On CARET model-checking of pushdown systems: 
application to malware detection

Huu Vu Nguyen

To cite this version:

HAL Id: tel-02167961
https://tel.archives-ouvertes.fr/tel-02167961
Submitted on 28 Jun 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
On CARET Model-Checking of Pushdown Systems: Application to Malware Detection

Par Huu Vu NGUYEN

Thèse de doctorat en Informatique
Dirigée par Tayssir TOUILI
Présentée et soutenue publiquement à Villetaneuse le 05/07/2018

Président du jury: BOUAJJANI Ahmed Prof Université Paris Diderot - Paris 7
Rapporteurs: CAUCAL Didier Prof Université Paris-Est Marne-la-Vallée
LUGIEZ Denis Prof Aix-Marseille Université
VOJNAR Tomáš Prof Brno University of Technology
Examineurs: ESPARZA Javier Prof Technische Universität München
LEROUX Jérôme DR CNRS, LaBRI et Université de Bordeaux
SCHWOON Stefan MdC École Normale Supérieure de Cachan
Directrice de thèse: TOUILI Tayssir DR CNRS, LIPN et Université Paris 13
Titre: CARET MODEL-CHECKING D’AUTOMATES À PILES: APPLICATION À LA DETECTION DE MALWARE

Résumé: Cette thèse s’attaque au problème de détection de malware en utilisant des techniques de model-checking: les automates à pile sont utilisés pour modéliser les programmes binaires, et la logique CARET (et ses variantes) sont utilisées pour représenter les comportements malicieux. La détection de malware est alors réduite au problème de model-checking des automates à pile par rapport à ces logiques CARET.

Cette thèse propose alors différents algorithmes de model-checking des automates à pile par rapport à ces logiques CARET et montre comment ceci peut s’appliquer pour la détection de malware.

Mots clefs: détection de malware, automates à pile, vérification, automates
Title: On CARET Model-Checking of Pushdown Systems: Application to Malware Detection

Abstract: The number of malware is growing significantly fast. Traditional malware detectors based on signature matching or code emulation are easy to get around. To overcome this problem, model-checking emerges as a technique that has been extensively applied for malware detection recently. Pushdown systems were proposed as a natural model for programs, since they allow to keep track of the stack, while extensions of LTL and CTL were considered for malicious behavior specification. However, LTL and CTL like formulas don’t allow to express behaviors with matching calls and returns. In this thesis, we propose to use CARET (a temporal logic of calls and returns) for malicious behavior specification. CARET model checking for Pushdown Systems (PDSs) was never considered in the literature. Previous works only dealt with the model checking problem for Recursive State Machine (RSMs). While RSMs are a good formalism to model sequential programs written in structured programming languages like C or Java, they become non suitable for modeling binary or assembly programs, since, in these programs, explicit push and pop of the stack can occur. Thus, it is very important to have a CARET model checking algorithm for PDSs. We tackle this problem in this thesis. We reduce it to the emptiness problem of Büchi Pushdown Systems.

Since CARET formulas for malicious behaviors are huge, we propose to extend CARET with variables, quantifiers and predicates over the stack. This allows to write compact formulas for malicious behaviors. Our new logic is called Stack linear temporal Predicate logic of CAlls and RETurns (SPCARET). We reduce the malware detection problem to the model checking problem of PDSs against SPCARET formulas, and we propose efficient algorithms to model check SPCARET formulas for PDSs. We implemented our algorithms in a tool for malware detection. We obtained encouraging results.

We then define the Branching temporal logic of CAlls and RETurns (BCARET) that allows to write branching temporal formulas while taking into account the matching between calls and returns and we proposed model-checking algorithms of PDSs for BCARET formulas.

Finally, we consider Dynamic Pushdown Networks (DPNs) as a natural model for multithreaded programs with (recursive) procedure calls and thread creation. We show that the model-checking problem of DPNs against CARET formulas is decidable.

Keywords: malware detection, pushdown systems, model-checking, automata
Acknowledgements

This thesis would have been impossible without the guidance, aid and support of many people. I cannot thank them enough for all their help.

First of all, I would like to express my deepest gratitude and appreciation for Prof. Tayssir Touili, my supervisor, for her patient guidance, enthusiastic encouragement and kind advices. It was an honor for me to share her immense scientific knowledge during years as her student. Her positive outlook and confidence in research inspired me a lot and made me more confident. Her detailed comments contributed enormously to the production of this thesis. Without her constant support, this thesis would not have been completed.

I would like to express my sincere gratitude to Prof. Didier Caucal, Prof. Denis Lugiez and Prof. Tomáš Vojnar for accepting to be my thesis referees, and to Prof. Ahmed Bouajjani, Prof. Javier Esparza, Prof. Jérôme Leroux and Prof. Stefan Schwoon for accepting to be a member of my thesis defense jury.

I would like to express my special thanks to Prof. Jean-Pierre Astruc, Prof. Ahmed Bouajjani, Prof. Frédérique Bassino and Madam. Brigitte Guéveneux for their generous support in prolonging my residence permit. Without their help, I cannot stay in France until the thesis defense day.

I am very grateful to many staff members of the Institut de Recherche en Informatique Fondamentale (IRIF) and Laboratoire d’Informatique de Paris Nord (LIPN) for their logistics help.

Many thanks to all my labmates and my friends for being there, for sharing good and bad times, for the encouragements and for our parties. Your presence made my life better and enjoyable during my graduate student years.

I would like to thank my family, especially Mom and Dad, for their unconditional support, love, and encouragement throughout my time in graduate school. I could not go this far without them.

Finally, this thesis would not have been possible without the financial support of the FUI project AiC 2.0.
## Contents

1 Introduction ................................................................................. 1
   1.1 CARET Model Checking For Pushdown Systems ............... 2
   1.2 CARET Model Checking for Malware Detection ............ 3
   1.3 Branching Temporal Logic of Calls and Returns for Pushdown Systems .................................................. 4
   1.4 BCARET Model Checking for Malware Detection ........ 6
   1.5 CARET analysis of multithreaded programs .................. 6
   1.6 Thesis Organization ............................................................. 7

2 CARET Model Checking For Pushdown Systems ...................... 9
   2.1 Linear Temporal Logic of Calls and Returns - CARET .... 9
   2.2 CARET for Pushdown Systems ............................................ 12
      2.2.1 Pushdown Systems: A model for sequential programs . 12
      2.2.2 CARET for Pushdown Systems ................................ 13
   2.3 CARET Model-Checking for Pushdown Systems .......... 14
      2.3.1 Büchi Pushdown Systems ............................................. 14
      2.3.2 From CARET Model-Checking for PDSs to the Emptiness Problem of BPDSs ........................................... 15
   2.4 CARET Model Checking for PDS with Regular Valuations . 40
   2.5 Applications .................................................................... 41
      2.5.1 Modeling Malicious Behaviors using CARET .......... 41
      2.5.2 Checking API Usage rules ......................................... 43
      2.5.3 Security Properties ..................................................... 44
   2.6 Experiments .................................................................... 44
   2.7 Conclusion ..................................................................... 46

3 CARET Model Checking for Malware Detection ....................... 49
   3.1 Malicious Behaviour Specification ................................. 49
      3.1.1 Environments, Predicates and Regular Variable Expressions ........................................................... 50
      3.1.2 The Stack linear temporal Predicate logic of CALLs and RETurns - SPCARET .............................. 50
      3.1.3 Modelling Malicious Behaviours Using SPCARET .......................... 52
   3.2 SPCARET Model-Checking for Pushdown Systems .......... 53
      3.2.1 Using CARET Model-Checking ................................. 53
      3.2.2 SPCARET\textsuperscript{c} .............................................. 54
   3.3 PCARET\textsuperscript{c} Model-Checking for Pushdown Systems .. 54
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3.1</td>
<td>Symbolic Büchi Pushdown Systems</td>
<td>55</td>
</tr>
<tr>
<td>3.3.2</td>
<td>From PCARET\textsuperscript{c} Model-Checking for PDSs to the Emptiness Problem of SBPDSs</td>
<td>55</td>
</tr>
<tr>
<td>3.4</td>
<td>SPCARET\textsuperscript{c} Model-Checking For PDS</td>
<td>77</td>
</tr>
<tr>
<td>3.5</td>
<td>Experiments</td>
<td>77</td>
</tr>
<tr>
<td>3.6</td>
<td>Conclusion</td>
<td>78</td>
</tr>
<tr>
<td>4.1</td>
<td>Branching Temporal Logic of Calls and Returns - BCARET</td>
<td>81</td>
</tr>
<tr>
<td>4.2</td>
<td>Application</td>
<td>83</td>
</tr>
<tr>
<td>4.3</td>
<td>BCARET Model-Checking for Pushdown Systems</td>
<td>85</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Alternating Büchi Pushdown Systems (ABPDSs)</td>
<td>85</td>
</tr>
<tr>
<td>4.3.2</td>
<td>From BCARET model checking of PDSs to the membership problem in ABPDSs</td>
<td>86</td>
</tr>
<tr>
<td>4.4</td>
<td>BCARET model-checking for PDSs with regular valuations</td>
<td>108</td>
</tr>
<tr>
<td>4.4.1</td>
<td>Multi Automata</td>
<td>109</td>
</tr>
<tr>
<td>4.4.2</td>
<td>From BCARET model checking of PDSs with regular valuations to the membership problem in ABPDSs</td>
<td>109</td>
</tr>
<tr>
<td>4.5</td>
<td>Conclusion</td>
<td>113</td>
</tr>
<tr>
<td>5.1</td>
<td>Stack Branching temporal Predicate logic of CAlls and RETurns</td>
<td>115</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Environments, Predicates and Regular Variable Expressions</td>
<td>115</td>
</tr>
<tr>
<td>5.1.2</td>
<td>The Stack Branching temporal Predicate logic of CAlls and RETurns - SBPCARET</td>
<td>116</td>
</tr>
<tr>
<td>5.1.3</td>
<td>Modelling Malicious Behaviours Using SBPCARET</td>
<td>118</td>
</tr>
<tr>
<td>5.2</td>
<td>SBPCARET Model-Checking for Pushdown Systems</td>
<td>119</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Variable Automata</td>
<td>119</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Symbolic Alternating Büchi Pushdown Systems (SABPDSs)</td>
<td>119</td>
</tr>
<tr>
<td>5.2.3</td>
<td>From SBPCARET model checking of PDSs to the membership problem in SABPDSs</td>
<td>122</td>
</tr>
<tr>
<td>5.3</td>
<td>Conclusion</td>
<td>137</td>
</tr>
<tr>
<td>6.1</td>
<td>Dynamic Pushdown Networks (DPNs)</td>
<td>139</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Definitions</td>
<td>139</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Single-indexed CARET for DPNs</td>
<td>141</td>
</tr>
<tr>
<td>6.2</td>
<td>Applications</td>
<td>141</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The number of malware is growing fast recently. Traditional malware detection techniques including signature matching and code emulation are not efficient enough. While malware writers can use obfuscation techniques to bypass the signature based malware detectors easily, code emulation can only monitor programs in certain execution paths during a short time. To overcome these limitations, model-checking emerges as an efficient technique for malware detection, as model-checking allows to check the behaviors of a program in all its execution traces without executing it.

A lot of efforts have been made to apply model-checking for malware detection [BDD+01, KKS05, CJ03, ST13a, ST12a, ST12b]. In [KKS05], the authors proposed to use finite state graphs to model the program and use the temporal logic CTPL to describe malicious behaviours. However, finite graphs are not precise enough to model programs, as they don’t allow to keep track of the program’s stack.

Being able to record the program’s stack is very important for malware detection as explained in [LKV05]. Indeed, push and pop instructions are frequently used by malware writers for code obfuscation. Moreover, in binary codes and assembly programs, parameters are passed to functions via the stack (i.e., they are pushed on the stack before invoking the function). The values of these parameters determine whether the program has a malicious behavior or not. Thus, it is very important to track the program’s stack for malware detection. To this aim, [ST13a, ST12a, ST12b, ST13c] proposed to use pushdown systems (PDSs) to model programs, and defined extensions of LTL and CTL (called SLTPL and SCTPL) to precisely and succinctly represent malicious behaviors. However, these logics cannot describe properties that require matchings of calls and returns, which is necessary to specify malicious behaviours. For instance, let us consider a typical behaviour of a spyware: it consists in hunting for personal information (emails, bank account information,…) on local drives by searching files matching certain conditions. To do this, the spyware first calls the API function FindFirstFileA to obtain the first matching file. FindFirstFileA will return a search handle $h$. To obtain all matching files, the spyware must continuously call the function FindNextFileA with $h$ as parameter. This behaviour cannot be specified by LTL or CTL since it requires that the return value of the API FindFirstFileA must be used as the
input of the function `FindNextFileA`. CARET was introduced to express these properties that involve matchings of calls and returns [AEM04]. However, only CARET model-checking for Recursive State Machines (RSMs) was considered. RSMs can be seen as a natural model to express the control flow graph of sequential programs with recursive procedure calls. Each procedure is modelled as a module. The invocation to a procedure is modelled as a call node; the return from a module corresponds to a ret node; and the remaining statements are considered as internal nodes in the RSMs. Thus, RSMs are a good formalism to model sequential programs written in structured programming languages like C or Java. However, they become non suitable for modelling binary or assembly programs; since, in these programs, explicit push and pop instructions can occur. This makes impossible the use of RSMs to model assembly programs and binary codes directly.

Thus, it is very important to be able to model check binary and assembly programs against CARET formulas. One can argue that from a binary/assembly program, one can compute a PDS as described in [ST12a] and then apply the translation in [ABE+05] to obtain a RSM and then apply the CARET model-checking algorithm of [AEM04] on this RSM. However, by doing so, we loose the explicit manipulation of the program’s stack. Explicit push and pop instructions are not represented in a natural way anymore, and the stack of the RSM does not correspond to the stack of the assembly program anymore. Thus, it is not possible to state intuitive formulas that correspond to properties of the program’s behaviors on the obtained RSM. Especially, when these formulas talk about the content of the program’s stack. Thus, it is very important to have a direct algorithm for CARET model-checking of PDSs.

1.1 CARET Model Checking For Pushdown Systems

In this thesis, we first consider CARET model-checking for Pushdown Systems. We also consider CARET model checking with regular valuations, where the set of configurations in which an atomic proposition holds is a regular language. Our main contributions in this part are:

1. We propose an algorithm to model-check PDSs against CARET formulas. We also consider CARET model checking for PDSs with regular valuations. We reduce these problems to the emptiness problem of Büchi Pushdown Systems. This latter problem is already solved in [BEM97, ES01].
2. We implemented our techniques in a tool, and we applied it to different case studies. Our results are encouraging. In particular, we were able to apply our tool to detect several malwares.

The results of this part were published in [NT17c]. They are discussed in Chapter 2.

1.2 CARET Model Checking for Malware Detection

Using CARET, the behavior of a spyware (described above) can be expressed by the following formula:

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

where the $\bigvee$ is taken over all possible memory addresses $d$ which contain the values of search handles $h$ in the program. $F^g$ is the standard LTL $F$ operator (in the future), while $F^a$ is a CARET operator that means "in the future, in the same procedural context", and $X^a$ is a CARET operator that means "at the return point of the called procedure" if it is applied at a call point.

In binary codes and assembly programs, the return value of an API function is put in the register $eax$. Thus, the return value of $FindFirstFileA$ is the value of $eax$ at its corresponding return-point. Then, the subformula $F^g(\text{call(FindFirstFileA)} \land X^a(eax = d))$ expresses that there is a call to the API $FindFirstFileA$ and the return value of this function is $d$. When $FindNextFileA$ is invoked, one of its required parameters is the search handle and this search handle must be put on top of the program stack (since parameters are passed through the stack in assembly). The requirement that $d$ is on top of the program stack is expressed by the regular valuation $d \Gamma^*$. Thus, the subformula $\text{call(FindNextFileA)} \land d \Gamma^*$ expresses that $FindNextFileA$ is called with $d$ as parameter ($d$ stores the information of the search handle). Thus, $\psi_{sf}$ expresses then that there is a call to the API $FindFirstFileA$ with the return value $d$ (the search handle), followed by a call to the function $FindNextFileA$ with $d$ on the top of the stack.

However, this formula is huge, as it considers the disjunction (of different CARET formulas) over all possible memory addresses $d$ which contain the information of search handles $h$ in the program. To represent it in a more succinct fashion, we follow the idea of [KKSV05, ST13a, ST12b] and extend CARET with variables, quantifiers, and predicates over the stack. We call our new logic SPCARET. We define also PCARET formulas to be SPCARET formulas that do not use predicates over the stack. The above formula can be compactly represented in SPCARET as follows:
\[ \psi_{sf2} = \exists x F^g (\text{call(FindFirstFileA)} \land X^a (\varepsilon a x = x) \land F^a (\text{call(FindNextFileA)} \land x \Gamma^a)) \]

Our main contributions in this part are:

1. We introduce the PCARET and SPCARET logics and show how they can be used to succinctly and precisely describe different malicious behaviors. We identify the sublogics PCARET\text{^c} and SPCARET\text{^c}, which are subclasses of PCARET and SPCARET respectively where the time operators on caller paths are removed. We show that PCARET\text{^c} and SPCARET\text{^c} are sufficient to describe most malicious behaviors.

2. We propose efficient algorithms to model check PCARET\text{^c} and SPCARET\text{^c} formulas for PDSs. Our algorithms are based on reducing the model checking problem to the emptiness problem of Symbolic Büchi Pushdown Systems. This latter problem is solved in [ES01].

3. We implemented our techniques in a tool for malware detection. We obtained encouraging results. Our tool was able to detect several malwares and to determine that benign programs are benign. We compared the performance of our new SPCARET tool against our CARET model checking tool. Our new tool behaves much better, as the CARET tool timeout in most cases.

The results of this part were published in [NT17b]. They are discussed in Chapter 3.

### 1.3 Branching Temporal Logic of Calls and Returns for Pushdown Systems

However, CARET and SPCARET are not sufficient to describe several kinds of properties, such as branching-time properties that require matchings of calls and returns. In [ACM06], the authors introduced VP-\(\mu\), a branching-time temporal logic that allows to talk about matchings between calls and returns, and proposed an algorithm to model-check VP-\(\mu\) formulas for Recursive State Machines (RSMs) [ABE+05]. VP-\(\mu\) can be seen as an extension of the modal \(\mu\)-calculus which allows to talk about matching of calls and returns. However, as discussed before, RSMs become non suitable for modelling binary or assembly programs; since, in these programs, explicit push and pop instructions can occur. Thus, it is very important to have a direct algorithm for model-checking a branching-time temporal logic with matching of calls and returns for PDSs.

In addition, VP-\(\mu\) is a heavy formalism that can’t be used by novice users. Indeed, VP-\(\mu\) can be seen as an extension of the modal \(\mu\)-calculus with
1.3. Branching Temporal Logic of Calls and Returns for Pushdown Systems

several modalities $⟨\text{loc}⟩$, $[\text{loc}]$, $⟨\text{call}⟩$, $[\text{call}]$, $⟨\text{ret}⟩$, $[\text{ret}]$ that allow to distinguish between calls, returns, and other statements (neither calls nor returns). Writing a simple specification in VP-$\mu$ is complicated. For example, the following simple property stating that "the configuration $e$ can be reached in the same procedural context as the current configuration" can be described (as shown in [ACM06]) by the complex VP-$\mu$ formula $\varphi'_2 = \mu X (e \lor ⟨\text{loc}⟩ X \lor ⟨\text{call}⟩ \varphi'_3 \{X\})$ where $\varphi'_3 = \mu Y (⟨\text{ret}⟩ R_1 \lor ⟨\text{loc}⟩ Y \lor ⟨\text{call}⟩ Y \{Y\})$. Thus, we need to define a more intuitive branching-time temporal logic (in the style of CTL) that allows to talk naturally and intuitively about matching calls and returns.

Therefore, we define in the third part of the thesis the Branching temporal logic of CAlls and RETurns BCARET. BCARET can be seen as an extension of CTL with operators that allow to talk about matchings between calls and returns. Using BCARET, the above reachability property can be described in a simple way by the formula $EF^a e$ where $EF^a$ is a BCARET operator that means "there exists a run on which eventually in the future in the same procedural context". We consider the model-checking problem of PDSs against BCARET formulas with "standard" valuations (where an atomic proposition holds at a configuration $c$ or not depends only on the control state of $c$, not on its stack) as well as regular valuations (where the set of configurations in which an atomic proposition holds is a regular set of configurations). We show that these problems can be effectively solved by a reduction to the emptiness problem of Alternating Büchi Pushdown Systems (ABPDSs). The latter problem can be solved effectively in [ST11]. Note that the regular valuation case cannot be solved by translating the PDSs to RSMs since as said previously, by doing the translation of PDSs to obtain RSMs, we loose the structure of the program’s stack. Our main contributions in this part are:

1. We define the Branching temporal logic of CAlls and RETurns BCARET and show how it can be used to specify malicious behaviors.

2. We propose algorithms to model-check PDSs against BCARET formulas. We also consider BCARET model checking for PDSs with regular valuations, where the set of configurations in which an atomic proposition holds is a regular language. We reduce these problems to the emptiness problem of Alternating Büchi Pushdown Systems.

The results of this part were published in [NT18]. They are discussed in Chapter 4.
1.4 BCARET Model Checking for Malware Detection

As for CARET, in order to write efficient and succinct BCARET formulas to specify malicious behaviors, we extend BCARET with variables, quantifiers, and predicates over the stack. We call our new logic Stack Branching temporal Predicate logic of CAlls and RETurns (SBPCARET). Our main contributions in this part are:

1. We introduce the SBPCARET logic and show how it can be used to succinctly and precisely describe different malicious behaviors.

2. We propose a symbolic algorithm to model check SBPCARET formulas for PDSs. Our algorithm is based on reducing the model checking problem to the emptiness problem of Symbolic Alternating Büchi Pushdown Systems.

The results of this part are discussed in Chapter 5.

1.5 CARET analysis of multithreaded programs

As mentioned previously, Pushdown Systems (PDSs) are known to be a natural model for sequential programs [Sch02]. Therefore, networks of pushdown systems are a natural model for concurrent programs where each PDS represents a sequential component of the system. In this context, Dynamic pushdown Networks (DPNs) [BMT05] were introduced by Bouajjani et al. as a natural model of multithreaded programs with procedure calls and thread creation. Intuitively, a DPN is a network of pushdown processes \( \{P_1, ..., P_n\} \) where each process, represented by a Pushdown system (PDS), can perform basic pushdown actions, call procedures, and spawn new instances of pushdown processes. A lot of previous researches focused on investigating automated methods to verify DPNs. In [BMT05, Lug11, LMW09, GLM+11], the reachability analysis of DPNs are considered. While the model-checking problem for DPNs against double-indexed properties is undecidable, i.e., properties where the satisfiability of an atomic proposition depends on control states of two or more threads [KG06], it is decidable to model-check DPNs against the linear temporal logic (LTL) and the computation tree logic (CTL) with single-indexed properties [ST13b], i.e., properties where the satisfiability of an atomic proposition depends on control states of only one thread.
1.6. Thesis Organization

As mentioned before, CARET is needed to describe several important properties such as malicious behaviors. Thus, to be able to analyse such properties for multithreaded programs, we need to be able to check CARET formulas for DPNs. We tackle this problem in this part. As LTL is a subclass of CARET, CARET model-checking for DPNs with double-indexed properties is also undecidable. Thus, in this part, we consider the model-checking problem for DPNs against single-indexed CARET formulas and show that it is decidable. A single-indexed CARET formula is a formula in the form $\bigwedge f_i$ where $f_i$ is a CARET formula over a certain PDS $P_i$. A DPN satisfies $\bigwedge f_i$ iff all instances of the PDS $P_i$ created in the network satisfy the subformula $f_i$.

The model-checking problem of DPNs against single-indexed CARET formulas is non-trivial because the number of instances of pushdown processes in DPNs can be unbounded. It is not sufficient to check if every PDS $P_i$ satisfies the corresponding formula $f_i$. Indeed, we need to ensure that all instances of $P_i$ created during a run of DPN satisfy the formula $f_i$. Also, an instance of $P_i$ should not be checked if it is not created during the run of DPNs. In this part, we solve these problems. Our main contributions in this part are:

1. We show how to use single-indexed CARET formulas to specify malicious behaviors.

2. We show that single-indexed CARET model checking is decidable for DPNs. We reduce the problem of checking whether Dynamic Pushdown Networks satisfy single-indexed CARET formulas to the membership problem for Büchi Dynamic Pushdown Networks (BDPNs).

3. We show that single-indexed CARET model checking is decidable for Dynamic Pushdown Networks communicating via nested locks.

The results of this part were published in [NT17a]. They are discussed in Chapter 6.

1.6 Thesis Organization

In Chapter 2, we show how to model-check CARET formulas for PDSs with both simple valuations and regular valuations. This chapter contains several definitions that are used in the rest of the thesis. Chapter 3 shows how SPCARET can be useful and effective for malware detection. In Chapter 4, we go one step further and define BCARET. Our algorithm for BCARET model-checking for PDSs is described in this chapter. Chapter 5 shows how SBPCARET can be useful and effective for detecting malware. Chapter 6
describes our algorithm for CARET model-checking for DPNs. Finally, the conclusion and future work can be found in Chapter 7.
Chapter 2

CARET Model Checking For Pushdown Systems

CARET (A temporal logic of calls and returns) was introduced by Alur et al. This logic allows to write linear temporal logic formulas while taking into account matching of calls and returns. However, CARET model checking for Pushdown Systems (PDSs) was never considered in the literature. Previous works only dealt with the model checking problem for Recursive State Machine (RSMs). While RSMs are a good formalism to model sequential programs written in structured programming languages like C or Java, they become non suitable for modeling binary or assembly programs, since, in these programs, explicit push and pop of the stack can occur. Thus, it is very important to have a CARET model checking algorithm for PDSs. We tackle this problem in this chapter. We also consider CARET model checking with regular valuations, where the set of configurations in which an atomic proposition holds is a regular language. We reduce these problems to the emptiness problem of Büchi Pushdown Systems (BPDSs). We implemented our technique in a tool, and we applied it to different case studies. Our results are encouraging. In particular, we were able to apply our tool to detect several malwares.

Outline. In Section 2.1, we recall the definition of CARET. In Section 2.2, we define the CARET model-checking problem for PDSs. In Section 2.3, we present our algorithm to reduce the CARET model-checking problem to the emptiness problem of BPDSs. Model checking for PDSs with regular valuations is discussed in Section 2.4. Section 2.5 shows how CARET can be used to specify different malicious behaviors and to describe API usage rules. Our experiments are presented in Section 2.6. Finally, we conclude in Section 2.7.

2.1 Linear Temporal Logic of Calls and Returns - CARET

In this section, we recall the definition of CARET with respect to sequential programs. A CARET formula is interpreted on an infinite path where each state on the path is associated with a tag in the set \{call, ret, int\}. A call-state
denotes an invocation to a procedure of a program while the corresponding \textit{ret-state} denotes the \textit{ret} statement of that procedure. A \textit{simple} statement (neither a \textit{call} nor a \textit{ret} statement) is called an \textit{internal} statement and its associated state is called \textit{int-state}.

Let $\omega = s_0s_1...$ be an infinite path where each state on the path is associated with a tag in the set \{\textit{call}, \textit{ret}, \textit{int}\}. Over $\omega$, three kinds of successors are defined for every position $s_i$:

- \textit{global-successor}: The global-successor of $s_i$ is $s_{i+1}$.
- \textit{abstract-successor}: The abstract-successor of $s_i$ is determined by its associated tag.
  - If $s_i$ is a \textit{call}, there are two cases: (1) if $s_i$ has $s_k$ as a corresponding return-point in $\omega$, then, the abstract successor of $s_i$ is $s_k$; (2) if $s_i$ does not have any corresponding return-point in $\omega$, then, the abstract successor of $s_i$ is $\bot$.
  - If $s_i$ is a \textit{int}, the abstract successor of $s_i$ is $s_{i+1}$.
  - If $s_i$ is a \textit{ret}, the abstract successor of $s_i$ is defined as $\bot$.
- \textit{caller-successor}: The caller-successor of $s_i$ is the most inner unmatched call if there is such a \textit{call}. Otherwise, it is defined as $\bot$.

For example, in Figure 2.1:

- The global-successor of $s_1$ and $s_2$ are $s_2$ and $s_3$ respectively.
- The abstract-successor of $s_2$ and $s_5$ are $s_k$ and $s_9$ respectively.
- The caller-successor of $s_6$, $s_7$, $s_8$ is $s_5$ while the caller-successor of $s_3$, $s_4$, $s_5$, $s_9$ is $s_2$. Note that the caller-successor of $s_0$, $s_1$, $s_2$, $s_k$ is $\bot$. 

Figure 2.1: Three kinds of successors of CARET
2.1. Linear Temporal Logic of Calls and Returns - CARET

A global-path is obtained by applying repeatedly the global-successor operator. Similarly, an abstract-path or a caller-path are obtained by repeatedly applying the abstract-successor and caller-successor respectively. In Figure 2.1, from $s_4$, the global-path is $s_4s_5s_6s_7s_8s_9s_{10}...$, the abstract-path is $s_4s_5s_9s_{10}...$ while the caller-path is $s_4s_2$. Note that the caller-path is always finite.

Given a state $s$, let $\mathcal{P}(s)$ be the procedure to which $s$ belongs. For example, in Figure 2.1, all states on the abstract-path starting from $s_3$ belong to the procedure $proc$, i.e., $\mathcal{P}(s_3) = proc$, $\mathcal{P}(s_9) = proc$, ...

Given a finite set of atomic propositions $AP$. Let $AP' = AP \cup \{call, ret, int\}$. A CARET formula over $AP$ is defined as follows (where $e \in AP'$):

$$
\psi := e \mid \psi \lor \psi \mid \neg \psi \mid X^g\psi \mid X^a\psi \mid X^c\psi \mid \psi U^a\psi \mid \psi U^g\psi \mid \psi U^c\psi
$$

Let $\Sigma = 2^{AP} \times \{call, ret, int\}$. Let $\pi = \pi(0)|\pi(1)|\pi(2)...$ be an $\omega$-word over $\Sigma$. Let $\pi(i)$ be the suffix of $\pi$ starting from $\pi(i)$. Let $next^g_i, next^a_i, next^c_i$ be the global-successor, abstract-successor and caller-successor of $\pi(i)$ respectively. The satisfiability relation is defined inductively as follows:

- $(\pi, i) \models e$, where $e \in AP'$, iff $\pi(i) = (Y, d)$ and $e \in Y$ or $e = d$
- $(\pi, i) \models \psi_1 \lor \psi_2$ iff $(\pi, i) \models \psi_1$ or $(\pi, i) \models \psi_2$
- $(\pi, i) \models \neg \psi$ iff $(\pi, i) \not\models \psi$
- $(\pi, i) \models X^g\psi$ iff $(\pi, next^g_i) \models \psi$
- $(\pi, i) \models X^a\psi$ iff $next^a_i \not= \perp$ and $(\pi, next^a_i) \models \psi$
- $(\pi, i) \models X^c\psi$ iff $next^c_i \not= \perp$ and $(\pi, next^c_i) \models \psi$
- $(\pi, i) \models \psi_1 U^b\psi_2$ (with $b \in \{g, a, c\}$) iff there exists a sequence of positions $h_0, h_1, ..., h_{k-1}, h_k$ where $h_0 = i$, for every $0 \leq j \leq k - 1 : h_{j+1} = next^b_{h_j}, (\pi, h_j) \models \psi_1$ and $\pi(h_k) \models \psi_2$

Then, $\pi \models \psi$ iff $(\pi, 0) \models \psi$. Other CARET operators can be expressed by the above operators: $F^g\psi = true U^g\psi$, $G^g\psi = \neg F^g\neg\psi$, $F^a\psi = true U^a\psi$, $G^a\psi = \neg F^a\neg\psi$, $F^c\psi = true U^c\psi$, $G^c\psi = \neg F^c\neg\psi$.

