- Constructive characterization of $\delta_{\mathbb{T}}$

Control Code Obfuscation

Control Code Obfuscation affects the control flow of the program often by inserting opaque predicates.



Attackers and Completeness

- Precise detection of Opaque Predicates:

  An attacker φ breaks an opaque predicate *Pr*
  if φ recognises the always true value of *Pr*

Attackers and Completeness

- Precise detection of Opaque Predicates:

  An attacker φ breaks an opaque predicate *Pr*
  if φ recognises the always true value of *Pr*

- $\forall x \in \mathbb{Z} : g(x) = h(x)$

  [M. Dalla Preda and R. Giacobazzi SEFM'05]: if attacker φ is $\mathcal{F}$-complete for g and h, then φ breaks the opaque predicate $\forall x \in \mathbb{Z} : g(x) = h(x)$

- $\forall x \in \mathbb{Z} : n \textit{ mod } f(x)$

  [M. Dalla Preda and R. Giacobazzi AMAST'06]: if attacker φ is $\mathcal{B}$-complete for the elementary functions composing f then φ breaks the opaque predicate $\forall x \in \mathbb{Z} : n \textit{ mod } f(x)$

Attackers and Completeness

- Precise detection of Opaque Predicates:

  An attacker φ breaks an opaque predicate *Pr*
  if φ recognises the always true value of *Pr*

- $\forall x \in \mathbb{Z} : g(x) = h(x)$

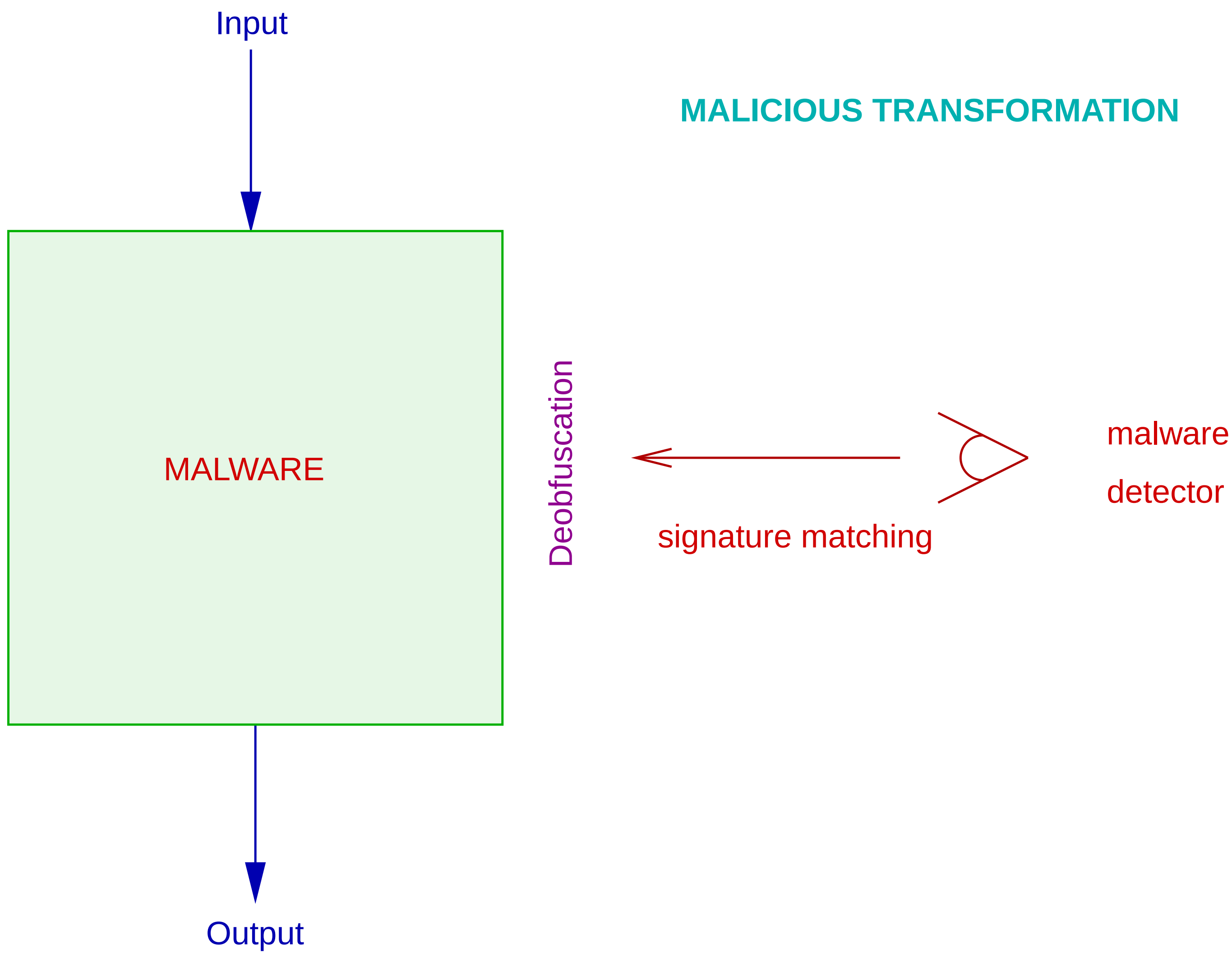
  [M. Dalla Preda and R. Giacobazzi SEFM'05]: if attacker φ is $\mathcal{F}$-complete for g and h, then φ breaks the opaque predicate $\forall x \in \mathbb{Z} : g(x) = h(x)$

- $\forall x \in \mathbb{Z} : n \textit{ mod } f(x)$

  [M. Dalla Preda and R. Giacobazzi AMAST'06]: if attacker φ is $\mathcal{B}$-complete for the elementary functions composing f then φ breaks the opaque predicate $\forall x \in \mathbb{Z} : n \textit{ mod } f(x)$

- consider OP_1 and OP_2: if $\mathcal{C}_{OP_1}(\varphi) \sqsubseteq \mathcal{C}_{OP_2}(\varphi)$, then OP_1 is more resilient than OP_2 in contrasting φ

Obfuscation and Malware



Semantic Malware Detector

[M. Dalla Preda et al. POPL'07]

A program P is infected by malware M, denoted $M \hookrightarrow P$
if (a part) of P execution is similar to that of M:

Semantic Malware Detector

[M. Dalla Preda et al. POPL'07]

A program P is infected by malware M, denoted $M \hookrightarrow P$ if (a part) of P execution is similar to that of M:

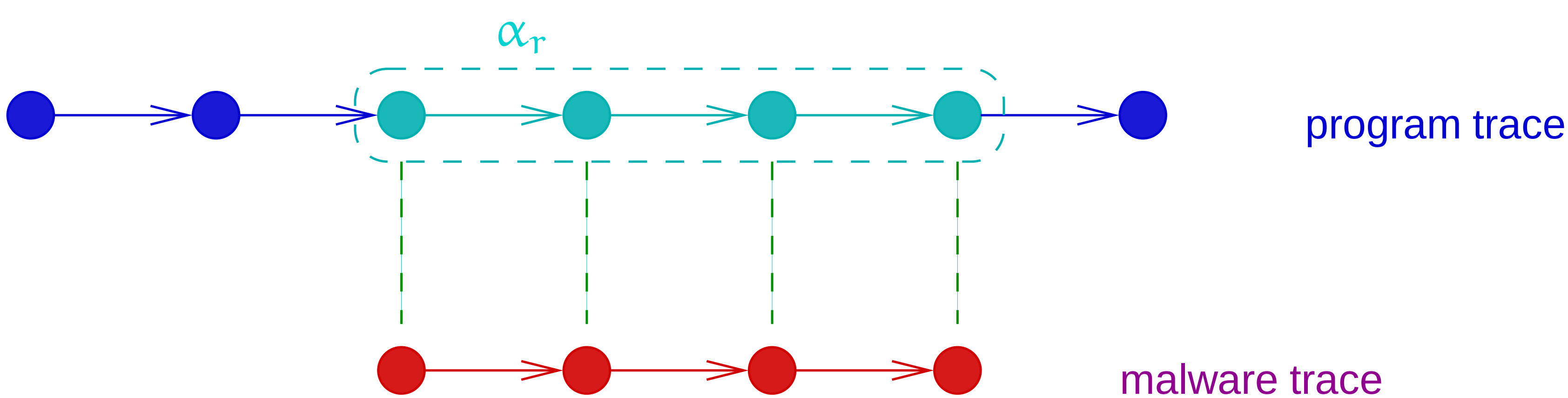
$$S[\![M]\!] \quad \subseteq \quad S[\![P]\!]$$

