CARET analysis of multithreaded programs

Huu-Vu Nguyen[1], Tayssir Touili[2]
[1] University Paris Diderot and LIPN
[2] LIPN, CNRS and University Paris 13

Motivation

Malware detection is a big challenge.

Existing Techniques (not robust)

- Signature-matching based technique: can easily be overcome by obfuscation techniques
- Code emulation based techniques: limitation in execution time

Motivation

Malware detection is a big challenge.

Existing Techniques (not robust)

- Signature-matching based technique: can easily be overcome by obfuscation techniques
- Code emulation based techniques: limitation in execution time

Solution to have a robust technique

Model-checking for malware detection

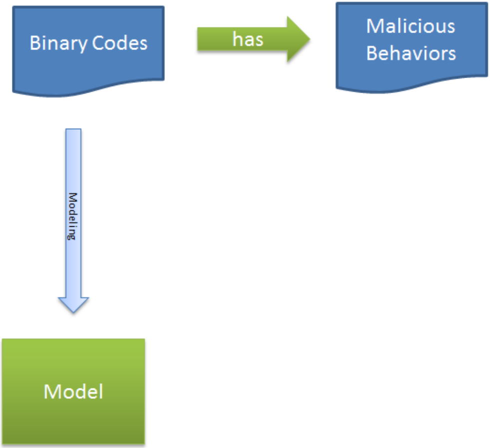
- allow us to analyse the behaviors (not the syntax) of the program without executing it