**Remark 1.** LTL can be seen as the subclass of CARET where the operators $X^c, U^c, X^a, U^a$ are not considered.

Let $\psi$ be a CARET formula over $AP$. The closure of $\psi$, denoted $Cl(\psi)$, is the smallest set that contains $\psi$, call, ret and int and satisfies the following properties:
• if \( \neg \psi' \in \text{Cl}(\psi) \), then \( \psi' \in \text{Cl}(\psi) \)
• if \( X^b \psi' \in \text{Cl}(\psi) \) (with \( b \in \{g, a, c\} \)), then \( \psi' \in \text{Cl}(\psi) \)
• if \( \psi_1 \lor \psi_2 \in \text{Cl}(\psi) \), then \( \psi_1 \in \text{Cl}(\psi), \psi_2 \in \text{Cl}(\psi) \)
• if \( \psi_1 U^b \psi_2 \in \text{Cl}(\psi) \) (with \( b \in \{g, a, c\} \)), then \( \psi_1 \in \text{Cl}(\psi), \psi_2 \in \text{Cl}(\psi), X^b(\psi_1 U^b \psi_2) \in \text{Cl}(\psi) \)
• if \( \psi' \in \text{Cl}(\psi) \), and \( \psi' \) is not in the form \( \neg \psi'' \) then \( \neg \psi' \in \text{Cl}(\psi) \)

A set \( A \subseteq \text{Cl}(\psi) \) is an atom of \( \psi \) if it satisfies the following properties:

• \( \forall \psi' \in \text{Cl}(\psi), \psi' \in A \iff \neg \psi' \notin A \)
• \( \forall \psi' \lor \psi'' \in \text{Cl}(\psi), \psi' \lor \psi'' \in A \iff \psi' \in A \) or \( \psi'' \in A \)
• \( \forall \psi' U^b \psi'' \in \text{Cl}(\psi) \), where \( b \in \{g, a, c\} \), \( \psi' U^b \psi'' \in A \iff \psi'' \in A \) or (\( \psi' \in A \) and \( X^b(\psi' U^b \psi'') \in A \))
• \( A \) includes exactly one element of the set \{call, ret, int\}

Let \( \text{Atoms}(\psi) \) be the set of atoms of \( \psi \). Let \( A \) and \( A' \) be two atoms, we define the following predicates:

• \( \text{AbsNext}(A, A') = \text{true} \) iff for every \( X^a \phi' \in \text{Cl}(\psi) : (X^a \phi' \in A \iff \phi' \in A') \).
• \( \text{GlNext}(A, A') = \text{true} \) iff for every \( X^g \phi' \in \text{Cl}(\psi) : (X^g \phi' \in A \iff \phi' \in A') \)
• \( \text{CallerNext}(A, A') = \text{true} \) iff for every \( X^c \phi' \in \text{Cl}(\psi) : (X^c \phi' \in A \iff \phi' \in A') \).

We define \( \text{NexCallerForms}(A) \) to be a function which returns the caller-formulas in \( A \). Formally, \( \text{NexCallerForms}(A) = \{X^c \phi' \mid X^c \phi' \in A\} \). Similarly, let \( \text{NexAbsForms}(A) = \{X^a \phi' \mid X^a \phi' \in A\} \) be a function which returns the abstract-formulas in \( A \).

### 2.2 CARET for Pushdown Systems

#### 2.2.1 Pushdown Systems: A model for sequential programs

Pushdown systems is a standard model that was extensively used to model sequential programs. Translations from sequential programs to PDSs can be
2.2. CARET for Pushdown Systems

found e.g. in [Sch02]. We apply the translation of [ST12a] together with the tools IDA Pro [IDA] and Jakstab [KV08] to obtain Pushdown Systems from binary programs. In order to be able to define CARET formulas on PDSs, we need to adapt the PDS model in order to record whether a rule of a PDS corresponds to a call, a return, or another statement:

Definition 1. A Labelled Pushdown System (PDS) \( \mathcal{P} \) is a tuple \( (P, \Gamma, \Delta) \), where \( P \) is a finite set of control locations, \( \Gamma \) is a finite set of stack alphabet, and \( \Delta \) is a finite subset of \( ((P \times \Gamma) \times (P \times \Gamma^*) \times \{ \text{call, ret, int} \}) \). If \( ((p, \gamma), (q, \omega), t) \in \Delta \) \( (t \in \{ \text{call, ret, int} \}) \), we also write \( (p, \gamma) \xrightarrow{t} (q, \omega) \in \Delta \). Rules of \( \Delta \) are of the following form, where \( p, q \in P, \gamma, \gamma_1, \gamma_2 \in \Gamma, \) and \( \omega \in \Gamma^* \):

\[
\begin{align*}
\bullet \ (r_1): & \quad (p, \gamma) \xrightarrow{\text{call}} (q, \gamma_1 \gamma_2) \\
\bullet \ (r_2): & \quad (p, \gamma) \xrightarrow{\text{ret}} (q, \varepsilon) \\
\bullet \ (r_3): & \quad (p, \gamma) \xrightarrow{\text{int}} (q, \omega)
\end{align*}
\]

Intuitively, a rule of the form \( (p, \gamma) \xrightarrow{\text{call}} (q, \gamma_1 \gamma_2) \) corresponds to a call statement. Such a rule usually models a statement of the form \( \gamma \xrightarrow{\text{call proc}} \gamma_2 \). In this rule, \( \gamma \) is the control point of the program where the function call is made, \( \gamma_1 \) is the entry point of the called procedure, and \( \gamma_2 \) is the return point of the call. A rule \( r_2 \) models a return statement, whereas a rule \( r_3 \) corresponds to a simple statement (neither a call nor a return). A configuration of \( \mathcal{P} \) is a pair \( (p, \omega) \), where \( p \) is a control location and \( \omega \in \Gamma^* \) is the stack content. \( \mathcal{P} \) defines a transition relation \( \Rightarrow_{\mathcal{P}} \) as follows: If \( (p, \gamma) \xrightarrow{t} (q, \omega) \) \( (t \in \{ \text{call, ret, int} \}) \), then for every \( \omega' \in \Gamma^* \), \( (p, \gamma \omega') \Rightarrow_{\mathcal{P}} (q, \omega') \). In other words, \( (q, \omega') \) is an immediate successor of \( (p, \gamma \omega') \). A run (or an execution) of \( \mathcal{P} \) from \( (p_0, \omega_0) \) is an infinite sequence \( (p_0, \omega_0) (p_1, \omega_1) \ldots \) where \( (p_i, \omega_i) \in P \times \Gamma^* \) and for every \( i \geq 0 \), \( (p_{i+1}, \omega_{i+1}) \) is an immediate successor of \( (p_i, \omega_i) \). Let \( \Rightarrow_{\mathcal{P}} \) be the reflexive and transitive closure of \( \Rightarrow_{\mathcal{P}} \). Given a configuration \( (p, \omega) \), let \( \text{Traces}(p, \omega) \) be the set of all possible runs starting from \( (p, \omega) \).

2.2.2 CARET for Pushdown Systems

Our goal in this chapter is to define and perform CARET model checking for PDSs. Let then \( \lambda : P \rightarrow 2^{AP} \) be a labeling function that assigns to each control point of \( P \) a set of atomic propositions. Let \( \pi = (p_0, \omega_0) (p_1, \omega_1) \ldots \) be an execution of \( \mathcal{P} \). We associate to each configuration \( (p_i, \omega_i) \) of \( \pi \) a tag \( t_i \) in \{ call, int, ret \} as follows:

\[
\bullet \text{ If } (p_i, \omega_i) \Rightarrow_{\mathcal{P}} (p_{i+1}, \omega_{i+1}) \text{ corresponds to a call statement, then } t_i = \text{call}.
\]
Chapter 2. CARET Model Checking for Pushdown Systems

- If \( \langle p_i, \omega_i \rangle \Rightarrow_P \langle p_{i+1}, \omega_{i+1} \rangle \) corresponds to a return statement, then \( t_i = \text{ret} \).

- If \( \langle p_i, \omega_i \rangle \Rightarrow_P \langle p_{i+1}, \omega_{i+1} \rangle \) corresponds to a simple statement (neither a call statement nor a return statement), then \( t_i = \text{int} \).

Let \( \psi \) be a CARET formula over \( AP \). Then we say that \( \pi \models \psi \) iff \((\lambda(p_0), t_0)(\lambda(p_1), t_1) \cdots \models \psi \).

Let \( \langle p, \omega \rangle \) be a configuration of \( P \). We say that \( \langle p, \omega \rangle \models \psi \) iff there exists a path \( \pi \) that starts at \( \langle p, \omega \rangle \) such that \( \pi \models \psi \).

2.3 CARET Model-Checking for Pushdown Systems

In this section, we show how to reduce the CARET model checking problem for Pushdown Systems to the emptiness problem of Büchi Pushdown Systems. The latter problem is already solved in [BEM97, ES01].

2.3.1 Büchi Pushdown Systems

Definition 2. A Büchi Pushdown System (BPDS) is a tuple \((P, \Gamma, \Delta, F)\) where \((P, \Gamma, \Delta)\) is a Pushdown System (PDS) and \(F \subseteq P\) is a finite set of accepting control locations. A run of a BPDS is accepted iff it visits infinitely often some control locations in \(F\).

Definition 3. A Generalized Büchi Pushdown System (GBPDS) is a tuple \((P, \Gamma, \Delta, F)\), where \((P, \Gamma, \Delta)\) is a PDS and \(F = \{F_1, \ldots, F_k\}\) is a set of sets of accepting control locations. A run of a GBPDS is accepting iff it visits infinitely often some control locations in \(F_i\) for every \(1 \leq i \leq k\).

For a BPDS or a GBPDS \(BP\), we let \(\mathcal{L}(BP)\) be the set of configurations \(\langle p, \omega \rangle\) such that \(BP\) has an accepting run from \(\langle p, \omega \rangle\). We have the following properties:

Proposition 1. [ST13a] Given a Generalized Büchi Pushdown System \(BP\), we can effectively compute a Büchi Pushdown System \(BP'\) s.t. \(\mathcal{L}(BP) = \mathcal{L}(BP')\).

Theorem 1. [ES01] Given a Büchi Pushdown System \(P = (P, \Gamma, \Delta, F)\), for every configuration \(\langle p, \omega \rangle \in P \times \Gamma^*\), whether or not \(\langle p, \omega \rangle\) is in \(\mathcal{L}(BP)\) can be decided in time \(O(|P| \cdot |\Delta|^2)\).
2.3. CARET Model-Checking for Pushdown Systems

2.3.2 From CARET Model-Checking for PDSs to the Emptiness Problem of BPDSs

Let $\mathcal{P} = (P, \Gamma, \Delta)$ be a PDS, $\lambda : P \rightarrow 2^{AP}$ be a labeling function, $\psi$ be a CARET formula over $AP$. In this section, we show how to build a Generalized Büchi Pushdown System $BP_\psi$ s.t. $\mathcal{P}$ has an execution $\pi$ from $\langle p, \omega \rangle$ s.t. $\pi$ satisfies $\psi$ iff there exists an atom $A \in \text{Atoms}(\psi)$, $\psi \in A$ and $\text{NexCallerForms}(A) = \emptyset$, such that $BP_\psi$ has an accepting run from $\langle \langle p, A, \text{unexit} \rangle, \omega \rangle$ where $\text{unexit}$ is a label expressing that from the configuration $\langle p, \omega \rangle$, the execution of the procedure of $\langle p, \omega \rangle$, $\mathcal{P}(\langle p, \omega \rangle)$, in $\pi$ is never finished (since $\pi$ is an infinite run and $\langle p, \omega \rangle$ is the initial configuration of $\pi$).

Note that the requirement $\text{NexCallerForms}(A) = \emptyset$ comes from the fact that on $BP_\psi$, $\langle \langle p, A \rangle, \omega \rangle$ is the initial configuration, thus, it has no caller-successor. This requirement does not make any restriction for our algorithm.

Let $cl_U(\psi) = \{\phi_1 U^g \chi_1, ..., \phi_k U^g \chi_k\}$ and $cl_{U^a}(\psi) = \{\xi_1 U^a \tau_1, ..., \xi_k U^a \tau_k\}$ be the set of $U^g$-formulas and $U^a$-formulas of $\text{Cl}(\psi)$ respectively. Let $\text{Label} = \{\text{exit, unexit}\}$. We define $BP_\psi = (P', \Gamma', \Delta', F)$ as follows:

- $P' = \{\langle p, A, l \rangle \mid p \in P, l \in \text{Label}, A \in \text{Atoms}(\psi) \text{ and } A \cap AP = \lambda(p) \}$ is the finite set of control locations of $BP_\psi$.
- $\Gamma' = \Gamma \cup (\Gamma \times \text{Atoms}(\psi) \times \text{Label})$ is the finite set of stack symbols of $BP_\psi$.

The transition relation $\Delta'$ of $BP_\psi$ is the smallest set of transition rules satisfying the following:

- $(\alpha_1)$ for every $\langle p, \gamma \rangle \xrightarrow{\text{call}} \langle q, \gamma' \gamma'' \rangle \in \Delta$: $\langle \langle p, A, l \rangle, \gamma \rangle \rightarrow \langle \langle q, A', l' \rangle, \gamma' \gamma'', A, l \rangle \in \Delta'$ for every $A, A' \in \text{Atoms}(\psi)$; $l, l' \in \text{Label}$ such that:
  - $(\beta_0)$ $A \cap \{\text{call, ret, int}\} = \{\text{call}\}$
  - $(\beta_1)$ $A \cap AP = \lambda(p)$
  - $(\beta_2)$ $A' \cap AP = \lambda(q)$
  - $(\beta_3)$ $\text{GlNext}(A, A')$
  - $(\beta_4)$ $\text{CallerNext}(A', A)$
  - $(\beta_5)$ $l' = \text{unexit}$ implies ($l = \text{unexit}$ and $\text{NexAbsForms}(A) = \emptyset$)

- $(\alpha_2)$ for every $\langle p, \gamma \rangle \xrightarrow{\text{ret}} \langle q, \varepsilon \rangle \in \Delta$: $\langle \langle p, A, \text{exit} \rangle, \gamma \rangle \rightarrow \langle \langle q, A', l' \rangle, \varepsilon \rangle \in \Delta'$ for every $A, A' \in \text{Atoms}(\psi)$; $l' \in \text{Label}$ such that:
Chapter 2. CARET Model Checking For Pushdown Systems

\[ (\beta_0) \ A \cap \{ \text{call, ret, int} \} = \{ \text{ret} \} \]
\[ (\beta_1) \ A \cap AP = \lambda(p) \]
\[ (\beta_2) \ A' \cap AP = \lambda(q) \]
\[ (\beta_3) \ \text{GlNext}(A, A') \]
\[ (\beta_4) \ \text{NexAbsForms}(A) = \emptyset \]
\[ (\alpha_2) \ (q, A', l', \gamma_0, A_0, l_0) \rightarrow (q, A', l') \in \Delta' \text{ for every } \gamma_0 \in \Gamma, A_0, A' \in \text{Atoms}(\psi); l', l_0 \in \text{Label} \text{ such that:} \]
\[ (\beta_2) \ \text{AbsNext}(A_0, A') \]
\[ (\beta_3) \ \text{NexCallerForms}(A') = \text{NexCallerForms}(A_0) \]
\[ (\beta_4) \ A' \cap AP = \lambda(q) \]
\[ (\beta_5) \ l_0 = l' \]
\[ (\alpha_3) \text{ for every } \langle p, \gamma \rangle \xrightarrow{\text{int}} \langle q, \omega \rangle \in \Delta: \langle p, A, l \rangle, \gamma \rightarrow \langle q, A', l \rangle, \omega \rangle \in \Delta' \text{ for every } A, A' \in \text{Atoms}(\psi), l \in \text{Label} \text{ such that:} \]
\[ (\beta_0) \ A \cap \{ \text{call, ret, int} \} = \{ \text{int} \} \]
\[ (\beta_1) \ A \cap AP = \lambda(p) \]
\[ (\beta_2) \ A' \cap AP = \lambda(q) \]
\[ (\beta_3) \ \text{GlNext}(A, A') \]
\[ (\beta_4) \ \text{AbsNext}(A, A') \]
\[ (\beta_5) \ \text{NexCallerForms}(A) = \text{NexCallerForms}(A') \]

The generalized Büchi accepting condition \( F \) of \( \mathcal{B}\mathcal{P}_\psi \) is defined as: \( F = \{ F_1 \} \cup F_2 \cup F_3 \) where

1. \( F_1 = P \times 2^{\text{C}(\psi)} \times \{ \text{unexit} \} \)
2. \( F_2 = \{ F_{\psi, \lambda, x_i}^1, ..., F_{\psi, \lambda, x_i}^k \} \) where \( F_{\psi, \lambda, x_i}^i = \{ P \times F_{\phi_{\lambda, x_i}} \times \text{Label} \} \) s.t. \( F_{\phi_{\lambda, x_i}} = \{ A \in \text{Atoms}(\psi) \mid \text{if } \phi_{\lambda, x_i} \in A \text{ then } x_i \in A \} \) for every \( 1 \leq i \leq k \).
3. \( F_3 = \{ F_{\xi, \omega, \tau_i}^1, ..., F_{\xi, \omega, \tau_i}^{k'} \} \) where \( F_{\xi, \omega, \tau_i}^i = \{ P \times F_{\xi, \omega, \tau_i} \times \{ \text{unexit} \} \} \) s.t. \( F_{\xi, \omega, \tau_i} = \{ A \in \text{Atoms}(\psi) \mid \text{if } \xi U_{\tau_i} \in A \text{ then } \tau_i \in A \} \) for every \( 1 \leq i \leq k' \).

**Intuition.** Roughly speaking, we construct \( \mathcal{B}\mathcal{P}_\psi \) as a kind of product of \( P \) and \( \psi \) which ensures that \( \mathcal{B}\mathcal{P}_\psi \) has an accepting run from \( \langle p, A, \text{unexit} \rangle, \omega \rangle \) where \( \psi \in A \) and \( \text{NexCallerForms}(A) = \emptyset \) iff \( P \) has an execution \( \pi \) starting at \( \langle p, \omega \rangle \) s.t. \( \pi \vdash \psi \). The form of control locations of \( \mathcal{B}\mathcal{P}_\psi \) is \( \langle p, A, l \rangle \) where \( A \) contains all sub-formulas of \( \psi \) which are satisfied at the configuration \( \langle p, \omega \rangle \), \( l \) is a label to determine whether the execution of the procedure of \( \langle p, \omega \rangle \),
2.3. CARET Model-Checking for Pushdown Systems 17

Figure 2.2: Case of $X^a\phi' \in A_i$

$P(\langle p, \omega \rangle)$ (as defined in Section 2.1), terminates on $\pi$. A configuration $\langle p, \omega \rangle$ is labeled with $\textit{exit}$ means that the execution of $P(\langle p, \omega \rangle)$ is finished in $\pi$, i.e., the run $\pi$ will run through the procedure $P(\langle p, \omega \rangle)$, reaches its ret statement and exits $P(\langle p, \omega \rangle)$ after that. On the contrary, $\langle p, \omega \rangle$ is labeled with $\textit{unexit}$ means that in $\pi$, the execution of the procedure $P(\langle p, \omega \rangle)$ never terminates, i.e., the run $\pi$ will be stuck in and never exits the procedure $P(\langle p, \omega \rangle)$. Let $\pi = \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \ldots$ be a run of $P$ and let $\langle \langle p_0, A_0, l_0 \rangle, \omega_0 \rangle \langle \langle p_1, A_1, l_1 \rangle, \omega_1 \rangle \ldots$ be a corresponding run of $BP_{\psi}$. Our construction aims to obtain a $BP_{\psi}$, such that $BP_{\psi}$ has an accepting run from $\langle \langle p_i, A_i, l_i \rangle, \omega_i \rangle$ iff $\langle p_i, \omega_i \rangle \models \phi$ for every $\phi \in A_i$. To obtain such a $BP_{\psi}$, in rules ($\alpha_1$), ($\alpha_2$) and ($\alpha_3$), the first class of conditions ($\beta_0$) ensures that the tags $\{\textit{call}, \textit{ret}, \textit{int}\}$ assigned to each configuration of the run are guessed correctly. The second class of conditions ($\beta_1$) and ($\beta_2$) expresses that for every $e \in AP$, ($\pi, i) \models e$ iff $e \in \lambda(p_i)$, and the class of conditions ($\beta_3$) expresses that ($\pi, i) \models X^a\phi'$ iff ($\pi, i + 1) \models \phi'$. Now, let us consider the two most delicate cases:

1. If $\phi = X^a\phi' \in A_i$. There are two possibilities:

- $\langle p_i, \omega_i \rangle \Rightarrow_P \langle p_{i+1}, \omega_{i+1} \rangle$ corresponds to a call statement. Let us consider Figure 2.2 to explain this case. Let $\langle p_k, \omega_k \rangle$ be the abstract-successor of $\langle p_i, \omega_i \rangle$. ($\pi, i) \models X^a\phi'$ iff ($\pi, k) \models \phi'$. Thus, we must have $\phi' \in A_k$. This is ensured by rules $\alpha_1$ and $\alpha_2$: rules $\alpha_1$ allow to record $X^a\phi'$ in the return point of the call, and rules $\alpha_2$ allow to extract and validate $\phi'$ when the return-point is reached. In what follows, we show in more details how this works: Let $\langle p_i, \gamma \rangle \xrightarrow{\text{call}} \langle p_{i+1}, \gamma'\gamma'' \rangle$ be the rule associated with the transition $\langle p_i, \omega_i \rangle \Rightarrow_P \langle p_{i+1}, \omega_{i+1} \rangle$, then we have $\omega_i = \gamma\omega'$ and $\omega_{i+1} = \gamma'\gamma''\omega'$. Let $\langle p_{k-1}, \omega_{k-1} \rangle \Rightarrow_P \langle p_k, \omega_k \rangle$ be the transition that corresponds to the $\textit{ret}$ statement of this call. Let then $\langle p_{k-1}, \beta \rangle \xrightarrow{\text{ret}} \langle p_k, \varepsilon \rangle \in \Delta$ be the corresponding return rule. Then, we have necessarily $\omega_{k-1} = \beta\gamma\omega'$, since as explained in Section 2.2.1, $\gamma''$ is the return address of the call. After applying this rule, $\omega_k = \gamma\omega'$. In other words, $\gamma''$ will be the topmost stack symbol at the corresponding return point of the call.
So, in order to recover $\phi'$ in $A_k$, we proceed as follows: At the call $\langle p_i, \gamma \rangle \xrightarrow{\text{call}} \langle p_{i+1}, \gamma' \gamma'' \rangle$, we encode $A_i$ into $\gamma''$ by the rule $(\alpha_1)$ stating that $\langle p_i, A_i, l_i, \gamma \rangle \rightarrow \langle p_{i+1}, A_{i+1}, l_{i+1}, \gamma' \gamma'', A_i, l_i \rangle \in \Delta_i$. This allows us to record $X^c\phi'$ in the corresponding return point of the stack. After that, $\langle \gamma'', A_i, l_i \rangle$ will be the topmost stack symbol at the corresponding return-point of this call. At the return-point, the condition $(\beta_5)$ in $(\alpha_{2.2})$ stating that $\text{AbsNext}(A_i, A_k)$ and the fact that $\phi = X^c\phi' \in A_i$ imply that $\phi' \in A_k$.

- $\langle p_i, \omega_i \rangle \Rightarrow p \langle p_{i+1}, \omega_{i+1} \rangle$ corresponds to a simple statement. Then, the abstract successor of $\langle p_i, \omega_i \rangle$ is $\langle p_{i+1}, \omega_{i+1} \rangle$ (see Figure 2.3). $(\pi, i) \models X^c\phi' \iff (\pi, i+1) \models \phi'$. Thus, we must have $\phi' \in A_{i+1}$. This is ensured by condition $(\beta_4)$ in $(\alpha_3)$ stating that $\text{AbsNext}(A_i, A_{i+1}) = \text{true}$.

2. The other delicate case is when $\phi = X^c\phi' \in A_i$. This means that $(\pi, i) \models X^c\phi'$. This case is handled by the conditions $(\beta_4)$ in $(\alpha_1)$, $(\beta_6)$ in $(\alpha_2)$ and $(\beta_5)$ in $(\alpha_3)$. Let us consider the example in Figure 2.3 to illustrate this case. In this figure, the caller-successor of $\langle p_i, \omega_i \rangle$ is $\langle p_{i-1}, \omega_{i-1} \rangle$. Thus, $(\pi, i) \models X^c\phi' \iff (\pi, i-1) \models \phi'$. Then we need to ensure that $\phi' \in A_{i-1}$. This is done as follows:

- $\langle p_{i-1}, \omega_{i-1} \rangle \Rightarrow p \langle p_i, \omega_i \rangle$ corresponds to a simple statement, so we require $\text{NexCallerForms}(A_{i-1}) = \text{NexCallerForms}(A_i)$ (by the condition $(\beta_5)$ in $(\alpha_3)$). This implies $X^c\phi' \in A_{i-1}$. Similarly, we have $X^c\phi' \in A_{i-k_2}$.

- $\langle p_{i-k_1}, \omega_{i-k_1} \rangle$ and $\langle p_{i-k_2}, \omega_{i-k_2} \rangle$ is a pair of call and return-point. Then, by applying the condition $(\beta_6)$ in $(\alpha_2)$, we have $\text{NexCallerForms}(A_{i-k_1}) = \text{NexCallerForms}(A_{i-k_2})$. This implies $X^c\phi' \in A_{i-k_1}$.

- The transitions from $\langle p_{i-(t-1)}, \omega_{i-(t-1)} \rangle$ to $\langle p_{i-k_1}, \omega_{i-k_1} \rangle$ correspond to simple statements. By the condition $(\beta_5)$ in $(\alpha_3)$, we obtain $X^c\phi' \in A_{i-(t-1)}$.

- $\langle p_{i-t}, \omega_{i-t} \rangle \Rightarrow p \langle p_{i-(t-1)}, \omega_{i-(t-1)} \rangle$ corresponds to a call statement, so we require $\text{callerNext}(A_{i-(t-1)}, A_{i-t})$ (by the condition $(\beta_4)$ in $(\alpha_3)$) which means that if $X^c\phi' \in A_{i-(t-1)}$ then $\phi' \in A_{i-t}$.

The labels. Now, let us explain how the label $l$ is used in the transition rules to ensure the correctness of the formulas. Note that our explanation above makes implicitly the assumption that along the run $\pi$, every call to a procedure $\text{proc}$ will eventually reach its corresponding return point, i.e., the
2.3. CARET Model-Checking for Pushdown Systems

Figure 2.3: Case of $X^c\phi' \in A_i$

Figure 2.4: $\langle p_i, \omega_i \rangle$ never reach its corresponding return-point

run $\pi$ will finally exit proc, then, we can encode formulas at the call and validate them at its corresponding return-point. However, it might be the case that at a certain point in the procedure proc, there will be a loop, and $\pi$ never exits proc. To solve this problem, we annotate the control states by the label $l \in \{\text{exit}, \text{unexit}\}$ to determine whether $\pi$ can complete the execution of the procedure $\mathcal{P}(\langle p, \omega \rangle)$. In the following, we explain three cases corresponding to three kinds of statements:

- Let us consider Figure 2.2. $\langle p_i, \omega_i \rangle \Rightarrow p (\langle p_{i+1}, \omega_{i+1} \rangle)$ corresponds to a call statement. Note that $\mathcal{P}(\langle p_{i+1}, \omega_{i+1} \rangle) = \text{proc}$ in this case. There are two possibilities. If proc terminates, then the call at $\langle p_i, \omega_i \rangle$ will reach its corresponding return-point. In this case, $\langle p_{i+1}, \omega_{i+1} \rangle$ is labelled by exit. If proc never terminates, then the call at $\langle p_i, \omega_i \rangle$ will never reach its corresponding return-point. In this case, $\langle p_{i+1}, \omega_{i+1} \rangle$ is labelled by unexit. If $\langle p_{i+1}, \omega_{i+1} \rangle$ is labelled by exit, then $\langle p_i, \omega_i \rangle$ can be labelled by exit or unexit. However, if $\langle p_{i+1}, \omega_{i+1} \rangle$ is labelled by unexit, then $\langle p_i, \omega_i \rangle$ must be labelled by unexit. This is ensured by the condition ($l' = \text{unexit}$ implies $l = \text{unexit}$) in the rule ($\alpha_1$). In addition, if $\langle p_{i+1}, \omega_{i+1} \rangle$ is labelled by unexit, then $\langle p_i, \omega_i \rangle$ never reaches its corresponding return-point. Thus, $\langle p_i, \omega_i \rangle$ does not satisfy any formula in the form $X^a\phi$. This is ensured by the condition ($l' = \text{unexit}$ implies $\text{NexAbsForms}(A) = \emptyset$) in the rule ($\alpha_1$).
• Again, let us consider Figure 2.2. \( \langle p_{k-1}, \omega_{k-1} \rangle \Rightarrow_p \langle p_k, \omega_k \rangle \) corresponds to a ret statement. At \( \langle p_{k-1}, \omega_{k-1} \rangle \), we are sure that \( \text{proc} \) will terminate. In this case, \( \langle p_{k-1}, \omega_{k-1} \rangle \) must be always labelled by \( \text{exit} \) and \( \langle p_k, \omega_k \rangle \) can be labelled by \( \text{exit} \) or \( \text{unexit} \). This is ensured by the rule \((\alpha_{2,1})\). Also, the abstract-successor of \( \langle p_{k-1}, \omega_{k-1} \rangle \) is \( \bot \), then, \( \langle p_{k-1}, \omega_{k-1} \rangle \) does not satisfy any formula in the form \( X^a \phi \). This is ensured by the condition \((\text{NexAbsForms}(A) = \emptyset)\) in the rule \((\alpha_{2,1})\).

• Finally, let us consider Figure 2.3. \( \langle p_i, \omega_i \rangle \Rightarrow_p \langle p_{i+1}, \omega_{i+1} \rangle \) corresponds to a simple statement. Then, \( \langle p_i, \omega_i \rangle \) and \( \langle p_{i+1}, \omega_{i+1} \rangle \) are in the same procedure \( \text{proc} \). Thus, the labels assigned to \( \langle p_i, \omega_i \rangle \) and \( \langle p_{i+1}, \omega_{i+1} \rangle \) should be the same. This is ensured by the transition rule \((\alpha_3)\).

The Büchi accepting condition. The generalized Büchi accepting condition \( F \) of \( \mathcal{B}P_{\psi} \) consists of three families of accepting conditions \( F_1, F_2 \) and \( F_3 \). The first set \( F_1 \) guarantees that an accepting run should go infinitely often through the label \( \text{unexit} \). The sets \( F_2 \) and \( F_3 \) ensure the liveness requirements of until-formulas on the infinite global path and the infinite abstract path:

• With regards to the second set of sets \( F_2 \), each set of \( F_2 \) ensures that the liveness requirement \( \phi_2 \) in \( \phi_1 U^g \phi_2 \) is eventually satisfied in \( \mathcal{P} \). Note that if \( \phi_1 U^g \phi_2 \in A_i \), then, \( (\pi, i) \models \phi_1 U^g \phi_2 \) iff \( (\pi, i) \models \phi_2 \) or \( ((\pi, i) \models \phi_1 \) and \( (\pi, i) \models X^g(\phi_1 U^g \phi_2) \). Because \( \phi_2 \) should hold eventually, to avoid the case where the run of \( \mathcal{B}P_{\psi} \) always carries \( \phi_1 \) and \( X^g(\phi_1 U^g \phi_2) \) and never reaches \( \phi_2 \), we set \( P \times F_{\phi_1 U^g \phi_2} = P \times \{ A \in \text{Atoms}(\psi) \mid \phi_1 U^g \phi_2 \in A \} \times \text{Label} \) as a set of Büchi generalized accepting condition. By this setting, the accepting run of \( \mathcal{B}P_{\psi} \) will infinitely often visit some control locations in \( P \times \{ A \in \text{Atoms}(\psi) \mid \phi_1 U^g \phi_2 \in A \} \times \text{Label} \) which ensures that \( \phi_2 \) will eventually hold.