Semantic Malware Detector

[M. Dalla Preda et al. POPL'07]

A program P is infected by malware M, denoted $M \hookrightarrow P$ if (a part) of P execution is similar to that of M:

$$\exists \text{ restriction } r : S[\![M]\!] \subseteq \alpha_r(S[\![P]\!])$$

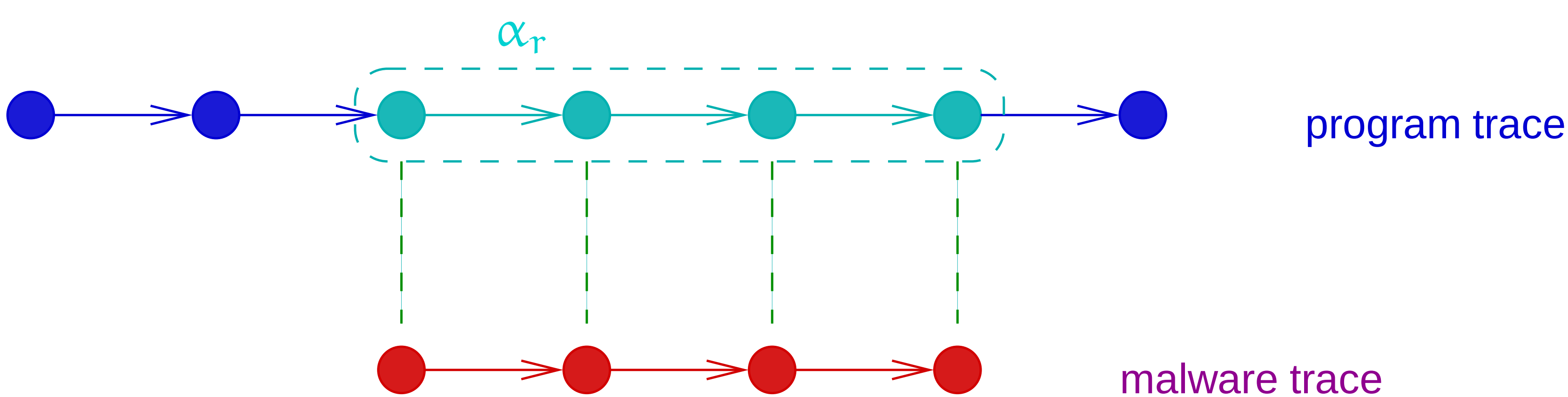


Semantic Malware Detector

[M. Dalla Preda et al. POPL'07]

A program P is infected by malware M, denoted $M \hookrightarrow P$ if (a part) of P execution is similar to that of M:

$$\exists \text{ restriction } r : S[\![M]\!] \subseteq \alpha_r(S[\![P]\!])$$



Vanilla Malware i.e. not obfuscated malware

Obfuscated Malware

[M. Dalla Preda et al. POPL'07]

- $\mathcal{O} : \mathbb{P} \to \mathbb{P}$ obfuscating transformation
- $\alpha_{\mathcal{O}}$: *Sem* $\to A$ abstraction that discards the details changed by the obfuscation while preserving maliciousness

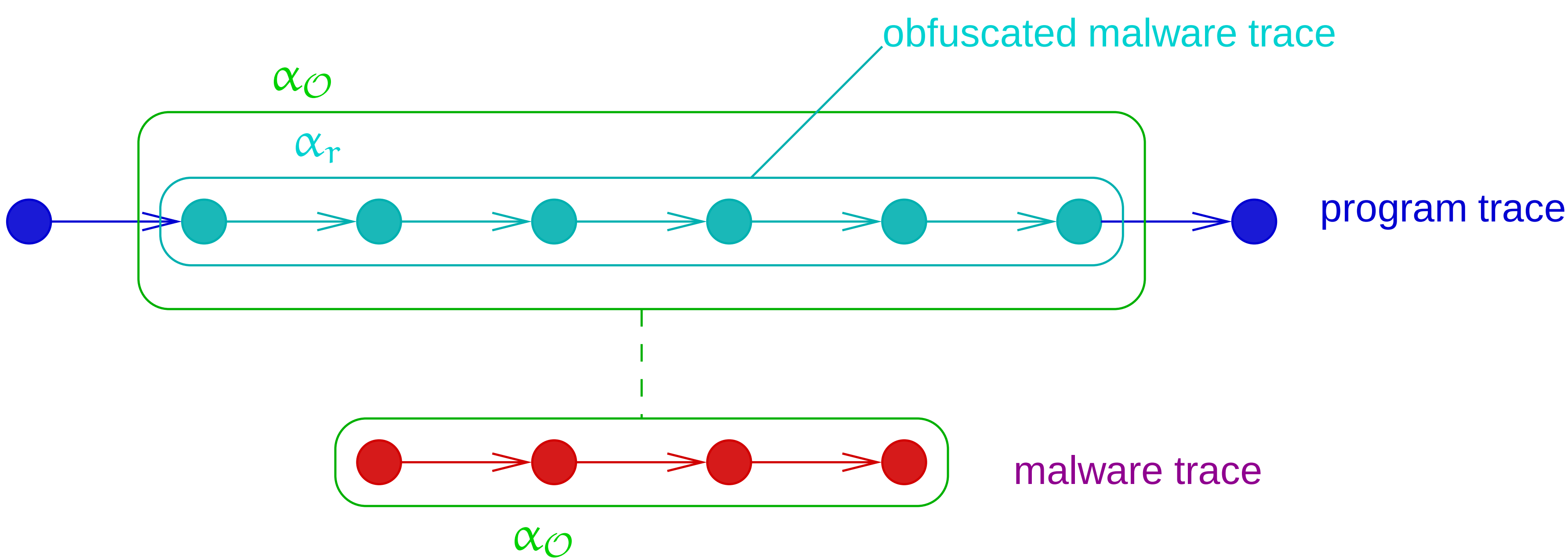
$$\exists \text{ restriction } r : \alpha_{\mathcal{O}}(S[\![M]\!]) \subseteq \alpha_{\mathcal{O}}(\alpha_r(S[\![P]\!]))$$

Obfuscated Malware

[M. Dalla Preda et al. POPL'07]

- $\mathcal{O} : \mathbb{P} \to \mathbb{P}$ obfuscating transformation
- $\alpha_{\mathcal{O}} : \mathit{Sem} \to A$ abstraction that discards the details changed by the obfuscation while preserving maliciousness

$$\exists \text{ restriction } r : \alpha_{\mathcal{O}}(S[\![M]\!]) \subseteq \alpha_{\mathcal{O}}(\alpha_r(S[\![P]\!]))$$



Sound vs Complete

[M. Dalla Preda et al. POPL'07]

- Precision of the SMD depends on the choice of $\alpha_{\mathcal{O}}$

Sound vs Complete

[M. Dalla Preda et al. POPL'07]

- Precision of the SMD depends on the choice of $\alpha_{\mathcal{O}}$
- A SMD on $\alpha_{\mathcal{O}}$ is complete w.r.t. a set $\mathbb{O}$ of transformations if $\forall \mathcal{O} \in \mathbb{O}$:

$$\mathcal{O}(M) \hookrightarrow P \Rightarrow \begin{cases} \exists \text{ restriction } r : \\ \alpha_{\mathcal{O}}(S[\![M]\!]) \subseteq \alpha_{\mathcal{O}}(\alpha_r(S[\![P]\!])) \end{cases}$$

always detects programs that are infected (no false negatives)

Sound vs Complete

[M. Dalla Preda et al. POPL'07]

- Precision of the SMD depends on the choice of $\alpha_{\mathcal{O}}$
- A SMD on $\alpha_{\mathcal{O}}$ is complete w.r.t. a set $\mathbb{O}$ of transformations if $\forall \mathcal{O} \in \mathbb{O}$:

$$\mathcal{O}(M) \hookrightarrow P \Rightarrow \begin{cases} \exists \text{ restriction } r : \\ \alpha_{\mathcal{O}}(S[\![M]\!]) \subseteq \alpha_{\mathcal{O}}(\alpha_r(S[\![P]\!])) \end{cases}$$

  always detects programs that are infected (no false negatives)