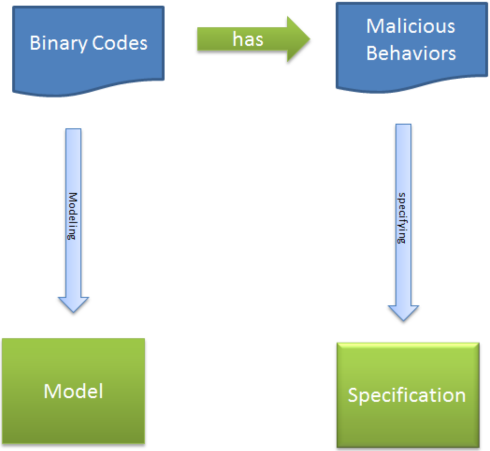
Model-checking for Malware Detection

Model-checking for Malware Detection

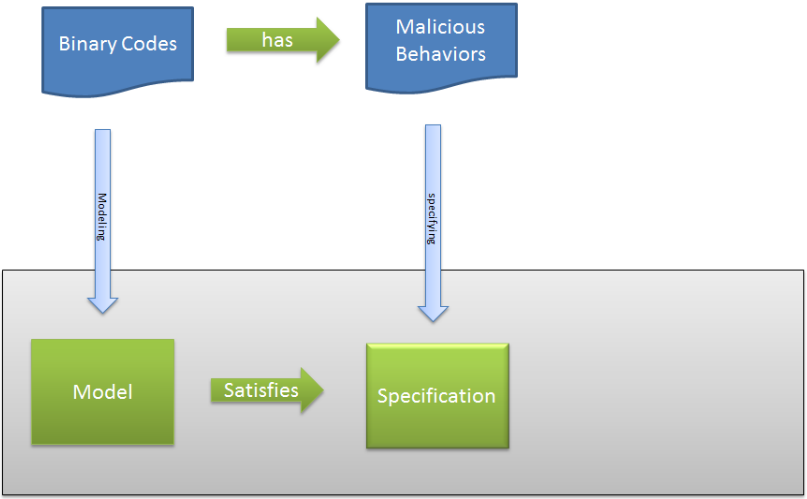
Model-checking for Malware Detection

Model-checking for Malware Detection

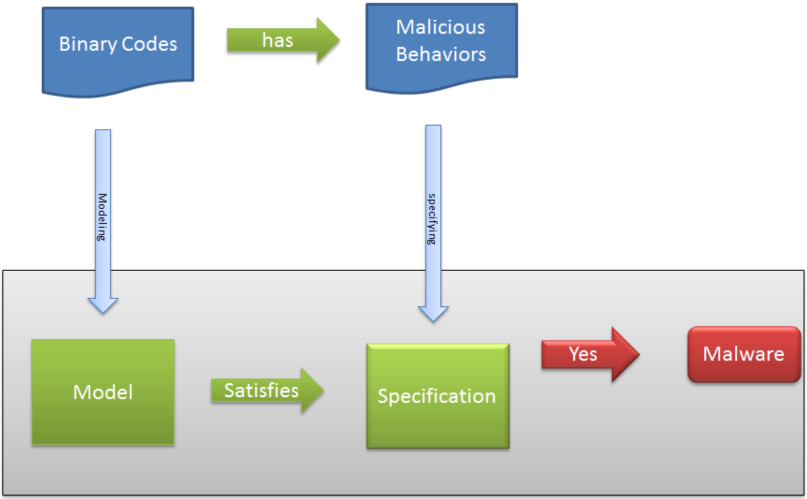



Binary Codes
has
Malicious Behaviors
Modeling
specifying
Model
Specification

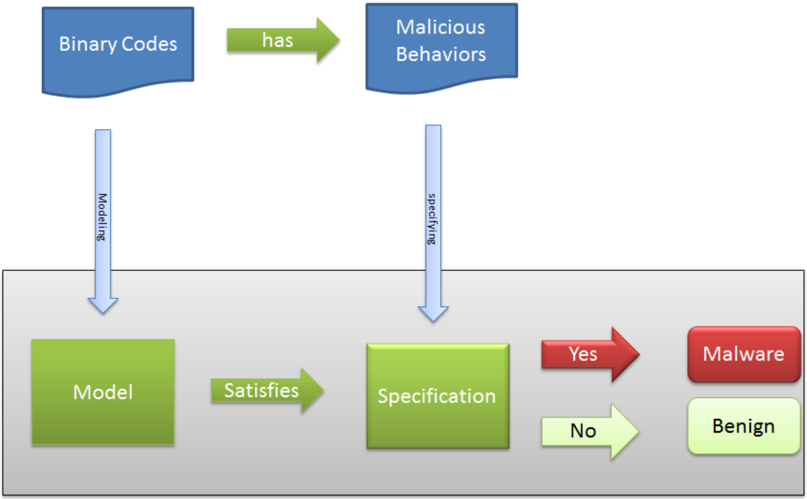
Model-checking for Malware Detection



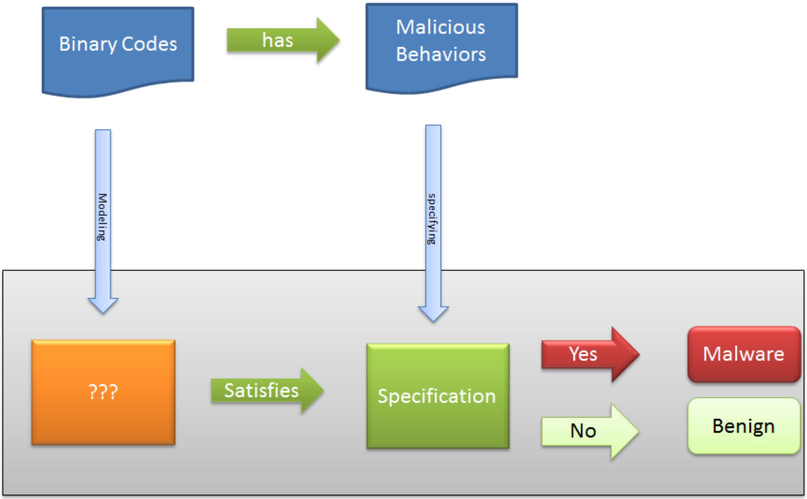
Model-checking for Malware Detection



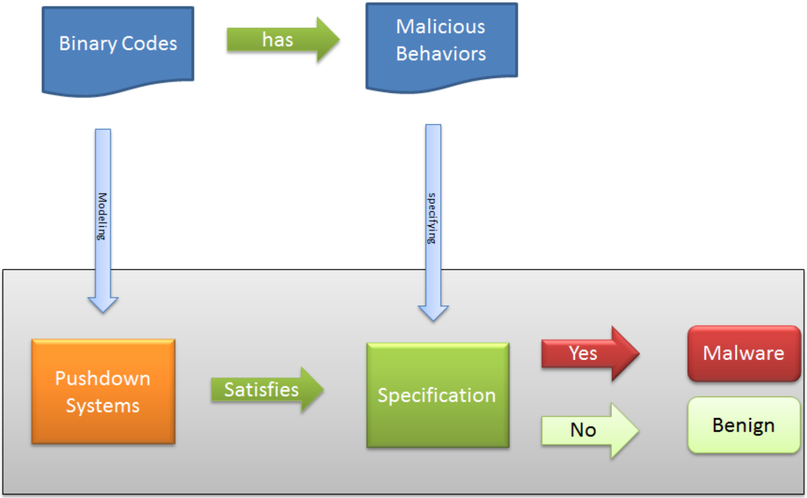
Model-checking for Malware Detection



Model-checking for Malware Detection



Model-checking for Malware Detection



Why Pushdown Systems?

Stack of binary codes

important for malware detection [Song and Touili 2012, 2013]

Pushdown Systems (PDSs)

- natural model of sequential programs
- allow taking into account the procedure contexts and stack content in the model

Why Pushdown Systems?

Stack of binary codes

important for malware detection [Song and Touili 2012, 2013]

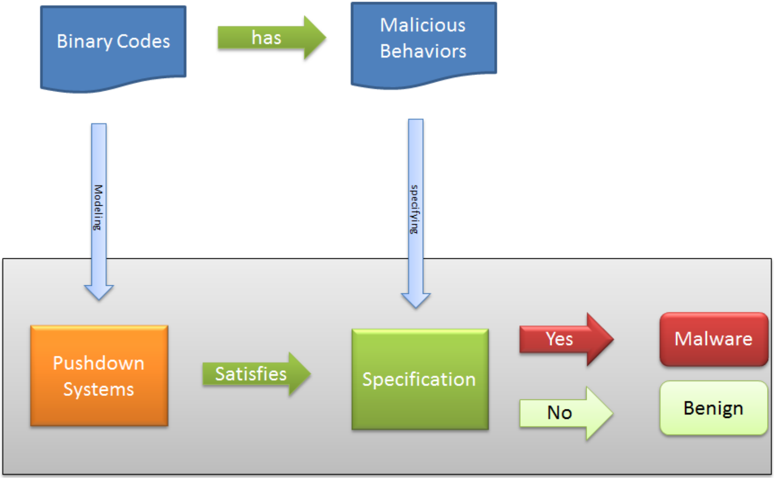
Pushdown Systems (PDSs)

- natural model of sequential programs
- allow taking into account the procedure contexts and stack content in the model

PDSs for Binary Codes

- Control locations of PDSs correspond to program points
- Stack of PDSs correspond to stack of binary programs

Model-checking for Malware Detection



$\Longrightarrow$ **Problem: This can be applied only for sequential programs. However, several malware is concurrent.**

Concurrent Malware Example

The email worm Bagle

is a multithreaded malware:

- Main thread: register itself into the registry listing: to be started at the boot time
- Thread 2: listen on port 6777 to receive different commands; allow the attackers to upload new file, ...
- Thread 3: contacts a list of websites every 10 minutes: to announce the infection of the current machine
- Thread 4: is spawn to search on local drives to look for valid email addresses, ...then send itself to these found emails.

Concurrent Malware Example

The email worm Bagle

is a multithreaded malware:

- Main thread: register itself into the registry listing: to be started at the boot time
- Thread 2: listen on port 6777 to receive different commands; allow the attackers to upload new file, ...
- Thread 3: contacts a list of websites every 10 minutes: to announce the infection of the current machine
- Thread 4: is spawn to search on local drives to look for valid email addresses, ...then send itself to these found emails.

How instances of threads are spawn?

- Thread 1 dynamically spawn instances of Thread 2,3,4 depending on the needs
- The number of instances is not fixed, depending on specific executions
- Instances of threads can be spawn dynamically during executions

Concurrent Malware Example

The email worm Bagle

is a multithreaded malware:

- Main thread: register itself into the registry listing: to be started at the boot time
- Thread 2: listen on port 6777 to receive different commands; allow the attackers to upload new file, ...
- Thread 3: contacts a list of websites every 10 minutes: to announce the infection of the current machine
- Thread 4: is spawn to search on local drives to look for valid email addresses, ...then send itself to these found emails.

How instances of threads are spawn?

- Thread 1 dynamically spawn instances of Thread 2,3,4 depending on the needs
- The number of instances is not fixed, depending on specific executions
- Instances of threads can be spawn dynamically during executions

$\Longrightarrow$ **Bagle is a multithreaded malware, with dynamic thread creation during its execution. How to model such a concurrent malware?**

How to model such concurrent malware?

Ideas

1. PDS is a natural model for sequential malware.
2. $\Longrightarrow$ networks of PDSs can model concurrent malware.
3. $\Longrightarrow$ networks of PDSs with dynamic creation can model concurrent malware with dynamic creations.
4. $\Longrightarrow$ Dynamic Pushdown Networks [Bouajjani, Müller-Olm and Touili 2005] match our needs.

How to model such concurrent malware?

Ideas

1. PDS is a natural model for sequential malware.
2. $\Longrightarrow$ networks of PDSs can model concurrent malware.
3. $\Longrightarrow$ networks of PDSs with dynamic creation can model concurrent malware with dynamic creations.
4. $\Longrightarrow$ Dynamic Pushdown Networks [Bouajjani, Müller-Olm and Touili 2005] match our needs.

Dynamic Pushdown Networks (DPNs)

- A DPN: a networks of Dynamic PDSs
- a Dynamic PDS: is a PDS with the ability to spawn new instances of PDSs during its runs

Definition of PDSs

A Pushdown System (PDS) $\mathcal{P}$ is a tuple (P, Γ, Δ), where

- P is a finite set of control locations
- Γ is a finite set of stack alphabet
- Δ is the set of transition rules of the following form:
  - (r_1): $p\gamma \xrightarrow{call} p_1\gamma_1\gamma_2$
  - (r_2): $p\gamma \xrightarrow{ret} p_1\epsilon$
  - (r_3): $p\gamma \xrightarrow{int} p_1\omega$

  where $p, p_1 \in P$, $\gamma, \gamma_1, \gamma_2 \in \Gamma$, $\omega \in \Gamma^*$

Definition of PDSs

A Pushdown System (PDS) $\mathcal{P}$ is a tuple (P, Γ, Δ), where

- P is a finite set of control locations
- Γ is a finite set of stack alphabet
- Δ is the set of transition rules of the following form:
  - (r_1): $p\gamma \xrightarrow{call} p_1\gamma_1\gamma_2$
  - (r_2): $p\gamma \xrightarrow{ret} p_1\epsilon$
  - (r_3): $p\gamma \xrightarrow{int} p_1\omega$

  where $p, p_1 \in P$, $\gamma, \gamma_1, \gamma_2 \in \Gamma$, $\omega \in \Gamma^*$

A rule of the form $p\gamma \xrightarrow{call} p_1\gamma_1\gamma_2$ corresponds to a call statement

- usually models a statement of the form $\gamma \xrightarrow{call \quad proc} \gamma_2$
- γ is the control point of the program where the function call is made, γ_1 is the entry point of the called procedure and γ_2 is the return point of the call.