• The idea behind the set \( F_3 \) is similar to the set \( F_2 \) except that we only need to ensure the liveness requirement for abstract-until formulas \( \phi_1 U^a \phi_2 \) on the infinite abstract path. Thus, the label in \( F_3 \) is always \( \text{unexit} \).

Finite caller and abstract paths. The liveness requirements of caller-until formulas on finite caller paths and abstract-until formulas on finite abstract paths are ensured by conditions in transition rules:

• With respect to caller-until formulas, note that caller paths are always finite. The liveness requirements of caller-until formulas are ensured by the condition \( \text{NexCallerForms}(A) = \emptyset \). This requirement guarantees
the liveness requirement $\phi_2$ in $\phi_1 U^c \phi_2$ eventually happens. Look at Figure 2.2 for an illustration. Assume that $\phi_1 U^c \phi_2 \in A_i$, then, $(\pi, i) \models \phi_1 U^c \phi_2$ iff $(\pi, i) \models \phi_2$ or $((\pi, i) \models \phi_1$ and $(\pi, i) \models X^c(\phi_1 U^c \phi_2))$. In other words, $\phi_1 U^c \phi_2 \in A_i$ iff $\phi_2 \in A_i$ or $(\phi_1 \in A_i$ and $X^c(\phi_1 U^c \phi_2) \in A_i)$. Since $\phi_2$ should eventually hold, $\phi_2$ should hold at $\pi(i)$ because $next^c_i = \bot$. To ensure this, we require that $NexCallerForms(A_0) = \emptyset$ which guarantees that $NexCallerForms(A_i) = \emptyset$. $NexCallerForms(A_i) = \emptyset$ ensures that the case $\phi_2 \in A_i$ occurs instead of $(\phi_1 \in A_i$ and $X^c(\phi_1 U^c \phi_2) \in A_i)$; which means that $(\pi, i) \models \phi_2$ and $\phi_2$ eventually holds. Notice that this requirement does not make any restriction for our algorithm: given a CARET formula $\psi$, we can always obtain at least one atom $A$ containing $\psi$ such that $NexCallerForms(A) = \emptyset$.

- The liveness requirements of abstract-until formulas on finite abstract paths $\langle p_{z_0}, \omega_{z_0} \rangle \langle p_{z_1}, \omega_{z_1} \rangle \cdots \langle p_{z_m}, \omega_{z_m} \rangle$ where $\langle p_{z_m}, \omega_{z_m} \rangle$ is associated with a tag $t_{z_m} = \text{ret}$ are ensured by the condition $NexAbsForms(A) = \emptyset$ in the transition rule $(\alpha_2)$. This requirement guarantees the liveness requirement $\phi_2$ in $\phi_1 U^a \phi_2$ eventually happens. Look at Figure 2.2 for an illustration. In this figure, for every $i + 1 \leq u \leq k - 1$, the abstract path starting from $\langle p_u, \omega_u \rangle$ is finite. Assume that $\phi_1 U^a \phi_2 \in A_{k-1}$, then, $(\pi, k-1) \models \phi_1 U^a \phi_2$ iff $(\pi, k-1) \models \phi_2$ or $((\pi, k-1) \models \phi_1$ and $(\pi, k-1) \models X^a(\phi_1 U^a \phi_2))$. In other words, $\phi_1 U^a \phi_2 \in A_{k-1}$ iff $\phi_2 \in A_{k-1}$ or $(\phi_1 \in A_{k-1}$ and $X^a(\phi_1 U^a \phi_2) \in A_{k-1})$. Since $\phi_2$ should eventually hold, $\phi_2$ should hold at $\pi(k-1)$ because $next^a_{k-1} = \bot$. To ensure this, we require that $NexAbsForms(A_{k-1}) = \emptyset$ at return statements by the condition $(\beta_4)$ in the transition rule $(\alpha_2)$. $NexAbsForms(A_{k-1}) = \emptyset$ ensures that the case $\phi_2 \in A_{k-1}$ occurs instead of $(\phi_1 \in A_{k-1}$ and $X^a(\phi_1 U^a \phi_2) \in A_{k-1})$; which means that $(\pi, k-1) \models \phi_2$ and $\phi_2$ eventually holds.

- The liveness requirements of abstract-until formulas on finite abstract paths $\langle p_{z_0}, \omega_{z_0} \rangle \langle p_{z_1}, \omega_{z_1} \rangle \cdots \langle p_{z_m}, \omega_{z_m} \rangle$ where $\langle p_{z_m}, \omega_{z_m} \rangle$ is associated with a tag $t_{z_m} = \text{call}$ but this call never reaches its corresponding return-point are ensured by the condition ($l' = \text{unexit}$ implies $NexAbsForms(A) = \emptyset$) in the transition rule $(\alpha_1)$. This requirement guarantees the liveness requirement $\phi_2$ in $\phi_1 U^a \phi_2$ eventually happens. Look at Figure 2.4 for an illustration. In this figure, for every $0 \leq u \leq i$, the abstract path starting from $\langle p_u, \omega_u \rangle$ is finite. Assume that $\phi_1 U^a \phi_2 \in A_i$, then, $(\pi, i) \models \phi_1 U^a \phi_2$ iff $(\pi, i) \models \phi_2$ or $((\pi, i) \models \phi_1$ and $(\pi, i) \models X^a(\phi_1 U^a \phi_2))$. In other words, $\phi_1 U^a \phi_2 \in A_i$ iff $\phi_2 \in A_i$ or $(\phi_1 \in A_i$ and $X^a(\phi_1 U^a \phi_2) \in A_i)$. Since $\phi_2$ should eventually hold, $\phi_2$ should hold at $\pi(i)$ because $next^a_i = \bot$. To ensure this, we require that $NexAbsForms(A_i) = \emptyset$ by the condition $(\beta_5)$ in the transition rule $(\alpha_1)$. $NexAbsForms(A_i) = \emptyset$ ensures that the
The projection of a CARET formula removing the atoms of the BP System.

Thus, we can show that:

**Theorem 2.** Given a PDS $\mathcal{P} = (P, \Gamma, \Delta)$, a labeling function $\lambda : P \rightarrow 2^{AP}$, and a CARET formula $\psi$, we can construct a Generalized Büchi Pushdown System $\mathcal{BP}_\psi = (P', \Gamma', \Delta', F)$ such that for every configuration $\langle p, \omega \rangle \in \mathcal{P} \times \Gamma^*$, $\langle p, \omega \rangle \models \psi$ iff there exists an atom $A \in \text{Atoms}(\psi)$ where $\psi \in A$ and $\text{NexCallerForms}(A) = \emptyset$ s.t. $\langle \langle p, A, \text{unexit} \rangle, \omega \rangle \in \mathcal{L}(\mathcal{BP}_\psi)$.

**Formal proof.** To prove formally this result, we need the following definitions:

**Definition 4.** Let $\pi'$ be a run of $\mathcal{BP}_\psi$. Let $\pi'(i) = \langle p'_i, \gamma'_i \gamma'_1 \ldots \gamma'_n \rangle$ where $p'_i$ is of the form $\langle p, A_i, l_i \rangle$, $\gamma'_i$ is of the form $\gamma_i$ or $\langle \gamma_i, A_i, l_i \rangle$, be a configuration of $\pi'$. The projection of $\pi'(i)$ on $\mathcal{P}$; $\text{pr}(\pi'(i)) := \langle p_i, \gamma_0 \gamma_1 \ldots \gamma_n \rangle$; is obtained by removing the atoms $A_i$ and the labels $l_i$ from the control location and the stack symbols of $\pi'(i)$.

Let $\pi' = \langle \langle p_0, A_0, l_0 \rangle, \omega'_0 \rangle \langle \langle p_1, A_1, l_1 \rangle, \omega'_1 \rangle \ldots$ be a run of $\mathcal{BP}_\psi$. Let $\pi = \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \ldots$ be the run obtained by projecting on $\mathcal{P}$ all the configurations of $\pi'$, then, it is easy to see that for every $i \geq 0$, either $\langle p_i, \omega_i \rangle = \langle p_{i+1}, \omega_{i+1} \rangle$ (in case $\langle \langle p_{i+1}, A_{i+1}, l_{i+1} \rangle, \omega'_{i+1} \rangle$ is obtained from $\langle \langle p_i, A_i, l_i \rangle, \omega'_i \rangle$ using a transition corresponding to the rule $(\alpha_{2,2})$), or $\langle p_i, \omega_i \rangle \Rightarrow_p \langle p_{i+1}, \omega_{i+1} \rangle$ in the other cases.

Then, to obtain from $\pi'$ a run of $\mathcal{P}$, we need to get rid of these duplicated configurations. Thus, we define the projection as follows:

**Definition 5.** Let $\pi' = \langle \langle p_0, A_0, l_0 \rangle, \omega'_0 \rangle \langle \langle p_1, A_1, l_1 \rangle, \omega'_1 \rangle \ldots$ be a run of $\mathcal{BP}_\psi$. The projection of $\pi'$ on $\mathcal{P}$; $\text{pr}(\pi') := \langle p_{z_0}, \omega_{z_0} \rangle \langle p_{z_1}, \omega_{z_1} \rangle \ldots$ where $\langle p_{z_i}, \omega_{z_i} \rangle = \text{pr}(\langle \langle p_{z_i}, A_{z_i}, l_{z_i} \rangle, \omega'_{z_i} \rangle)$; for $i \geq 0$

- $z_0 = 0$
- for $j > 0$, if the transition $\langle \langle p_{z_j}, A_{z_j}, l_{z_j} \rangle, \omega'_{z_j} \rangle \Rightarrow_{\mathcal{BP}_\psi} \langle \langle p_{z_j+1}, A_{z_j+1}, l_{z_j+1} \rangle, \omega'_{z_j+1} \rangle$

  corresponds to a transition of the form $(\alpha_{2,2})$ in $\mathcal{BP}_\psi$, then $z_{j+1} = z_j + 1$.

Otherwise, $z_{j+1} = z_j + 2$.

Then, it is easy to see that:
Lemma 1. Let $\pi' = (\langle p_0, A_0, l_0 \rangle, \omega_0') \langle p_1, A_1, l_1 \rangle, \omega_1' \rangle \ldots$ be a run of $\mathcal{BP}_\psi$, let $pr(\pi') = \langle p_{20}, \omega_{20} \rangle \langle p_{21}, \omega_{21} \rangle \ldots$ be the projection of $\pi'$ on $\mathcal{P}$, then, $pr(\pi')$ is a run in $\mathcal{P}$.

Proof. There are different cases depending on the nature of the transition $\langle p_{zi}, \omega_{zi} \rangle \Rightarrow p \langle p_{zi+1}, \omega_{zi+1} \rangle$

- If $\langle p_{zi}, \omega_{zi} \rangle \Rightarrow p \langle p_{zi+1}, \omega_{zi+1} \rangle$ corresponds to a call statement. The property is ensured by $(\beta_3)$ in $(\alpha_1)$ stating that $\text{GlNext}(A_{zi}, A_{zi+1}) = \text{true}$

- If $\langle p_{zi}, \omega_{zi} \rangle \Rightarrow p \langle p_{zi+1}, \omega_{zi+1} \rangle$ corresponds to a return statement. The property is ensured by $(\beta_3)$ in $(\alpha_2)$ stating that $\text{GlNext}(A_{zi}, A_{zi+1}) = \text{true}$

- If $\langle p_{zi}, \omega_{zi} \rangle \Rightarrow p \langle p_{zi+1}, \omega_{zi+1} \rangle$ corresponds to a simple statement. The property is ensured by $(\beta_3)$ in $(\alpha_3)$ stating that $\text{GlNext}(A_{zi}, A_{zi+1}) = \text{true}$

Lemma 3. Let $\pi' = (\langle p_0, A_0, l_0 \rangle, \omega_0') \langle p_1, A_1, l_1 \rangle, \omega_1' \rangle \ldots$ be an accepting run of $\mathcal{BP}_\psi$. Let $\pi = \langle p_{20}, \omega_{20} \rangle \langle p_{21}, \omega_{21} \rangle \ldots$ be the projection of $\pi'$ on $\mathcal{P}$. Let $i \geq 0$. Let $\pi(z_k)$ be the abstract successor of $\pi(z_i)$. We prove that if $\pi(z_k) \neq \bot$ then $\text{AbsNext}(A_{zi}, A_{zi+1}) = \text{true}$.

Proof. There are different cases depending on the nature of the transition $\langle p_{zi}, \omega_{zi} \rangle \Rightarrow p \langle p_{zi+1}, \omega_{zi+1} \rangle$. Since $\pi(z_k) \neq \bot$, this transition does not correspond to a return statement.

- If $\langle p_{zi}, \omega_{zi} \rangle \Rightarrow p \langle p_{zi+1}, \omega_{zi+1} \rangle$ corresponds to a call statement. As presented before, let us consider Figure 2.2 to explain this case. Let $\langle p_{zi}, \gamma \rangle \xrightarrow{\text{call}} \langle p_{zi+1}, \gamma' \gamma'' \rangle$ be the rule associated with the transition $\langle p_{zi}, \omega_{zi} \rangle \Rightarrow p \langle p_{zi+1}, \omega_{zi+1} \rangle$, then we have $\omega_{zi} = \gamma \omega'$ and $\omega_{zi+1} = \gamma' \gamma'' \omega'$. Let $\langle p_{zk-1}, \omega_{zk-1} \rangle \Rightarrow p \langle p_{zk}, \omega_{zk} \rangle$ be the transition that corresponds to the $\text{ret}$ statement of this call. Let then $\langle p_{zk-1}, \beta \rangle \xrightarrow{\text{ret}} \langle p_{zk}, \varepsilon \rangle \in \Delta$ be the corresponding return rule. Then, we have necessarily $\omega_{zk-1} = \beta \gamma'' \omega'$, since as explained in Section 2.2.1, $\gamma''$ is the return address of the call. After applying this rule, $\omega_{zk} = \gamma'' \omega'$. In other words, $\gamma''$ will be the topmost stack symbol at the corresponding return point of the call.
Chapter 2. CARET Model Checking For Pushdown Systems

Lemma 4. \( i > 0 : \) Let \( \pi \) be an accepting run of \( \mathcal{BP}_\psi \). Let \( \pi = \langle p_0, A_0, l_0, \omega_0 \rangle \langle p_1, A_1, l_1, \omega_1 \rangle \ldots \) be an accepting run of \( \mathcal{BP}_\psi \). Let \( \pi = \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \ldots \) be a projection of \( \pi \) on \( P \). Let \( i \geq 0 \). Let \( \pi(z_{i-1}) \) be the caller successor of \( \pi(z_i) \) \( (t \geq 1) \). We prove that for every \( i > 0 : \) \( \text{CallerNext}(A_{z_i}, A_{z_{i-1}}) = true \).

Proof. There are two cases:

1. If there is no matched pair of calls and returns between \( \pi(z_{i-1}) \) and \( \pi(z_i) \). Let us consider Figure 2.5 to prove this case. We apply \( (\beta_5) \) in \( (\alpha_{2.2}) \) stating that \( \text{AbsNext}(A_{z_i}, A_{z_{i-1}}) = true \). The property holds for this case.

* If \( \langle p_{z_i}, \omega_{z_i} \rangle \Rightarrow_P \langle p_{z_{i+1}}, \omega_{z_{i+1}} \rangle \) corresponds to a simple statement. As presented before, the abstract successor of \( \langle p_{z_i}, \omega_{z_i} \rangle \) is \( \langle p_{z_{i+1}}, \omega_{z_{i+1}} \rangle \) (see Figure 2.5). \( \langle \pi, z_i \rangle \models X^a \phi' \) iff \( \langle \pi, z_{i+1} \rangle \models \phi' \). Thus, we must have \( \phi' \in A_{z_{i+1}} \). This is ensured by condition \( (\beta_4) \) in \( (\alpha_3) \) stating that \( \text{AbsNext}(A_{z_i}, A_{z_{i+1}}) = true \). The property holds for this case.

\( \blacksquare \)

24 Chapter 2. CARET Model Checking For Pushdown Systems

So, in order to recover \( \phi' \) in \( A_{z_i} \), we proceed as follows: At the call \( (p_{z_i}, \gamma) \xrightarrow{\text{call}} (p_{z_{i+1}}, \gamma') \), we encode \( A_{z_i} \) into \( \gamma'' \) by the rule \( (\alpha_1) \) stating that \( \langle (p_{z_i}, A_{z_i}, L_{z_i}), \gamma \rangle \rightarrow \langle (p_{z_{i+1}}, A_{z_{i+1}}, L_{z_{i+1}}), \gamma' \rangle \in \Delta' \). After that, \( \langle \gamma'', A_{z_i}, L_{z_i} \rangle \) will be the topmost stack symbol at the corresponding return-point of this call. At this point, we apply \( (\beta_5) \) in \( (\alpha_{2.2}) \) stating that \( \text{AbsNext}(A_{z_i}, A_{z_{i-1}}) = true \). The property holds for this case.

- If \( \langle p_{z_i}, \omega_{z_i} \rangle \Rightarrow_P \langle p_{z_{i+1}}, \omega_{z_{i+1}} \rangle \) corresponds to a simple statement. As presented before, the abstract successor of \( \langle p_{z_i}, \omega_{z_i} \rangle \) is \( \langle p_{z_{i+1}}, \omega_{z_{i+1}} \rangle \) (see Figure 2.5). \( \langle \pi, z_i \rangle \models X^a \phi' \) iff \( \langle \pi, z_{i+1} \rangle \models \phi' \). Thus, we must have \( \phi' \in A_{z_{i+1}} \). This is ensured by condition \( (\beta_4) \) in \( (\alpha_3) \) stating that \( \text{AbsNext}(A_{z_i}, A_{z_{i+1}}) = true \). The property holds for this case.

\( \blacksquare \)

Lemma 4. Let \( \pi' = \langle \langle p_0, A_0, l_0, \omega_0 \rangle, \langle p_1, A_1, l_1, \omega_1 \rangle, \ldots \rangle \) be an accepting run of \( \mathcal{BP}_\psi \). Let \( \pi = \langle p_0, \omega_0 \rangle, \langle p_1, \omega_1 \rangle, \ldots \rangle \) be the projection of \( \pi' \) on \( P \). Let \( i \geq 0 \). Let \( \pi(z_{i-1}) \) be the caller successor of \( \pi(z_i) \) \( (t \geq 1) \). We prove that for every \( i > 0 : \) \( \text{CallerNext}(A_{z_i}, A_{z_{i-1}}) = true \).

Proof. There are two cases:

1. If there is no matched pair of calls and returns between \( \pi(z_{i-1}) \) and \( \pi(z_i) \). Let us consider Figure 2.5 to prove this case. We need to prove that \( \text{CallerNext}(A_{z_i}, A_{z_{i-1}}) = true \). We apply a second induction on \( t \) to show that \( \text{CallerNext}(A_{z_i}, A_{z_{i-1}}) = true \).

   - Base case: \( (t = 1) \)
     - \( (t = 1) \), then, \( \langle p_{z_{i-1}}, \omega_{z_{i-1}} \rangle \) and \( \langle p_{z_i}, \omega_{z_i} \rangle \) are two consequent configurations on \( P \). Since \( \pi(z_{i-1}) \) is the caller successor of \( \pi(z_i) \), then, \( \langle p_{z_{i-1}}, \omega_{z_{i-1}} \rangle \Rightarrow_P \langle p_{z_i}, \omega_{z_i} \rangle \) must correspond to a call statement. As a result, \( \text{CallerNext}(A_{z_i}, A_{z_{i-1}}) = true \) (by the condition \( (\beta_4) \) in \( (\alpha_1) \)). The property holds.

   - Induction step: \( (t > 1) \)
     - Since there is no matched pair of calls and returns between \( \pi(z_{i-1}) \) and \( \pi(z_i) \), then, \( \langle p_{z_{i-1}}, \omega_{z_{i-1}} \rangle \Rightarrow_P \langle p_{z_i}, \omega_{z_i} \rangle \) must correspond to a simple statement, so we have \( \text{NexCallerForms}(A_{z_{i-1}}) = \text{NexCallerForms}(A_{z_i}) \) (by the condition \( (\beta_5) \) in \( (\alpha_3) \))
     - \( \text{CallerNext}(A_{z_{i-1}}, A_{z_{i-1}}) = true \) (by the second induction hypothesis)
2.3. CARET Model-Checking for Pushdown Systems

2. If there are matched pairs of calls and returns between $\pi(z_{i-t})$ and $\pi(z_i)$ (see Figure 2.6). Let $\pi(z_{i-k_1})$ and $\pi(z_{i-k_2})$ ($1 \leq k_1 \leq (t-1)$) be a matched pair of call and ret. Since $\pi'$ is an accepting run of $\mathcal{BP}_\psi$, we obtain $\text{NexCallerForms}(A_{z_i-k_1}) = \text{NexCallerForms}(A_{z_i-k_2})$ (by the condition ($\beta_6$) in ($\alpha_2.2$)).

Secondly, $\text{NexCallerForms}(A_{z_i-k_2}) = \text{NexCallerForms}(A_{z_i})$ (since $\langle p_{z_i-k_2}, \omega_{z_i-k_2} \rangle \ldots \langle p_{z_i}, \omega_{z_i} \rangle$ is a sequence of simple statements)

Thirdly, $\text{NexCallerForms}(A_{z_i-(t-1)}) = \text{NexCallerForms}(A_{z_i-k_1})$ (since $\langle p_{z_i-(t-1)}, \omega_{z_i-(t-1)} \rangle \ldots \langle p_{z_i-k_1}, \omega_{z_i-k_1} \rangle$ is a sequence of simple statements)

Thus, we have:

\[
\text{NexCallerForms}(A_{z_i-(t-1)}) = \text{NexCallerForms}(A_{z_i}) \quad (2.1)
\]

Also, $\langle p_{z_i-t}, \omega_{z_i-t} \rangle \Rightarrow p \langle p_{z_i-(t-1)}, \omega_{z_i-(t-1)} \rangle$ corresponds to a call statement, then, by the condition ($\beta_4$) in ($\alpha_1$), we have

\[
\text{CallerNext}(A_{z_i-(t-1)}, A_{z_i-t}) = true \quad (2.2)
\]

From (2.1) and (2.2), we obtain $\text{CallerNext}(A_{z_i}, A_{z_i-t})$. The property holds for this case.

\[\square\]

Proof of Theorem 2. Now, we are ready to prove Theorem 2. We prove these 2 directions:
Assume that there exists an atom \( A_0 \in Atoms(\psi) \) where \( \psi \in A_0 \) and \( \text{NexCallerForms}(A_0) = \emptyset \) s.t \( \langle [p, A_0, \text{unexit}], \omega \rangle \in L(\mathcal{BP}_\psi) \). In other words, there exists an accepting run \( \pi' = \langle [p_0, A_0, l_0], \omega_0' \rangle \langle [p_1, A_1, l_1], \omega_1' \rangle \ldots \) of \( \mathcal{BP}_\psi \) where \( l_0 = \text{unexit}, \psi \in A_0 \) and \( \text{NexCallerForms}(A_0) = \emptyset \). Let \( \pi = \langle [p_0, \omega_0] \rangle \langle [p_1, \omega_1] \rangle \ldots \) be the projection of \( \pi' \) on \( \mathcal{P} \), note that \( \pi \) is a run of \( \mathcal{P} \) (by Lemma 1). We need to prove that \( \pi \models \psi \).

**Proof.** It is sufficient to show that for every \( i \geq 0, \phi \in A_{zi} \) iff \( (\pi, z_i) \models \phi \). This is proved by induction on the structure of \( \phi \).

- **Base case:**
  - \( \phi = e \ (e \in AP) \):
    \( e \in A_{zi} \) iff \( e \in \lambda(p_{zi}) \) (by the condition \( A_{zi} \cap AP = \lambda(p_{zi}) \) in \( \alpha_1 \), \( \alpha_2 \) and \( \alpha_3 \)) iff \( (\pi, z_i) \models \phi \) (by the semantics of CARET). The property holds for this case.
  - \( \phi = d \in A_{zi} \) (\( d \in \{\text{call, ret, int}\} \))

    * \( d = \text{call} \)
      \( \text{call} \in A_{zi} \) iff \( \langle p_{zi}, \omega_{zi} \rangle \not\rightarrow \mathcal{P} \langle p_{zi+1}, \omega_{zi+1} \rangle \) corresponds to a call statement (by the conditions \( \beta_0 \) in \( \alpha_1 \), \( \alpha_2 \) and \( \alpha_3 \)) iff \( \langle p_{zi}, \omega_{zi} \rangle \) is tagged with a \text{call} (by the definition in Section 2.2.2) iff \( (\pi, z_i) \models \text{call} \) (by the semantics of CARET). The property holds for this case.

    * \( d = \text{ret} \) and \( d = \text{int} \). Similarly, we obtain \( d \in A_{zi} \) iff \( (\pi, z_i) \models d \). The property holds for these cases.

- **Induction Step:**
2.3. CARET Model-Checking for Pushdown Systems

- \( \phi = \neg \phi' \)
  \( \phi = \neg \phi' \in A_{z_i} \) iff \( \phi' \notin A_{z_i} \) (by the definition of Atom) iff \( (\pi, z_i) \not\models \phi' \) (by induction hypothesis) iff \( (\pi, z_i) \models \neg \phi' \) (by the semantics of CARET) iff \( (\pi, z_i) \models \phi \). The property holds.

- \( \phi = \phi_1 \lor \phi_2 \)
  \( \phi = \phi_1 \lor \phi_2 \in A_{z_i} \) iff \( \phi_1 \in A_{z_i} \) or \( \phi_2 \in A_{z_i} \) (by the definition of atom) iff \( (\pi, z_i) \models \phi_1 \) or \( (\pi, z_i) \models \phi_2 \) (by induction hypothesis) iff \( (\pi, z_i) \models \phi_1 \lor \phi_2 \) (by the semantics of CARET). In other words, \( (\pi, z_i) \models \phi \). The property holds.

- \( \phi = X^g \phi' \)
  By Lemma 2, we have \( GlNext(A_{z_i}, A_{z_{i+1}}) = true \). Note that \( \pi(z_{i+1}) \) is the global-successor of \( \pi(z_i) \).
  * \( \phi = X^g \phi' \in A_{z_i} \) iff \( \phi' \in A_{z_{i+1}} \) (by \( GlNext(A_{z_i}, A_{z_{i+1}}) \)) iff \( (\pi, z_{i+1}) \models \phi' \) (by induction hypothesis) iff \( (\pi, z_i) \models X^g \phi' \) (by the fact that \( \pi(z_{i+1}) = next^g_{\pi(z_i)} \)). The property holds for this case.

- \( \phi = X^a \phi' \)
  Let \( \pi(z_u) \) be the abstract-successor of \( \pi(z_i) \). Note that \( u > i \). By Lemma 3, we have \( AbsNext(A_{z_i}, A_{z_u}) = true \).
  * \( \phi = X^a \phi' \in A_{z_i} \) iff \( \phi' \in A_{z_u} \) (by \( AbsNext(A_{z_i}, A_{z_u}) \)) iff \( (\pi, z_u) \models \phi' \) (by induction hypothesis) iff \( (\pi, z_i) \models X^a \phi' \) (by the fact that \( \pi(z_u) = next^a_{\pi(z_i)} \)). The property holds for this case.

- \( \phi = X^c \phi' \)
  Let \( \pi(z_u) \) be the caller-successor of \( \pi(z_i) \). Note that \( u < i \). By Lemma 4, we have \( CallerNext(A_{z_i}, A_{z_u}) = true \).
  * \( \phi = X^c \phi' \in A_{z_i} \) iff \( \phi' \in A_{z_u} \) (by \( CallerNext(A_{z_i}, A_{z_u}) \)) iff \( (\pi, z_u) \models \phi' \) (by induction hypothesis) iff \( (\pi, z_i) \models X^c \phi' \) (by the fact that \( \pi(z_u) = next^c_{\pi(z_i)} \)). The property holds for this case.

- \( \phi = \phi_1 U^g \phi_2 \)
  a) Firstly, let \( \phi_1 U^g \phi_2 \in A_{z_i} \), we prove that \( (\pi, z_i) \models \phi_1 U^g \phi_2 \).
  Note that \( \pi(z_{i+1}) \) is the global-successor of \( \pi(z_i) \). By Lemma 2, we have \( GlNext(A_{z_i}, A_{z_{i+1}}) = true \). Since \( (\pi', z_i) \) is also an accepting run of \( B\mathcal{P}_\phi \), there must exist a \( k \geq z_i \) such that the control location of \( \pi'(k) \) belongs to \( F^x \) where \( F^x \) is an element (a set) in the Büchi
accepting condition set of $\mathcal{BP}_\psi$ corresponding to the formula $\phi_1 U^g \phi_2$. Choose the least such $k$. We apply a second induction on $(k - z_i)$ to show that $(\pi, z_i) \models \phi$.

* Base case: $(k - z_i = 0)$

\[ k - z_i = 0 \implies \langle p_{z_i}, A_{z_i} \rangle \in F^x. \]
Since $\phi_1 U^g \phi_2 \in A_{z_i}$, the only way for $\langle p_{z_i}, A_{z_i} \rangle \in F^x$ is that $\phi_2$ must also belong to $A_{z_i}$ (by the definition of $F$ of $\mathcal{BP}_\psi$). Consequently, by applying the main induction hypothesis, we obtain $(\pi, z_i) \models \phi_2$. $(\pi, z_i) \models \phi_2$ implies that $(\pi, z_i) \models \phi_1 U^g \phi_2$ (by the semantics of the modality $U^g$). The property holds for this case.

* Induction Step: $(k - z_i > 0)$

\[ \cdot \text{Induction Step: } (k - z_i > 0) \]
\[ \cdot \text{Induction Step: } (k - z_i > 0) \]
\[ \cdot \text{Induction Step: } (k - z_i > 0) \]
\[ \cdot \text{Induction Step: } (k - z_i > 0) \]
\[ \cdot \text{Induction Step: } (k - z_i > 0) \]

b) Conversely, suppose $(\pi, z_i) \models \phi_1 U^g \phi_2$, we must show that $\phi = \phi_1 U^g \phi_2 \in A_{z_i}$.
Based on the semantic of $U^g$, there exists $k \geq i$ such that $(\pi, z_k) \models \phi_2$ and for all $i \leq j < k$: $(\pi, z_j) \models \phi_1$. We apply a second induction on $(k - i)$ to prove that $\phi \in A_{z_i}$.

* Base case: $(k - i = 0)$

\[ k - i = 0 \implies (\pi, z_i) \models \phi_2 \implies \phi_2 \in A_{z_i} \] (by the main induction hypothesis) $\implies \phi_1 U^g \phi_2 \in A_{z_i}$ (by the definition of Atom). The property holds for the base case.

* Induction Step: $(k - i > 0)$

From the semantics of $U^g$, $(\pi, z_i) \models \phi_1 U^g \phi_2$ imply $(\pi, z_i) \models \phi_1$ and $(\pi, l_{i+1}) \models \phi_1 U^g \phi_2$.

\[ \cdot \text{Induction Step: } (k - i > 0) \]
\[ \cdot \text{Induction Step: } (k - i > 0) \]
\[ \cdot \text{Induction Step: } (k - i > 0) \]
\[ \cdot \text{Induction Step: } (k - i > 0) \]
\[ \cdot \text{Induction Step: } (k - i > 0) \]


\[ (\pi, l_{i+1}) \models \phi_1 U^g \phi_2 \implies \phi_1 U^g \phi_2 \in A_{z_i+1} \] (by second induction on \((k - i)\) since \(k - (i + 1) < k - i\) \(\implies X^g(\phi_1 U^g \phi_2) \in A_{z_i} \) (by \(\text{GINext}(A_{z_i}, A_{z_i+1})\))

\[ \phi_1 \in A_{z_i} \text{ and } X^g(\phi_1 U^g \phi_2) \in A_{z_i} \implies \phi_1 U^g \phi_2 \in A_{z_i} \] (by the definition of Atom)

\[ \implies \text{The property holds for this case.} \]

\[- \phi = \phi_1 U^a \phi_2 \]

\[ \text{a) Firstly, let } \phi_1 U^a \phi_2 \in A_{z_i}, \text{ we prove that } (\pi, z_i) \models \phi_1 U^a \phi_2. \]

Let \(\pi_a = \langle p_{j_0}, \omega_{j_0} \rangle \langle p_{j_1}, \omega_{j_1} \rangle \ldots \langle p_{j_k}, \omega_{j_k} \rangle \) be the abstract path starting from \(\pi(z_i)\), then, we get that \(j_0 = z_i, j_{m+1} = \{ t \mid \text{next}^a_{j_m} = \pi(t) \}\) for every \(0 \leq m \leq k - 1\). There are three possibilities:

**Case 1:** \(\pi_a = \langle p_{j_0}, \omega_{j_0} \rangle \langle p_{j_1}, \omega_{j_1} \rangle \ldots \langle p_{j_k}, \omega_{j_k} \rangle \) is finite and \(\langle p_{j_k}, \omega_{j_k} \rangle \) is associated with a tag \(t_{j_k} = \text{ret} \). We apply a second induction on \(k\) to show that \((\pi, z_i) \models \phi\).