- A SMD on $\alpha_{\mathcal{O}}$ is sound w.r.t. a set $\mathbb{O}$ of transformations if:

$$\left.\begin{array}{l} \exists \text{ restriction } r : \\ \alpha_{\mathcal{O}}(S[\![M]\!]) \subseteq \alpha_{\mathcal{O}}(\alpha_r(S[\![P]\!])) \end{array}\right\} \Rightarrow \exists \mathcal{O} \in \mathbb{O} : \mathcal{O}(M) \hookrightarrow P$$

  never erroneously claims a program is infected (no false positives)

Main Results

[M. Dalla Preda et al. POPL'07]

- Classification of obfuscating transformations w.r.t. their effects on program semantics
  - conservative
  - non-conservative
- Abstraction that is both sound and complete for conservative obfuscation
- Possible strategies in order to handle non-conservative obfuscations
- Flexibility of the abstract interpretation-based approach
- Prove completeness of the semantics-aware malware detector [Christodorescu et al. 2005]

Open issues (related to RE-TRUST)

- Metamorphic viruses and monitor factory
- Composition of elementary obfuscations
- The checker may verify the satisfaction of some properties of program behaviour (abstract semantics)

Thank you!

Sound vs Complete

[M. Dalla Preda et al. POPL'07]

- If $\alpha_{\mathcal{O}}$ is preserved by $\mathcal{O}$ then the SMD on $\alpha_{\mathcal{O}}$ is complete w.r.t. $\mathcal{O}$:

$$\forall P \in \mathbb{P} : \alpha_{\mathcal{O}}(S[\![P]\!]) = \alpha_{\mathcal{O}}(S[\![\mathcal{O}[\![P]\!]]\!])$$

Sound vs Complete

[M. Dalla Preda et al. POPL'07]

- If $\alpha_{\mathcal{O}}$ is preserved by $\mathcal{O}$ then the SMD on $\alpha_{\mathcal{O}}$ is complete w.r.t. $\mathcal{O}$:

$$\forall P \in \mathbb{P} : \alpha_{\mathcal{O}}(S[\![P]\!]) = \alpha_{\mathcal{O}}(S[\![\mathcal{O}[\![P]\!]]\!])$$

- Given an abstraction α, consider the set $\mathbb{O}$ of transformations such that $\forall P, T \in \mathbb{P}$:

$$(\alpha(S[\![T]\!])) \subseteq \alpha(S[\![P]\!])) \Rightarrow (\exists \mathcal{O} \in \mathbb{O} : S[\![\mathcal{O}[\![T]\!]]\!]) \subseteq S[\![P]\!])$$

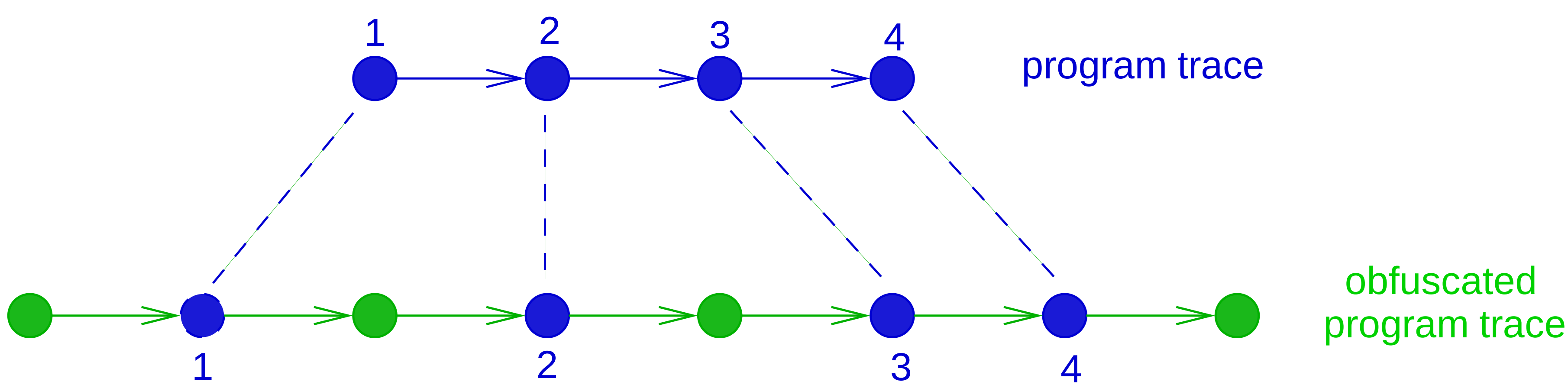
then, the SMD on α is sound w.r.t. $\mathbb{O}$

Classifying Transformations

[M. Dalla Preda et al. POPL'07]

- $\mathcal{O} : \mathbb{P} \to \mathbb{P}$ is a conservative transformation if

  $\forall$ trace1 $\in S[\![P]\!]$, $\exists$ trace2 $\in S[\![\mathcal{O}[\![P]\!]]\!]$: trace1 is sub-sequence of trace2



- $\mathcal{O} : \mathbb{P} \to \mathbb{P}$ is a non-conservative transformation if $\mathcal{O}$ is not conservative

Conservative

[M. Dalla Preda et al. POPL'07]

Suitable abstraction for conservative transformations:

$$\alpha_c[X](Y) = X \cap \textit{SubSequences}(Y)$$

returns all the traces in X that are sub-sequences of a *trace* in Y

$$\alpha_c[S[\![M]\!]](S[\![\mathcal{O}_c[\![M]\!]]\!]) = S[\![M]\!]$$

- $\mathcal{O}_c(M) \hookrightarrow P$ iff $\exists$ restriction r: $\alpha_c[S[\![M]\!]](S[\![M]\!]) \subseteq \alpha_c[S[\![M]\!]](\alpha_r(S[\![P]\!]))$

Conservative

[M. Dalla Preda et al. POPL'07]

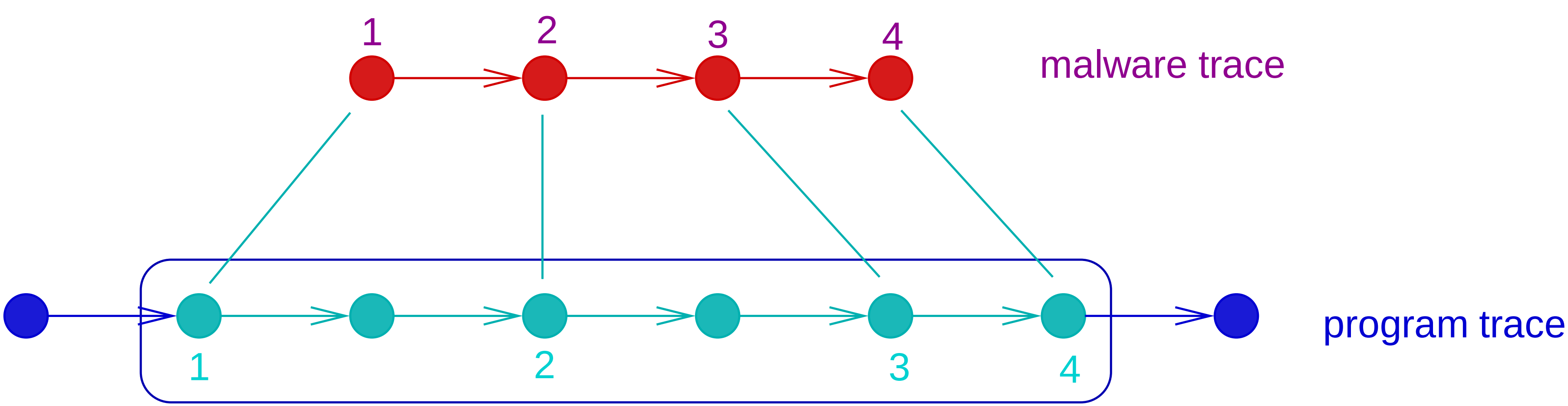
Suitable abstraction for conservative transformations:

$$\alpha_c[X](Y) = X \cap \mathit{SubSequences}(Y)$$

returns all the traces in X that are sub-sequences of a *trace* in Y

$$\alpha_c[S[\![M]\!]](S[\![\mathcal{O}_c[\![M]\!]]\!]) = S[\![M]\!]$$

- $\mathcal{O}_c(M) \hookrightarrow P$ iff $\exists$ restriction r: $\alpha_c[S[\![M]\!]](S[\![M]\!]) \subseteq \alpha_c[S[\![M]\!]](\alpha_r(S[\![P]\!]))$