A Pushdown System (PDS) $\mathcal{P}$ is a tuple (P, Γ, Δ), where

- P is a finite set of control locations
- Γ is a finite set of stack alphabet
- Δ is the set of transition rules of the following form:
  - (r_1): $p\gamma \xrightarrow{call} p_1\gamma_1\gamma_2$
  - (r_2): $p\gamma \xrightarrow{ret} p_1\epsilon$
  - (r_3): $p\gamma \xrightarrow{int} p_1\omega$

  where $p, p_1 \in P$, $\gamma, \gamma_1, \gamma_2 \in \Gamma$, $\omega \in \Gamma^*$

A rule of the form $p\gamma \xrightarrow{call} p_1\gamma_1\gamma_2$ corresponds to a call statement

- usually models a statement of the form $\gamma \xrightarrow{call \quad proc} \gamma_2$
- γ is the control point of the program where the function call is made, γ_1 is the entry point of the called procedure and γ_2 is the return point of the call.

A configuration: $p\omega$ where $p \in P$ is the current control location, $\omega \in \Gamma^*$ is the current stack content.

Definition of DPNs

A Dynamic Pushdown Network (DPN) $\mathcal{M}$ is a set $\{\mathcal{P}_1, ..., \mathcal{P}_n\}$ s.t. for every $1 \leq i \leq n$, $\mathcal{P}_i = (P_i, \Gamma_i, \Delta_i)$ is a Dynamic Pushdown System (DPDS)

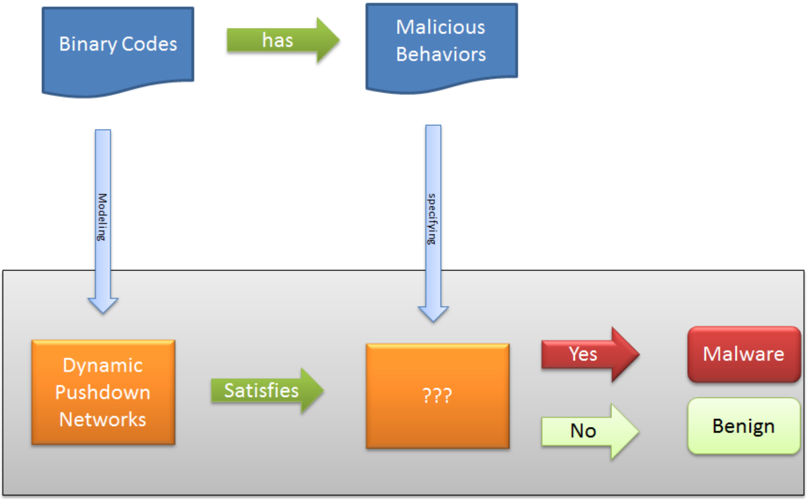
- $(NonSpawn)(r_1)$ $p\gamma \xrightarrow{call}_i p_1\gamma_1\gamma_2$
- $(NonSpawn)(r_2)$ $p\gamma \xrightarrow{ret}_i p_1\epsilon$
- $(NonSpawn)(r_3)$ $p\gamma \xrightarrow{int}_i p_1\omega_1$

Definition of DPNs

A Dynamic Pushdown Network (DPN) $\mathcal{M}$ is a set $\{\mathcal{P}_1, ..., \mathcal{P}_n\}$ s.t. for every $1 \leq i \leq n$, $\mathcal{P}_i = (P_i, \Gamma_i, \Delta_i)$ is a Dynamic Pushdown System (DPDS) where $p_s\omega_s \in \bigcup_{1\leq j\leq n} P_j \times \Gamma_j^*$

- $(NonSpawn)(r_1)\ p\gamma \xrightarrow{call}_i p_1\gamma_1\gamma_2$
- $(NonSpawn)(r_2)\ p\gamma \xrightarrow{ret}_i p_1\epsilon$
- $(NonSpawn)(r_3)\ p\gamma \xrightarrow{int}_i p_1\omega_1$
- $(Spawn)\ (r_4)\ p\gamma \xrightarrow{call}_i p_1\gamma_1\gamma_2 \rhd p_s\omega_s$
- $(Spawn)\ (r_5)\ p\gamma \xrightarrow{ret}_i p_1\epsilon \rhd p_s\omega_s$
- $(Spawn)\ (r_6)\ p\gamma \xrightarrow{int}_i p_1\omega_1 \rhd p_s\omega_s$

Model-checking for Malware Detection



Specification Formalisms for Malware Behaviors

Recent works: extensions of LTL, CTL were used as specifications

- CTPL [Kinder, Katzenbeisser,Schallhart and Veith 2005]
- SLTPL, SCTPL [Song and Touili 2012, 2013]

However, these are not expressive enough for malicious behaviors

Malicious Behavior Example

Spyware Behavior

search directories for personal information (emails, bank account info, ...)

Malicious Behavior Example

Spyware Behavior

search directories for personal information (emails, bank account info, ...)

To do that

- Firstly, call the API FindFirstFileA $\Longrightarrow$ return a search handle h
- After that, call the API FindNextFileA with h as parameter $\Longrightarrow$ search remaining matching files

Malicious Behavior Example

Spyware Behavior

search directories for personal information (emails, bank account info, ...)

To do that

- Firstly, call the API FindFirstFileA $\Longrightarrow$ return a search handle h
- After that, call the API FindNextFileA with h as parameter $\Longrightarrow$ search remaining matching files

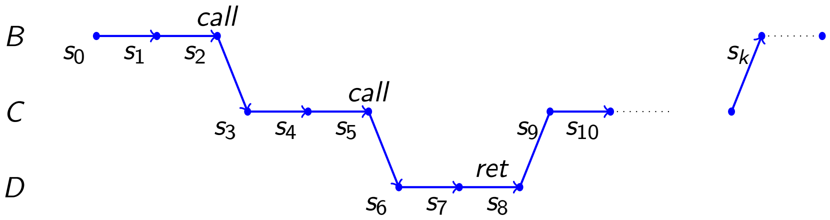
Then,..

- Cannot be expressed by LTL or CTL since it requires that the return value of the function FindFirstFileA should be used as the input to the function FindNextFileA
- $\Longrightarrow$ we need a formalism that can talk about matching calls and returns $\Longrightarrow$ CARET.