* Base case: \((k = 0)\)

\[ k = 0 \implies \text{NextAbsForms}(A_{j_0}) = \emptyset \] (by the condition \((\beta_4)\) in the transition rule \((\alpha_2)\)) which means that there are no formulas of the form \(X^a \psi \in A_{z_i}\) (since \(z_i = j_0\)). In addition, \(\phi_1 U^a \phi_2 \in A_{z_i} \implies \phi_2 \in A_{z_i} \text{ or } (\phi_1 \in A_{z_i} \text{ and } X^a(\phi_1 U^a \phi_2) \in A_{z_i}) \) (by the definition of Atom). So, the only way for \(A_{z_i}\) in this case is that \(\phi_2 \in A_{z_i}, \phi_2 \in A_{z_i} \implies (\pi, z_i) \models \phi_2 \) (by the main induction hypothesis) \(\implies (\pi, z_i) \models \phi_1 U^a \phi_2 \) (by the semantics of the \(U^a\) modality).

* Induction Step: \((k > 0)\)

\[ \phi_1 U^a \phi_2 \in A_{z_i} \implies \phi_2 \in A_{z_i} \text{ or } (\phi_1 \in A_{z_i} \text{ and } X^a(\phi_1 U^a \phi_2) \in A_{z_i}) \] (by the definition of Atom)

\[ \text{Case } \phi_2 \in A_{z_i} \implies (\pi, z_i) \models \phi_2 \] (by the main induction hypothesis) \(\implies (\pi, z_i) \models \phi_1 U^a \phi_2 \) (by the semantics of the \(U^a\) modality). The property holds.

\[ \text{Case } \phi_1 \in A_{z_i} \text{ and } X^a(\phi_1 U^a \phi_2) \in A_{z_i}. \]

\[ X^a(\phi_1 U^a \phi_2) \in A_{z_i} \implies \phi_1 U^a \phi_2 \in A_{j_1} \] (by Lemma 3) \(\implies (\pi, j_1) \models \phi_1 U^a \phi_2 \) (by the second induction hypothesis). Combining with the fact \(\phi_1 \in A_{z_i}\), we obtain \((\pi, z_i) \models \phi_1 U^a \phi_2 \) (by the semantics of \(U^a\) modality). In other words, \((\pi, z_i) \models \phi_1 U^a \phi_2 \). The property holds.

**Case 2:** \(\pi_a = \langle p_{j_0}, \omega_{j_0} \rangle \langle p_{j_1}, \omega_{j_1} \rangle \ldots \langle p_{j_k}, \omega_{j_k} \rangle \) is finite and \(\langle p_{j_k}, \omega_{j_k} \rangle \) is associated with a tag \(t_{j_k} = \text{call} \) where this call never reaches its
matching return-point.

We apply a second induction on $k$ to show that $(\pi, z_i) \models \phi$.

* Base case: $(k = 0)$
  
  $k = 0 \implies \text{NexAbsForms}(A_{j_0}) = \emptyset$ (by the condition $(\beta_5)$ in the transition rule $(\alpha_1)$) which means that there are no formulas of the form $X^a \psi \in A_{z_i}$ (since $z_i = j_0$). In addition, $\phi_1 U^a \phi_2 \in A_{z_i} \implies \phi_2 \in A_{z_i}$ or $(\phi_1 \in A_{z_i}$ and $X^a(\phi_1 U^a \phi_2) \in A_{z_i}$) (by the definition of Atom). So, the only way for $A_{z_i}$ in this case is that $\phi_2 \in A_{z_i}$. $\phi_2 \in A_{z_i}$.

* Induction Step: $(k > 0)$
  
  Case $\phi_2 \in A_{z_i} \implies (\pi, z_i) \models \phi_2$ (by the main induction hypothesis) \implies (\pi, z_i) \models \phi_1 U^a \phi_2$ (by the semantics of the $U^a$ modality). The property holds.

  Case $\phi_1 \in A_{z_i}$ and $X^a(\phi_1 U^a \phi_2) \in A_{z_i}$.
  
  $X^a(\phi_1 U^a \phi_2) \in A_{z_i} \implies \phi_1 U^a \phi_2 \in A_{j_1}$ (by Lemma 3) \implies (\pi, j_1) \models \phi_1 U^a \phi_2$ (by the second induction hypothesis). Combining with the fact $\phi_1 \in A_{z_i}$, we obtain $(\pi, z_i) \models \phi_1 U^a \phi_2$ (by the semantics of $U^a$ modality). In other words, $(\pi, z_i) \models \phi_1 U^a \phi_2$. The property holds.

**Case 3:** $\pi_0 = \langle p_{j_0}, \omega_{j_0} \rangle \langle p_{j_1}, \omega_{j_1} \rangle \ldots$ is infinite.

By Lemma 3, we have $\text{AbsNext}(A_{z_i}, A_{j_1}) = \text{true}$. Since $(\pi', z_i)$ is also an accepting run, there must exist a $k \geq 0$ such that the control location of $\pi'(j_k) \in F^x$ where $F^x$ is an element (a set) in the Büchi accepting condition set of $\mathcal{BP}_\psi$ (by the generalized Büchi accepting condition of $\mathcal{BP}_\psi$). Choose the least such $k$. We apply a second induction on $k$ to show that $(\pi, z_i) \models \phi$.

* Base case: $(k = 0)$
  
  $k = 0 \implies \langle p_{z_i}, A_{z_i} \rangle \subseteq F^x$. Since $\phi_1 U^a \phi_2 \in A_{z_i}$, the only way for $\langle p_{z_i}, A_{z_i} \rangle \subseteq F^x$ is that $\phi_2$ must also belong to $A_{z_i}$. Consequently, by applying the main induction hypothesis, we obtain $(\pi, z_i) \models \phi_2 \implies (\pi, z_i) \models \phi_1 U^a \phi_2$ (by the semantics of the modality $U^a$). The property holds for this case.
2.3. CARET Model-Checking for Pushdown Systems

* Induction Step: \((k > 0)\)
  
  - \(k > 0 \implies \langle p_{z_i}, A_{z_i} \rangle \notin F^x \implies \phi_2 \notin A_{z_i}\) since if \(\phi_2 \in A_{z_i}\), then \(\langle p_{z_i}, A_{z_i} \rangle\) must belong to \(F^x\) (by the definition of the generalized Büchi accepting condition of \(BP_\phi\)).
  
  - \(\phi^a_1 U^a_2 \in A_{z_i}\) and \(\phi_2 \notin A_{z_i}\) \(\implies\) both \((\phi_1, X^a(\phi_1 U^a_2))\) must be in \(A_{z_i}\) (by the definition of Atom)
  
  - \(\phi_1 \in A_{z_i} \implies (\pi, z_i) \models \phi_1\) (by the main induction hypothesis).
  
  - \(X^a(\phi_1 U^a_2) \in A_{z_i} \implies \phi_1 U^a_2 \in A_{j_1}\) (by \(AbsNext(A_{z_i}, A_{j_1})\)) \(\implies\) \((\pi, j_1) \models \phi_1 U^a_2\) (by the second induction hypothesis).
  
  - \((\pi, z_i) \models \phi_1\) and \((\pi, j_1) \models \phi_1 U^a_2\) \(\implies\) \((\pi, z_i) \models \phi_1 U^a_2\) (by the semantics of the modality \(U^a\)). The property holds.

\[\implies\] The property holds for this case.

b) Conversely, suppose \((\pi, z_i) \models \phi_1 U^a_2\), we must show that \(\phi = \phi_1 U^a_2 \in A_{z_i}\). From Lemma 3, we have \(AbsNext(A_{z_i}, A_{j_1}) = true\). Based on the semantic of \(U^a\), there exists \(k \geq 0\) such that \((\pi, j_k) \models \phi_2\) and for all \(0 \leq m < k\): \((\pi, j_m) \models \phi_1\). We apply a second induction on \(k\) to prove that \(\phi \in A_{z_i}\).

* Base case: \((k = 0)\)

\[k = 0 \implies (\pi, j_0) \models \phi_2 \implies \phi_2 \in A_{j_0}\] (by the main induction hypothesis) \(\implies\) \(\phi_1 U^a_2 \phi_2 \in A_{j_0}\) (by the definition of Atom) \(\implies\) \(\phi_1 U^a_2 \phi_2 \in A_{z_i}\) (since \(z_i = j_0\)). The property holds for this case.

* Induction step: \((k > 0)\)

From the semantics of \(U^a\), we have \((\pi, j_0) \models \phi_1\) and \((\pi, j_1) \models \phi_1 U^a_2\).

- \((\pi, j_0) \models \phi_1 \implies \phi_1 \models A_{z_i}\) (by the main induction hypothesis and the fact that \(A_{j_0} = A_{z_i}\)).

- \((\pi, j_1) \models \phi_1 U^a_2 \implies \phi_1 U^a_2 \models A_{j_1}\) (by the second induction hypothesis) \(\implies\) \(X^a(\phi_1 U^a_2) \models A_{z_i}\) (by the fact that \(AbsNext(A_{j_0}, A_{j_1}) = true\) and \(A_{j_0} = A_{z_i}\))

- \(\phi_1 \models A_{z_i}\) and \(X^a(\phi_1 U^a_2) \models A_{z_i}\) \(\implies\) \(\phi_1 U^a_2 \models A_{z_i}\) (by the definition of Atom)

\[\implies\] The property holds for this case.

- \(\phi = \phi_1 U^c_2\).

a) Firstly, let \(\phi_1 U^c_2 \phi_2 \in A_{z_i}\), we prove that \((\pi, z_i) \models \phi_1 U^c_2\).

Let \(j_0, j_1, \ldots, j_k\) be the maximum sequence of positions of \(\pi\) where
\( j_0 = z_i, j_{m+1} = \{ t \mid next^e_{j_m} = \pi(t) \} \) for every \( 0 \leq m \leq k - 1 \). Note that the caller-path is always finite. We apply a second induction on \( k \) to show that \((\pi, z_i) \models \phi\).

* Base case: \((k = 0)\)

- \( k = 0 \implies \text{NextCallerForms}(A_{j_0}) = \emptyset \) which means that there are no formulas of the form \( X^c \psi \in A_{z_i} \) (since \( z_i = j_0 \)).

In addition, \( \phi_1 U^c \phi_2 \in A_{z_i} \implies \phi_2 \in A_{z_i} \) or \((\phi_1 \in A_{z_i} \) and \(X^c(\phi_1 U^c \phi_2) \in A_{z_i}) \) (by the definition of Atom). So, the only way for \( A_{z_i} \) in this case is that \( \phi_2 \in A_{z_i} \). \( \phi_2 \in A_{z_i} \implies (\pi, z_i) \models \phi_2 \) (by the main induction hypothesis) \( \implies (\pi, z_i) \models \phi_1 U^c \phi_2 \) (by the semantics of the \( U^c \) modality).

* Induction Step: \((k > 0)\)

\( \phi_1 U^c \phi_2 \in A_{z_i} \implies \phi_2 \in A_{z_i} \) or \((\phi_1 \in A_{z_i} \), and \(X^c(\phi_1 U^c \phi_2) \in A_{z_i}) \) (by the definition of Atom)

- Case \( \phi_2 \in A_{z_i} \implies (\pi, z_i) \models \phi_2 \) (by the main induction hypothesis) \( \implies (\pi, z_i) \models \phi_1 U^c \phi_2 \) (by the semantics of the \( U^c \) modality). The property holds.

- Case \( \phi_1 \in A_{z_i} \) and \(X^c(\phi_1 U^c \phi_2) \in A_{z_i} \).

\( X^c(\phi_1 U^c \phi_2) \in A_{z_i} \implies \phi_1 U^c \phi_2 \in A_{j_1} \) (by Lemma 4) \( \implies (\pi, j_1) \models \phi_1 U^c \phi_2 \) (by the second induction hypothesis). Combining with the fact \( \phi_1 \in A_{z_i} \), we obtain \((\pi, z_i) \models \phi_1 U^c \phi_2 \) (by the semantics of \( U^c \) modality). In other words, \((\pi, z_i) \models \phi_1 U^c \phi_2 \). The property holds.

b) Conversely, suppose \((\pi, z_i) \models \phi_1 U^c \phi_2 \), we must show that \( \phi = \phi_1 U^c \phi_2 \in A_{z_i} \). From Lemma 4, we have \( \text{CallerNext}(A_{z_i}, A_{j_1}) = \text{true} \).

Based on the semantic of \( U^c \), there exists \( k \geq 0 \) such that \((\pi, j_k) \models \phi_2 \) and for all \( 0 < m < k: (\pi, j_m) \models \phi_1 \). We apply a second induction on \( k \) to prove that \( \phi \in A_{z_i} \).

* Base case: \((k = 0)\)

\( k = 0 \implies (\pi, j_0) \models \phi_2 \implies \phi_2 \in A_{j_0} \) (by the main induction hypothesis) \( \implies \phi_1 U^c \phi_2 \in A_{j_0} \) (by the definition of Atom) \( \implies \phi_1 U^c \phi_2 \in A_{z_i} \) (since \( z_i = j_0 \)). The property holds for this case.

* Induction step: \((k > 0)\)

From the semantics of \( U^c \), we have \((\pi, j_0) \models \phi_1 \) and \((\pi, j_1) \models \phi_1 U^c \phi_2 \).

- \((\pi, j_0) \models \phi_1 \implies \phi_1 \in A_{z_i} \) (by the main induction hypothesis and the fact that \( A_{j_0} = A_{z_i} \)).
2.3. CARET Model-Checking for Pushdown Systems

\[
\cdot (\pi, j_1) \models \phi_1 U^c \phi_2 \implies \phi_1 U^c \phi_2 \in A_{j_1} \quad \text{(by the second induction hypothesis)} \implies X^c(\phi_1 U^c \phi_2) \in A_{z_i} \quad \text{(by the fact that \(\text{CallerNext}(A_{j_0}, A_{j_1}) = \text{true}\) and \(A_{j_0} = A_{z_i}\))}
\]

\[
\cdot \phi_1 \in A_{z_i} \text{ and } X^c(\phi_1 U^c \phi_2) \in A_{z_i} \implies \phi_1 U^c \phi_2 \in A_{z_i} \quad \text{(by the definition of Atom)}
\]

\[
\implies \text{The property holds for this case.}
\]

\[
(\implies) \quad \text{Assume that there exists an execution}
\]

\[
\pi = \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \ldots \langle p_n, \omega_n \rangle \ldots
\]

of \(P\) such that \(\pi \models \psi\), we have to show that there exists an atom \(A_0\) where \(\psi \in A_0\) and \(\text{NexCallerForms}(A_0) = \emptyset\) such that \(\langle \langle p_0, A_0, \text{unexit} \rangle, \omega_0 \rangle \in \mathcal{L}(BP_\psi)\). In other words, we have to show an accepting run of \(BP_\psi\) starting from \(\langle \langle p_0, A_0, \text{unexit} \rangle, \omega_0 \rangle\) where \(\psi \in A_0\) and \(\text{NexCallerForms}(A_0) = \emptyset\). In the following, we show how we can compute such an accepting run of \(BP_\psi\).

**Proof.**

For every \(i \geq 0\), let \(A_i = \{ \phi \in Cl(\psi) \mid (\pi, i) \models \phi \} \cup \{ t_i \}\) where \(t_i\) is the tag associated with the configuration \(\langle p_i, \omega_i \rangle\). Firstly, we prove that each \(A_i\) is an atom:

- For every \(\phi \in A_i\) (\(\phi \in Cl(\psi)\)), we need to show that \(\phi \in A_i\) iff \(\neg \phi \notin A_i\), \(\phi \in A_i\) iff \((\pi, i) \models \phi\) (by the way we choose \(A_i\)) iff \((\pi, i) \not\models \neg \phi\) (by the semantics of CARET) iff \(\neg \phi \notin A_i\) (by the way we choose \(A_i\)) . The property holds.

- For every \(\phi_1 \lor \phi_2 \in A_i\), we need to show that \(\phi_1 \lor \phi_2 \in A_i\) iff \(\phi_1 \in A_i\) or \(\phi_2 \in A_i\), \(\phi_1 \lor \phi_2 \in A_i\) iff \((\pi, i) \models \phi_1 \lor \phi_2\) (by the way we choose \(A_i\)) iff \((\pi, i) \not\models \phi_1\) or \((\pi, i) \not\models \phi_2\) (by the semantics of CARET) iff \(\phi_1 \in A_i\) or \(\phi_2 \in A_i\) (by the way we choose \(A_i\)). The property holds.

- For every \(\phi_1 U^b \phi_2 \in A_i\) where \(b \in \{ g, a \}\), we need to show that \(\phi_1 U^b \phi_2 \in A_i\) iff \(\phi_2 \in A_i\) or \((\phi_1 \in A_i\) and \(X^b(\phi_1 U^b \phi_2) \in A_i\))

Let \(i_0, i_1, i_2 \ldots\) be a sequence of positions of \(\pi\) where \(i_0 = i, i_{m+1} = next^b_{i_m}\) for every \(m \geq 0\).

\[
\phi_1 U^b \phi_2 \in A_i \quad \text{iff} \quad (\pi, i) \models \phi_1 U^b \phi_2 \quad \text{iff} \quad (\pi, i) \models \phi_2 \quad \text{iff} \quad (\pi, i) \models \phi_1 \quad \text{and}
\]

...
(π, i_1) ⊨ φ_1 U^b φ_2) (by the semantics of U^b) iff (π, i) ⊨ φ_2 or ((π, i) ⊨ φ_1 and (π, i) ⊨ X^b φ_1 U^b φ_2) (since π(i_1) = \text{next}^b_i(π(i))) iff φ_2 ∈ A_i or (φ_1 ∈ A_i and X^b(φ_1 U^b φ_2) ∈ A_i) (by the way we choose A_i). The property holds.

- For every φ_1 U^c φ_2 ∈ A_i, we need to show that φ_1 U^c φ_2 ∈ A_i iff φ_2 ∈ A_i or (φ_1 ∈ A_i and X^c(φ_1 U^c φ_2) ∈ A_i)

Let i_0, i_1, ..., i_k be a sequence of positions of π where i_0 = i, i_{m+1} = \text{next}^c_i for every 0 ≤ m ≤ (k - 1).

φ_1 U^c φ_2 ∈ A_i iff (π, i) ⊨ φ_1 U^c φ_2 iff (π, i) ⊨ φ_2 or ((π, i) ⊨ φ_1 and (π, i_1) ⊨ φ_1 U^c φ_2) (by the semantics of U^b) iff (π, i) ⊨ φ_2 or ((π, i) ⊨ φ_1 and (π, i) ⊨ X^c(φ_1 U^c φ_2)) (since π(i_1) = \text{next}^c_i(π(i))) iff φ_2 ∈ A_i or (φ_1 ∈ A_i and X^c(φ_1 U^c φ_2) ∈ A_i) (by the way we choose A_i). The property holds.

- We need to show that A_i includes exactly one element of the set \{call, int, ret\}. Note that by the definition in Section 2.2.2, each transition \(\langle p_i, ω_i \rangle \Rightarrow_p \langle p_{i+1}, ω_{i+1} \rangle\) is tagged with only one element in the set \{call, int, ret\}. So, this property holds.

Let π'(0) = \(\langle p_0, A_0, \text{unexit}, ω_0 \rangle\) where A_0 = \{φ ∈ Cl(ψ) | (π, 0) ⊨ φ\} \cup \{t_0\}.
Let π'(k) = \(\langle p_i, A_i, l_i, γ_m γ_{m-1} ... γ_0 \rangle\) where A_i ∈ Atoms(ψ), l_i ∈ Label, γ_i is of the form γ_i or \(\langle γ_{t_i}, A_i, l_i \rangle\) for every 0 ≤ t ≤ m. We get that pr(π'(k)) = \(\langle p_i, γ_m γ_{m-1} ... γ_0 \rangle\). Let ω_i = γ_m γ_{m-1} ... γ_0, we get that pr(π'(k)) = \(\langle p_i, ω_i \rangle\).

Now we show that for every k ≥ 0, we can compute from π'(k) its immediate successor π'(k + 1) = \(\langle q, A', l', ω' \rangle\). During this construction, we maintain the following property:

"For every k + 1 ≥ 0, l' is the label expressing whether the execution of the procedure \(\mathcal{P}(\text{pr}(π'(k + 1)))\) terminates or not from \(\text{pr}(π'(k + 1))\); and for every φ ∈ A', \(\text{pr}(⟨π', k + 1⟩)\) satisfies φ"
2.3. CARET Model-Checking for Pushdown Systems

There are two cases: 

Induction Step ($\text{Atoms} = \phi \Rightarrow \text{the property holds for this case.}$)

1. Since $\text{Cl}$ corresponds to the transition procedure $A$ to $\phi$ for every $\phi \in A_0$, $(\pi, 0)$ satisfies $\phi$. Therefore, for every $\phi \in A_0$, $\text{pr}((\pi', 0))$ satisfies $\phi$ (since $\text{pr}((\pi', 0)) = (\pi, 0)$). In other words, for every $\phi \in A'$, $\text{pr}((\pi', 0))$ satisfies $\phi$ (since $A' = A_0$). The property related to $A'$ in the above property is satisfied.

$\implies$ The property holds for this case.

- Induction Step $(k + 1 > 0)$

There are two cases: $\gamma'_m = \gamma_m$ or $\gamma'_m = \langle \gamma_m, A'', l'' \rangle$ where $\gamma_m \in \Gamma, A'' \in \text{Atoms}(\psi), l'' \in \text{Label}$. 

1. $\gamma'_m = \gamma_m$, then, $\pi'(k) = \langle \langle p_i, A_i, l_i \rangle, \gamma_m, \gamma_{m-1} \ldots \gamma'_0 \rangle$. Let $A' = \{ \phi \in \text{CI}(\psi) \mid (\pi, i + 1) \models \phi \} \cup \{ t_{i+1} \}$. Let $l' = \text{exit}$ if the execution of the procedure $\mathcal{P}(\langle p_{i+1}, \omega_{i+1} \rangle)$ from $\langle p_{i+1}, \omega_{i+1} \rangle$ terminates; otherwise $l' = \text{unexit}$. In what follows, we use $A'$ and $l'$ to compute $\pi'(k + 1)$.

Our construction is based on different kinds of statements corresponding to the transition $\langle p_i, \omega_i \rangle \Rightarrow \pi \langle p_{i+1}, \omega_{i+1} \rangle$. There are three possibilities:

- if $\langle p_i, \omega_i \rangle \Rightarrow \pi \langle p_{i+1}, \omega_{i+1} \rangle$ corresponds to a call statement.

  In this case, we will use the transition rules in $(\alpha_1)$ to compute $\pi'(k + 1)$. Firstly, we show that $A_i$ and $A'$ satisfy the required conditions related to atoms in $(\alpha_1)$.

  * The conditions $(\beta_0)$, $(\beta_1)$, $(\beta_2)$ in $(\alpha_1)$ are true based on the way we choose $A_i$ and $A'$.

  * If $X^c \phi' \in A_i$, then $(\pi, i) \models X^c \phi'$ (by the way we choose $A_i$) $\implies (\pi, i + 1) \models \phi'$ (by the semantics of CARET) $\implies \phi' \in A'$ (by the way we choose $A'$) ($X^c \phi' \in A_i \implies \phi' \in A'$) implies that $\text{GlNext}(A_i, A') = true$. Then, the condition $(\beta_3)$ in $(\alpha_1)$ is true.

  * If $X^c \phi' \in A' \implies (\pi, i + 1) \models X^c \phi'$ (by the way we choose $A'$ and the fact that $\pi(i)$ is the caller-successor of $\pi(i + 1)$) $\implies (\pi, i) \models \phi'$ (by the semantics of CARET) $\implies \phi' \in A_i$ (by the way we choose $A_i$). ($X^c \phi' \in A' \implies \phi' \in A_i$) implies that $\text{CallerNext}(A', A_i) = true$. Then, the condition $(\beta_4)$ in $(\alpha_1)$ is true.
* We need to show that if \( l' = \text{unexit} \) then \( \text{NexAbsForms}(A_i) = \emptyset \) (by \( (\beta_2) \) in \( (\alpha_1) \)). \( l' = \text{unexit} \) implies that the execution of the procedure \( P((p_{i+1}, \omega_{i+1})) \) from \( (p_{i+1}, \omega_{i+1}) \) never terminates \( \implies (p_i, \omega_i) \) can never reach its corresponding return-point \( \implies \) the abstract successor of \( (p_i, \omega_i) \) is \( \bot \) (by the definition of abstract successor) \( \implies (p_i, \omega_i) \not\in X^a \phi \) (by the semantics of CARET) \( \implies \text{NexAbsForms}(A_i) = \emptyset \). Then, this condition holds.

Now, we need to show that the conditions related to labels in the transition rule \( (\alpha_1) \) are satisfied. In other words, we need to show that if \( l' = \text{unexit} \) then \( l_i = \text{unexit} \). \( l' = \text{unexit} \) implies that the execution of the procedure \( P((p_{i+1}, \omega_{i+1})) \) from \( (p_{i+1}, \omega_{i+1}) \) never terminates. In addition, we get that \( (p_i, \omega_i) \models p (p_{i+1}, \omega_{i+1}) \). Thus, the execution of the procedure \( P((p_i, \omega_i)) \) from \( (p_i, \omega_i) \) never terminates. As a result, \( l_i = \text{unexit} \). The property is proved.

Therefore, if \( (p_i, \omega_i) \models p (p_{i+1}, \omega_{i+1}) \) corresponds to a call statement, let \( (p_i, \gamma_m) \xrightarrow{\text{call}} (p_{i+1}, \gamma' \gamma'') \) be the rule associated to this transition. Then, we apply the rules in \( (\alpha_1) \) with the pair of atoms \( (A_i, A') \), the pair of labels \( (l_i, l') \), and we select \( \pi(k + 1) = \langle (p_{i+1}, A', l'), \gamma' \gamma'' A_i, l_0 \gamma' \gamma'' \cdots \gamma' \rangle \). By the way we select \( A' \) and \( l' \), we get that for every \( \phi \in A' \), \( pr((\pi', k + 1)) \) satisfies \( \phi \) and \( l' \) is the label expressing the execution of the procedure \( P(pr(\pi'(k + 1))) \) from \( pr(\pi'(k + 1)) \) can terminate or not. The property holds for this case.

- if \( (p_i, \omega_i) \models p (p_{i+1}, \omega_{i+1}) \) corresponds to a return statement.

In this case, we will use the transition rules in \( (\alpha_{2.1}) \) to compute \( \pi'(k + 1) \). Firstly, we show that \( A_i \) and \( A' \) satisfy the required conditions related to atoms in \( (\alpha_{2.1}) \).

* The conditions \( (\beta_0), (\beta_1), (\beta_2) \) in \( (\alpha_{2.1}) \) are true based on the way we choose \( A_i \) and \( A' \).

* If \( X^g \phi' \in A_i \implies (\pi, i) \models X^g \phi' \) (by the way we choose \( A_i \)) \implies (\pi, i + 1) \models \phi' \) (by the semantics of CARET) \implies \phi' \in A' \) (by the way we choose \( A' \)). \( (X^g \phi' \in A_i \implies \phi' \in A') \) implies that \( GLNext(A', A_i) = \text{true} \). Then, the condition \( (\beta_3) \) in \( (\alpha_{2.1}) \) is true.

* \( (p_i, \omega_i) \models p (p_{i+1}, \omega_{i+1}) \) corresponds to a return statement \( \implies l_i = \text{ret} \) (by the way we associate a tag to a configuration) \implies \) the abstract successor of \( (p_i, \omega_i) \) is \( \bot \) (by the definition of abstract successor) \implies (p_i, \omega_i) \not\in X^a \phi \) (by
the semantics of CARET) $\implies \text{NexAbsForms}(A_i) = \emptyset$.

Then, the condition $(\beta_4)$ in $(\alpha_{2,1})$ is true.

Now, we need to show that the condition related to labels in the transition rule $(\alpha_{2,1})$ is satisfied. In other words, we need to prove that $l_i = \text{exit}$, this is ensured because $\langle p_i, \omega_i \rangle \Rightarrow_p \langle p_{i+1}, \omega_{i+1} \rangle$ corresponds to a return statement, then, at this point, we know that the execution of the procedure $\mathcal{P}(\langle p_i, \omega_i \rangle)$ can terminate.

Therefore, if $\langle p_i, \omega_i \rangle \Rightarrow_p \langle p_{i+1}, \omega_{i+1} \rangle$ corresponds to a ret statement, let $\langle p_i, \gamma_m \rangle \rightarrow^\text{ret} \langle p_{i+1}, \varepsilon \rangle$ be the rule associated with this transition. Then, we apply the rules in $(\alpha_{2,1})$ with the pair of atoms $(A_i, A')$, the pair of labels $(l_i, l')$, and we select $\pi(k + 1) = \langle \langle p_{i+1}, A', l' \rangle, \gamma_{m-1} \ldots \gamma_0 \rangle$. By the way we select $A'$ and $l'$, we get that for every $\phi \in A'$, $\text{pr}(\langle \pi', k + 1 \rangle)$ satisfies $\phi$; and $l'$ is the label expressing whether the execution of the procedure $\mathcal{P}(\text{pr}(\pi(k + 1)))$ can terminate or not. The property holds for this case.

- if $\langle p_i, \omega_i \rangle \Rightarrow_p \langle p_{i+1}, \omega_{i+1} \rangle$ corresponds to a simple statement. In this case, we will use the transition rules in $(\alpha_3)$ to compute $\pi'(k + 1)$. Firstly, we show that $A_i$ and $A'$ satisfy the required conditions related to atoms in $(\alpha_3)$.

  - The conditions $(\beta_0)$, $(\beta_1)$, $(\beta_2)$ in $(\alpha_3)$ are true based on the way we choose $A_i$ and $A'$.
  
  * If $X^a \phi' \in A_i \implies (\pi, i) \models X^a \phi'$ (by the way we choose $A_i$) $\implies (\pi, i + 1) \models \phi'$ (by the semantics of CARET) $\implies \phi' \in A'$ (by the way we choose $A'$) $\implies \text{GlNext}(A_i, A') = \text{true}$.
   
   Then, the condition $(\beta_3)$ in $(\alpha_3)$ is true.

  * If $X^a \phi' \in A_i \implies (\pi, i) \models X^a \phi'$ (by the way we choose $A_i$) $\implies (\pi, i + 1) \models \phi'$ (by the semantics of CARET and the fact that $\pi(i + 1)$ is the abstract-successor of $\pi(i)$ when this corresponds to a simple statement) $\implies \phi' \in A'$ (by the way we choose $A'$). Thus, we obtain $\text{AbsNext}(A_i, A') = \text{true}$. Then, the condition $(\beta_4)$ in $(\alpha_3)$ is true.