Conservative

[M. Dalla Preda et al. POPL'07]

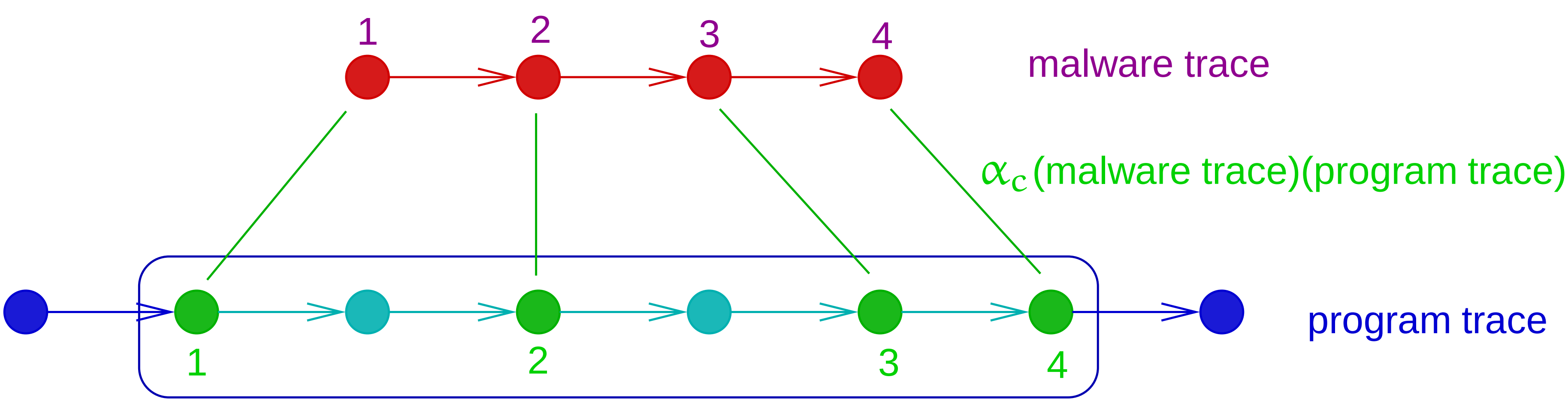
Suitable abstraction for conservative transformations:

$$\alpha_c[X](Y) = X \cap \textit{SubSequences}(Y)$$

returns all the traces in X that are sub-sequences of a *trace* in Y

$$\alpha_c[S[\![M]\!]](S[\![\mathcal{O}_c[\![M]\!]]\!]) = S[\![M]\!]$$

- $\mathcal{O}_c(M) \hookrightarrow P$ iff $\exists$ restriction r: $\alpha_c[S[\![M]\!]](S[\![M]\!]) \subseteq \alpha_c[S[\![M]\!]](\alpha_r(S[\![P]\!]))$



Conservative Transformations

[M. Dalla Preda et al. POPL'07]

- The property of being conservative is preserved by composition

Conservative Transformations

[M. Dalla Preda et al. POPL'07]

- The property of being conservative is preserved by composition

- Opaque Predicate Insertion
- Code Reordering: changes the order in which commands are written while maintaining the execution order
- Semantic NOP insertion: inserts irrelevant commands (es. $x := x + 0$)
- Substitution of equivalent commands

Example

[M. Dalla Preda et al. POPL'07]

M

L_1 :	$\mathtt{assign}(L_B, B) \rightarrow L_2$
L_2 :	$\mathtt{assign}(L_A, A) \rightarrow L_c$
L_c :	$\mathtt{cond}(A) \rightarrow \{L_T, L_F\}$
L_T :	$B := \mathrm{Dec}(A) \rightarrow L_{T_1}$
L_{T_1} :	$\mathtt{assign}(\mathit{succ}(B), B) \rightarrow L_{T_2}$
L_{T_2} :	$\mathtt{assign}(\mathit{succ}(A), A) \rightarrow L_C$
L_F	$\mathit{skip} \rightarrow L_B$

$\mathcal{O}_c[\![M]\!]$

L_1 :	$\mathtt{assign}(L_B, B) \rightarrow L_2$
L_2 :	$\mathit{skip} \rightarrow L_4$
L_c :	$\mathtt{cond}(A) \rightarrow \{L_O, L_F\}$
L_4 :	$\mathtt{assign}(L_A, A) \rightarrow L_5$
L_5 :	$\mathit{skip} \rightarrow L_c$
L_O :	$P^T \rightarrow \{L_N, L_k\}$
L_N :	$X := X - 3 \rightarrow L_{N_1}$
L_{N_1} :	$X := X + 3 \rightarrow L_T$
L_T :	$B := \mathrm{Dec}(A) \rightarrow L_{T_1}$
L_{T_1} :	$\mathtt{assign}(\mathit{succ}(B), B) \rightarrow L_{T_2}$
L_{T_2} :	$\mathtt{assign}(\mathit{succ}(A), A) \rightarrow L_c$
L_k :	$\ldots$
L_F :	$\mathit{skip} \rightarrow L_B$

Non-Conservative

[M. Dalla Preda et al. POPL'07]

- Identify the set of all possible modifications induced by a non-conservative transformation and fix a canonical one
  - Variable renaming
  - Canonical rename: V_1 first variable, V_2 second variable...
  - Apply variable renaming with canonical renaming to $\alpha_r(S[\![P]\!])$ and to $S[\![M]\!]$
  - Verify infection
  - Complete and Sound
- Derive the most concrete preserved property as seen before
- Use further abstractions

Composition

[M. Dalla Preda et al. POPL'07]

- Completeness
  - SMD on α_1 is complete for $\mathcal{O}_1$ and SMD on α_2 is complete for $\mathcal{O}_2$
  - $\alpha_1 \circ \alpha_2 = \alpha_2 \circ \alpha_2$

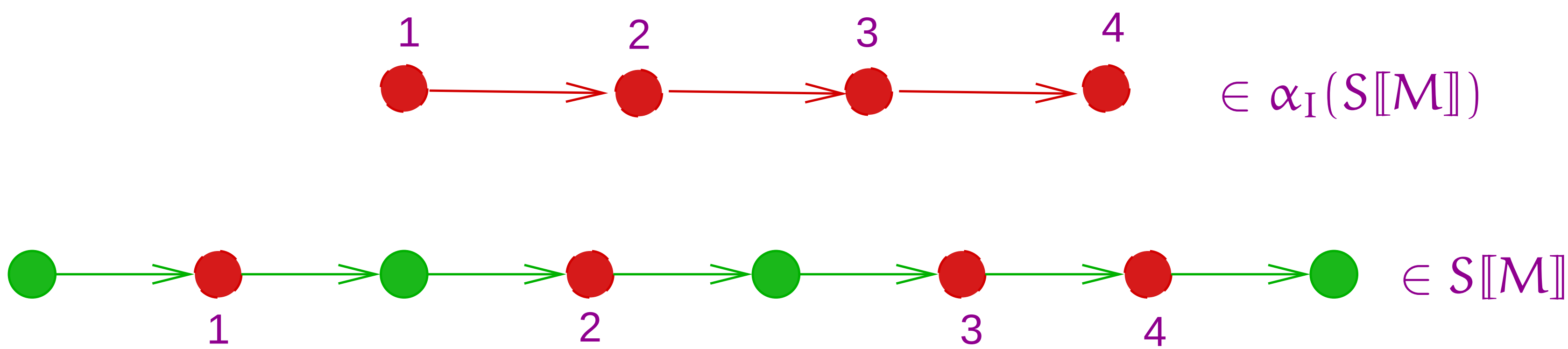
    $\Rightarrow$ SMD on $\alpha_1 \circ \alpha_2$ is complete w.r.t. $\{\mathcal{O}_1 \circ \mathcal{O}_2, \mathcal{O}_2 \circ \mathcal{O}_1\}$
- Soundness
  - SMD on α_1 is sound for $\mathcal{O}_1$ and SMD on α_2 is sound for $\mathcal{O}_2$
  - $\alpha_1(X) \subseteq \alpha_1(Y) \Rightarrow X \subseteq Y$