- linear temporal logic of Calls and Returns [Alur, Etessami and Madhusudan 2004]
- Interpreted over transition systems where each state is associated with a tag in the set {call, ret, int}
  - call : a call statement
  - ret : a return statement
  - int : an internal statement (neither call nor return)

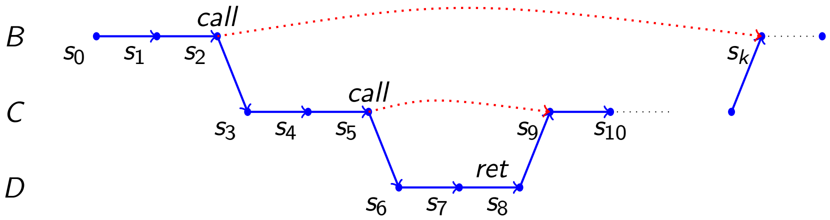
Global Successor

- Global Successor(X^g): standard successor ($X^g(s_i) = s_{i+1}$)
- Global Path: standard path like for LTL



Abstract Successor

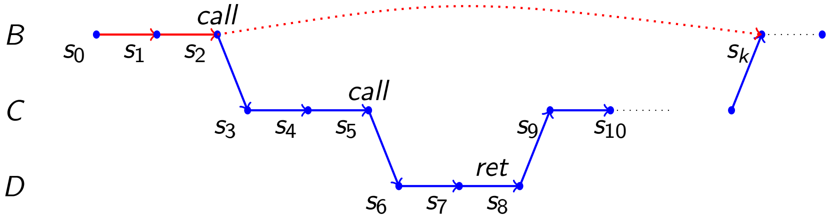
- Abstract Successor (X^a)
  - The abstract successor of a *call* is its corresponding return-point
- Abstract Path: apply repeatedly the abstract successor



Abstract Path

Abstract path:

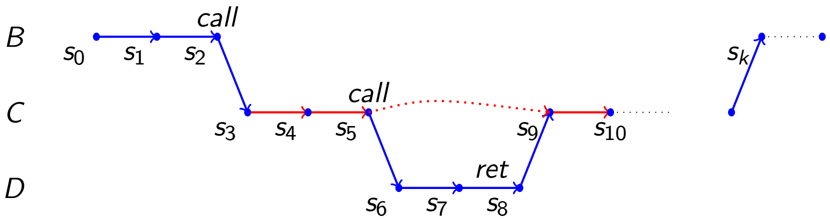
- From s_0: $s_0 s_1 s_2 s_k \ldots$.



Abstract Path

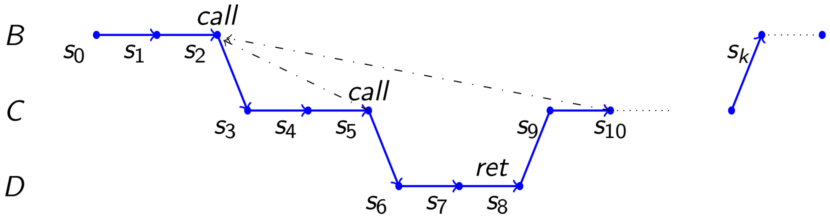
Abstract path:

- From s_3: $s_3 s_4 s_5 s_9 s_{10} \ldots$.

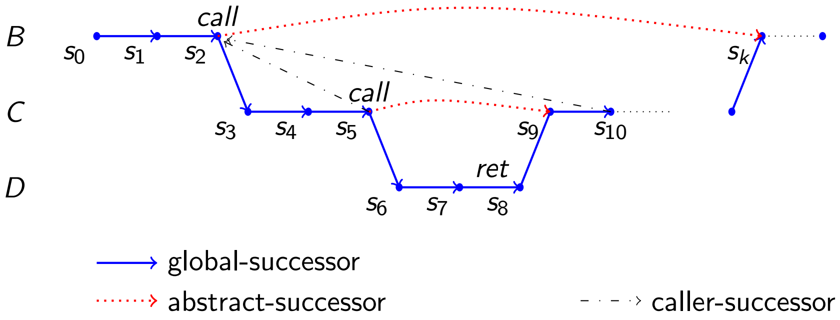


Caller Successor

- Caller Successors (X^c)
  - the caller successor of a point is the caller point of the current procedure
- Caller Path: apply repeatedly the caller successor



CARET successors



CARET Definition

Given a finite set of atomic propositions AP. A CARET formula over AP is defined as follows:

$$\psi := e \mid \{call, ret, int\} \mid \psi \vee \psi \mid \neg\psi \mid X^g\psi \mid X^a\psi \mid X^c\psi \mid \psi U^a\psi \mid \psi U^g\psi \mid \psi U^c\psi$$

where

- $e \in AP$: atomic proposition
- X^g: global successor
- X^a: abstract successor
- X^c: caller successor
- U^g: until operator on global path
- U^a: until operator on abstract path
- U^c: until operator on caller path

Malicious Behavior Example

Spyware Behavior

search directories for personal information (emails, bank account info, ...)

To do that

- Firstly, call the API FindFirstFileA $\Longrightarrow$ return a search handle h
- After that, call the API FindNextFileA with h as parameter $\Longrightarrow$ search remaining matching files

Malicious Behavior Example

Spyware Behavior

search directories for personal information (emails, bank account info, ...)

To do that

- Firstly, call the API FindFirstFileA $\Longrightarrow$ return a search handle h
- After that, call the API FindNextFileA with h as parameter $\Longrightarrow$ search remaining matching files

Using CARET to describe ...

$$\psi_{sf} = \bigvee_{d \in D} F^g(\text{call}(\text{FindFirstFileA}) \wedge X^a(eax = d) \wedge F^a(\text{call}(\text{FindNextFileA}) \wedge d\Gamma^*))$$

Spyware Behavior Formula

Malicious behavior by CARET

$$\psi_{sf} = \boxed{\bigvee_{d \in D}} F^g(\text{call}(\text{FindFirstFileA}) \wedge X^a(eax = d) \wedge F^a(\text{call}(\text{FindNextFileA}) \wedge d\Gamma^*))$$

- $\bigvee_{d \in D}$: disjunction over all possible memory addresses d containing search handles

Spyware Behavior Formula

Malicious behavior by CARET

$$\psi_{sf} = \bigvee_{d \in D} F^g\big(\boxed{\text{call(FindFirstFileA)} \wedge X^a(eax = d)} \wedge F^a(\text{call(FindNextFileA)} \wedge d\Gamma^*)\big)$$

- $\bigvee_{d \in D}$: disjunction over all possible memory addresses d containing search handles
- $\text{call(FindFirstFileA)} \wedge X^a(eax = d)$
  - eax: contain the return value of an API function when the function finish its execution
  - X^a of a call is its corresponding return point

$\rightarrow$ there is a call to *FindFirstFileA* and the return value is d

Spyware Behavior Formula

Malicious behavior by CARET

$$\psi_{sf} = \bigvee_{d \in D} F^g(\text{call}(\text{FindFirstFileA}) \wedge X^a(eax = d) \wedge F^a(\text{call}(\text{FindNextFileA}) \wedge d\Gamma^*))$$

- $\bigvee_{d \in D}$: disjunction over all possible memory addresses d containing search handles
- $\text{call}(\text{FindFirstFileA}) \wedge X^a(eax = d)$
  - eax: contain the return value of an API function when the function finish its execution
  - X^a of a call is its corresponding return point

$\rightarrow$ there is a call to *FindFirstFileA* and the return value is d

- $\text{call}(\text{FindNextFileA}) \wedge d\Gamma^*$
  - $d\Gamma^*$: d is on top of the stack
  - parameters: passed to function by pushing on the stack

$\rightarrow$ there is a call to *FindNextFileA* where d is used as parameter.