  * Now, we prove the condition $(\beta_5)$ in $(\alpha_3)$. Note that $\pi(i)$ and $\pi(i + 1)$ have the same caller-successor. Let denote this caller-successor be $\pi(x)$. If $X^c \phi' \in A_i \implies (\pi, i) \models X^c \phi'$ (by the way we choose $A_i$) $\implies (\pi, x) \models \phi'$ (by the semantics of CARET) $\implies (\pi, i + 1) \models X^c \phi'$ (because $\pi(x)$ is the caller-successor of $\pi(i + 1)$) $\implies X^c \phi' \in A'$ (by the way we choose $A'$). Consequently, $\text{NextCallerForms}(A_i) =$
Chapter 2. CARET Model Checking For Pushdown Systems

NextCallerForms(A'), the condition (β₅) in (α₃) is true. Now, we need to show that the conditions related to labels in the transition rule (α₃) are satisfied. In other words, we need to prove that \( l' = l_i \). This is always ensured by the fact that \( \mathcal{P}(\langle p_i, \omega_i \rangle) = \mathcal{P}(\langle p_{i+1}, \omega_{i+1} \rangle) \) since this transition corresponds to a simple statement, i.e., the procedural context doesn’t change.

Therefore, if \( \langle p_i, \omega_i \rangle \Rightarrow \langle p_{i+1}, \omega_{i+1} \rangle \) corresponds to a ret statement, let \( \langle p_i, \gamma_m \rangle \xrightarrow{\text{inv}} \langle p_{i+1}, \omega \rangle \) be the rule associated with this transition. Then, we apply the rules in (α₂₁) with the pair of atoms \((A_i, A')\), the pair of labels \((l_i, l')\), and we select \( \pi'(k + 1) = \langle \langle p_{i+1}, A', l' \rangle, \omega, \gamma_{m-1}^{\prime}, \gamma'_{m}^{3}, \gamma'_{m-1}^{2} \rangle \). By the way we select \( A' \) and \( l' \), we get that for every \( \phi \in A' \), \( pr((\pi', k + 1)) \) satisfies \( \phi \); and \( l' \) is the label expressing whether the execution of the procedure \( \mathcal{P}(pr(\pi'(k + 1))) \) from \( pr(\pi'(k + 1)) \) can terminate or not. The property holds for this case.

2. \( \gamma'_m = \langle \gamma_m, A'', l'' \rangle \), then, \( \pi'(k) = \langle \langle p_i, A_i, l_i \rangle, \langle \gamma_m, A'', l'' \rangle \rangle \). Note that this case only occurs at return-points. Let \( \pi(u) = \langle p_u, \omega_u \rangle \) be the corresponding call of this ret. Then, we get that \( A'' \) and \( l'' \) are the atom and the label of \( \pi(u) \), i.e, \( A'' = A_u \) and \( l'' = l_u \).

In this case, we will use the transition rules in (α₂₂) to compute \( \pi'(k + 1) \). Firstly, we show that the required conditions related to atoms in (α₂₂) are satisfied.

- If \( X^a \phi' \in A'' \Rightarrow (\pi, u) \models X^a \phi' \) (by the way we choose \( A'' \) at \( \pi(u) \)) \( \Rightarrow (\pi, i + 1) \models \phi' \) (by the semantics of CARET and the fact that \( \pi(i + 1) \) is the return-point of \( \pi(u) \)) \( \Rightarrow \phi' \in A' \) (by the way we choose \( A' \)). \( (X^a \phi' \in A'' \Rightarrow \phi' \in A') \) implies that \( \text{AbsNext}(A'', A') = \text{true} \). Then, the condition (β₅) in (α₂₂) is true.

- Now, we prove the condition (β₆) in (α₂₂). Notice that \( \pi(i) \) and \( \pi(u) \) have the same caller-successor. Let us denote this caller-successor by \( \pi(x) \). If \( X^c \phi' \in A' \Rightarrow (\pi, i) \models X^c \phi' \) (by the way we choose \( A' \) \( \Rightarrow (\pi, x) \models \phi' \) (by the semantics of CARET) \( \Rightarrow (\pi, u) \models X^c \phi' \) (because \( \pi(x) \) is the caller-successor of \( \pi(u) \)) \( \Rightarrow X^c \phi' \in A_u \) (by the way we choose \( A_u \)) \( \Rightarrow X^c \phi' \in A'' \) (because \( A'' = A_u \)). Thus, we have \( X^c \phi' \in A' \Rightarrow X^c \phi' \in A'' \). Consequently, \( \text{NextCallerForms}(A') = \text{NextCallerForms}(A'') \), the condition (β₆) in (α₂₂) is true.

- The conditions (β₇) in (α₂₂) are true based on the way we choose \( A' \).
The conditions \((\beta_8)\) in \((\alpha_{2.2})\) are true based on the fact that \(\pi(u)\) and \(\pi(k)\) have the same caller successor.

Let \(l' = l_i\) in this case. Now, we need to show that the condition related to labels in the transition rule \((\alpha_{2.2})\) is satisfied. In other words, we need to prove that \(l_i = l''\) where \(l''\) is the label of \(\langle p_u, \omega_u \rangle\). This is always satisfied because a call point \(\langle p_u, \omega_u \rangle\) and its corresponding return point \(\langle p_i, \omega_i \rangle\) always belong to the same procedure. Therefore, we always obtain that \(l_i = l''\).

Therefore, if \(\pi'(k)\) is of the form \(\langle \langle p_i, A_i, t_i \rangle, \langle \gamma_m, A', \phi' \rangle \gamma_{m-1} \ldots \gamma_0 \rangle\), we apply the transition rules in \((\alpha_{2.2})\) and select \(\pi'(k + 1) = \langle \langle p_i, A_i, t_i \rangle, \gamma_m \gamma_{m-1} \ldots \gamma_0 \rangle\). By applying the induction hypothesis, we obtain that for every \(\phi \in A_i\), \(pr((\pi', k))\) satisfies \(\phi\). Since \(pr((\pi', k + 1)) = pr((\pi', k))\), we get that for every \(\phi \in A_i\), \(pr((\pi', k + 1))\) satisfies \(\phi\). In addition, by the way we choose \(l', l''\) is the label expressing whether the execution of the procedure \(\mathcal{P}(pr((\pi', k + 1)))\) from \(pr((\pi', k + 1))\) can terminate or not. Therefore, the property holds for this case.

Moreover, \(\pi \models \psi\) implies the following properties:

- \(\psi \in A_0\) (by the way we choose \(A_i\))
- \(NexCallerForms(A_0) = \emptyset\) (by the way we choose \(A_i\)) and by the fact that \(\langle p_0, \omega_0 \rangle\) is the root, then, it has no caller-successor)

Now, we prove that \(\pi'\) is an accepting run of \(\mathcal{B}_\psi\). We prove that each set of the Büchi accepting condition of \(\mathcal{B}_\psi\) is visited infinitely often by \(\pi'\). Suppose that this is not the case, then there exists a set \(F_{\phi_1 U b \phi_2}\) where \(b \in \{g, a\}\) such that \(\pi'\) does not visit infinitely often any control location in \(P \times F_{\phi_1 U b \phi_2} \times \text{Label}\). This means that there exists \(k\) where the suffix of \(\pi'\) starting from \(\pi'(k)\) (denoted \((\pi', k)\)) does not visit any control location in \(P \times F_{\phi_1 U b \phi_2} \times \text{Label}\). It implies that for every \(t \geq k\), where \(\pi'(t) = \langle \langle p_t, A_t, t \rangle, \omega_i \rangle\), we must have \(\phi_2 \notin A_t\) and \(\phi_1 U^b \phi_2 \in A_t\) (otherwise, \(\langle p_t, A_t, t \rangle\) belongs to \(P \times F_{\phi_1 U b \phi_2} \times \text{Label}\)).

- \(\phi_1 U^b \phi_2 \in A_t \implies pr((\pi', t)) \models \phi_1 U^b \phi_2\). \(pr((\pi', t)) \models \phi_1 U^b \phi_2\) implies that \(\phi_2\) eventually holds.
- \(\phi_2 \notin A_t \implies pr((\pi', t)) \n
Note that the second fact that for every \(t \geq k\), \(pr((\pi', t)) \n
Consequently, the run \(\pi'\) visits infinitely often some control locations in \(P \times
F_{\phi_1 \land \phi_2} \cdot \pi' \text{ visits infinitely often each set of the Büchi accepting condition of } \mathcal{BP}_\psi \text{ implies that } \pi' \text{ is an accepting run of } \mathcal{BP}_\psi.

In conclusion, from a run } \pi \text{ of } \mathcal{P} \text{ such that } \pi \models \psi, \text{ we can always obtain an accepting run } \pi' \text{ of } \mathcal{BP}_\psi \text{ starting from } \langle \langle p_0, A_0, unexit \rangle, \omega_0 \rangle \text{ such that } \psi \in A_0 \text{ and } \text{NexCallerForms}(A_0) = \emptyset.

The number of control locations of } \mathcal{BP} \text{ is at most } |P| \times 2^{O(|\psi|)} \text{ and the number of transitions is at most } |\Delta||\Gamma| \times 2^{O(|\psi|)}. \text{ From Theorem 1, the membership problem can be solved in time } |P|.|\Delta|^2.|\Gamma|^2.2^{O(|\psi|)}. \text{ Thus, we get:}

**Theorem 3.** Given a PDS } \mathcal{P} = (P, \Gamma, \Delta) \text{, a labeling function } \lambda : P \rightarrow 2^{AP} \text{ and a CARET formula } \psi, \text{ for every configuration } \langle p, \omega \rangle, \text{ whether or not } \langle p, \omega \rangle \text{ satisfies } \psi \text{ can be solved in time } |P|.|\Delta|^2.|\Gamma|^2.2^{O(|\psi|)}.

### 2.4 CARET Model Checking for PDS with Regular Valuations

In this section, we discuss how to do CARET model-checking for PDSs with regular valuations, where the set of configurations in which an atomic proposition holds is a regular language.

**Definition 6.** Let } \mathcal{P} = (P, \Gamma, \Delta) \text{ be a PDS. A set of configurations is regular if it can be written as the union of sets of the form } E_p, \text{ where } p \in P \text{ and } E_p = \{(p, w) \mid w \in L_p\}, \text{ where } L_p \text{ is a regular set over } \Gamma^*.

**Definition 7.** Let } \mathcal{P} = (P, \Gamma, \Delta) \text{ be a PDS. Let } AP \text{ be a finite set of atomic propositions. Let } \nu : AP \rightarrow 2^{P \times \Gamma^*} \text{ be a valuation. } \nu \text{ is called regular if for every } e \in AP, \nu(e) \text{ is a regular set of configurations.}

Let } \nu : AP \rightarrow 2^{P \times \Gamma^*} \text{ be a regular valuation. We define } \lambda_\nu : P \times \Gamma^* \rightarrow 2^{AP} \text{ such that } \lambda_\nu((p, \omega)) = \{e \in AP \mid \langle p, \omega \rangle \in \nu(e)\}. \text{ Let } \pi = \langle \langle p_0, \omega_0 \rangle, \langle p_1, \omega_1 \rangle, \ldots \rangle \text{ be an execution of } \mathcal{P}. \text{ We associate to each configuration } \langle p_i, \omega_i \rangle \text{ of } \pi \text{ a tag } t_i \text{ in } \{\text{call, int, ret}\} \text{ as done in Section 2.2.2}. \text{ Let } \psi \text{ be a CARET formula over } AP. \text{ The satisfiability relation wrt the regular valuation } \nu \text{ is defined as follows:}

\[ \pi \models_{\nu} \psi \text{ iff } (\lambda_\nu((p_0, \omega_0)), t_0)(\lambda_\nu((p_1, \omega_1)), t_1) \cdots \models \psi \]

**Theorem 4.** [EKS03] LTL model-checking with regular valuations for PDSs can be reduced to standard LTL model checking for PDSs.
2.5. Applications

Given a PDS \( \mathcal{P} = (P, \Gamma, \Delta) \) and a regular valuation \( \nu : AP \rightarrow 2^{P \times \Gamma} \), the above result is based on translating the PDS \( \mathcal{P} \) into a PDS \( \mathcal{P}' = (P', \Gamma, \Delta') \) where the regular valuation requirements are encoded in \( P' \). The same reduction is still valid for CARET with regular valuations. We refer the reader to [EKS03] for details. Thus, given a CARET formula with regular valuations and a PDS \( \mathcal{P} = (P, \Gamma, \Delta) \), we apply the reduction of [EKS03] to get a PDS \( \mathcal{P}' = (P', \Gamma, \Delta') \) such that CARET model checking of \( \mathcal{P} \) wrt the regular valuation \( \nu \) can be reduced to standard CARET model checking of \( \mathcal{P}' \). Thus, we get that:

**Theorem 5.** CARET model-checking with regular valuations for PDSs can be reduced to standard CARET model checking for PDSs.

2.5 Applications

In this section, we show how CARET can be used to describe various kinds of properties.

2.5.1 Modeling Malicious Behaviors using CARET

CARET is more expressive than LTL, thus, all the malicious behaviors that are expressible in LTL [ST13a] can always be specified by CARET. We showed in the introduction how we can use CARET to describe the typical malicious behavior of a spyware. Now, we show in this section how CARET allows to precisely specify several malicious behaviors that cannot be described by LTL. We also show how describing some malicious behaviors using LTL as done in [ST13a] may lead to false alarms which can be avoided if we use CARET instead of LTL.

Most of the malicious behaviors that we consider need to be expressed using \textit{CARET with regular valuations}. For succinctness, we write \( d^{\Gamma^*} \) (resp. \( 0d^{\Gamma^*} \)) to express the regular valuation that means “the content of the stack is in \( d^{\Gamma^*} \) (resp. in \( 0d^{\Gamma^*} \)).

**Open and listen on a specific port:** Malware writers often configure the malware to listen to a specific port to receive information (such as updates, new attack targets,...). To do this, it needs to invoke the API \textit{socket} to create a socket, followed by a call to the API \textit{bind} to associate a local address with the socket and a call to \textit{listen} to put the socket in the listening state. The call to the API \textit{socket} returns a descriptor referencing the new socket which is used as input of the calls to the APIs \textit{bind} and \textit{listen}. Thus, when \textit{bind} and \textit{listen} are invoked, the socket descriptor must be on top of the program’s stack. This malicious behavior can be specified by CARET as follows:
\[ \psi_{lp} = \bigvee_{d \in D} F^g \left( \text{call(socket)} \land X^a(eax = d) \land F^a \left( \text{call(bind)} \land d\Gamma^* \land F^a \left( \text{call(listen)} \land d\Gamma^* \right) \right) \right) \]

where the \( \bigvee \) is taken over all possible memory addresses \( d \) which contain the values of descriptors referencing the new socket.

As mentioned before, in binary codes and assembly programs, the return value of an API function is put in the register \( eax \). Thus, the return value of \( \text{socket} \) is the value of \( eax \) at its corresponding return-point. Then, the subformula \( F^g(\text{call(socket)} \land X^a(eax = d)) \) states that there is a call to the API \( \text{socket} \) and the return value of this function is \( d \) (the abstract successor of a call is its return-point). When \( \text{bind} \) and \( \text{listen} \) are invoked, one of their required parameters is the socket descriptor and this socket descriptor must be put on top of the program stack (since parameters are passed through the stack in assembly). The requirement that \( d \) is on top of the program stack is expressed by the regular valuation \( d\Gamma^* \). Thus, the subformula \( \text{call(bind)} \land d\Gamma^* \) expresses that \( \text{bind} \) is called with \( d \) as parameter (\( d \) stores the information of the descriptor). Similarly, the subformula \( \text{call(listen)} \land d\Gamma^* \) expresses that \( \text{listen} \) is called with \( d \) as parameter. Thus, \( \psi_{lp} \) expresses then that there is a call to the API \( \text{socket} \) with the return value \( d \) (the descriptor), followed by a call to the function \( \text{bind} \) and a call to the function \( \text{listen} \) with \( d \) on the top of the stack. These behaviors allow the malware to open and listen on a port.

**Remark 2.** In our experiments, the domain \( D \) of possible values of memory addresses \( d \) is computed using the tool Jakstab [KV08].

**Registry Key Injecting:** Malware writers often create registry entries to set the malware as an authenticated program or make it started at the boot time. To do that, it calls the API \( \text{GetModuleFileNameA} \) with 0 and a memory address \( d \) as parameters; followed by a call to the API \( \text{RegSetValueExA} \) with the same parameter \( d \). After execution of \( \text{GetModuleFileNameA} \), the file name of the malware is stored at the address \( d \). After that, the API \( \text{RegSetValueExA} \) is invoked and adds the file name stored in \( d \) to the registry key listing. This malicious behavior can be expressed by the following CARET formula:

\[ \psi_{rk} = \bigvee_{d \in D} F^g \left( \text{call(GetModuleFileNameA)} \land 0d\Gamma^* \land F^a \left( \text{call(RegSetValueExA)} \land d\Gamma^* \right) \right) \]

where the \( \bigvee \) is taken over all possible memory addresses \( d \) which can store the file name of the malware. This formula states that the API function \( \text{GetModuleFileNameA} \) is called with 0 and a value \( d \) on the top of the stack (i.e., with 0 and \( d \) as parameters), followed by a call to the API function \( \text{RegSetValueExA} \) with the same \( d \) on the top of the stack. This behavior was described in [ST13a] using the following LTL formula:

\[ \psi_{rk} = \bigvee_{d \in D} F^g \left( \text{call(GetModuleFileNameA)} \land 0d\Gamma^* \land F^g \left( \text{call(RegSetValueExA)} \land d\Gamma^* \right) \right) \]
that uses the standard $F^g$ operator instead of CARET’s $F^a$. It can be seen that this LTL formula $\psi_{rk}$ is not as precise as the CARET formula $\psi_{rk2}$, as it may be satisfied even for paths where the call to `RegSetValueExA` is made e.g. before the function `GetModuleFileNameA` returns. Thus, this LTL formula $\psi_{rk}$ may lead to false alarms that can be avoided using our CARET formula $\psi_{rk2}$.

**Email Worm:** The typical characteristic of an email worm is to copy itself to other locations. To do this, the worm first calls the API `GetModuleFileNameA` with 0 and $d$ as parameters ($d$ is a memory address which is used to store the file name of the current executable when the API is executed). After that, the worm calls `CopyFileA` with the same $d$ as parameter. `CopyFileA` will use the file name stored at $d$ to copy itself to another location. This malicious behavior can be expressed by CARET as follows:

$$\psi_{em} = \bigvee_{d \in D} F^g(\text{call(GetModuleFileNameA)} \land 0d \Gamma^* \land F^a(\text{call(CopyFileA)} \land d \Gamma^*))$$

As previously, this formula $\psi_{em}$ is not as precise as our CARET formula $\psi_{em2}$ since $\psi_{em}$ does not discard e.g. the case where `CopyFileA` is called before `GetModuleFileNameA` returns. Thus, $\psi_{em}$ may lead to false alarms that can be avoided using our CARET formula $\psi_{em2}$.

### 2.5.2 Checking API Usage rules

Modern softwares increasingly utilize third-party libraries which can be accessed through application programming interfaces (APIs) to shorten development time. These libraries usually impose several constraints on how APIs should be used (API usage rules). Following these rules in programming is very important to avoid unexpected side-effects. Therefore, it is crucial to check API usage rule correctness for programs. We show in what follows how these rules can be described by CARET formulas.

**File Operation Using Rules.** One typical constraint for operations on files is that the opened files should be closed after being used. Note that closing files is very critical to maintain performance since longtime running programs will absorb a huge amount of resources if the opened files are not closed. This requirement can be expressed by the API rule "Calling the API `fopen` in some procedure `proc` must be followed by a call to the function `fclose` before the
procedure \textit{proc} terminates”. This API usage rule can be described in CARET by the following succinct formula: $$\phi = G^g(fopen \implies F^a fclose)$$. $$\phi_2$$ states that in any execution path, at a certain point of a certain procedure \textit{proc}, if the function \textit{fopen} is invoked, then, in the same function \textit{proc}, there must be a call to the API \textit{fclose} to close the opened file. Note that the operator $$F^a$$ allows us to ensure that \textit{fopen} and \textit{fclose} are called in the same function \textit{proc}.

**Socket Using Rules.** The socket library is popularly used in server programs. When a client sends a connection request, the server creates a new socket by calling the API function \textit{socket} to communicate with that client. When the communication ends, the socket should be closed by calling the API \textit{close}. Note that if the socket is not closed, the address to which the socket is bound cannot be used by other sockets. If the bound address is a popular one, it will lead to a serious problem. To avoid this problem, one typical API usage rule for socket operations is that "The socket should be closed by calling the API function \textit{close} whenever that socket is created by calling the API \textit{socket}”. Similar to the file operation usage rules, we can specify this API usage rule by the CARET formula: $$\phi = G^g(socket \implies F^a close)$$. 

### 2.5.3 Security Properties

The basic security mechanism of Java Development Kit JDK is based on permissions. In this mechanism, a piece of code is assigned to a set of permissions. If a critical operation \textit{op} is invoked by a method \textit{p}_1, then, \textit{p}_1 must have the permission to do that. Also, if \textit{p}_1 is invoked in the body of another method \textit{p}_2, then \textit{p}_2 is also required to have the permission to execute \textit{op}. In general, to execute a critical operation \textit{op}, all the code leading to the position where \textit{op} is executed must have the corresponding permissions. We show in what follows how CARET can be used to represent this property. Let \textit{per} be the permission to execute the operation \textit{op}. Then, the requirement above can be described by the following CARET formula:

$$\phi = G^g(per \implies G^c per)$$

Note that the subformula $$G^c per$$ ensures that all the methods that lead to the current execution of the operation \textit{op} also have the permission \textit{per}.

### 2.6 Experiments

We implemented our algorithms in a tool and carried out different experiments. We use Moped [ES01] as a tool to check emptiness of BPDSs. We first applied
our tool on randomly generated PDSs to see the efficiency of our procedure. The results are reported in Table 2.1.

Then, we applied our tool for malware detection. A program is declared as a malware if it satisfies one of the CARET formulas described previously. For each CARET formula, if we consider the disjunction over all possible values of the domain $\mathcal{D}$, our tool does not terminate. Thus, we consider only some values of $\mathcal{D}$ (these values are determined by Jakstab [KV08]). We obtained encouraging results. As malwares are executables, i.e., binary codes, we use the translation of [ST12a] together with the tools IDAPRO [IDA] and Jakstab [KV08] to generate a PDS from a binary code. Our tool was able to detect several malwares and to show that benign programs are benign as described in Table 2.2. The result Yes (resp. No) denotes that the program is a malware (resp. is benign). The malware samples are taken from "http://vxheaven.org/". Benign programs are taken from Microsoft Windows operating system. Note that, as mentioned in the introduction, it is not possible to obtain a RSM from these binary codes because they contain explicit push and pop instructions. Moreover, if we apply the translation in [ABE+05] to compute a bisimilar RSM, then we still cannot apply malware detection on these RSMs since the

| $|P|$ | $|\Gamma|$ | $|\Delta|$ | $|\psi|$ | Time(s) | $|P|$ | $|\Gamma|$ | $|\Delta|$ | $|\psi|$ | Time(s) |
|------|------|------|------|-------|------|------|------|------|-------|-------|
| 549  | 286  | 3050 | 11   | 12.5  | 89   | 126  | 47   | 15   | 44.7  |
| 344  | 532  | 3658 | 11   | 14.1  | 328  | 432  | 194  | 15   | 48.1  |
| 338  | 56   | 514  | 12   | 16.0  | 817  | 636  | 936  | 14   | 49.8  |
| 50   | 210  | 3286 | 9    | 16.8  | 852  | 928  | 3306 | 12   | 54.2  |
| 230  | 305  | 1618 | 14   | 17.4  | 13   | 77   | 2217 | 11   | 58.0  |
| 455  | 349  | 907  | 14   | 18.3  | 514  | 490  | 990  | 12   | 61.1  |
| 466  | 302  | 1162 | 12   | 21.1  | 657  | 907  | 3329 | 11   | 65.2  |
| 377  | 695  | 1394 | 12   | 24.5  | 363  | 194  | 4949 | 11   | 67.2  |
| 465  | 249  | 2105 | 11   | 27.4  | 850  | 740  | 3255 | 12   | 72.7  |
| 304  | 288  | 3495 | 11   | 28.9  | 770  | 520  | 4751 | 11   | 77.9  |
| 432  | 61   | 1030 | 12   | 30.5  | 341  | 39   | 4448 | 11   | 79.9  |
| 802  | 149  | 2330 | 15   | 32.8  | 349  | 89   | 1129 | 14   | 81.4  |
| 421  | 916  | 1359 | 7    | 34.1  | 863  | 946  | 3560 | 12   | 85.6  |
| 205  | 637  | 2887 | 6    | 35.3  | 645  | 923  | 3865 | 12   | 89.3  |
| 233  | 330  | 134  | 12   | 36.2  | 215  | 218  | 4988 | 15   | 92.5  |
| 197  | 331  | 2398 | 13   | 37.2  | 798  | 458  | 4936 | 12   | 96.4  |
| 856  | 353  | 3159 | 12   | 42.1  | 587  | 46   | 3592 | 12   | 99.3  |
| 487  | 392  | 4927 | 15   | 101.3 | 194  | 286  | 111  | 14   | 103.1 |

Table 2.1: Model Checking random PDSs against CARET formulas
malicious behaviors need CARET with regular valuations on the stack content, and by doing the translation from PDSs to RSMs the stack of the RSM does not correspond to the stack of the assembly program anymore. Thus, our techniques are crucial for malware detection.

2.7 Conclusion

In this chapter, we present an algorithm for model-checking PDSs against CARET formulas where whether a configuration of a PDS satisfies an atomic proposition or not depends only on the control location of that configuration. In addition, we consider CARET model-checking for PDSs with regular valuations where whether a configuration of a PDS satisfies an atomic proposition or not depends on both the control location and the stack content of that configuration. Our approach consists of reducing these problems to the emptiness problem of Büchi Pushdown Systems. The techniques are implemented in a tool that we applied to different case studies. Our experimental results are encouraging.
<table>
<thead>
<tr>
<th>Malware</th>
<th>Time (s)</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backdoor.Win32.Small.dm</td>
<td>59.1</td>
<td>Yes</td>
</tr>
<tr>
<td>Trojan-Downloader.Win32.Agent.cv</td>
<td>25.2</td>
<td>Yes</td>
</tr>
<tr>
<td>Email-Worm.Win32.Bater.a</td>
<td>8.3</td>
<td>Yes</td>
</tr>
<tr>
<td>Backdoor.Win32.Cocoazul.e</td>
<td>13.5</td>
<td>Yes</td>
</tr>
<tr>
<td>Backdoor.Win32.Agent.jp</td>
<td>41.2</td>
<td>Yes</td>
</tr>
<tr>
<td>HackTool.Win32.Tups.122</td>
<td>20.3</td>
<td>Yes</td>
</tr>
<tr>
<td>Email-Worm.Win32.Bloored.a</td>
<td>10.2</td>
<td>Yes</td>
</tr>
<tr>
<td>Email-Worm.Win32.Klez.b</td>
<td>13.4</td>
<td>Yes</td>
</tr>
<tr>
<td>Email-Worm.Win32.Krynos.b</td>
<td>9.8</td>
<td>Yes</td>
</tr>
<tr>
<td>Email-Worm.Win32.Lohack.b</td>
<td>17.3</td>
<td>Yes</td>
</tr>
<tr>
<td>Email-Worm.Win32.Lohack.c</td>
<td>17.5</td>
<td>Yes</td>
</tr>
<tr>
<td>Trojan-Downloader.Win32.Agent.dh</td>
<td>25.2</td>
<td>Yes</td>
</tr>
<tr>
<td>Email-Worm.Win32.Seliz</td>
<td>29.2</td>
<td>Yes</td>
</tr>
<tr>
<td>Email-Worm.Win32.Mydoom.az</td>
<td>19.3</td>
<td>Yes</td>
</tr>
<tr>
<td>Trojan-Downloader.Win32.Dadobra.z</td>
<td>35.5</td>
<td>Yes</td>
</tr>
<tr>
<td>Trojan-Downloader.Win32.Dyfuca.ae</td>
<td>40.3</td>
<td>Yes</td>
</tr>
<tr>
<td>Backdoor.Win32.IRCBot.jm</td>
<td>39.2</td>
<td>Yes</td>
</tr>
<tr>
<td>Backdoor.Win32.Lithium.10.b5</td>
<td>35.5</td>
<td>Yes</td>
</tr>
<tr>
<td>Backdoor.Win32.SdBot.as</td>
<td>40.9</td>
<td>Yes</td>
</tr>
<tr>
<td>Backdoor.Win32.SdBot.v</td>
<td>42.2</td>
<td>Yes</td>
</tr>
<tr>
<td>Backdoor.Win32.Wisdoor.at</td>
<td>39.5</td>
<td>Yes</td>
</tr>
<tr>
<td>Email-Worm.Win32.Ardurk.d</td>
<td>26.2</td>
<td>Yes</td>
</tr>
<tr>
<td>Email-Worm.Win32.Cholera</td>
<td>54.3</td>
<td>Yes</td>
</tr>
<tr>
<td>Benign</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cmd.exe</td>
<td>3.2</td>
<td>No</td>
</tr>
<tr>
<td>ipv6.exe</td>
<td>4.2</td>
<td>No</td>
</tr>
<tr>
<td>shutdown.exe</td>
<td>4.1</td>
<td>No</td>
</tr>
<tr>
<td>ping.exe</td>
<td>3.7</td>
<td>No</td>
</tr>
<tr>
<td>print.exe</td>
<td>3.4</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 2.2: Detection of real malwares
CARET Model Checking for Malware Detection

The number of malware is growing significantly fast. Traditional malware detectors based on signature matching or code emulation are easy to get around. To overcome this problem, model-checking emerges as a technique that has been extensively applied for malware detection recently. Pushdown systems were proposed as a natural model for programs, since they allow to keep track of the stack, while extensions of LTL and CTL were considered for malicious behavior specification. However, LTL and CTL like formulas don’t allow to express behaviors with matching calls and returns. In this thesis, we propose to use CARET for malicious behavior specification. Since CARET formulas for malicious behaviors are huge, we propose in this chapter to extend CARET with variables, quantifiers and predicates over the stack. Our new logic is called Stack linear temporal Predicate logic of CAlls and RETurns (SPCARET). We reduce the malware detection problem to the model checking problem of PDSs against SPCARET formulas, and we propose efficient algorithms to model check SPCARET formulas for PDSs. We implemented our algorithms in a tool for malware detection. We obtained encouraging results.

Outline. Section 3.1 introduces our logic SPCARET and shows how it can be used to precisely and succinctly describe malicious behaviors. Model checking SPCARET is discussed in Sections 3.2, 3.3 and 3.4. In Section 3.5, we present our experimental results. Finally, we conclude in Section 3.6.

3.1 Malicious Behaviour Specification

In this section, we define the Stack linear temporal Predicate logic of CAlls and RETurns (SPCARET) as an extension of the linear temporal logic of CAlls and RETurns (CARET) with variables and regular predicates over the stack contents. The predicates contain variables that can be quantified existentially or universally. Regular predicates are expressed by regular variable expressions and are used to describe the stack content of PDSs.
3.1.1 Environments, Predicates and Regular Variable Expressions

Let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be a finite set of variables over a finite domain $\mathcal{D}$. Let $B : \mathcal{X} \cup \mathcal{D} \to \mathcal{D}$ be an environment that associates each variable $x \in \mathcal{X}$ with a value $d \in \mathcal{D}$ s.t. $B(d) = d$ for every $d \in \mathcal{D}$. Let $B[x \leftarrow d]$ be an environment obtained from $B$ such that $B[x \leftarrow d](x) = d$ and $B[x \leftarrow d](y) = B(y)$ for every $y \neq x$. Let $\mathcal{B}$ be the set of all environments. Let $\theta_{id} = \{(B, B') \in \mathcal{B} \times \mathcal{B} \mid B = B'\}$ be the identity relation for environments, and for $x \in \mathcal{X}$, let $\theta_x = \{(B, B') \in \mathcal{B} \times \mathcal{B} \mid \forall y \in \mathcal{X}, y \neq x, B(y) = B'(y)\}$ be the relation that abstracts away the value of $x$.

Let $\mathcal{AP} = \{a, b, c, \ldots\}$ be a finite set of atomic propositions. Let $\mathcal{AP}_\mathcal{D}$ be a finite set of atomic predicates of the form $b(\alpha_1, \ldots, \alpha_m)$ such that $b \in \mathcal{AP}$ and $\alpha_i \in \mathcal{D}$ for every $1 \leq i \leq m$. Let $\mathcal{AP}_{\mathcal{X}}$ be a finite set of atomic predicates $b(\alpha_1, \ldots, \alpha_n)$ such that $b \in \mathcal{AP}$ and $\alpha_i \in \mathcal{X} \cup \mathcal{D}$ for every $1 \leq i \leq n$.