    $\Rightarrow$ SMD on $\alpha_1 \circ \alpha_2$ is sound w.r.t. $\mathcal{O}_1 \circ \mathcal{O}_2$

Further Abstraction

[M. Dalla Preda et al. POPL'07]

- Interesting States:



Let I be the set of interesting states:

$$M \hookrightarrow P \text{ if } \exists r : \alpha_I(S[\![M]\!]) \subseteq \alpha_I(\alpha_r(S[\![P]\!]))$$

Example: reordering of independent statements

Further Abstraction

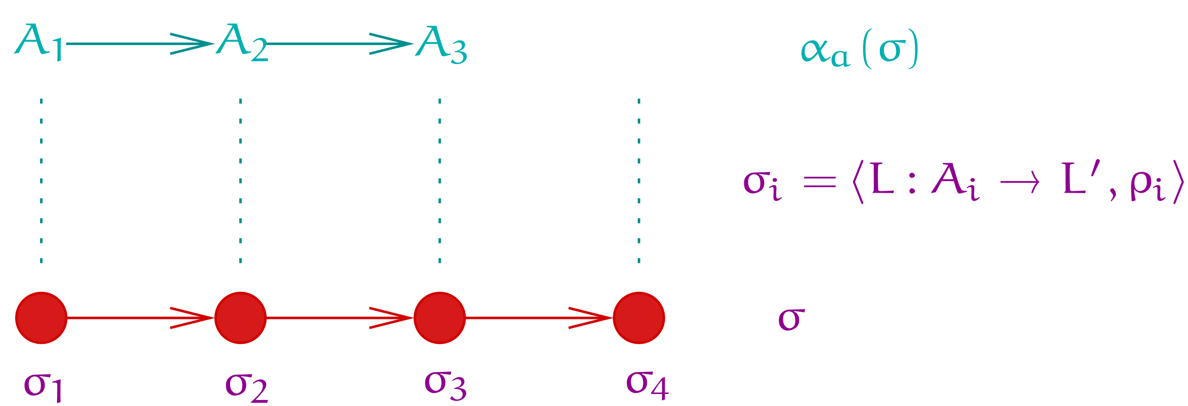
[M. Dalla Preda et al. POPL'07]

- Interesting Behaviours: consider a significant set of behaviours $X \subseteq S[\![M]\!]$:

$$M \hookrightarrow P \text{ if } \exists r : X \subseteq \alpha_r(S[\![P]\!])$$

- Interesting Actions: consider a significant set of program actions *Bad*

$$M \hookrightarrow P \text{ if } \exists r : \alpha_a(S[\![M]\!]) \subseteq \alpha_a(\alpha_r(S[\![P]\!]))$$



Proving Soundness/Completeness of MD

[M. Dalla Preda et al. POPL'07]

- Identifying the class of obfuscators for which a malware detector is resilient can be a complex and error-prone task
- There exists many obfuscation technique often defined using different languages
- obfuscators and detectors can be expressed on executions traces
  - express the malware detector as an algorithm on traces
  - prove soundness and completeness w.r.t. a class of obfuscating techniques
- Case study: Semantics-Aware Malware Detection Algorithm proposed by [Christodorescu et al. 2005]

Future work

Malware Detection:

- Systematically derive a suitable abstraction $\alpha_{\mathcal{O}}$ (Data mining)
- Model Checking
  - Abstraction $\alpha_{\mathcal{O}}$ identifies sets of program traces
  - Express such set of traces as formulae in some linear-branching temporal logic
- Program semantics is in general not computable, what happens when considering CFG or dependency graphs?
- Investigate code obfuscation composition

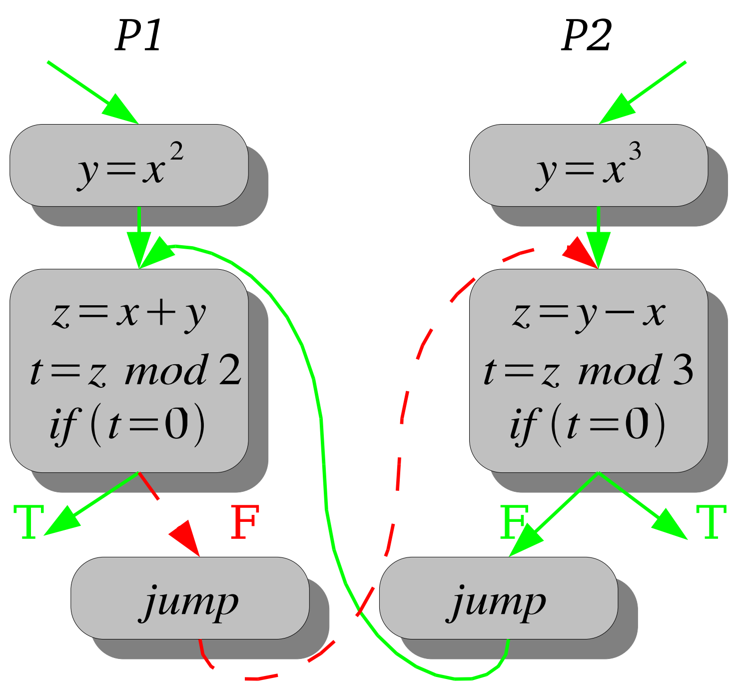
Semantic Obfuscation

- Given an attacker φ derive an obfuscation (possibly the simplest) able to defeat it.

Future Work

Opaque Predicates Detection

- Investigate the composition of opaque predicates

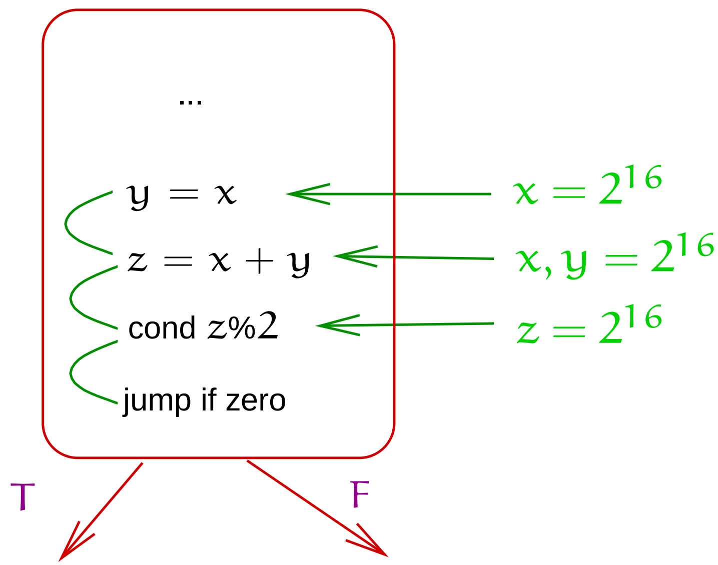


- Investigate a wider class of opaque predicates
- Use complex abstract domain to design opaque predicates (Polyhedra)

Brute Force Detection

[M. Dalla Preda et al. AMAST'06]

- Example: $\forall x \in \mathbb{Z} : 2\ \mathit{mod}\ (x + x)$ is decomposed into x and $x + y$

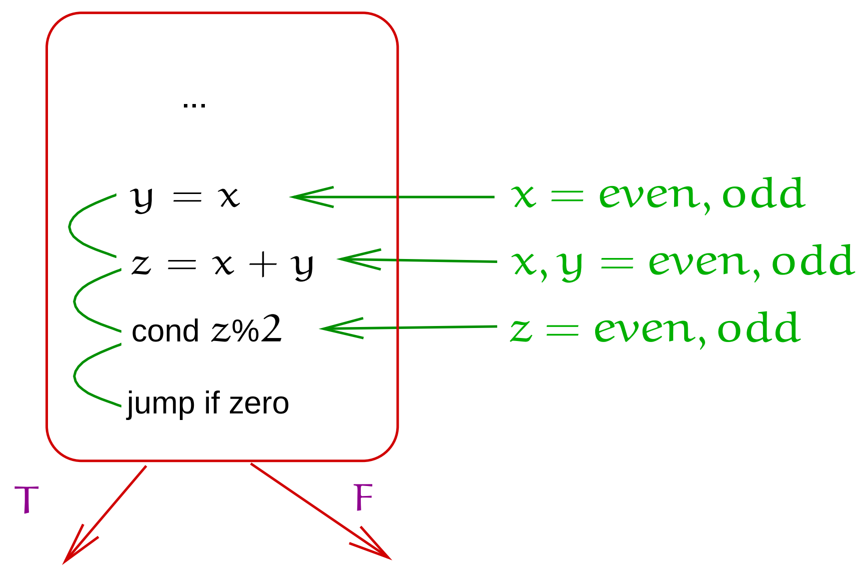


- 16-bit x86 environment: 3 instructions and each variable: $x = 2^{16}$
- Time 8.83 seconds
- Hybrid Static-Dynamic attack is time consuming