Spyware Behavior Formula

Malicious behavior by CARET

$$\psi_{sf} = \bigvee_{d \in D} \boxed{F^g} (\text{call(FindFirstFileA)} \wedge X^a(eax = d) \wedge \boxed{F^a} (\text{call(FindNextFileA)} \wedge d\Gamma^*))$$

- $\bigvee_{d \in D}$: disjunction over all possible memory addresses d containing search handles
- call(FindFirstFileA) $\wedge X^a(eax = d)$
  - eax: contain the return value of an API function when the function finish its execution
  - X^a of a call is its corresponding return point

  $\rightarrow$ there is a call to *FindFirstFileA* and the return value is d
- call(FindNextFileA) $\wedge d\Gamma^*$
  - $d\Gamma^*$: d is on top of the stack
  - parameters: passed to function by pushing on the stack

  $\rightarrow$ there is a call to *FindNextFileA* where d is used as parameter.
- F^g: the standard F operator
- F^a: in the future after call(FindFirstFileA) finishes

Spyware Behavior Formula

Malicious behavior by CARET

$$\psi_{sf} = \bigvee_{d \in D} F^g(\text{call}(\text{FindFirstFileA}) \wedge X^a(eax = d) \wedge F^a(\text{call}(\text{FindNextFileA}) \wedge d\Gamma^*))$$

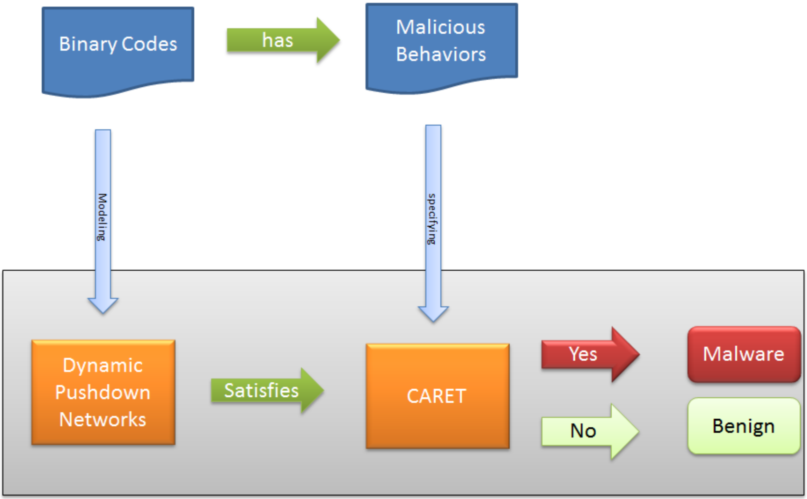
- $\bigvee_{d \in D}$: disjunction over all possible memory addresses d containing search handles
- $\text{call}(\text{FindFirstFileA}) \wedge X^a(eax = d)$
    - eax: contain the return value of an API function when the function finish its execution
    - X^a of a call is its corresponding return point

  $\rightarrow$ there is a call to *FindFirstFileA* and the return value is d
- $\text{call}(\text{FindNextFileA}) \wedge d\Gamma^*$
    - $d\Gamma^*$: d is on top of the stack
    - parameters: passed to function by pushing on the stack

  $\rightarrow$ there is a call to *FindNextFileA* where d is used as parameter.
- F^g: the standard F operator
- F^a: in the future after call(FindFirstFileA) finishes

$\Longrightarrow$ ψ_{sf}: there exists a path s.t there is a call to *FindFirstFileA* where the return value is d, and after this call finishes, there is a call to *FindNextFileA* s.t d is used as parameter.

Model-checking for Malware Detection



Problem: $DPNs \models CARET$??

1. model-checking LTL properties for networks of PDSs is undecidable [Kahlon and Gupta 2006], e.g., for properties that mix different indices of different threads like $F(a_i \wedge b_j)$

Problem: $DPNs \models CARET$??

1. model-checking LTL properties for networks of PDSs is undecidable [Kahlon and Gupta 2006], e.g., for properties that mix different indices of different threads like $F(a_i \wedge b_j)$
2. LTL is a subclass of CARET

Problem: $DPNs \models CARET$??

1. model-checking LTL properties for networks of PDSs is undecidable [Kahlon and Gupta 2006], e.g., for properties that mix different indices of different threads like $F(a_i \wedge b_j)$
2. LTL is a subclass of CARET
3. $\Longrightarrow$ model-checking CARET properties for networks of PDSs is undecidable

Problem: $DPNs \models CARET$??

1. model-checking LTL properties for networks of PDSs is undecidable [Kahlon and Gupta 2006], e.g., for properties that mix different indices of different threads like $F(a_i \wedge b_j)$
2. LTL is a subclass of CARET
3. $\Longrightarrow$ model-checking CARET properties for networks of PDSs is undecidable
4. $\Longrightarrow$ We consider: model-checking single-indexed CARET properties for DPNs, where:

Problem: $DPNs \models CARET$??

1. model-checking LTL properties for networks of PDSs is undecidable [Kahlon and Gupta 2006], e.g., for properties that mix different indices of different threads like $F(a_i \wedge b_j)$
2. LTL is a subclass of CARET
3. $\Longrightarrow$ model-checking CARET properties for networks of PDSs is undecidable
4. $\Longrightarrow$ We consider: model-checking single-indexed CARET properties for DPNs, where:
   - single-indexed properties: properties in the form $f = f_1 \wedge f_2 \ldots \wedge f_n$, where f_i is the CARET formula corresponding to $\mathcal{P}_i$

Problem to solve

Given:

- a DPN $\mathcal{M} = \{\mathcal{P}_1, \mathcal{P}_2, ..., \mathcal{P}_n\}$
- a single-indexed CARET formula $f = f_1 \wedge f_2 ... \wedge f_n$

Model-checking problem:

- Does there exist an execution of $\mathcal{M}$ s.t. every instance of the DPDS $\mathcal{P}_i$ satisfies the corresponding CARET formula f_i?

Theorem

Single-indexed CARET Model Checking for DPNs is decidable.

Intuition:

- We reduce this problem to the emptiness problem of Büchi Dynamic Pushdown Networks (BDPNs) [Song and Touili 2013, 2016].

Single-indexed CARET Model Checking for DPNs

Theorem

Single-indexed CARET Model Checking for DPNs is decidable.

Intuition:

- We reduce this problem to the emptiness problem of Büchi Dynamic Pushdown Networks (BDPNs) [Song and Touili 2013, 2016].
  - a BDPN $\mathcal{BM}$ is a set $\{\mathcal{BP}_1, ..., \mathcal{BP}_n\}$ where $\mathcal{BP}_i (1 \leq i \leq n)$ is a Büchi Dynamic Pushdown System

Theorem

Single-indexed CARET Model Checking for DPNs is decidable.

Intuition:

- We reduce this problem to the emptiness problem of Büchi Dynamic Pushdown Networks (BDPNs) [Song and Touili 2013, 2016].
  - a BDPN $\mathcal{BM}$ is a set $\{\mathcal{BP}_1, ..., \mathcal{BP}_n\}$ where $\mathcal{BP}_i(1 \leq i \leq n)$ is a Büchi Dynamic Pushdown System
  - a Büchi Dynamic Pushdown System $\mathcal{BP}_i = (P_i, \Gamma_i, \Delta_i, F_i)$ is a PDS with a set of accepting control locations F_i

Theorem

Single-indexed CARET Model Checking for DPNs is decidable.