Let $\mathcal{P} = (P, \Gamma, \Delta)$ be a Labelled PDS. A Regular Variable Expression (RVE) $e$ over $\mathcal{X} \cup \Gamma$ is defined by $e ::= \varepsilon \mid a \in \mathcal{X} \cup \Gamma \mid e + e \mid e.e \mid e^*$. The language $L(e)$ of a RVE $e$ is a subset of $P \times \Gamma^* \times \mathcal{B}$ and is defined as follows:

- $L(\varepsilon) = \{(\langle p, \varepsilon \rangle, B) \mid p \in P, B \in \mathcal{B}\}$
- for $x \in \mathcal{X}$, $L(x) = \{(\langle p, \gamma \rangle, B) \mid p \in P, \gamma \in \Gamma, B \in \mathcal{B} \text{ s.t. } B(x) = \gamma\}$
- for $\gamma \in \Gamma$, $L(\gamma) = \{(\langle p, \gamma \rangle, B) \mid p \in P, B \in \mathcal{B}\}$
- $L(e_1, e_2) = \{(\langle p, \omega_2 \omega_1 \rangle, B) \mid (\langle p, \omega_1 \rangle, B) \in L(e_1); (\langle p, \omega_2 \rangle, B) \in L(e_2)\}$
- $L(e^*) = \{(\langle p, \omega \rangle, B) \mid \omega \in \{v \in \Gamma^* \mid (\langle p, v \rangle, B) \in L(e)\}^*\}$

3.1.2 The Stack linear temporal Predicate logic of CAlls and RETurns - SPCARET

A SPCARET formula is a CARET [AEM04] formula where predicates and RVEs are used as atomic propositions and where quantifiers are applied to variables. For technical reasons, we assume w.l.o.g. that formulas are written in positive normal form, where negations are applied only to atomic predicates, and we use the release operator $R$ as the dual of the until operator $U$. From now on, we fix a finite set of variables $\mathcal{X}$, a finite set of atomic propositions $\mathcal{AP}$, a finite domain $\mathcal{D}$, and a finite set of RVEs $\mathcal{V}$. A SPCARET formula is defined as follows, where $v \in \{g, a, c\}$, $x \in \mathcal{X}$, $e \in \mathcal{V}$, $b(\alpha_1, \ldots, \alpha_n) \in \mathcal{AP}_\mathcal{X}$:

$\psi ::= b(\alpha_1, \ldots, \alpha_n) \mid \neg b(\alpha_1, \ldots, \alpha_n) \mid e \mid \neg e \mid \psi \lor \psi \mid \psi \land \psi \mid \forall x \psi \mid \exists x \psi \mid X^\psi \mid \psi U^\psi \mid \psi R^\psi$

Let $\lambda : P \to 2^{\mathcal{AP}}$ be a labelling function which associates each control location to a set of atomic predicates. Let $\psi$ be a SPCARET formula over
3.1. Malicious Behaviour Specification

AP. Let \( \langle p, \omega \rangle \) be a configuration of \( \mathcal{P} \). Then we say that \( \mathcal{P} \) satisfies \( \psi \) at \( \langle p, \omega \rangle \) (denoted by \( \langle p, \omega \rangle \models_\lambda \psi \)) iff there exists an environment \( B \in \mathcal{B} \), a path \( \pi = \langle p_0, \omega_0, p_1, \omega_1 \rangle \ldots \) starting from \( \langle p, \omega \rangle \) such that \( \pi \) satisfies \( \psi \) under \( B \) (denoted by \( \pi \models^B_\lambda \psi \)). Let \( \text{next}^g_i \), \( \text{next}^a_i \) and \( \text{next}^c_i \) be the global-successor, abstract-successor and caller-successor of \( \langle p_i, \omega_i \rangle \) respectively. Let \( (\pi, i) \) be the suffix of \( \pi \) starting from \( \langle p_i, \omega_i \rangle \). Then, \( \pi \models^B_\lambda \psi \) iff \( (\pi, 0) \models^B_\lambda \psi \) where \( (\pi, i) \models^B_\lambda \psi \) is defined inductively as follows:

- \( (\pi, i) \models^B_\lambda b(\alpha_1, \ldots, \alpha_n) \), iff \( b(B(\alpha_1), \ldots, B(\alpha_n)) \in \lambda(p_i) \)
- \( (\pi, i) \models^B_\lambda \neg b(\alpha_1, \ldots, \alpha_n) \), iff \( b(B(\alpha_1), \ldots, B(\alpha_n)) \notin \lambda(p_i) \)
- \( (\pi, i) \models^B_\lambda e \) iff \( ((p_i, \omega_i), B) \in L(e) \)
- \( (\pi, i) \models^B_\lambda \neg e \) iff \( ((p_i, \omega_i), B) \notin L(e) \)
- \( (\pi, i) \models^B_\lambda \psi_1 \lor \psi_2 \) iff \( (\pi, i) \models^B_\lambda \psi_1 \) or \( (\pi, i) \models^B_\lambda \psi_2 \)
- \( (\pi, i) \models^B_\lambda \psi_1 \land \psi_2 \) iff \( (\pi, i) \models^B_\lambda \psi_1 \) and \( (\pi, i) \models^B_\lambda \psi_2 \)
- \( (\pi, i) \models^B_\lambda X^g \psi \) iff \( (\pi, \text{next}^g_i) \models^B_\lambda \psi \)
- \( (\pi, i) \models^B_\lambda X^a \psi \) iff \( \text{next}^a_i \neq \bot \) and \( (\pi, \text{next}^a_i) \models^B_\lambda \psi \)
- \( (\pi, i) \models^B_\lambda X^c \psi \) iff \( \text{next}^c_i \neq \bot \) and \( (\pi, \text{next}^c_i) \models^B_\lambda \psi \)
- \( (\pi, i) \models^B_\lambda \forall x \psi \) iff for every \( d \in \mathcal{D} \), \( (\pi, i) \models^{B[d]}_\lambda \psi \)
- \( (\pi, i) \models^B_\lambda \exists x \psi \) iff there exists \( d \in \mathcal{D} \), \( (\pi, i) \models^{B[d]}_\lambda \psi \)

- \( (\pi, i) \models^B_\lambda \psi_1 U^v \psi_2 \) (with \( v \in \{ g, a, c \} \)) iff there exists a sequence of positions \( h_0, h_1, \ldots, h_{k-1}, h_k \) where \( h_0 = i \), for every \( 0 \leq j \leq k-1 \): \( h_{j+1} = \text{next}_{h_j}^v, (\pi, h_j) \models^B_\lambda \psi_1 \) and \( (\pi, h_k) \models^B_\lambda \psi_2 \)
- \( (\pi, i) \models^B_\lambda \psi_1 R^v \psi_2 \) (with \( v \in \{ g, a, c \} \)) iff there exists a sequence of positions \( h_0, h_1, \ldots, h_{k-1}, h_k \) where \( h_0 = i \), for every \( 0 \leq j \leq k \): \( h_{j+1} = \text{next}_{h_j}^v, (\pi, h_j) \models^B_\lambda \psi_2 \) and \( (\pi, h_k) \models^B_\lambda \psi_1 \)

Other CARET operators can be represented by the above operators: \( F^g \psi = \text{true} U^g \psi, G^g \psi = \text{false} R^g \psi, F^a \psi = \text{true} U^a \psi, G^a \psi = \text{false} R^a \psi, F^c \psi = \text{true} U^c \psi, G^c \psi = \text{false} R^c \psi, \ldots \)

Let a PCARET formula be an SPCARET formula that does not use any regular variable expression.

CARET with regular valuations is an extension of CARET where the set of configurations where an atomic proposition hold can be expressed by a regular language. Since the domain \( \mathcal{D} \) is finite, we get:
Proposition 2. PCARET and CARET (resp. SPCARET and CARET with regular valuations) have the same expressive power. SPCARET is more expressive than CARET.

Let $\psi$ be a SPCARET formula. The closure of $\psi$, denoted $\text{Cl}(\psi)$, is the smallest set that contains $\psi$ and satisfies the following properties:

- if $X^v\psi' \in \text{Cl}(\psi)$ (with $v \in \{g, a, c\}$), then $\psi' \in \text{Cl}(\psi)$
- if $\forall x \psi' \in \text{Cl}(\psi)$, then $\psi' \in \text{Cl}(\psi)$ and for every $d \in D$, $\psi'_d \in \text{Cl}(\psi)$ where $\psi'_d$ is $\psi'$ in which $x$ is substituted by $d$
- if $\exists x \psi' \in \text{Cl}(\psi)$, then $\psi' \in \text{Cl}(\psi)$ and for every $d \in D$, $\psi'_d \in \text{Cl}(\psi)$ where $\psi'_d$ is $\psi'$ in which $x$ is substituted by $d$
- if $\psi_1 \lor \psi_2 \in \text{Cl}(\psi)$, then $\psi_1 \in \text{Cl}(\psi), \psi_2 \in \text{Cl}(\psi)$
- if $\psi_1 \land \psi_2 \in \text{Cl}(\psi)$, then $\psi_1 \in \text{Cl}(\psi), \psi_2 \in \text{Cl}(\psi)$
- if $\psi_1 U^v\psi_2 \in \text{Cl}(\psi)$ (with $v \in \{g, a, c\}$), then $\psi_1 \in \text{Cl}(\psi), \psi_2 \in \text{Cl}(\psi), X^v(\psi_1 U^v\psi_2) \in \text{Cl}(\psi)$
- if $\psi_1 R^v\psi_2 \in \text{Cl}(\psi)$ (with $v \in \{g, a, c\}$), then $\psi_1 \in \text{Cl}(\psi), \psi_2 \in \text{Cl}(\psi), X^v(\psi_1 R^v\psi_2) \in \text{Cl}(\psi)$

### 3.1.3 Modelling Malicious Behaviours Using SPCARET

In this section, we show how SPCARET can be used to succinctly specify the malicious behaviors presented in Section 2.5.1.

**Open and listen on a specific port:** The CARET formula $\psi_{lp}$ described in Section 2.5.1 can be represented by the SPCARET formula:

$$
\psi'_{lp} = \exists x F^g \left( \text{call(socket)} \wedge X^a(eax = x) \wedge F^a \left( \text{call(bind)} \wedge x \Gamma^* \wedge F^a \left( \text{call(listen)} \wedge x \Gamma^* \right) \right) \right)
$$

$\psi'_{lp}$ expresses that there is a call to the API `socket` with a return value $x$, followed by a call to the function `bind` and a call to the function `listen` with $x$ on top of the stack. Note that in this case, $x$ is the memory address storing the socket descriptor. It can be seen that $\psi'_{lp}$ is much more compact than $\psi_{lp}$.

**Registry Key Injecting:** The CARET formula $\psi_{rk}$ described in Section 2.5.1 can be represented by the SPCARET formula:

$$
\psi'_{rk} = \exists x F^g \left( \text{call(GetModuleFileNameA)} \wedge 0 \Gamma^* \wedge F^a \left( \text{call(RegSetValueExA)} \wedge x \Gamma^* \right) \right)
$$

This formula states that there is a call to the API `GetModuleFileNameA` with 0 and $x$ on the top of the stack (i.e., with 0 and $x$ as parameters), followed by
3.2. SPCARET Model-Checking for Pushdown Systems

3.2.1 Using CARET Model-Checking

We can show that:

**Theorem 6.** Model-checking a PCARET formula against PDSs can be reduced to model-checking a CARET formula against PDSs. Model-checking a SPCARET formula against PDSs can be reduced to model-checking a CARET formula with regular valuations against PDSs.

The reduction underlying this theorem is based on enumerating all possible values for each variable that occurs in the given SPCARET (PCARET) formula. For example, the PCARET formula $\psi = \exists x_1 \exists x_2 \text{push}(x_1) \land X^g \text{push}(x_2)$ where $x_1$ and $x_2$ are variables over the domain $D = \{eax, ebx, ecx, ...\}$ can be rewritten as the huge CARET formula $\bigvee_{d_1, d_2 \in D} \text{push}(d_1) \land X^g \text{push}(d_2)$.

More precisely, we get:

**Proposition 3.** Let $\psi$ be a SPCARET formula, let $|X|$ be the number of variables in $\psi$, let $D$ be the domain of variables, we can compute an equivalent CARET formula with regular valuations $\psi'$ such that $|\psi'| = |\psi| \times O(|D|^{|X|})$.

CARET model checking for PDSs was solved in the previous chapter. From Theorem 3 and Proposition 3, we get:

**Theorem 7.** Given a PDS $\mathcal{P} = (P, \Gamma, \Delta)$, a labeling function $\lambda : P \to 2^{APD}$ and a SPCARET formula $\psi$, for every configuration $\langle p, \omega \rangle$, checking whether $\langle p, \omega \rangle$ satisfies $\psi$ by translating $\psi$ to an equivalent CARET formula $\psi'$ can be solved in time $|P| \cdot |\Delta|^2 \cdot |\Gamma|^2 \cdot 2^O(|\psi||D|^{|X|})$.
3.2.2 **SPCARET\(^c\)**

It is obvious to see that the above approach that consists in translating a SPCARET formula to an equivalent CARET formula is not efficient since the size of the domain \(\mathcal{D}\) is big when the formula specifies a malicious behavior, where \(\mathcal{D}\) is usually all possible register names, or all possible values of the memory addresses or all possible values of the stack. Thus, we need a direct model checking algorithm that does not go through the translation to CARET and that is more efficient than the above approach. One possible idea to have a direct algorithm consists in reducing the model checking problem to the emptiness problem of Symbolic Büchi Pushdown Systems (SBPDSs), by computing a kind of product between the SPCARET formula \(\psi\) and the PDS \(\mathcal{P}\), which gives a Symbolic Büchi Pushdown System. The key idea would be the use of Symbolic BPDSs. This allows to move the complexity of dealing with variables over a big domain to the **symbolic** transitions of the BPDS, which can be efficiently dealt with using BDDs as described in [ES01].

Intuitively, when computing the product, each state of the computed Symbolic BPDS ensures the satisfiability of a certain subformula at some state of the PDS. To be able to apply this approach in the presence of variables, the semantic correctness of a certain subformula at one state is ensured by the semantic correctness of the formulas of its **successor** state. However, this cannot apply for caller-paths since from a state, the correctness of \(X^c, U^c\), and \(R^c\) are ensured backward not forward (i.e., by looking at the predecessors, not the successors). Thus, to be able to apply this idea, we define a subclass of SPCARET that does not involve the \(X^c, U^c\), and \(R^c\) operators. This subclass is called SPCARET\(^c\):

**Definition 8.** A SPCARET\(^c\) (PCARET\(^c\)) formula is a SPCARET (PCARET) formula that does not use the operators \(X^c, U^c\), and \(R^c\).

We believe that these operators \(X^c, U^c\), and \(R^c\) are not useful to specify malicious behaviours. Indeed, a malicious behaviour can often be described as a sequence of API function calls with corresponding register as well as stack values at calls and matching return-points, combined with a sequence of certain assembly instructions (mov, push, pop,...). The operators \(X^g, U^g\), \(R^g\), \(X^a, U^a\), and \(R^a\) are sufficient to express such behaviors.

### 3.3 **PCARET\(^c\)** Model-Checking for Pushdown Systems

In this section, we show how to reduce PDSs model-checking for PCARET\(^c\) to the emptiness problem of Symbolic Büchi Pushdown Systems. The latter
problem is already solved in [ES01].

3.3.1 Symbolic Büchi Pushdown Systems

Definition 9. A Symbolic Pushdown System (SPDS) \( P \) is a tuple \((P, \Gamma, \Delta)\) where \( P \) is a finite set of control locations, \( \Gamma \) is a finite set of stack alphabet and \( \Delta \) is a finite set of symbolic transition rules in the form \( \langle p, \gamma \rangle \xrightarrow{\theta} \langle q, \omega \rangle \) where \( p, q \in P, \gamma \in \Gamma, \omega \in \Gamma^* \) and \( \theta \subseteq B \times B \).

A symbolic transition rule \( \langle p, \gamma \rangle \xrightarrow{\theta} \langle q, \omega \rangle \) represents the set of transition rules: \( \langle (p, B), \gamma \rangle \rightarrow \langle (q, B'), \omega \rangle \) such that \( B, B' \in B \) and \( (B, B') \in \theta \). For every \( \omega' \in \Gamma' \), \( \langle (q, B'), \omega' \rangle \) is an immediate successor of \( \langle (p, B), \gamma \omega' \rangle \). A run of \( P \) starting from \( \langle (p_0, B_0), \omega_0 \rangle \) is a sequence \( \langle (p_0, B_0), \omega_0 \rangle \xrightarrow{(p_1, B_1), \omega_1} ... \xrightarrow{(p_{i+1}, B_{i+1}), \omega_{i+1}} \) such that \( (p_{i+1}, B_{i+1}, \omega_{i+1}) \) is an immediate successor of \( (p_i, B_i, \omega_i) \) for every \( i \geq 0 \).

Definition 10. A Symbolic Büchi Pushdown System (SBPDS) is a tuple \((P, \Gamma, \Delta, F)\), where \((P, \Gamma, \Delta)\) is a SPDS and \( F \subseteq P \) is a set of accepting control locations. A run of a SBPDS is accepting if it visits infinitely often some control locations in \( F \).

Definition 11. A Generalized Symbolic Büchi Pushdown System (GSBPDS) is a tuple \((P, \Gamma, \Delta, F)\), where \((P, \Gamma, \Delta)\) is a SPDS and \( F = \{F_1, ..., F_k\} \) is a set of sets of accepting control locations. A run of a GSBPDS is accepting if it visits infinitely often some control locations in \( F_i \) for every \( 1 \leq i \leq k \).

Let \( BP \) be a SBPDS (resp. GSBPDS), \( L(BP) \) is the set of configurations \( \langle (p, B), \omega \rangle \in P \times B \times \Gamma^* \) such that \( BP \) has an accepting run from \( \langle p, \omega \rangle \). We have the following properties:

Proposition 4. [ST13a] Given a GSBPDS \( BP \), we can compute a SBPDS \( BP' \) s.t. \( L(BP) = L(BP') \).

Theorem 8. [ES01, ST13a] Given a SBPDS \( BP = (P, \Gamma, \Delta, F) \), for every configuration \( \langle (p, B), \omega \rangle \in P \times B \times \Gamma^* \), whether or not \( \langle (p, B), \omega \rangle \) is in \( L(BP) \) can be decided in time \( O(|P| \cdot |\Delta|^2 \cdot |D|^{3|X|}) \).

3.3.2 From PCARET^c Model-Checking for PDSs to the Emptiness Problem of SBPDSs

Let \( P = (P, \Gamma, \Delta) \) be a PDS, \( \lambda : P \rightarrow 2^{AP} \) be a labelling function, \( \psi \) be a PCARET^c formula. In this section, we show how to build a Generalized Symbolic Büchi Pushdown System \( BP_\psi \) s.t. \( P \) has an execution \( \pi \) from \( \langle p, \omega \rangle \) s.t. \( \pi \) satisfies \( \psi \) under \( B \) iff \( BP_\psi \) has an accepting run from
\(\langle \langle p, \{v\}, \text{unexit} \rangle, B, \omega \rangle\) where unexit is a label expressing that from the configuration \(\langle p, \omega \rangle\), the execution of the procedure of \(\langle p, \omega \rangle\), \(\mathcal{P}(\langle p, \omega \rangle)\), in \(\pi\) is never finished (since \(\pi\) is an infinite run and \(\langle p, \omega \rangle\) is the initial configuration of \(\pi\)). Let \(\text{Label} = \{\text{exit}, \text{unexit}\}\).

We define \(\mathcal{BP}_\phi = (P', \Gamma', \Delta', F)\) as follows:

- \(P' = P \times 2^{\text{Cl}(\psi)} \times \text{Label}\)
- \(\Gamma' = \Gamma \cup (\Gamma \times 2^{\text{Cl}(\psi)} \times \text{Label})\) is the finite set of stack symbols of \(\mathcal{BP}_\phi\).

\(\Delta'\) is the smallest set of transition rules defined as follows\(^1\): for every \(\Phi \subseteq \text{Cl}(\psi), p \in P, \gamma \in \Gamma; l, l' \in \text{Label}\):

\(\beta_1\) if \(\phi = b(\alpha_1, \ldots, \alpha_n) \in \Phi\), then, \(\langle \langle p, \Phi, \emptyset l \rangle, \gamma \rangle \xrightarrow{\theta} \langle \langle p, \Phi \setminus \{\phi\}, \emptyset l, \gamma \rangle \in \Delta'\)
where \(\theta = \{(B, B) \mid B \in \mathcal{B} \land b(B(\alpha_1), \ldots, B(\alpha_n)) \notin \lambda(p)\}\)

\(\beta_2\) if \(\phi = \neg b(\alpha_1, \ldots, \alpha_n) \in \Phi\), then, \(\langle \langle p, \Phi, \emptyset l \rangle, \gamma \rangle \xrightarrow{\theta} \langle \langle p, \Phi \setminus \{\phi\}, \emptyset l, \gamma \rangle \in \Delta'\)
where \(\theta = \{(B, B) \mid B \in \mathcal{B} \land b(B(\alpha_1), \ldots, B(\alpha_n)) \notin \lambda(p)\}\)

\(\beta_3\) if \(\phi = \phi_1 \land \phi_2 \in \Phi\), then, \(\langle \langle p, \Phi, \emptyset l \rangle, \gamma \rangle \xrightarrow{\theta_{id}} \langle \langle p, \Phi \setminus \{\phi\} \cup \{\phi_1, \phi_2\}, \emptyset l, \gamma \rangle \in \Delta'\)

\(\beta_4\) if \(\phi = \phi_1 \lor \phi_2 \in \Phi\), then, \(\langle \langle p, \Phi, \emptyset l \rangle, \gamma \rangle \xrightarrow{\theta_{id}} \langle \langle p, \Phi \setminus \{\phi\} \cup \{\phi_1, \phi_2\}, \emptyset l, \gamma \rangle \in \Delta'\)
and \(\langle \langle p, \Phi, \emptyset l \rangle, \gamma \rangle \xrightarrow{\theta_{id}} \langle \langle p, \Phi \setminus \{\phi\} \cup \{\phi_2, \phi_1\}, \emptyset l, \gamma \rangle \in \Delta'\)

\(\beta_5\) if \(\phi = \exists x \phi' \in \Phi\), then:

\(\beta_{5,1}\) if \(x\) is not a free variable of any formula in \(\Phi\), then, \(\langle \langle p, \Phi, \emptyset l \rangle, \gamma \rangle \xrightarrow{\theta_{id}} \langle \langle p, \Phi \cup \{\phi'\} \setminus \{\phi\}, \emptyset l, \gamma \rangle \in \Delta'\)

\(\beta_{5,2}\) otherwise, for every \(c \in \mathcal{D}\), \(\langle \langle p, \Phi, \emptyset l \rangle, \gamma \rangle \xrightarrow{\theta_{id}} \langle \langle p, \Phi \cup \{\phi'_c\} \setminus \{\phi\}, \emptyset l, \gamma \rangle \in \Delta'\) where \(\phi'_c\) is \(\phi'\) such that \(x\) is substituted by \(c\).

\(\beta_6\) if \(\phi = \forall x \phi' \in \Phi\), then, \(\langle \langle p, \Phi, \emptyset l \rangle, \gamma \rangle \xrightarrow{\theta_{id}} \langle \langle p, \Phi \cup \{\phi_c' \mid c \in \mathcal{D}\} \setminus \{\phi\}, \emptyset l, \gamma \rangle \in \Delta'\) where \(\phi'_c\) is \(\phi'\) such that \(x\) is replaced by \(c\).

\(\beta_7\) if \(\phi = \phi_1 \cup v \phi_2 \in \Phi \ (v \in \{g, a\})\), then, \(\langle \langle p, \Phi, \emptyset l \rangle, \gamma \rangle \xrightarrow{\theta_{id}} \langle \langle p, \Phi \cup \{\phi_2\} \setminus \{\phi\}, \emptyset l, \gamma \rangle \in \Delta'\)
and \(\langle \langle p, \Phi, \emptyset l \rangle, \gamma \rangle \xrightarrow{\theta_{id}} \langle \langle p, \Phi \cup \{\phi_1, X^v \phi\} \setminus \{\phi\}, \emptyset l, \gamma \rangle \in \Delta'\)

\(\beta_8\) if \(\phi = \phi_1 \cup v \phi_2 \in \Phi \ (v \in \{g, a\})\), then, \(\langle \langle p, \Phi, \emptyset l \rangle, \gamma \rangle \xrightarrow{\theta_{id}} \langle \langle p, \Phi \cup \{\phi_2\} \setminus \{\phi\}, \emptyset l, \gamma \rangle \in \Delta'\)
and \(\langle \langle p, \Phi, \emptyset l \rangle, \gamma \rangle \xrightarrow{\theta_{id}} \langle \langle p, \Phi \cup \{\phi_1, X^v \phi\} \setminus \{\phi\}, \emptyset l, \gamma \rangle \in \Delta'\)

\(^1\theta_{id}\) and \(\theta_{id}\) are as defined in Section 3.1.1
3.3. PCARETv Model-Checking for Pushdown Systems

(β₀) if \( \Phi = \Phi_g \cup \Phi_a \) where \( \Phi_g = \{\langle Y^g \phi_1, ..., Y^g \phi_n \rangle \}, \Phi_a = \{\langle X^a \phi_1, ..., X^a \phi_m \rangle \} \) (\( \Phi_g \) or \( \Phi_a \) can be empty), then:

(β₀₁) for every \( \langle p, \gamma \rangle \xrightarrow{\text{call}} \langle q, \gamma' \gamma'' \rangle \in \Delta' \):
\[
\langle \langle p, \gamma \rangle, \gamma' \rangle \xrightarrow{\theta_\text{call}} \langle \langle q, \{\phi_1, ..., \phi_n\}, l' \rangle, \gamma' \gamma'' \rangle \in \Delta' \text{ iff } (l' = \text{unexit}) \text{ implies } (l = \text{unexit} \text{ and } \Phi_a = \emptyset)
\]

(β₀₂) for every \( \langle p, \gamma \rangle \xrightarrow{\text{ret}} \langle q, \varepsilon \rangle \in \Delta' \):
\[
\begin{align*}
(β₀₂₁) & \quad \langle \langle p, \Phi, \text{exit} \rangle, \gamma \rangle \xrightarrow{\theta_\text{exit}} \langle \langle q, \{\phi_1, ..., \phi_n\}, l' \rangle, \varepsilon \rangle \in \Delta' \text{ iff } \Phi_a = \emptyset \\
(β₀₂₂) & \quad \langle \langle q, \{\phi_1, ..., \phi_n\}, l' \rangle, l_0 \rangle, \gamma_0 \rangle \in \Delta' \text{ for every } \gamma_0 \in \Gamma \text{ iff } l_0 = l'
\end{align*}
\]

(β₀₃) for every \( \langle p, \gamma \rangle \xrightarrow{\text{int}} \langle q, \omega \rangle \in \Delta' \):
\[
\langle \langle p, \Phi, \gamma \rangle \xrightarrow{\theta_\text{int}} \langle \langle q, \{\phi_1, ..., \phi_n\}, \{\phi_1, ..., \phi_n\}, l \rangle, \omega \rangle \in \Delta'
\]

\[
\{X^g \phi_1, ..., X^g \phi_n, X^a \phi_1, ..., X^a \phi_m \} \quad \text{encoded \& passed down} \quad \{\gamma''', \{\phi_1, ..., \phi_m\} \}
\]

Figure 3.1: \( \langle p_i, \omega_i \rangle \Rightarrow_p \langle p_{i+1}, \omega_{i+1} \rangle \) corresponds to a call statement

Let \( c\{U^g(\psi) = \{\xi_1 U^a \tau_1, ..., \xi_k U^a \tau_k \} \) \) and \( c\{U^a(\psi) = \{\xi_1 U^a \tau_1, ..., \xi_k U^a \tau_k \} \) \) be the set of \( U^g \)-formulas and \( U^a \)-formulas of \( C\{\psi \) \) respectively. The generalized Büchi accepting condition \( F \) of \( BP_\psi \) is defined as: \( F = \{F_1\} \cup F_2 \cup F_3 \) where

- \( F_1 = P \times 2^{C\{\psi \)} \times \{\text{unexit}\}
- \( F_2 = \{F^g_1, ..., F^g_k\} \) where \( F^g_i = P \times F_{\phi_i U^g \chi_i} \times \text{Label} \) where \( F_{\phi_i U^g \chi_i} = \{\Phi \subseteq C\{\psi \} | \phi_i U^g \chi_i \notin \Phi \text{ and } X^g(\phi_i U^g \chi_i) \notin \Phi \} \) for every \( 1 \leq i \leq k \).
- \( F_3 = \{F^a_1, ..., F^a_k\} \) where \( F^a_i = P \times F_{\xi_i U^a \tau_i} \times \{\text{unexit}\} \) where \( F_{\xi_i U^a \tau_i} = \{\Phi \subseteq C\{\psi \} | \xi_i U^a \tau_i \notin \Phi \text{ and } X^a(\xi_i U^a \tau_i) \notin \Phi \} \) for every \( 1 \leq i \leq k' \).

Intuition. Roughly speaking, we construct \( BP_\psi \) as a kind of product between \( P \) and \( \psi \) which ensures that \( BP_\psi \) has an accepting run from \( \langle \langle p, \{\psi\}, \text{unexit} \rangle, B \rangle, \omega \rangle \) iff \( P \) has an execution \( \pi \) starting at \( \langle p, \omega \rangle \) s.t. \( \pi \) satisfies \( \psi \) under \( B \). The form of the control locations of \( BP_\psi \) is \( \langle p, \Phi, \ell \rangle \) where \( \Phi \) is a set of formulas that are satisfied (under \( B \)) at the configuration \( \langle p, \omega \rangle \),
l is a label to determine whether the execution of the procedure of \(\langle p, \omega \rangle\), \(P(p, \omega)\) (as defined in Section 2.1), can be terminated on \(\pi\). A configuration \(\langle p, \omega \rangle\) is labeled with \textit{exit} means that the execution of \(P(p, \omega)\) is finished in \(\pi\), i.e., the run \(\pi\) will run through the procedure \(P(p, \omega)\), reaches its ret statement and exits \(P(p, \omega)\) after that. On the contrary, \(\langle p, \omega \rangle\) is labeled with \textit{unexit} means that in \(\pi\), the execution of the procedure \(P(p, \omega)\) never terminates, i.e., the run \(\pi\) will be stuck in and never exits the procedure \(P(p, \omega)\). Let \(\pi = \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \ldots\) be a run of \(P\). Let us write \(\pi \models^B \Phi\) to express that \(\pi\) satisfies all formulas \(\phi \in \Phi\) under \(B\). To obtain such a \(B \Phi\), intuitively, we proceed as follows:

- If \(b(\alpha_1, \ldots, \alpha_n) \in \Phi\), then \(\pi \models^{B} \Phi\) if \(\pi\) satisfies \(b(\alpha_1, \ldots, \alpha_n)\) and \(\pi\) satisfies all the remaining formulas in \(\Phi\) under \(B\). This is ensured by the transition rules in \((\beta_1)\) stating that \(B \Phi\) has an accepting run from \(\langle\langle p, \Phi, l \rangle, B, \omega \rangle\) iff \(b(B(\alpha_1), \ldots, B(\alpha_n)) \in \lambda(p)\) and \(B \Phi\) has an accepting run from \(\langle\langle p, \Phi \setminus \{b(\alpha_1, \ldots, \alpha_n)\}, l \rangle, B, \omega \rangle\).