Abstract Detection

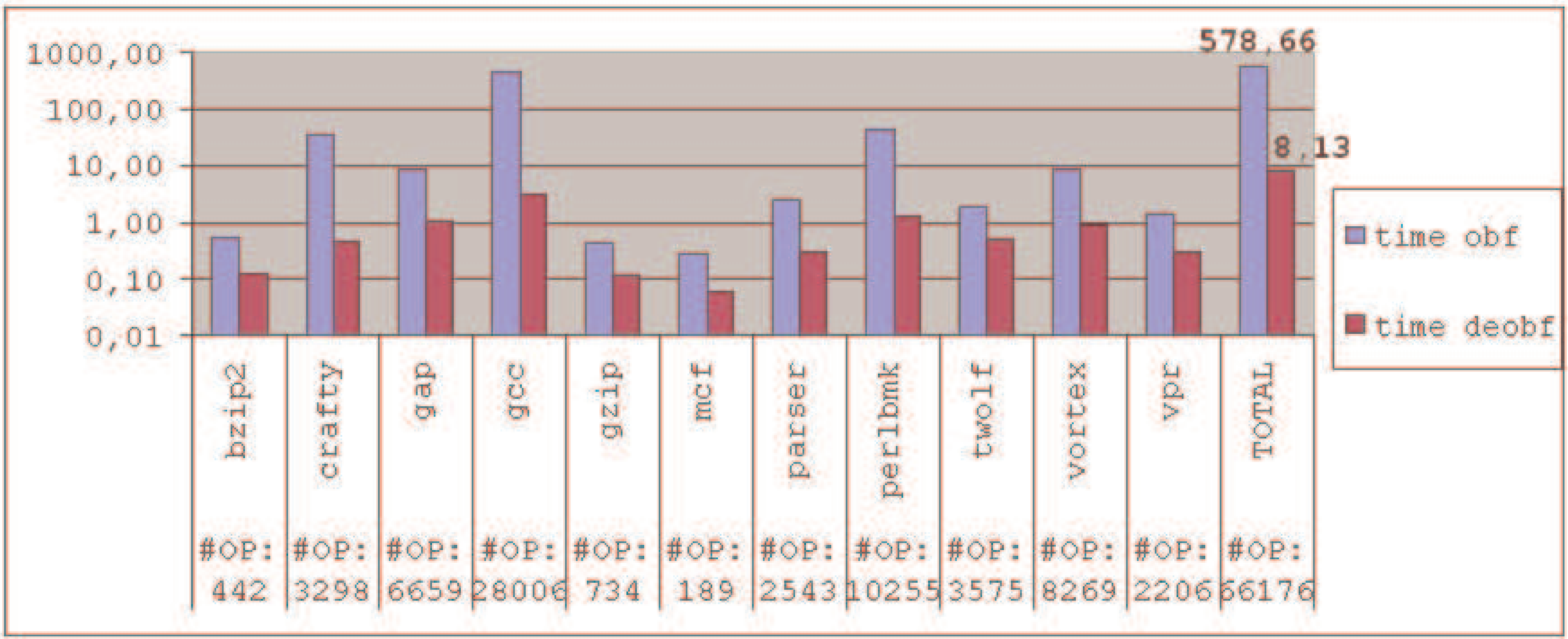
[M. Dalla Preda et al. AMAST'06]

- $\forall x \in \mathbb{Z} : 2 \ \mathit{mod} \ (x + x)$
- Abstract domain (Attacker): $\mathtt{Parity} = \left\{ \ \mathbb{Z}, \mathit{even}, \mathit{odd}, \varnothing \ \right\}$



Abstract Detection Results

[M. Dalla Preda et al. AMAST’06]



SPECInt2000 benchmarks obfuscated with:
$\forall x \in \mathbb{Z} : 2 \; mod \; (x^2 + x)$ and $\forall x \in \mathbb{Z} : 2 \; mod \; (x + x)$

Hybrid static-dynamic attack 8.83 sec to deobfuscate one opaque predicate

Example

Let $\perp$ denote the undefined function. $\alpha_c[\alpha_e(S[\![M]\!])](\alpha_e(S[\![M]\!]))$ is given by:

$$(\perp, \perp)$$
$$((B \rightsquigarrow L_B), \perp)$$
$$((B \rightsquigarrow L_B, A \rightsquigarrow L_A), \perp)^2$$
$$((B \rightsquigarrow L_B, A \rightsquigarrow L_A), (\rho(B) \leftarrow \textbf{Dec}(A)))$$
$$((B \rightsquigarrow \textit{succ}(m(\rho(B)), A \rightsquigarrow L_A), (\rho(B) \leftarrow \textbf{Dec}(A)))$$
$$((B \rightsquigarrow \textit{succ}(m(\rho(B)), A \rightsquigarrow \textit{succ}(m(\rho(A)))), (\rho(B) \leftarrow \textbf{Dec}(A)))$$

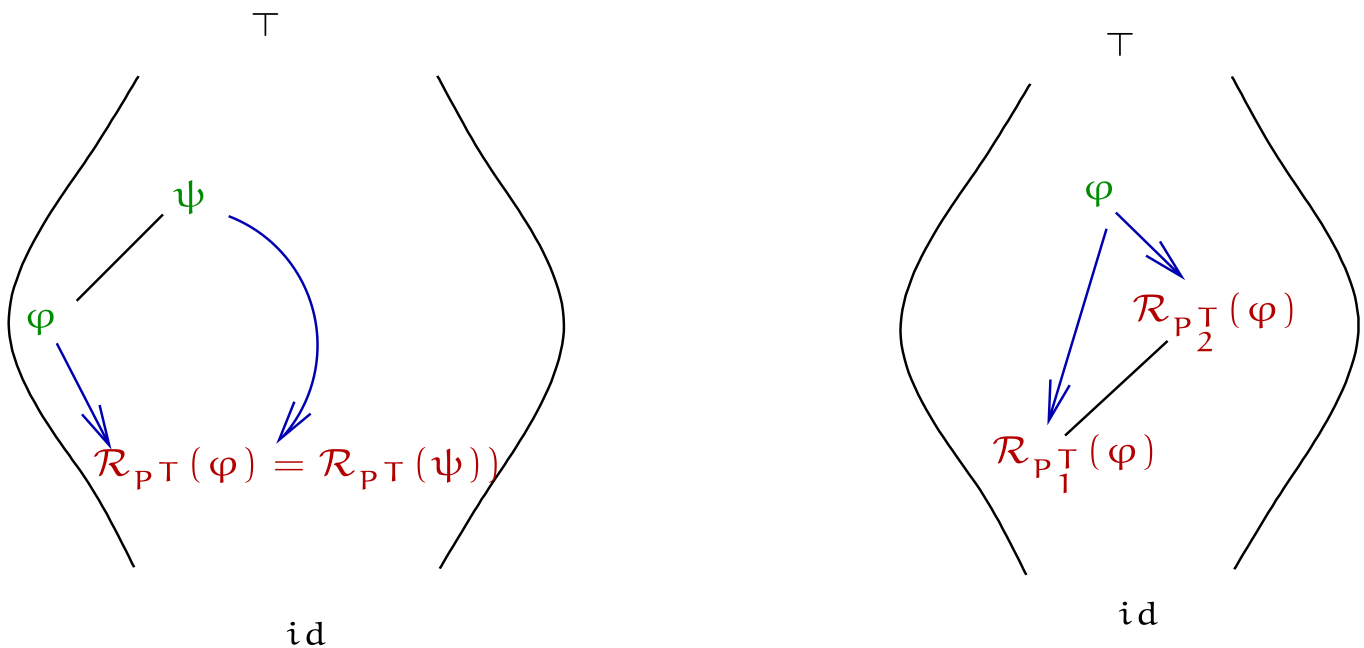
...