Intuition:

- We reduce this problem to the emptiness problem of Büchi Dynamic Pushdown Networks (BDPNs) [Song and Touili 2013, 2016].
  - a BDPN $\mathcal{BM}$ is a set $\{\mathcal{BP}_1, ..., \mathcal{BP}_n\}$ where $\mathcal{BP}_i (1 \leq i \leq n)$ is a Büchi Dynamic Pushdown System
  - a Büchi Dynamic Pushdown System $\mathcal{BP}_i = (P_i, \Gamma_i, \Delta_i, F_i)$ is a PDS with a set of accepting control locations F_i
- We compute BDPNs $\mathcal{BM} = \{\mathcal{BP}_1, ..., \mathcal{BP}_n\}$ such that $\mathcal{BP}_i$ is a kind of product between $\mathcal{P}_i$ and the CARET formula f_i which ensures that:

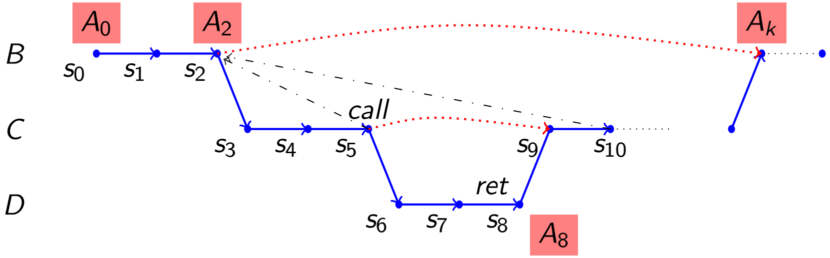
Theorem

Single-indexed CARET Model Checking for DPNs is decidable.

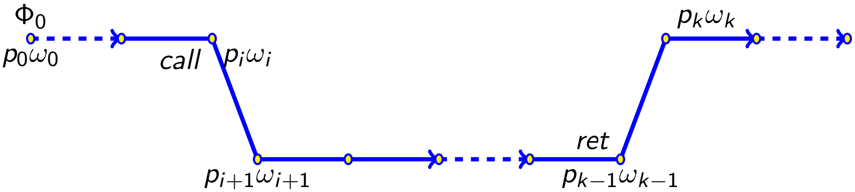
Intuition:

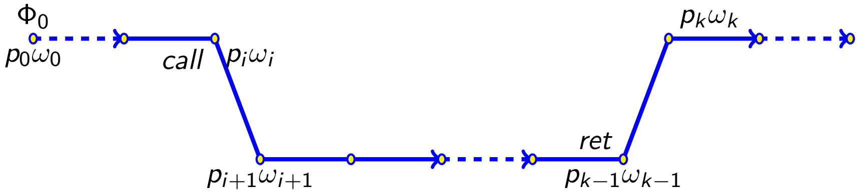
- We reduce this problem to the emptiness problem of Büchi Dynamic Pushdown Networks (BDPNs) [Song and Touili 2013, 2016].
  - a BDPN $\mathcal{BM}$ is a set $\{\mathcal{BP}_1, ..., \mathcal{BP}_n\}$ where $\mathcal{BP}_i (1 \leq i \leq n)$ is a Büchi Dynamic Pushdown System
  - a Büchi Dynamic Pushdown System $\mathcal{BP}_i = (P_i, \Gamma_i, \Delta_i, F_i)$ is a PDS with a set of accepting control locations F_i
- We compute BDPNs $\mathcal{BM} = \{\mathcal{BP}_1, ..., \mathcal{BP}_n\}$ such that $\mathcal{BP}_i$ is a kind of product between $\mathcal{P}_i$ and the CARET formula f_i which ensures that:
  - The problem of checking whether an instance of $\mathcal{P}_i$ starting from $p\omega$ satisfies f_i can be reduced to the membership problem of $\mathcal{BP}_i$

At state s_i, we encode a set of formulas A_i such that for every $\phi \in A_i$, ϕ holds at s_i



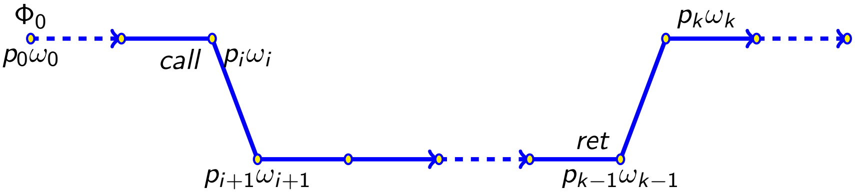
BDPDS Computation-X Operators-Call statements





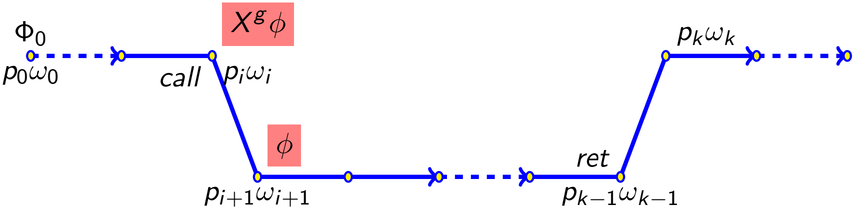
for $p_i\gamma \xrightarrow{call} p_{i+1}\gamma'\gamma''$ in $\mathcal{P}_i$:

BDPDS Computation-X Operators-Call statements



for $p_i\gamma \xrightarrow{call} p_{i+1}\gamma'\gamma''$ in $\mathcal{P}_i$:

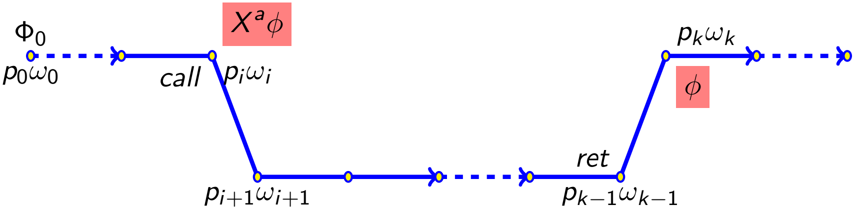
- $(\!| p_i, \{X^g\phi\} |\!)\gamma \rightarrow (\!| p_{i+1}, \{\phi\} |\!)\gamma\gamma''$ in $\mathcal{BP}_i$



$p_i\omega_i \models X^g\phi$ iff $p_{i+1}\omega_{i+1} \models \phi$

for $p_i\gamma \xrightarrow{call} p_{i+1}\gamma'\gamma''$ in $\mathcal{P}_i$:

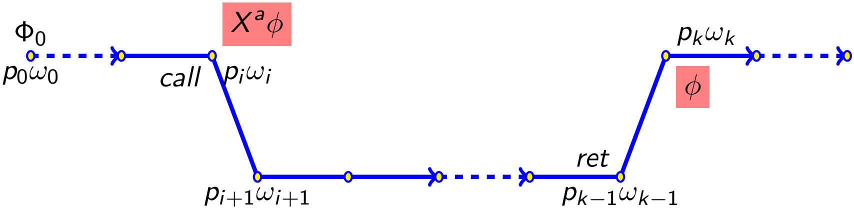
- $(\!|p_i, \{X^g\phi\}|\!)\gamma \rightarrow (\!|p_{i+1}, \{\phi\}|\!)\gamma\gamma''$ in $\mathcal{BP}_i$