- If \(\neg b(\alpha_1, \ldots, \alpha_n) \in \Phi\), then \(\pi \models^B \Phi\) if \(\pi\) satisfies \(\neg b(\alpha_1, \ldots, \alpha_n)\) and \(\pi\) satisfies all the remaining formulas in \(\Phi\) under \(B\). This is ensured by the transition rules in \((\beta_2)\) stating that \(B \Phi\) has an accepting run from \(\langle\langle p, \Phi, l \rangle, B, \omega \rangle\) iff \(b(B(\alpha_1), \ldots, B(\alpha_n)) \notin \lambda(p)\) and \(B \Phi\) has an accepting run from \(\langle\langle p, \Phi \setminus \neg b(\alpha_1, \ldots, \alpha_n) \setminus \lambda(p) \rangle, l, B, \omega \rangle\).

- If \(\phi_1 \land \phi_2 \in \Phi\), then \(\pi \models^{B} \Phi\) if \(\pi\) satisfies \(\phi_1 \land \phi_2\) and \(\pi\) satisfies all the remaining formulas in \(\Phi\) under \(B\). Note that \(\pi \models^B \phi_1 \land \phi_2\) iff \(\pi \models^B \phi_1\) and \(\pi \models^B \phi_2\). This is ensured by the transition rules in \((\beta_3)\) stating that \(B \Phi\) has an accepting run from \(\langle\langle p, \Phi, l \rangle, B, \omega \rangle\) iff \(B \Phi\) has an accepting run from \(\langle\langle p, \Phi \cup \{\phi_1, \phi_2\} \setminus \{\phi_1 \land \phi_2\}, l, B, \omega \rangle\). Item \((\beta_3)\) is similar to \((\beta_3)\)

- If \(\forall x \phi' \in \Phi\), then \(\pi \models^{B} \Phi\) if \(\pi\) satisfies \(\forall x \phi'\) and \(\pi\) satisfies all the remaining formulas in \(\Phi\) under \(B\). Note that \(\pi \models^B \forall x \phi'\) iff \(\pi \models^B \wedge_{x \in D} \phi'_c\) where \(\phi'_c\) is \(\phi'\) in which \(x\) is replaced by \(c\). Thus, this is ensured by the transition rules in \((\beta_4)\) stating that \(B \Phi\) has an accepting run from \(\langle\langle p, \Phi, l \rangle, B, \omega \rangle\) iff \(B \Phi\) has an accepting run from \(\langle\langle p, \Phi \cup \{\phi'_c | c \in D \setminus \{x \phi'\}, l, B, \omega \rangle\).\)

- If \(\exists x \phi' \in \Phi\), then \(\pi \models^B \Phi\) if \(\pi \models^B \exists x \phi'\) and \(\pi \models^B \Phi \setminus \{\exists x \phi'\}\). In other words, \(\pi \models^B \Phi\) if there exists \(c \in D\) s.t. \(\pi \models^B \phi'\) and \(\pi \models^B \Phi \setminus \{\exists x \phi'\}\). We consider two possibilities:
  - if \(x\) is not a free variable of any formula in \(\Phi\), then \(\pi \models^B \Phi \setminus \{\exists x \phi'\}\)
    \(\pi \models^B \Phi \setminus \{\exists x \phi'\}\) for every \(c \in D\). This means that \(\pi \models^B \Phi\) iff
3.3. PCARE\textsuperscript{tc} Model-Checking for Pushdown Systems

Figure 3.2: $\langle p_i, \omega_i \rangle \Rightarrow_p \langle p_{i+1}, \omega_{i+1} \rangle$ corresponds to a simple statement

there exists $c \in D$ s.t. $\pi \models_{B[x\leftarrow c]} \phi'$ and $\pi \models_{B[x\leftarrow c]} \Phi \setminus \{\exists x\phi'\}$. This is ensured by the transition rules in $(\beta_{5,1})$ stating that $BP_\psi$ has an accepting run from $\langle(\langle p, \Phi, l \rangle, B), \omega \rangle$ iff there exists $c \in D$ s.t. $BP_\psi$ has an accepting run from $\langle(\langle p, \Phi \cup \{\phi'\} \setminus \{\exists x\phi'\}, l \rangle, B[x \leftarrow c]), \omega \rangle$ (as $(B, B[x \leftarrow c]) \in \theta_x$)

if $x$ is a free variable of some formula in $\Phi$, then, it may occur the case that $\phi'$ is satisfied only when $x = c (\pi \models_{B[x\leftarrow c]} \phi')$, $\phi''$ is not satisfied when $x = c (\pi \not\models_{B[x\leftarrow c]} \phi'')$, while $\pi \models_{B[x\leftarrow c]} \{\exists x\phi', \phi''\}$. Thus, we cannot apply the transition rules in $(\beta_{5,1})$ for this case. Note that $\pi \models_{B[x\leftarrow c]} \Phi$ iff there exists $c \in D$ s.t. $\pi \models_{B[x\leftarrow c]} \phi'$ and $\pi \models_{B[x\leftarrow c]} \Phi \setminus \{\exists x\phi'\}$. This is ensured by the transition rule $(\beta_{5,2})$ stating that $BP_\psi$ has an accepting run from $\langle(\langle p, \Phi, l \rangle, B), \omega \rangle$ iff there exists $c \in D$ s.t. $BP_\psi$ has an accepting run from $\langle(\langle p, \Phi \cup \{\phi'\} \setminus \{\exists x\phi'\}, l \rangle, B), \omega \rangle$ (since $(B, B) \in \theta_{id}$).

- If $\phi_1 U^r \phi_2 \in \Phi$, then $\pi \models_{B} \Phi$ iff $\pi \models_{B} \phi_1 U^r \phi_2$ and $\pi \models_{B} \Phi \setminus \{\phi_1 U^r \phi_2\}$. Note that $\pi \models_{B} \phi_1 U^r \phi_2$ iff $\pi \models_{B} \phi_2$ or ($\pi \models_{B} \phi_1$ and $\pi \models_{B} X^r(\phi_1 U^r \phi_2)$). This is ensured by the transition rules in $(\beta_7)$ stating that $BP_\psi$ has an accepting run from $\langle(\langle p, \Phi, l \rangle, B), \omega \rangle$ iff either $BP_\psi$ has an accepting run from $\langle(\langle p, \Phi \cup \{\phi_2\} \setminus \{\phi_1 U^r \phi_2\}, l \rangle, B), \omega \rangle$, or $BP_\psi$ has an accepting run from $\langle(\langle p, \Phi \cup \{\phi_1, \phi_1 X^r(\phi_1 U^r \phi_2)\} \setminus \{\phi_1 U^r \phi_2\}, l \rangle, B), \omega \rangle$. Item $(\beta_8)$ is similar to $(\beta_7)$.

- If $\Phi = \{X^q \phi_1, ..., X^q \phi_n, X^q \phi_1, ..., X^q \phi_m\}$. Let $\langle p_k, \omega_k \rangle$ be the abstract-successor of $\langle p_i, \omega_i \rangle$, then, ($\pi, i) \models_{B} \Phi$ iff ($\pi, i + 1) \models_{B} \{\phi_1, ..., \phi_n\}$ and ($\pi, k) \models_{B} \{\phi_1, ..., \phi_m\}$). Now we show how we can ensure these:

$$(\pi, i + 1) \models_{B} \{\phi_1, ..., \phi_m\}$$
The labels. Now, let us explain how the label $l$ is used in the transition rules to ensure the correctness of the formulas. Note that our explanation

- To ensure $(\pi, k) \models B \{\varphi_1, \ldots, \varphi_m\}$ There are two possibilities:
  
  * If $\langle p_i, \omega_i \rangle \Rightarrow \langle p_{i+1}, \omega_{i+1} \rangle$ corresponds to a call statement.
  
  Let us consider Figure 3.1 to explain this case. $(\pi, i) \models B \{X^a \varphi_1, \ldots, X^a \varphi_m\}$ if $(\pi, k) \models \lambda \varphi_1, \ldots, \varphi_m \}$. This is ensured by rules $(\beta_{0.1})$ and $(\beta_{0.2})$: rules $(\beta_{0.1})$ allow to record $\varphi_1, \ldots, \varphi_m$ in the return point of the call, and rules $(\beta_{0.2})$ allow to extract and validate $\varphi_1, \ldots, \varphi_m$ when the return-point is reached. In what follows, we show in more details how this works: Let $\langle p_i, \gamma \rangle \xrightarrow{\text{call}} \langle p_{i+1}, \gamma' \gamma'' \rangle$ be the rule associated with the transition $\langle p_i, \omega_i \rangle \Rightarrow \langle p_{i+1}, \omega_{i+1} \rangle$, then we have $\omega_i = \gamma \omega'$ and $\omega_{i+1} = \gamma' \gamma'' \omega'$. Let $\langle p_{k-1}, \omega_{k-1} \rangle \Rightarrow \langle p_k, \omega_k \rangle$ be the transition that corresponds to the $\text{return}$ statement of this call. Let then $\langle p_{k-1}, \beta \rangle \xrightarrow{\text{return}} \langle p_k, \varepsilon \rangle \in \Delta$ be the corresponding return rule. Then, we have necessarily $\omega_{k-1} = \beta \gamma'' \omega'$, since as explained in Section 2.2.1, $\gamma''$ is the return address of the call. After applying this rule, $\omega_k = \gamma'' \omega'$. In other words, $\gamma''$ will be the topmost stack symbol at the corresponding return point of the call. So, in order to ensure that $(\pi, k) \models B \{\varphi_1, \ldots, \varphi_m\}$, we proceed as follows: At the call $\langle p_i, \gamma \rangle \xrightarrow{\text{call}} \langle p_{i+1}, \gamma' \gamma'' \rangle$, we encode formulas which are required to be true at the corresponding return-point of the call $\{\varphi_1, \ldots, \varphi_m\}$ into $\gamma''$ by the rule $(\beta_{0.1})$ stating that $\langle \langle p_i, \{X^a \varphi_1, \ldots, X^a \varphi_n, X^a \varphi_1, \ldots, X^a \varphi_m\} \rangle, \gamma \rangle \xrightarrow{\beta_{0.1}} \langle \langle p_{i+1}, \{\varphi_1, \ldots, \varphi_n\} \rangle, \gamma' \gamma'' \rangle \in \Delta'$. This allows to record $\varphi_1, \ldots, \varphi_m$ in the corresponding return point of the stack. After that, $\{\gamma'', \varphi_1, \ldots, \varphi_m\}$ will be the topmost stack symbol at the corresponding return-point of this call. At the return-point, the transition rule $(\beta_{0.2})$ ensures that $(\pi, k) \models B \varphi_1, \ldots, \varphi_m \} by adding $\varphi_1, \ldots, \varphi_m$ to the set of formulas which are required to be satisfied at $\langle p_k, \omega_k \rangle$.

* If $\langle p_i, \omega_i \rangle \Rightarrow \langle p_{i+1}, \omega_{i+1} \rangle$ corresponds to a simple statement (see Figure 3.2). Then, the abstract successor of $\langle p_i, \omega_i \rangle$ is $\langle p_{i+1}, \omega_{i+1} \rangle$. Thus, we must have $(\pi, i+1) \models \lambda \varphi_1, \ldots, \varphi_m \}$. This is ensured by the rules in $(\beta_{0.3})$ stating that $B \varphi_1, \ldots, \varphi_m \} has an accepting run from $\langle \langle p_i, \Phi_i \rangle, \omega_i \rangle \xrightarrow{B} \varphi_1, \ldots, \varphi_m \} \rangle \xrightarrow{B} \varphi_1, \ldots, \varphi_m \} has an accepting run from $\langle \langle p_{i+1}, \{\varphi_1, \ldots, \varphi_n, \varphi_1, \ldots, \varphi_m\} \rangle, B_{i+1}, \omega_{i+1} \rangle$. 

The labels. Now, let us explain how the label $l$ is used in the transition rules to ensure the correctness of the formulas. Note that our explanation
above makes implicitly the assumption that along the run $\pi$, every call to a procedure $\text{proc}$ will eventually reach its corresponding return point, i.e., the run $\pi$ will finally exit $\text{proc}$, then, we can encode formulas at the call and validate them at its corresponding return-point. However, it might be the case that at a certain point in the procedure $\text{proc}$, there will be a loop, and $\pi$ never exits $\text{proc}$. To solve this problem, we annotate the control states by the label $l \in \{\text{exit}, \text{unexit}\}$ to determine whether $\pi$ can complete the execution of the procedure $\mathcal{P}(p, \omega)$. In the following, we explain three cases corresponding to three kinds of statements:

- Let us consider Figure 3.1. $\langle p_i, \omega_i \rangle \Rightarrow p \langle p_{i+1}, \omega_{i+1} \rangle$ corresponds to a call statement. Note that $\mathcal{P}(\langle p_{i+1}, \omega_{i+1} \rangle) = \text{proc}$ in this case. There are two possibilities. If $\text{proc}$ terminates, then the call at $\langle p_i, \omega_i \rangle$ will reach its corresponding return-point. In this case, $\langle p_{i+1}, \omega_{i+1} \rangle$ is labelled by $\text{exit}$. If $\text{proc}$ never terminates, then the call at $\langle p_i, \omega_i \rangle$ will never reach its corresponding return-point. In this case, $\langle p_{i+1}, \omega_{i+1} \rangle$ is labelled by $\text{unexit}$. If $\langle p_{i+1}, \omega_{i+1} \rangle$ is labelled by $\text{exit}$, then $\langle p_i, \omega_i \rangle$ can be labelled by $\text{exit}$ or $\text{unexit}$. However, if $\langle p_{i+1}, \omega_{i+1} \rangle$ is labelled by $\text{unexit}$, then $\langle p_i, \omega_i \rangle$ must be labelled by $\text{unexit}$. This is ensured by the condition ($l' = \text{unexit}$ implies $l = \text{unexit}$) in the rule $(\beta_{0.1})$. In addition, if $\langle p_{i+1}, \omega_{i+1} \rangle$ is labelled by $\text{unexit}$, then $\langle p_i, \omega_i \rangle$ never reaches its corresponding return-point. Thus, $\langle p_i, \omega_i \rangle$ does not satisfy any formula in the form $X^a \phi$. This is ensured by the condition ($l' = \text{unexit}$ implies $\Phi_a = \emptyset$) in the rule $(\beta_{0.1})$.

- Again, let us consider Figure 3.1. $\langle p_{k-1}, \omega_{k-1} \rangle \Rightarrow p \langle p_k, \omega_k \rangle$ corresponds to a ret statement. At $\langle p_{k-1}, \omega_{k-1} \rangle$, we are sure that $\text{proc}$ will terminate. In this case, $\langle p_{k-1}, \omega_{k-1} \rangle$ must be always labelled by $\text{exit}$ and $\langle p_k, \omega_k \rangle$ can be labelled by $\text{exit}$ or $\text{unexit}$. This is ensured by the rule $(\beta_{9, 2.1})$. Also, the abstract-successor of $\langle p_{k-1}, \omega_{k-1} \rangle$ is $\bot$, then, $\langle p_{k-1}, \omega_{k-1} \rangle$ does not satisfy any formula in the form $X^a \phi$. This is ensured by the condition ($\Phi_a = \emptyset$) in the rule $(\beta_{9, 2.1})$.

- Finally, let us consider Figure 3.2. $\langle p_i, \omega_i \rangle \Rightarrow p \langle p_{i+1}, \omega_{i+1} \rangle$ corresponds to a simple statement. Then, $\langle p_i, \omega_i \rangle$ and $\langle p_{i+1}, \omega_{i+1} \rangle$ are in the same procedure $\text{proc}$. Thus, the labels assigned to $\langle p_i, \omega_i \rangle$ and $\langle p_{i+1}, \omega_{i+1} \rangle$ should be the same. This is ensured by the transition rule $(\beta_{9, 3})$.

The Büchi accepting condition. The generalized Büchi accepting condition $F$ of $BP_\psi$ consists of three families of accepting conditions $F_1$, $F_2$ and $F_3$. The first set $F_1$ guarantees that an accepting run should go infinitely often through the label $\text{unexit}$. The sets $F_2$ and $F_3$ ensure the liveness requirements of until-formulas on the infinite global path and the infinite abstract path:
Each set of $F_2$ ensures that the liveness requirement $\phi_2$ in $\phi_1 U^g \phi_2$ is eventually satisfied in $P$. Note that $(\pi, i) \models^B \phi_1 U^g \phi_2$ iff $(\pi, i) \models^B \phi_2$ or $((\pi, i) \models^B \phi_1$ and $(\pi, i) \models^B X^g(\phi_1 U^g \phi_2))$. Because $\phi_2$ should hold eventually, to avoid the case where the run of $BP_\psi$ always follow the latter and never reaches $\phi_2$, we set $P \times \{\Phi \subseteq Cl(\psi) \mid \phi_1 U^g \phi_2 \notin \Phi \text{ and } X^g(\phi_1 U^g \phi_2) \notin \Phi\} \times \text{Label}$ as an accepting set. By this setting, the accepting run of $BP_\psi$ will infinitely often visit some control locations in $P \times \{\Phi \subseteq Cl(\psi) \mid \phi_1 U^g \phi_2 \notin \Phi \text{ and } X^g(\phi_1 U^g \phi_2) \notin \Phi\} \times \text{Label}$ which ensures that $\phi_2$ will eventually hold.

The idea behind the set $F_3$ is similar to the set $F_2$ except that the liveness requirement for a $U^a$-formula $\phi_1 U^a \phi_2$ is only required on the infinite abstract path (labelled by $unexit$).

**Finite abstract paths.** The liveness requirements of abstract-until formulas on finite abstract paths are ensured by conditions in transition rules:

- The liveness requirements of abstract-until formulas on finite abstract paths $\langle p_{z_0}, \omega_{z_0} \rangle \langle p_{z_1}, \omega_{z_1} \rangle \cdots \langle p_{z_m}, \omega_{z_m} \rangle$ where $\langle p_{z_m}, \omega_{z_m} \rangle$ is associated with a tag $t_{z_m} = \text{return}$ are ensured by the condition $\phi_a = \emptyset$ in the transition rule $(\beta_{a,2,1})$. This requirement guarantees the liveness requirement $\phi_2$ in $\phi_1 U^a \phi_2$ eventually happens. Look at Figure 3.1 for an illustration. In this figure, for every $i + 1 \leq u \leq k - 1$, the abstract path starting from $\langle p_u, \omega_u \rangle$ is finite. Suppose that we want to determine whether $(\pi, k - 1) \models^B \{\phi_1 U^a \phi_2\}$, then, we get that $(\pi, k - 1) \models^B \phi_1 U^a \phi_2$ iff $(\pi, k - 1) \models^B \phi_2$ or $((\pi, k - 1) \models^B \phi_1$ and $(\pi, k - 1) \models^B X^a(\phi_1 U^a \phi_2))$. In other words, $(\pi, k - 1) \models^B \{\phi_1 U^a \phi_2\}$ iff $(\pi, k - 1) \models^B \{\phi_2\}$ or $(\pi, k - 1) \models^B \{\phi_1, X^a(\phi_1 U^a \phi_2)\}$. Since $\phi_2$ should eventually hold, $\phi_2$ should hold at $\pi(k - 1)$ because $next_{k-1}^a \subseteq \bot$. To ensure this, we require that $\phi_a = \emptyset$ at return statements in the transition rule $(\beta_{a,2,1})$. $\phi_a = \emptyset$ will ensure that the case $(\pi, k - 1) \models^B \{\phi_1 U^a \phi_2\}$ if $(\pi, k - 1) \models^B \{\phi_2\}$ occurs instead of $(\pi, k - 1) \models^B \{\phi_1 U^a \phi_2\}$ if $(\pi, k - 1) \models^B \{\phi_1, X^a(\phi_1 U^a \phi_2)\}$; which means that $(\pi, k - 1) \models^B \phi_2$ and $\phi_2$ eventually holds.
3.3. PCARET\^c Model-Checking for Pushdown Systems

- The liveness requirements of abstract-until formulas on finite abstract paths \( \langle p_{z_0}, \omega_{z_0} \rangle \langle p_{z_1}, \omega_{z_1} \rangle \ldots \langle p_{z_m}, \omega_{z_m} \rangle \) where \( \langle p_{z_m}, \omega_{z_m} \rangle \) is associated with a call \( t_{z_m} = \text{call} \) but this call never reaches its corresponding return-point are ensured by the condition \( \phi_a = \emptyset \) in the transition rule (\( \beta_{9,1} \)). This requirement guarantees the liveness requirement \( \phi_2 \) in \( \phi_1 U^a \phi_2 \) eventually happens. Look at Figure 3.3 for an illustration. In this figure, for every \( 0 \leq u \leq i \), the abstract path starting from \( \langle p_u, \omega_u \rangle \) is finite. Suppose that we want to determine whether \( (\pi, i) \models_B \{ \phi_1 U^a \phi_2 \} \), then, we get that \( (\pi, i) \not\models \phi_1 U^a \phi_2 \) if \( (\pi, i) \not\models \phi_1 \) or \( (\pi, i) \not\models \phi_2 \) or \( (\pi, i) \models \phi_1 \) and \( (\pi, i) \models X^a(\phi_1 U^a \phi_2) \). In other words, \( (\pi, i) \models_B \{ \phi_1 U^a \phi_2 \} \) iff \( (\pi, i) \not\models \phi_2 \) or \( (\pi, i) \models \phi_1 \) and \( (\pi, i) \models X^a(\phi_1 U^a \phi_2) \). Since \( \phi_2 \) should eventually hold, \( \phi_2 \) should hold at \( \pi(i) \) because \( \text{next}_a^u = \emptyset \). To ensure this, we require that \( \phi_a = \emptyset \) in the transition rule (\( \beta_{9,1} \)). \( \phi_a = \emptyset \) will ensure that the case \( (\pi, i) \models_B \{ \phi_1 U^a \phi_2 \} \) if \( (\pi, i) \not\models B \{ \phi_2 \} \) occurs instead of \( (\pi, i) \models_B \{ \phi_1, X^a(\phi_1 U^a \phi_2) \} \); which means that \( (\pi, i) \models_B \{ \phi_2 \} \) and \( \phi_2 \) eventually holds.

Thus, we can show that:

**Theorem 9.** Given a PDS \( \mathcal{P} = (P, \Gamma, \Delta) \), a labeling function \( \lambda : P \rightarrow 2^{AP^r} \), and a PCARET\(^c\) formula \( \psi \), we can construct a GSBPDS \( \mathcal{BP}_\psi = (P', \Gamma', \Delta', F) \) such that for every configuration \( \langle p, \omega \rangle \in P \times \Gamma^* \) and every \( B \in \mathcal{B} \), \( \langle p, \omega \rangle \) satisfies \( \psi \) under \( B \) iff \( \langle \langle p, \omega \rangle, \text{unexit}_B \rangle, B, \omega \rangle \in \mathcal{L}(\mathcal{BP}_\psi) \).

**Formal proof.** To prove formally this result, we need the following definitions:

**Definition 12.** Let \( \pi' \) be a run of \( \mathcal{BP}_\psi \). Let \( \pi'(i) = \langle p'_i, \gamma'_0 \gamma'_1 \ldots \gamma'_n \rangle \) where \( p'_i \) is of the form \( \langle p_i, \Phi_i, l_i \rangle \), \( \gamma'_i \) is of the form \( \gamma_i \) or \( \langle \gamma_i, \Phi_i, l_i \rangle \), be a configuration of \( \pi' \). The projection of \( \pi'(i) \) on \( \mathcal{P} \); \( pr(\pi'(i)) := \langle p_i, \gamma_0 \gamma_1 \ldots \gamma_n \rangle \) is obtained by removing the set of formulas \( \Phi_i \) and the labels \( l_i \) from the control location and the stack symbols of \( \pi'(i) \).

Let \( \pi' = \langle \langle p_0, \Phi_0, l_0 \rangle, \omega'_0 \rangle \langle p_1, \Phi_1, l_1 \rangle, \omega'_1 \rangle \ldots \) be a run of \( \mathcal{BP}_\psi \). Let \( \pi = \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \ldots \) be the run obtained by projecting on \( \mathcal{P} \) all the configurations of \( \pi' \), then, it is easy to see that for every \( i \geq 0 \), either \( \langle p_i, \omega_i \rangle \Rightarrow \mathcal{P} \langle p_{i+1}, \omega_{i+1} \rangle \) (in case \( \langle p_{i+1}, \Phi_{i+1}, l_{i+1} \rangle, \omega'_{i+1} \rangle \) is obtained from \( \langle \langle p_i, \Phi_i, l_i \rangle, \omega'_i \rangle \) using a transition corresponding to the rule (\( \beta_{9,1} \), (\( \beta_{9,2} \)), and (\( \beta_{9,3} \))), or \( \langle p_i, \omega_i \rangle = \langle p_{i+1}, \omega_{i+1} \rangle \) in the other cases. Then, to obtain from \( \pi' \) a run of \( \mathcal{P} \), we need to get rid of these duplicated configurations. Let \( pr(\pi') \) be the run obtained after removing these duplicated configurations. Then, it is easy to see that:
Lemma 5. Let \( \pi' = \langle \langle p_0, A_0, l_0 \rangle, \omega_0' \rangle \langle \langle p_1, A_1, l_1 \rangle, \omega_1' \rangle \ldots \) be a run of \( BP_\psi \), let \( pr(\pi') = \langle p_{z_0}, \omega_{z_0} \rangle \langle p_{z_1}, \omega_{z_1} \rangle \ldots \) be the projection of \( \pi' \) on \( \mathcal{P} \), then, \( pr(\pi') \) is a run in \( \mathcal{P} \).

Proof of Theorem 9. Now, we are ready to prove Theorem 9. We prove these 2 directions:

\( \iff \) Assume that \( \langle \langle p_0, \{ \psi \}, \text{unexit} \rangle, B_0 \rangle, \omega_0 \rangle \in \mathcal{L}(BP_\psi) \). In other words, there exists an accepting run

\[ \pi' = \langle \langle p_0, \Phi_0, l_0 \rangle, B_0 \rangle, \omega_0' \rangle \langle \langle p_1, \Phi_1, l_1 \rangle, B_1 \rangle, \omega_1' \rangle \ldots \]

of \( BP_\psi \) where \( \Phi_0 = \{ \psi \}, l_0 = \text{unexit} \). Let \( \pi = \langle p_{z_0}, \omega_{z_0} \rangle \langle p_{z_1}, \omega_{z_1} \rangle \ldots \) be the projection of \( \pi' \) on \( \mathcal{P} \), then, \( \pi \) is a run of \( \mathcal{P} \) (by Lemma 5). We need to prove that \( \pi \models_{B_0 \psi} \psi \).

Proof. Assume that \( \pi(z_i) \) corresponds to \( \pi'(z_i) \ldots \pi'(z_i + k_i) \) for every \( i \geq 0 \). It is sufficient to prove that \( \langle \pi, z_i \rangle \models_{B_0} \phi \) for every \( \phi \in \Phi_{z_i} \). The proof is by induction on the structure of \( \phi \).

- **Base case:**
  \[ \phi = b(\alpha_1, \ldots, \alpha_n) \ (b(\alpha_1, \ldots, \alpha_n) \in AP_\chi) \]
  then, \( \pi'(z_i + 1) \) is determined by the rule \( (\beta_1) \), we get that \( \pi'(z_i + 1) = \langle \langle p_{z_i}, \Phi_{z_i} \setminus \{b(\alpha_1, \ldots, \alpha_n)\}, l_{z_i}, \omega_{z_i} \rangle \rangle \). Also, we have \( b(B(\alpha_1), \ldots, B(\alpha_n)) \in \lambda(p_{z_i}) \) (by the condition in the rule \( (\beta_1) \)). Thus we get \( \langle \pi, z_i \rangle \models_{B_{z_i}} \phi \) (by the definition of SP-CARET). The property holds for this case.

- \[ \phi = \neg b(\alpha_1, \ldots, \alpha_n) \ (b(\alpha_1, \ldots, \alpha_n) \in AP_\chi) \]
  then, \( \pi'(z_i + 1) \) is determined by the rule \( (\beta_2) \), we get that \( \pi'(z_i + 1) = \langle \langle p_{z_i}, \Phi_{z_i} \setminus \{\neg b(\alpha_1, \ldots, \alpha_n)\}, l_{z_i}, B_{z_i}, \omega_{z_i} \rangle \rangle \). Also, we have \( b(B(\alpha_1), \ldots, B(\alpha_n)) \notin \lambda(p_{z_i}) \) (by the condition in the rule \( (\beta_2) \)). Thus we get \( \langle \pi, z_i \rangle \models_{B_{z_i}} \neg b(\alpha_1, \ldots, \alpha_n) \) (by the definition of SP-CARET). In other words, \( \langle \pi, z_i \rangle \models_{B_{z_i}} \phi \).

The property holds for this case.

- **Induction Step:**
  \[ \phi = \phi_1 \land \phi_2, \text{then}, \pi'(z_i + 1) \text{ is determined by the rule } (\beta_3) \]
  we get \( \pi'(z_i + 1) = \langle \langle p_{z_i}, \Phi_{z_i} \setminus \{\phi_1, \phi_2\} \setminus \{\phi\}, l_{z_i}, B_{z_i}, \omega_{z_i} \rangle \rangle \). Note that \( \pi'(z_i + 1) \) is also an accepting run of \( BP_\psi \), so, we obtain \( \langle \pi, z_i \rangle \models_{B_{z_i}} \phi_1 \) and \( \langle \pi, z_i \rangle \models_{B_{z_i}} \phi_2 \) (by the induction hypothesis).

Thus, \( \langle \pi, z_i \rangle \models_{B_{z_i}} \phi_1 \land \phi_2 \). The property holds.
3.3. PCARE\textsuperscript{\textcircled{c}} Model-Checking for Pushdown Systems

\( \phi = \phi_1 \lor \phi_2 \), then, \( \pi'(z_i + 1) \) is determined by the rule (\( \beta_4 \)), there are two possibilities:

* If \( \pi'(z_i + 1) = \langle ([p_{z_i}, \Phi_{z_i} \cup \{ \phi_1 \} \setminus \{ \phi \}, l_{z_i}], B_{z_i}), \omega_{z_i} \rangle \). Since \( (\pi', z_i + 1) \) is also an accepting run of \( \mathcal{BP}_\psi \), then, by applying the induction hypothesis, we get \( (\pi, z_i) \models_{B_{z_i}} \phi_1 \).

* If \( \pi'(z_i + 1) = \langle ([p_{z_i}, \Phi_{z_i} \cup \{ \phi_2 \} \setminus \{ \phi \}, l_{z_i}], B_{z_i}), \omega_{z_i} \rangle \). Since \( (\pi', z_i + 1) \) is also an accepting run of \( \mathcal{BP}_\psi \), then, by applying the induction hypothesis, we get \( (\pi, z_i) \models_{B_{z_i}} \phi_2 \).

Thus, we always have \( (\pi, z_i) \models_{B_{z_i}} \phi_1 \lor \phi_2 \). The property holds for this case.

\( \phi = \forall x \phi' \), then, \( \pi'(z_i + 1) \) is determined by the rule (\( \beta_6 \)), we get \( \pi'(z_i + 1) = \langle ([p_{z_i}, \Phi_{z_i} \cup \{ \phi'_c \mid c \in D \} \setminus \{ \forall x \phi' \}, l_{z_i}], B_{z_i}[x \leftarrow c]), \omega_{z_i} \rangle \) where \( \phi'_c = \phi' \) where \( x \) is replaced by \( c \). Since \( (\pi', z_i + 1) \) is also an accepting run of \( \mathcal{BP}_\psi \), then, by applying the induction hypothesis, we get \( (\pi, z_i) \models_{B_{z_i}} \phi'_c \) for every \( c \in D \). This implies that \( (\pi, z_i) \models_{B_{z_i}} \forall x \phi' \). The property holds.

\( \phi = \exists x \phi' \), then, there are two possibilities:

* If \( \Phi_{z_i} \) contains no formulas \( \varphi \) where \( x \) is a free variable, we get that \( (\pi', z_i + 1) = \langle ([p_{z_i}, \Phi_{z_i} \cup \{ \phi' \} \setminus \{ \exists x \phi' \}, l_{z_i}], B_{z_i}[x \leftarrow c]), \omega_{z_i} \rangle \) for some \( c \in D \). Since \( (\pi', z_i + 1) \) is also an accepting run of \( \mathcal{BP}_\psi \), then, by applying the induction hypothesis, we get \( (\pi, z_i) \models_{B_{z_i}} \exists x \phi' \) for some \( c \in D \). Since \( (B_{z_i}, B_{z_i}[x \leftarrow c]) \in \theta_x \), we obtain that \( (\pi, z_i) \models_{B_{z_i}} \exists x \phi' \). The property holds.