while $\alpha_e(S[\![\mathcal{O}_c(M)]\!])$ given by:

$$(\perp, \perp)$$
$$((B \rightsquigarrow L_B), \perp)^2$$
$$((B \rightsquigarrow L_B, A \rightsquigarrow L_A), \perp)^5$$
$$((B \rightsquigarrow L_B, A \rightsquigarrow L_A), (\rho(X) \leftarrow X + 3))$$
$$((B \rightsquigarrow L_B, A \rightsquigarrow L_A), (\rho(X) \leftarrow X + 3, \rho(X) \leftarrow X - 3))$$
$$((B \rightsquigarrow L_B, A \rightsquigarrow L_A), (\rho(B) \leftarrow \textbf{Dec}(A)))$$
$$((B \rightsquigarrow \textit{succ}(m(\rho(B)), A \rightsquigarrow L_A), (\rho(B) \leftarrow \textbf{Dec}(A)))$$
$$((B \rightsquigarrow \textit{succ}(m(\rho(B)), A \rightsquigarrow \textit{succ}(m(\rho(A)))), (\rho(B) \leftarrow \textbf{Dec}(A)))$$

...

Comparing attackers



- P^T obstructs ψ more than φ
- P_1^T is more efficient in obstructing the attacker φ than P_2^T

Reordering of independent statements

independent commands

$$\boxed{C_j; C_i \text{ is equivalent to } C_i; C_j}$$

Reordering of independent statements

independent commands

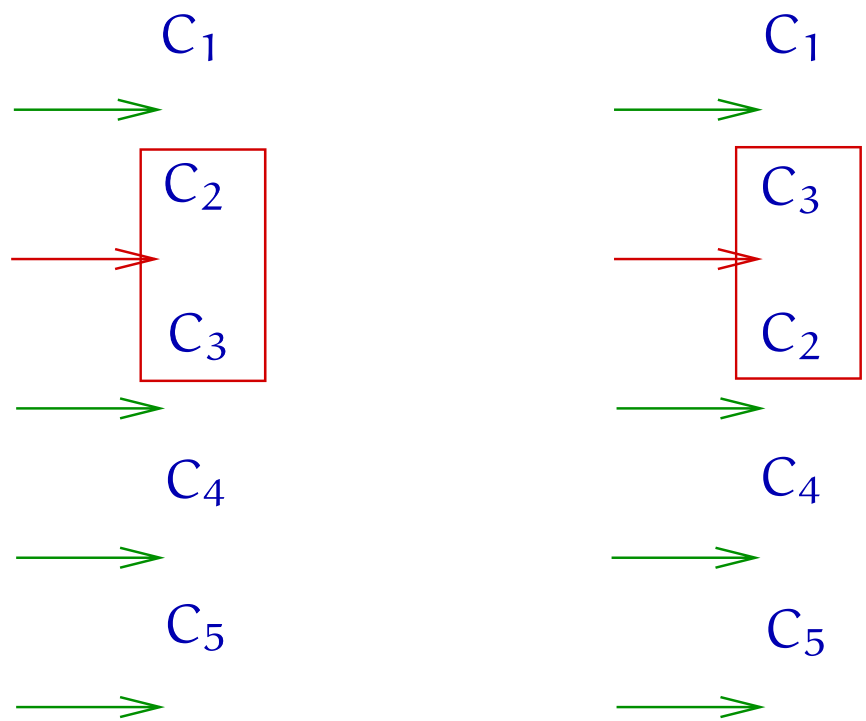
$C_j; C_i$ is equivalent to $C_i; C_j$

C_1	C_1
C_2	C_3
C_3	C_2
C_4	C_4
C_5	C_5

Reordering of independent statements

independent commands

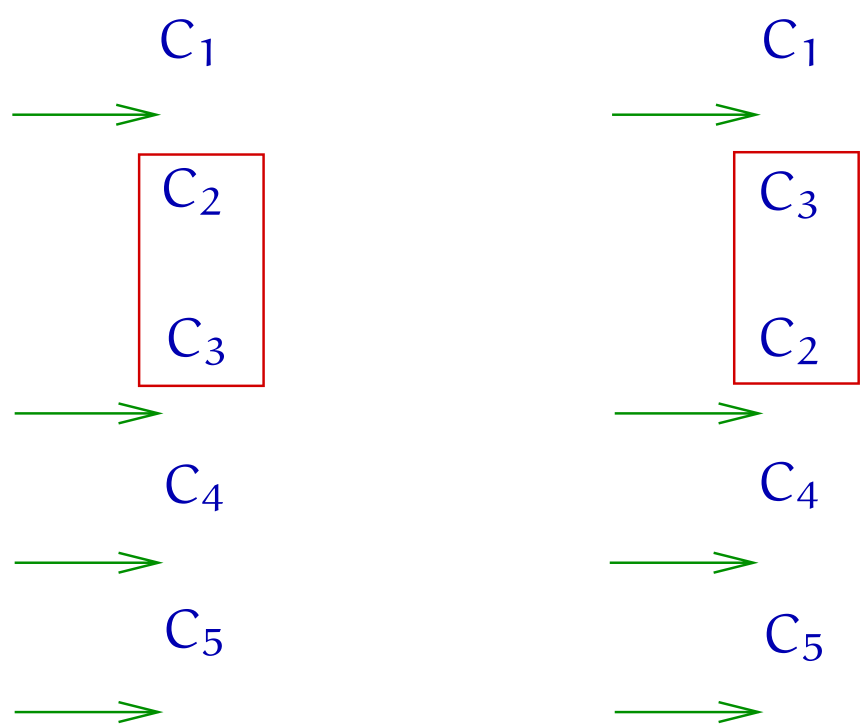
$C_j; C_i$ is equivalent to $C_i; C_j$



Reordering of independent statements

independent commands

$C_j; C_i$ is equivalent to $C_i; C_j$



Collberg's Definition

[Collberg POPL'98]

Let $\mathbb{T}$ be a transformation of a source program P into a target program P'. $\mathbb{T}$ is an obfuscating transformation if P and P' have the same observational behaviour. More precisely in order for $\mathbb{T}$ to be a legal obfuscating transformation the following conditions must hold:

- If P fails to terminate or terminates with an error condition, then P' may or may not terminate;
- Otherwise, P' must terminate and produce the same output as P.

Barak et al. Definition

[Barak et al. CRYPTO'01]

A probabilistic algorithm $\mathcal{O}$ is a TM obfuscator if the following conditions hold:

- functionality: For every TM M, the string $\mathcal{O}(M)$ describes a TM that computes the same function as M;
- polynomial slowdown: The description length and running time of $\mathcal{O}(M)$ are at most polynomially larger than that of M. That is, there exists a polynomial p such that for every TM M, $|\mathcal{O}(M)| \leq p(|M|)$, and if M halts in t steps on some input x, then $\mathcal{O}(M)$ halts within $p(t)$ steps on x;
- virtual black-box property:anything one can efficiently compute from the obfuscated program, one should be able to efficiently compute given just oracle access to the program (oracle says the input-output behaviour)

Relation with Signature Matching

- $\mathbb{P} = \wp(\mathbb{C})$, malware signature $S \subseteq M$
- syntactic test: $S \subseteq P$
- $\alpha_s(S[\![M]\!]) = S[\![S]\!]$
- semantic test: $\exists r : \alpha_s(S[\![M]\!]) \subseteq \alpha_r(S[\![P]\!])$
- Proposition: Semantic and syntactic test are equivalent
- Semantic test corresponds to SMD when $S = M$
- All the results still hold if considering abstraction $\alpha_s(S[\![M]\!])$