$p_i\omega_i \models X^a\phi$ iff $p_k\omega_k \models \phi$

for $p_i\gamma \xrightarrow{call} p_{i+1}\gamma'\gamma''$ in $\mathcal{P}_i$:

- $(\!|p_i, \{X^g\phi\}|\!)\gamma \to (\!|p_{i+1}, \{\phi\}|\!)\gamma\gamma''$ in $\mathcal{BP}_i$

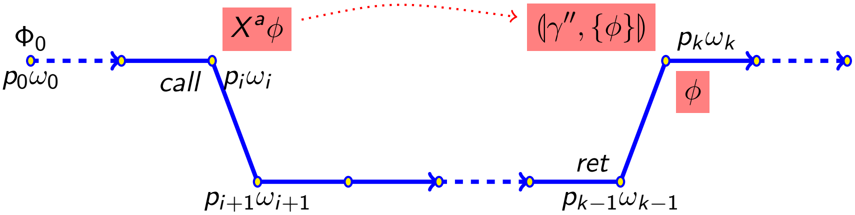


$p_i\omega_i \models X^a\phi$ iff $p_k\omega_k \models \phi$

for $p_i\gamma \xrightarrow{call} p_{i+1}\gamma'\gamma''$ in $\mathcal{P}_i$:

- $(\!|p_i, \{X^g\phi\}|\!)\gamma \rightarrow (\!|p_{i+1}, \{\phi\}|\!)\gamma\gamma''$ in $\mathcal{BP}_i$
- $(\!|p_i, \{X^a\phi\}|\!)\gamma \rightarrow p_{i+1}\gamma' \boxed{(\!|\gamma'', \{\phi\}|\!)}$ in $\mathcal{BP}_i$

BDPDS Computation-X Operators-Call statements

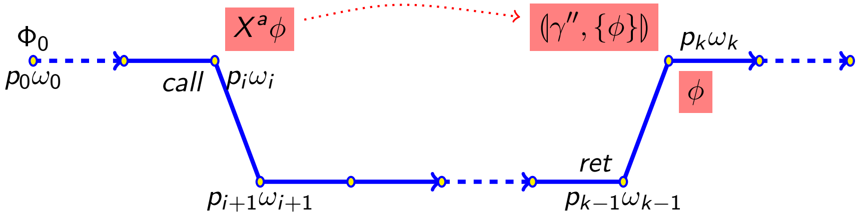


$p_i\omega_i \models X^a\phi$ iff $p_k\omega_k \models \phi$

for $p_i\gamma \xrightarrow{call} p_{i+1}\gamma'\gamma''$ in $\mathcal{P}_i$:

- $\llparenthesis p_i, \{X^g\phi\} \rrparenthesis \gamma \rightarrow \llparenthesis p_{i+1}, \{\phi\} \rrparenthesis \gamma\gamma''$ in $\mathcal{BP}_i$
- $\llparenthesis p_i, \{X^a\phi\} \rrparenthesis \gamma \rightarrow p_{i+1}\gamma' \boxed{\llparenthesis \gamma'', \{\phi\} \rrparenthesis}$ in $\mathcal{BP}_i$

BDPDS Computation-X Operators-Call statements

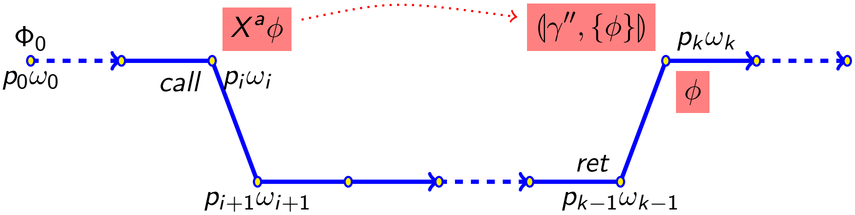


$p_i\omega_i \models X^a\phi$ iff $p_k\omega_k \models \phi$

for $p_i\gamma \xrightarrow{call} p_{i+1}\gamma'\gamma''$ in $\mathcal{P}_i$:

- $(\!|p_i, \{X^g\phi\}|\!)\gamma \rightarrow (\!|p_{i+1}, \{\phi\}|\!)\gamma\gamma''$ in $\mathcal{BP}_i$
- $(\!|p_i, \{X^a\phi\}|\!)\gamma \rightarrow p_{i+1}\gamma' \boxed{(\!|\gamma'', \{\phi\}|\!)}$ in $\mathcal{BP}_i$

for $p_{k-1}\beta \xrightarrow{ret} p_k\epsilon$ in $\mathcal{P}_i$



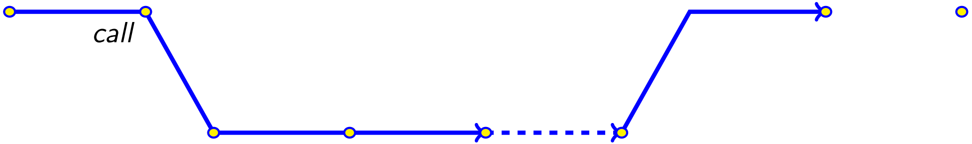
$p_i\omega_i \models X^a\phi$ iff $p_k\omega_k \models \phi$

for $p_i\gamma \xrightarrow{call} p_{i+1}\gamma'\gamma''$ in $\mathcal{P}_i$:

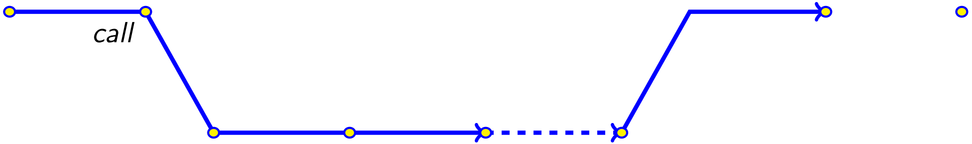
- $(\!|p_i, \{X^g\phi\}|\!)\gamma \rightarrow (\!|p_{i+1}, \{\phi\}|\!)\gamma\gamma''$ in $\mathcal{BP}_i$
- $(\!|p_i, \{X^a\phi\}|\!)\gamma \rightarrow p_{i+1}\gamma' \boxed{(\!|\gamma'', \{\phi\}|\!)}$ in $\mathcal{BP}_i$

for $p_{k-1}\beta \xrightarrow{ret} p_k\epsilon$ in $\mathcal{P}_i$:

- $p_k \boxed{(\!|\gamma'', \{\phi\}|\!)} \rightarrow (\!|p_k, \boxed{\{\phi\}}|\!)\gamma''$



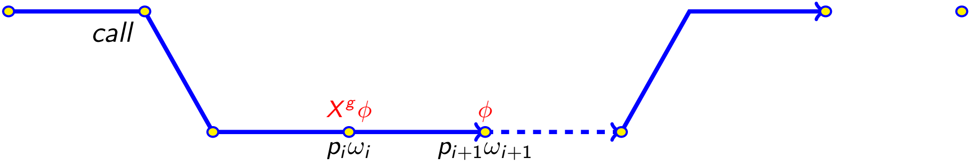
for $p_i\gamma \xrightarrow{int} p_{i+1}\omega$ in $\mathcal{P}_i$:



for $p_i\gamma \xrightarrow{int} p_{i+1}\omega$ in $\mathcal{P}_i$:

- $(\!| p_i, \{X^g\phi\} |\!)\gamma \rightarrow (\!| p_{i+1}, \{\phi\} |\!)\omega$ in $\mathcal{BP}_i$

BDPDS Computation-X Operators- Int statements

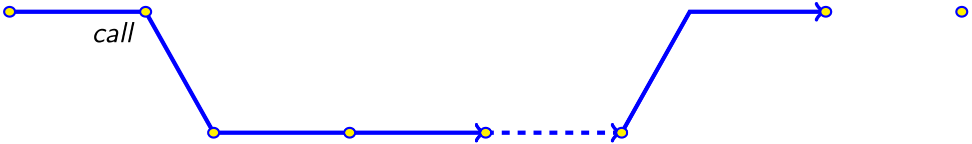


$p_i\omega_i \models X^g\phi$ iff $p_{i+1}\omega_{i+1} \models \phi$

for $p_i\gamma \xrightarrow{int} p_{i+1}\omega$ in $\mathcal{P}_i$:

- $(\!|p_i, \{X^g\phi\}|\!)\gamma \rightarrow (\!|p_{i+1}, \{\phi\}|\!)\omega$ in $\mathcal{BP}_i$

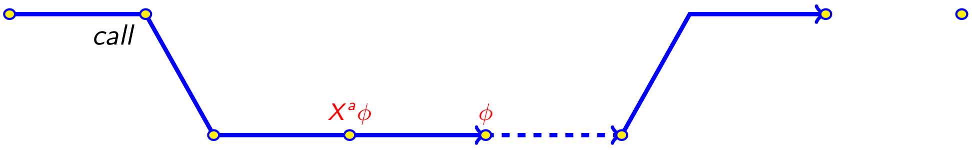
BDPDS Computation-X Operators- Int statements



for $p_i\gamma \xrightarrow{int} p_{i+1}\omega$ in $\mathcal{P}_i$:

- $(\!|p_i, \{X^g\phi\}|\!)\gamma \rightarrow (\!|p_{i+1}, \{\phi\}|\!)\omega$ in $\mathcal{BP}_i$
- $(\!|p_i, \{X^a\phi\}|\!)\gamma \rightarrow (\!|p_{i+1}, \{\phi\}|\!)\omega$ in $\mathcal{BP}_i$

BDPDS Computation-X Operators- Int statements

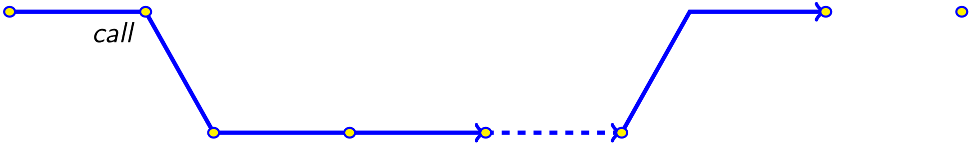


$$p_i\omega_i \models X^a\phi \text{ iff } p_{i+1}\omega_{i+1} \models \phi$$

for $p_i\gamma \xrightarrow{int} p_{i+1}\omega$ in $\mathcal{P}_i$:

- $(\!|p_i, \{X^g\phi\}|\!)\gamma \rightarrow (\!|p_{i+1}, \{\phi\}|\!)\omega$ in $\mathcal{BP}_i$
- $(\!|p_i, \{X^a\phi\}|\!)\gamma \rightarrow (\!|p_{i+1}, \{\phi\}|\!)\omega$ in $\mathcal{BP}_i$

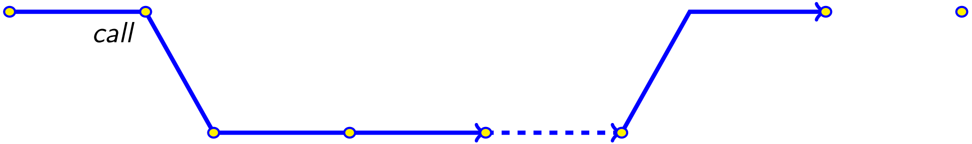
BDPDS Computation-X Operators- Int statements



for $p_i\gamma \xrightarrow{int} p_{i+1}\omega$ in $\mathcal{P}_i$:

- $(\!|p_i, \{X^g\phi\}|\!)\gamma \rightarrow (\!|p_{i+1}, \{\phi\}|\!)\omega$ in $\mathcal{BP}_i$
- $(\!|p_i, \{X^a\phi\}|\!)\gamma \rightarrow (\!|p_{i+1}, \{\phi\}|\!)\omega$ in $\mathcal{BP}_i$

for $p_i\gamma \xrightarrow{int} p_{i+1}\omega \triangleright p_s\omega_s$ in $\mathcal{P}_i$ ($p_s\omega_s \in \mathcal{P}_j$):



for $p_i\gamma \xrightarrow{int} p_{i+1}\omega$ in $\mathcal{P}_i$:

- $(\!|p_i, \{X^g\phi\}|\!)\gamma \rightarrow (\!|p_{i+1}, \{\phi\}|\!)\omega$ in $\mathcal{BP}_i$
- $(\!|p_i, \{X^a\phi\}|\!)\gamma \rightarrow (\!|p_{i+1}, \{\phi\}|\!)\omega$ in $\mathcal{BP}_i$

for $p_i\gamma \xrightarrow{int} p_{i+1}\omega \rhd p_s\omega_s$ in $\mathcal{P}_i$ ($p_s\omega_s \in \mathcal{P}_j$):

- $(\!|p_i, \{X^g\phi\}|\!)\gamma \rightarrow (\!|p_{i+1}, \{\phi\}|\!)\omega \rhd (\!|p_s, f_j|\!)\omega_s$ in $\mathcal{BP}_i$
- $(\!|p_i, \{X^a\phi\}|\!)\gamma \rightarrow (\!|p_{i+1}, \{\phi\}|\!)\omega \rhd (\!|p_s, f_j|\!)\omega_s$ in $\mathcal{BP}_i$

Theorem

Theorem

Given a DPN $\mathcal{M} = \{\mathcal{P}_1, ..., \mathcal{P}_n\}$, a single-indexed CARET formula $f = f_1 \wedge f_2 ... \wedge f_n$, we can compute a BDPN $\mathcal{BM} = \{\mathcal{BP}_1, ..., \mathcal{BP}_n\}$ such that $\mathcal{M} \models f$ iff $\mathcal{BM}$ has an accepting run.

DPNs communicating via Locks (L-DPNs)

L-DPNs

a L-DPN is a DPN where pushdown processes communicate via locks.

DPNs communicating via Locks (L-DPNs)

L-DPNs

a L-DPN is a DPN where pushdown processes communicate via locks.

Nested Lock Access

a L-DPNs with Nested Lock Access: is a L-DPN s.t. in all executions, the locks are accessed in a well-nested manner, i.e, an execution can only release the latest lock it acquired that is not released yet.

DPNs communicating via Locks (L-DPNs)

L-DPNs

a L-DPN is a DPN where pushdown processes communicate via locks.

Nested Lock Access

a L-DPNs with Nested Lock Access: is a L-DPN s.t. in all executions, the locks are accessed in a well-nested manner, i.e, an execution can only release the latest lock it acquired that is not released yet.

Theorem

Single-indexed CARET model-checking for L-DPNs with nested Lock access can be reduced to single-indexed CARET model-checking for DPNs

Q & A

Thank you for your listening!