Model Checking

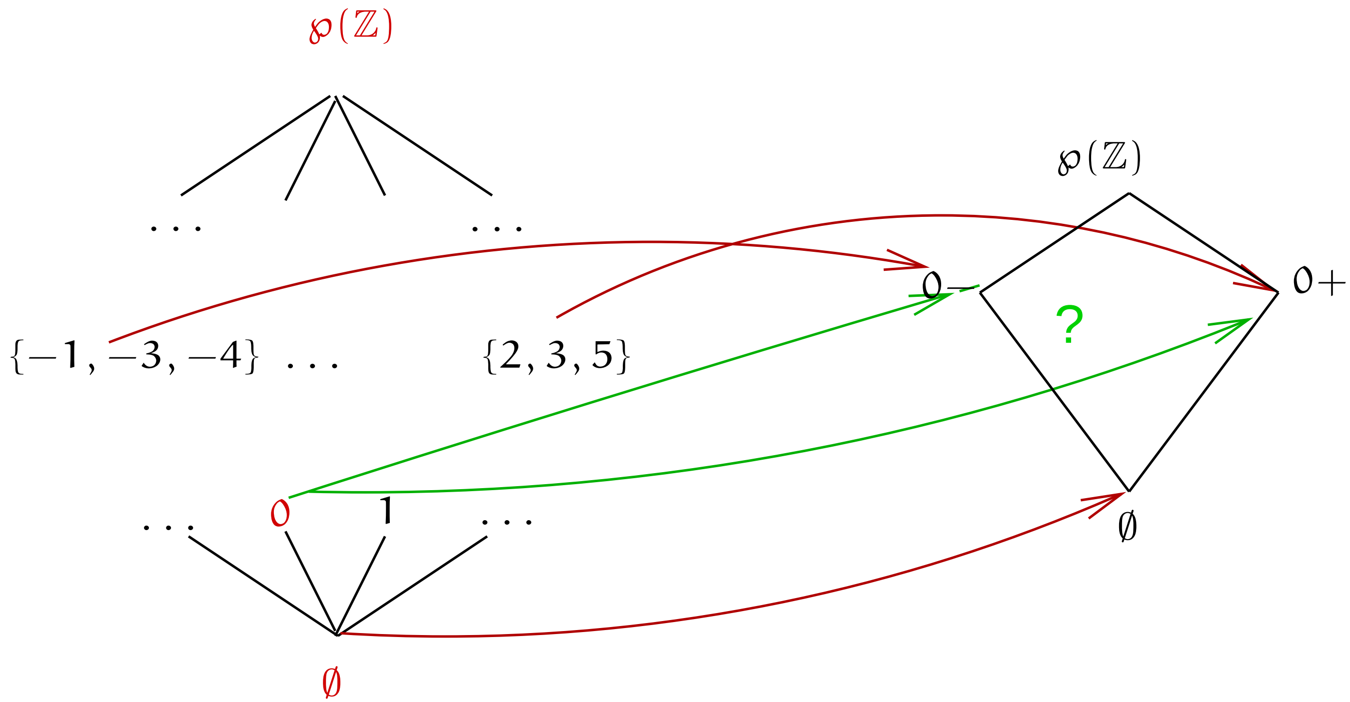
[Detecting Malicious Code by Model Checking, Kinder et al. 2005]

- Logic CTPL (Computation Tree Predicate Logic), extension of CTL: $p(x_1 ... x_n)$ where x_i are free variable in universe $\mathcal{U}$ or constants
- "In the code there exists a `mov` instruction that loads the constant 937 into *some* register, later the value contained in *this* register is always pushed into the stack":

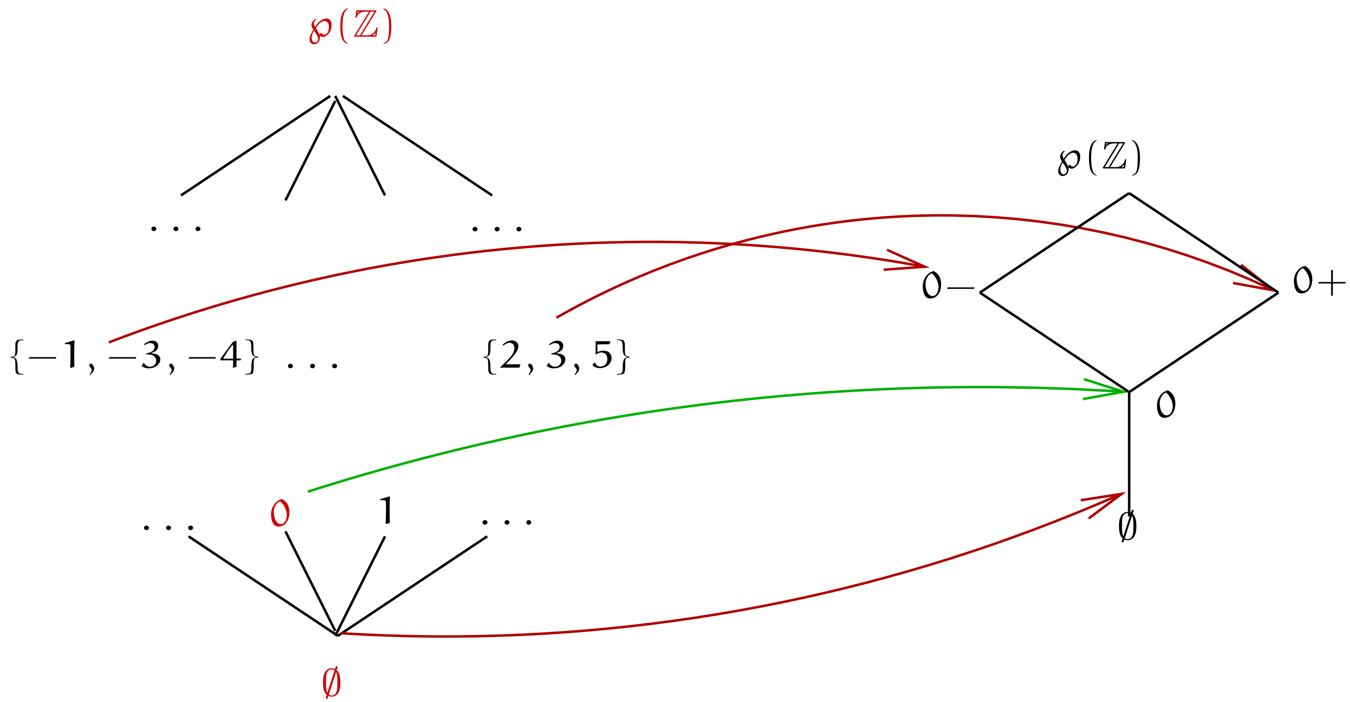
$$\exists r\,\mathbf{EF}(\mathtt{mov}(r, 937) \wedge \mathbf{AF}(\mathtt{push}(r)))$$

- Experimental results: carefully written CTPL specifications can apply to several families of worms
- Replace x86 instructions predicated with abstracted forms that capture their operational semantics

Why Galois Connections?

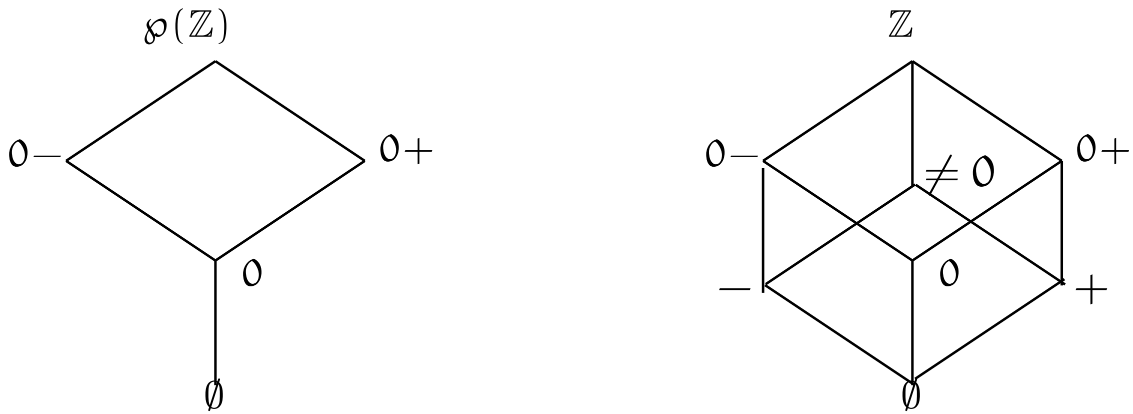


Why Galois Connections?



best correct representation of a concrete element in the abstract domain

Why Galois Connections?



Sign *Sign*$^+$

compare abstractions