* If \( \Phi_{z_i} \) contains no formulas \( \varphi \) where \( x \) is a free variable, we get that \( (\pi', z_i + 1) = \langle ([p_{z_i}, \Phi_{z_i} \cup \{ \phi' \} \setminus \{ \exists x \phi' \}, l_{z_i}], B_{z_i}), \omega_{z_i} \rangle \). Since \( (\pi', z_i + 1) \) is also an accepting run of \( \mathcal{BP}_\psi \), then, by applying the induction hypothesis, we get \( (\pi, z_i) \models_{B_{z_i}} \phi'_c \). Thus, \( (\pi, z_i) \models_{B_{z_i}} \exists x \phi' \). The property holds.

For the case \( \Phi = \{ X^g \phi_1, \ldots, X^g \phi_n, X^a \varphi_1, \ldots, X^a \varphi_m \} \). Note that this is always the case, since if \( \Phi \) contains a formula that is not in the form \( X^r \phi' \), we will process that formula first. In this case, \( \pi'(z_i + 1) \) is determined by the nature of the statements:

* If \( \langle p_{z_i}, \omega_{z_i} \rangle \models \Phi \rightarrow p \langle p_{z_i+1}, \omega_{z_i+1} \rangle \) corresponds to a call statement, then, \( \pi'(z_i + 1) \) is determined by the rule (\( \beta_{9,1} \)) which means that \( \pi'(z_i + 1) = \langle ([p_{z_i+1}, \{ \phi_1, \ldots, \phi_n \}, l_{z_i+1}], B_{z_i}), \omega_{z_i}' \rangle \). Since
(π’, z_i + 1) is also an accepting run of \(BP_\psi\), then, by applying the induction hypothesis, we get that \((π, z_i + 1) \models_{B_z} φ_t\) for every \(1 \leq t \leq n\). Thus, \((π, z_i) \models_{B_z} X^gφ_t\) for every \(1 \leq t \leq n\). Let \(π(z_a)\) be the abstract-successor of \(π(z_i)\). Since \(π’\) is an accepting run of \(BP_\psi\), we get that \(φ_1, ..., φ_m\) is validated at \(π(z_a)\). In other words, \((π, z_i) \models \{ X^gφ_1, ..., X^gφ_n, X^aφ_1, ..., X^aφ_m \}\). The property holds for this case.

* If \(\langle π_z, ω_z \rangle \models_P \langle π_{z+1}, ω_{z+1} \rangle\) corresponds to a return statement, then, \(π’(z_i + 1)\) is determined by the rule \((β_9, 2)\) which means that \(π’(z_i + 1) = (\langle [p_z, ω_z], \{ φ_1, ..., φ_n \}, B_z, ω_z' \rangle, z_i).\) Since \((π’, z_i + 1)\) is also an accepting run of \(BP_\psi\), then, by applying the induction hypothesis, we get that \((π, z_i + 1) \models_{B_z} φ_t\) for every \(1 \leq t \leq n\). Thus, \((π, z_i) \models_{B_z} X^gφ_t\) for every \(1 \leq t \leq n\). The property holds.

* If \(\langle π_z, ω_z \rangle \models_P \langle π_{z+1}, ω_{z+1} \rangle\) corresponds to a simple statement, then, \(π’(z_i + 1)\) is determined by the rule \((β_9, 3)\) which means that \(π’(z_i + 1) = (\langle [p_z, ω_z], \{ φ_1, ..., φ_n, ϕ_1, ..., ϕ_m \}, l_{z_i+1}, B_z, ω_z' \rangle, \) \(z_i).\) Since \((π’, z_i + 1)\) is also an accepting run of \(BP_\psi\), then, by applying the induction hypothesis, we get that \((π, z_i + 1) \models_{B_z} φ_t\) for every \(1 \leq t \leq n\) and \((π, z_i + 1) \models_{B_z} ϕ_x\) for every \(1 \leq x \leq m\). Thus, \((π, z_i) \models_{B_z} X^gφ_t\) for every \(1 \leq t \leq n\). Also, since \(π(z_{i+1})\) is the abstract-successor of \(π(z_i)\) in this case, we get \((π, z_i) \models_{B_z} X^aφ_x\) for every \(1 \leq x \leq m\). Therefore, \((π, z_i) \models_{B_z} \{ X^gφ_1, ..., X^gφ_n, X^aφ_1, ..., X^aφ_m \}\). The property holds for this case.

- \(φ = φ_1 U^vφ_2\) \((v \in \{g, a\})\), then, \(π’(z_i + 1)\) is determined by the rule \((β_9)\), there are two possibilities:

  * Firstly, \(π’(z_i + 1) = (\langle [p_z, ω_z], \{ φ_2 \}, l_{z_i}, B_z, ω_z' \rangle, \) \(z_i).\) Then, since \((π’, z_i + 1)\) is also an accepting run of \(BP_\psi\), by applying the induction hypothesis, we get \((π, z_i) \models_{B_z} φ_2\).

  * Secondly, note that all the formulas \(φ’ \in Φ_{z, i+1}\) which are not in the form \(X^gφ'\) will be processed until \(Φ_{z, i+1}\) is in the form \(Φ = \{ X^gφ_1, ..., X^gφ_n, X^aφ_1, ..., X^aφ_m \}\). For this case, we already prove above that \((π, z_i) \models_{B_z} X^gφ’\) for
every $X^v\phi' \in \Phi_{zi+1}$. Thus, we obtain $(\pi, zi) \models B_{zi} X^v \phi$ for this case.

In conclusion, we always have $(\pi, zi) \models B_{zi} \phi_2$ or $((\pi, zi) \models B_{zi} \phi_1$ and $(\pi, zi) \models B_{zi} X^v \phi_1 U^v \phi_2)$. In other words, $(\pi, zi) \models B_{zi} \phi_1 U^v \phi_2$ (by the semantics of $U^v$). The property holds.

- $\phi = \phi_1 R^v \phi_2 (v \in \{g, a\})$, then, $\pi'(zi + 1)$ is determined by the rule $(\beta_8)$. Similar to the case $\phi = \phi_1 U^v \phi_2$, we obtain that $(\pi, zi) \models B_{zi} \phi$.

$(\Longrightarrow)$ Assume that there exists an execution $\pi = \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \ldots$ of $P$ such that $\pi \models B_{zi} \psi$, we have to show that $\langle \langle p_0, \{\psi\}, unexit\rangle, B), \omega_0 \rangle \in \mathcal{L}(BP_\psi)$. In other words, we have to show an accepting run of $BP_\psi$ starting from $\langle \langle p_0, \{\psi\}, unexit\rangle, B), \omega_0 \rangle$.

Proof. Let $\pi'(0) = \langle \langle p_0, \{\psi\}, unexit\rangle, B), \omega_0 \rangle$.

Let $\pi'(i) = \langle \langle p_j, \Phi_j, l_j \rangle, B_j), \gamma'_0 \gamma'_m \cdots \gamma'_t \rangle$ where $\gamma'_t$ is of the form $\gamma_t$ or $\langle \gamma_t, \Phi_t, l_t \rangle$ for every $0 \leq t \leq m$. Now we show that for every $i \geq 0$, we can compute from $\pi'(i)$ its immediate successor $\pi'(i + 1) = \langle \langle \gamma_q, \Phi', l'_q \rangle, B'), \omega' \rangle$.

During this computation, we maintain the following property:

"(A) For every $i + 1 \geq 0$, $l'$ is the label expressing whether the execution of the procedure $P(pr(\pi'(i + 1)))$ terminates or not from $pr(\pi'(i + 1))$; and for every $\chi \in \Phi'$, $pr((\pi', i + 1)) \models B' \chi".$

The computation is shown by induction on $i + 1$.

- Base case ($i + 1 = 0$). We prove that $\pi'(0) = \langle \langle p_0, \{\psi\}, unexit\rangle, B), \omega_0 \rangle$ satisfies the property (A). In other words, we need to show that (A) is satisfied with $\Phi' = \{\psi\}, l' = unexit, B' = B$.

- Since $\pi$ is an infinite run and $\langle p_0, \omega_0 \rangle$ is the initial configuration of $\pi$, then, the execution of the procedure $P(\langle p_0, \omega_0 \rangle)$ never terminates. $\iff$ $\text{unexit}$ is the label expressing whether the execution of the procedure $P(\langle p_0, \omega_0 \rangle)$ terminates or not from $\langle p_0, \omega_0 \rangle$. $\iff$ $l' = \text{unexit}$ is the label expressing whether the execution of the procedure $P(pr(\pi'(0)))$ terminates or not from $pr(\pi'(0))$ (since $pr(\pi'(0)) = \langle p_0, \omega_0 \rangle$). In other words, the property related to $l'$ in (A) is satisfied.
Since \( \pi \models B \psi \), we get that \( (\pi, 0) \models B \psi \implies pr((\pi', 0)) \models B \psi \) (since \( pr((\pi', 0)) = (\pi, 0) \)) \implies for every \( \chi \in \{ \psi \} \), \( pr((\pi', 0)) \models B \chi \implies \) for every \( \chi \in \Phi' \), \( pr((\pi', 0)) \models B \chi \) (since \( \Phi' = \{ \psi \} \)). Therefore, the property related to \( \Phi' \) in (A) is satisfied.

\[ \implies \text{The property (A) holds for this case.} \]

- **Induction Step** \((i + 1 > 0)\). There are two cases: \( \gamma'_m = \gamma_m \) or \( \gamma'_m = \langle \gamma_m, \Phi'', l'' \rangle \) where \( \gamma_m \in \Gamma \), \( \Phi'' \subseteq Cl(\psi) \), \( l'' \in Label \).

1. \( \gamma'_m = \gamma_m \), then, \( \pi'(i) = \langle \langle p_j, \Phi_j, l_j \rangle, B_j \rangle, \gamma_m \gamma_{m-1} \ldots \gamma_0 \).

   Assume that the property (A) is satisfied at \( \pi'(i) \). Now, we show how we can compute the immediate successor \( \pi'(i+1) \) of \( \pi'(i) \) which maintain (A). This depends on the nature of \( \phi \in \Phi_j \):

   - Case \( \phi = b(\alpha_1, \ldots, \alpha_n) \in \Phi_j \), then, by applying the induction hypothesis, we obtain that for every \( \chi \in \Phi_j \), \( pr((\pi', i)) \models B_j \chi \) (1) \implies for every \( \chi \in \Phi_j \), \( (\pi, j) \models B_j \chi \) (since \( pr((\pi', i)) = (\pi, j) \)) \implies \( (\pi, j) \models B_j b(\alpha_1, \ldots, \alpha_n) \) (since \( b(\alpha_1, \ldots, \alpha_n) \in \Phi_j \)) \implies \( b(B(\alpha_1), \ldots, B(\alpha_n)) \in \lambda(p_j) \). Thus, by applying the rules in (\( \beta_1 \)), let \( \Phi' = \Phi_j \setminus \{b(\alpha_1, \ldots, \alpha_n)\} \), \( l'' = l_j, B' = B_j \), we obtain that \( \pi'(i+1) = \langle \langle p_j, \Phi_j \setminus \{b(\alpha_1, \ldots, \alpha_n)\}, l_j \rangle, B_j \rangle, \gamma_m \gamma_{m-1} \ldots \gamma_0 \).

   Firstly, we show that the property related to \( \Phi' \) in (A) is satisfied.

   * According to the way we select \( \pi'(i+1) \), we get that \( pr((\pi', i+1)) = (\pi, j) \). Therefore, \( pr((\pi', i)) = pr((\pi', i+1)) \) (since \( pr((\pi', i)) = (\pi, j) \)) (2)

   * From (2) and (1), we get that for every \( \chi \in \Phi_j \), \( pr((\pi', i + 1)) \models B_j \chi \). Thus, for every \( \chi \in \Phi_j \setminus \{b(\alpha_1, \ldots, \alpha_n)\}, pr((\pi', i + 1)) \models B_j \chi \). Therefore, for every \( \chi \in \Phi_j \setminus \{b(\alpha_1, \ldots, \alpha_n)\}, pr((\pi', i + 1)) \models B' \chi \) (since \( B' = B_j \)). In other words, for every \( \chi \in \Phi', pr((\pi', i + 1)) \models B' \chi \). The property related to \( \Phi' \) in (A) is satisfied in this case (3).

   Now, we show that the property related to \( l' \) in (A) is satisfied.

   * According to the way we select \( \pi'(i+1) \), we get that \( pr(\pi'(i+1)) = \pi(j) \). Therefore, \( pr(\pi'(i+1)) = pr(\pi'(i)) \) (since \( pr(\pi'(i)) = \pi(j) \)) (4)

   * By applying the induction hypothesis, we obtain that \( l_j \) is the label expressing whether the execution of the procedure \( \mathcal{P}(pr(\pi'(i))) \) terminates or not from \( pr(\pi'(i)) \) (5)
* From (4) and (5), we get that $l_j$ is the label expressing whether the execution of the procedure $\mathcal{P}(pr(\pi'(i + 1)))$ terminates or not from $pr(\pi'(i + 1))$. In addition, we get that $l' = l_j$ (by the way we select $l'$). Therefore, we obtain that $l'$ is the label expressing whether the execution of the procedure $\mathcal{P}(pr(\pi'(i + 1)))$ terminates or not from $pr(\pi'(i + 1))$. The property related to $l'$ in (A) is satisfied in this case (6).

From (3) and (6), we get that the property (A) holds for this case.

- Case $\phi = \neg b(\alpha_1, ..., \alpha_n) \in \Phi_j$, then, by applying the induction hypothesis, we obtain that for every $\chi \in \Phi_j$, $pr((\pi', i)) \models_{B_j} \chi$ (7) $\implies$ for every $\chi \in \Phi_j$, $(\pi, j) \models_{B_j} \chi$ (since $pr((\pi', i)) = (\pi, j)$) $\implies$ $(\pi, j) \models_{B_j} \neg b(\alpha_1, ..., \alpha_n)$ (since $\neg b(\alpha_1, ..., \alpha_n) \in \Phi_j$) $\implies$ $b(B(\alpha_1), ..., B(\alpha_n)) \notin \lambda(p_j)$. Thus, by applying the rules in ($\beta_2$), let $\Phi' = \Phi_j \setminus \{\neg b(\alpha_1, ..., \alpha_n)\}$, $l' = l_j$, $B' = B_j$, we obtain that $\pi'(i + 1) = \langle\{p_j, \Phi_j \setminus \{\neg b(\alpha_1, ..., \alpha_n)\}, l_j\}, B_j\rangle$, $\gamma_m \gamma_{m-1} ... \gamma_0$.

Firstly, we show that the property related to $\Phi'$ in (A) is satisfied.

* According to the way we select $\pi'(i + 1)$, we get that $pr((\pi', i + 1)) = (\pi, j)$. Therefore, $pr((\pi', i)) = pr((\pi', i + 1))$ (since $pr((\pi', i)) = (\pi, j)$) (8)

* From (8) and (7), we get that for every $\chi \in \Phi_j$, $pr((\pi', i + 1)) \models_{B_j} \chi$. Thus, for every $\chi \in \Phi_j \setminus \{\neg b(\alpha_1, ..., \alpha_n)\}$, $pr((\pi', i + 1)) \models_{B_j} \chi$. Therefore, for every $\chi \in \Phi_j \setminus \{\neg b(\alpha_1, ..., \alpha_n)\}$, $pr((\pi', i + 1)) \models_{B_j} \chi$ (since $B' = B_j$).

In other words, for every $\chi \in \Phi'$, $pr((\pi', i + 1)) \models_{B_j} \chi$. The property related to $\Phi'$ in (A) is satisfied in this case (9).

Secondly, similar to the case $\phi = b(\alpha_1, ..., \alpha_n)$, it can be seen that the property related to $l'$ in (A) is satisfied (10).

From (9) and (10), we get that the property (A) holds for this case.

- Case $\phi = \phi_1 \land \phi_2 \in \Phi_j$, then, by applying the rules in ($\beta_3$), let $\Phi' = \Phi_j \cup \{\phi_1, \phi_2\} \setminus \{\phi_1 \land \phi_2\}$, $l' = l_j$, $B' = B_j$, we obtain that $\pi'(i + 1) = \langle\{p_j, \Phi_j \cup \{\phi_1, \phi_2\} \setminus \{\phi_1 \land \phi_2\}, l_j\}, B_j\rangle$, $\gamma_m \gamma_{m-1} ... \gamma_0$.

By applying the induction hypothesis, we obtain that for every $\chi \in \Phi_j$, $pr((\pi', i)) \models_{B_j} \chi$ (11) $\implies$ for every $\chi \in \Phi_j$, $(\pi, j) \models_{B_j} \chi$ (since $pr((\pi', i)) = (\pi, j)$) $\implies$ $(\pi, j) \models_{B_j} \phi_1 \land \phi_2$ (since
\(\phi_1 \land \phi_2 \in \Phi_j\) \implies (\pi, j) \models_{\lambda} B^j \phi_1 \text{ and } (\pi, j) \models_{\lambda} B^j \phi_2 \implies pr((\pi', i)) \models_{\lambda} B^j \phi_1 \text{ and } pr((\pi', i)) \models_{\lambda} B^j \phi_2 \text{ (since } pr((\pi', i)) = (\pi, j)) \tag{12}

Firstly, we show that the property related to \(\Phi'\) in (A) is satisfied.

* According to the way we select \(\pi'(i+1)\), we get that \(pr((\pi', i+1)) = (\pi, j)\). Therefore, \(pr((\pi', i)) = pr((\pi', i+1))\) (since \(pr((\pi', i)) = (\pi, j)\)) \tag{13}

* From (13) and (11), we get that for every \(\chi \in \Phi_j\), \(pr((\pi', i + 1)) \models_{\lambda} B^j \chi\). Thus, for every \(\chi \in \Phi_j \setminus \{\phi_1 \land \phi_2\}\), \(pr((\pi', i + 1)) \models_{\lambda} B^j \chi\). Therefore, for every \(\chi \in \Phi_j \setminus \{\phi_1 \land \phi_2\}\), \(pr((\pi', i + 1)) \models_{\lambda} B^j \chi\) (since \(B' = B_j\)) \tag{14}

* From (13) and (12), we get that \(pr((\pi', i + 1)) \models_{\lambda} B^j \phi_1\) and \(pr((\pi', i + 1)) \models_{\lambda} B^j \phi_2\). Thus, we obtain that \(pr((\pi', i + 1)) \models_{\lambda} B^j \phi_1\) and \(pr((\pi', i + 1)) \models_{\lambda} B^j \phi_2\) (since \(B' = B_j\)) \tag{15}

* From (14) and (15), we get that for every \(\chi \in \Phi_j \cup \{\phi_1, \phi_2\} \setminus \{\phi_1 \land \phi_2\}\), \(pr((\pi', i + 1)) \models_{\lambda} B^j \chi\). In other words, for every \(\chi \in \Phi', \) \(pr((\pi', i + 1)) \models_{\lambda} B^j \chi\). The property related to \(\Phi'\) in (A) is satisfied in this case \(\tag{16}\).

Secondly, similar to the case \(\phi = b(\alpha_1, \ldots, \alpha_n)\), it can be seen that the property related to \(l'\) in (A) is satisfied \(\tag{17}\).

From \(\tag{16}\) and \(\tag{17}\), we get that the property (A) holds for this case.

- Case \(\phi = \phi_1 \lor \phi_2 \in \Phi_j\),

By applying the induction hypothesis, we obtain that for every \(\chi \in \Phi_j, \) \(pr((\pi', i)) \models_{\lambda} B^j \chi\) \(\implies\) for every \(\chi \in \Phi_j, \) \(pr((\pi', i)) \models_{\lambda} B^j \chi\) (since \(pr((\pi', i)) = (\pi, j)\)) \implies \(\pi, j) \models_{\lambda} B^j \phi_1 \lor \phi_2\) (since \(\phi_1 \lor \phi_2 \in \Phi_j\)) \implies \(\pi, j) \models_{\lambda} B^j \phi_1\) or \(\pi, j) \models_{\lambda} B^j \phi_2\) \implies \(pr((\pi', i)) \models_{\lambda} B^j \phi_1\) or \(pr((\pi', i)) \models_{\lambda} B^j \phi_2\) (since \(pr((\pi', i)) = (\pi, j)\)) \tag{19}

(a) Case \(pr((\pi', i)) \models_{\lambda} B^j \phi_1\) \(\text{ (19) }\), then, by applying the rules in (19), let \(\Phi' = \Phi_j \cup \{\phi_1\} \setminus \{\phi_1 \lor \phi_2\}\), \(l' = l_j, B' = B_j\), we obtain that \(\pi'(i + 1) = (((p_l, \Phi_j \cup \{\phi_1\} \setminus \{\phi_1 \lor \phi_2\}, l_j, B_j), \gamma_m \gamma_{m-1} \cdots \gamma_0)\).

Firstly, we show that the property related to \(\Phi'\) in (A) is satisfied.

* According to the way we select \(\pi'(i + 1)\), we get that \(pr((\pi', i + 1)) = (\pi, j)\). Therefore, \(pr((\pi', i)) = pr((\pi', i + 1))\) (since \(pr((\pi', i)) = (\pi, j)\)) \tag{20}
3.3. PCARET\textsuperscript{c} Model-Checking for Pushdown Systems

* From (20) and (18), we get that for every $\chi \in \Phi_j$, $pr((\pi', i + 1)) \models_{\lambda}^{B_j} \chi$. Thus, for every $\chi \in \Phi_j \setminus \{\phi_1 \lor \phi_2\}$, $pr((\pi', i + 1)) \models_{\lambda}^{B_j} \chi$. Therefore, for every $\chi \in \Phi_j \setminus \{\phi_1 \lor \phi_2\}$, $pr((\pi', i + 1)) \models_{\lambda}^{B'} \chi$ (since $B' = B_j$) (21).

* From (20) and (19), we get that $pr((\pi', i + 1)) \models_{\lambda}^{B_j} \phi_1$. Thus, we obtain that $pr((\pi', i + 1)) \models_{\lambda}^{B'} \phi_1$ (since $B' = B_j$) (22).

* From (21) and (22), we get that for every $\chi \in \Phi_j \cup \{\phi_1\} \setminus \{\phi_1 \lor \phi_2\}$, $pr((\pi', i + 1)) \models_{\lambda}^{B'} \chi$. In other words, for every $\chi \in \Phi'$, $pr((\pi', i + 1)) \models_{\lambda}^{B'} \chi$. The property related to $\Phi'$ in (A) is satisfied in this case (23).

Secondly, similar to the case $\phi = b(\alpha_1, ..., \alpha_n)$, it can be seen that the property related to $l'$ in (A) is satisfied (24). From (23) and (24), we get that the property (A) holds for this case.

(b) Case $pr((\pi', i)) \models_{\lambda}^{B_j} \phi_2$, then, by applying the rules in (\beta_1), let $\Phi' = \Phi_j \cup \{\phi_2\} \setminus \{\phi_1 \lor \phi_2\}$, $l' = l_j$, $B' = B_j$, we obtain that $\pi'(i + 1) = (((p_j, \Phi_j \cup \{\phi_2\} \setminus \{\phi_1 \lor \phi_2\}, l_j), B_j), \gamma_0 \gamma_{m-1} ... \gamma_0')$.

Similar to the above case, we get that the property (A) holds for this case.

- Case $\phi = \forall x \phi' \in \Phi_j$, then, by applying the rules in (\beta_6), let $\pi'(i + 1) = (((p_j, \Phi_j \cup \{\phi_1' \setminus \{\exists x \phi'\}, l_j), B_j), \gamma_0 \gamma_{m-1} ... \gamma_0')$ where $\phi_1'$ is $\phi'$ where $x$ is replaced by $c$. Since $(\pi, j) \models_{\lambda}^{B_j} \forall x \phi'$, we get that $(\pi, j) \models_{\lambda}^{B_j} \phi_c$ for all $\phi_c \in \Phi_j \setminus \{\exists x \phi'\}$ (by the induction assumption), the property holds.

- Case $\phi = \exists x \phi' \in \Phi_j$. We consider two possibilities:

  * If there exists a formula $\varphi \in \Phi_j$ s.t. $x$ is a free variable of $\varphi$, then, by applying the rules in (\beta_{5.2}), let $\pi'(i + 1) = (((p_j, \Phi_j \cup \{\phi_1' \setminus \{\exists x \phi'\}, l_j), B_j), \gamma_0 \gamma_{m-1} ... \gamma_0')$ where $\phi_1'$ is $\phi'$ where $x$ is substituted by $c$ and $(\pi, j) \models_{\lambda}^{B_j} \phi_c$. Since $(\pi, j) \models_{\lambda}^{B_j} \phi_c$ for every $\phi_c \in \Phi_j \setminus \{\exists x \phi'\}$ (by the induction assumption), the property holds.

  * If there are no formula $\varphi \in \Phi_j$ s.t. $x$ is a free variable of $\varphi$, then, by applying the rules in (\beta_{5.1}), let $\pi'(i + 1) = (((p_j, \Phi_j \cup \{\phi_1' \setminus \{\exists x \phi'\}, l_j), B_j[x \leftarrow c]), \gamma_0 \gamma_{m-1} ... \gamma_0')$ s.t. $(\pi, j) \models_{\lambda}^{B_j[x \leftarrow c]} \phi'$ for $c \in \mathcal{D}$. Since $x$ is not free variable of any formula in $\Phi_j \setminus \{\exists x \phi'\}$, we obtain that $(\pi, j) \models_{\lambda}^{B_j[x \leftarrow c]} \phi_c'$ for every $\phi'_c \in \Phi_j \setminus \{\exists x \phi'\}$. Therefore, the property holds.

- Case $\phi = \phi_1 U v \phi_2 \in \Phi_j$ ($v \in \{g, a\}$), then, by
applying the rules in \((\beta_2)\), we select \(\pi'(\ell + 1) = \langle(p_j, \Phi_j \cup \{\phi_2\} \setminus \{\phi_1 U^v \phi_2\}, l_j, B_j)\rangle\). Otherwise, \(\pi'(\ell + 1) = \langle(p_j, \Phi_j \cup \{\phi_1, X^v \phi\} \setminus \{\phi_1 U^v \phi_2\}, l_j, B_j)\rangle\). Since \((\pi, j) \models_{\lambda}^{B_j} \phi_1 U^v \phi_2\), we get that \((\pi, j) \models_{\lambda}^{B_j} \phi_2\) or \((\pi, j) \models_{\lambda}^{B_j} \phi\) and \((\pi, j) \models_{\lambda}^{B_j} X^v \phi_1 U^v \phi_2\). Therefore, the property holds.

- Case \(\phi = \phi_1 R^v \phi_2 \in \Phi_j\) \((v \in \{g, a\})\), then, similar to the case \(\phi = \phi_1 U^v \phi_2 \in \Phi_j\), we obtain that the property holds for this case.

- Case \(\Phi = \{X^g \phi_1, ..., X^g \phi_n, X^a \phi_1, ..., X^a \phi_m\}\)

Let \(\Phi_s = \{X^g \phi_1, ..., X^g \phi_n\}\), \(\Phi_a = \{X^a \phi_1, ..., X^a \phi_m\}\). Let \(l' = \text{exit}\) if the execution of the procedure \(\mathcal{D}(\langle p_{j+1}, \omega_{j+1}\rangle)\) from \(\langle p_{j+1}, \omega_{j+1}\rangle\) terminates; otherwise \(l' = \text{unexit}\). There are different cases depending on the nature of the transition \(\langle p_j, \omega_j\rangle \Rightarrow p \langle p_{j+1}, \omega_{j+1}\rangle\)

* If \(\langle p_j, \omega_j\rangle \Rightarrow p \langle p_{j+1}, \omega_{j+1}\rangle\) corresponds to a call statement.

Let \(\langle p_j, \omega_j\rangle \Rightarrow p \langle p_{j+1}, \omega_{j+1}\rangle\) be the rule associated with the transition \(\langle p_j, \omega_j\rangle \Rightarrow p \langle p_{j+1}, \omega_{j+1}\rangle\).

Firstly, we show that the conditions in the transition rule \((\beta_0)\) are satisfied. In other words, we need to show that if \(l' = \text{unexit}\) then \((l_j = \text{unexit} and \Phi_a = \emptyset)\).

\(l' = \text{unexit}\) implies that the execution of the procedure \(\mathcal{D}(\langle p_{j+1}, \omega_{j+1}\rangle)\) from \(\langle p_{j+1}, \omega_{j+1}\rangle\) never terminates. In addition, we get \(\langle p_j, \omega_j\rangle \Rightarrow p \langle p_{j+1}, \omega_{j+1}\rangle\). Thus, the execution of the procedure \(\mathcal{D}(\langle p_j, \omega_j\rangle)\) from \(\langle p_j, \omega_j\rangle\) never terminates. As a result, \(l_j = \text{unexit}\).

\(l' = \text{unexit}\) implies that the execution of the procedure \(\mathcal{D}(\langle p_{j+1}, \omega_{j+1}\rangle)\) from \(\langle p_{j+1}, \omega_{j+1}\rangle\) never terminates \(\Rightarrow\) the abstract successor of \(\langle p_j, \omega_j\rangle\) is \(\perp\) (by the definition of abstract successor) \(\Rightarrow\) \(\langle p_j, \omega_j\rangle \notin X^a \phi\) (by the semantics of SPCARET) \(\Rightarrow\) \(\Phi_a = \emptyset\).

\(\Rightarrow\) the conditions in the transition rule \((\beta_0)\) holds. Then, we apply the rules in \((\beta_0)\) and we select \(\pi'(\ell + 1) = \langle(p_j, \Phi_j \cup \{\phi_1 \ldots \phi_n\}, l_j', B_j)\rangle\). Since \((\pi, j) \models_{\lambda}^{B_j} X^g \phi_t\) for every \(1 \leq t \leq n\), then, \((\pi, j + 1) \models_{\lambda}^{B_j} \phi_t\) for every \(1 \leq t \leq n\). Therefore, the property holds for this case.

* If \(\langle p_j, \omega_j\rangle \Rightarrow p \langle p_{j+1}, \omega_{j+1}\rangle\) corresponds to a return state-
3.3. PCARET\(\text{c}\) Model-Checking for Pushdown Systems

Carletti and Montanari [73] introduced an extension of PCARET to pushdown systems.

Let \(〈p_j, γ_m⟩ \xrightarrow{\text{rel}} 〈p_{j+1}, ε⟩\) be the rule associated with the transition \(〈p_j, ω_j⟩ \Rightarrow_p 〈p_{j+1}, ω_{j+1}⟩\). Firstly, we need to show that the conditions in \((β_{9.2})\) are satisfied.

- \(〈p_j, ω_j⟩ \Rightarrow_p 〈p_{j+1}, ω_{j+1}⟩\) corresponds to a return statement \(t_j = \text{ret}\) (by the way we associate a tag to a configuration) \(⇒\) the abstract successor of \(〈p_j, ω_j⟩\) is \(⊥\) (by the definition of abstract successor) \(⇒\) \(〈p_j, ω_j⟩ \not\models X^a \phi\) (by the semantics of SPCARET) \(⇒\) \(Φ_a = ∅\). Then, the condition related to the set of formulas in \((β_{9.2})\) is satisfied.

- Also, we need to show that the condition related to labels in the transition rule \((β_{9.2})\) is satisfied. In other words, we need to prove that \(l_j = \text{exit}\), this is ensured because \(〈p_j, ω_j⟩ \Rightarrow_p 〈p_{j+1}, ω_{j+1}⟩\) corresponds to a return statement, then, at this point, we know that the execution of the procedure \(Ψ(〈p_j, ω_j⟩)\) can terminate. Therefore \(l_j = \text{exit}\).

Therefore, we can apply the transition rule in \((β_{9.2})\) and we select \(π(i + 1) = 〈⟨p_{j+1}, \{ϕ_1...ϕ_n\}, l', B_j⟩, γ_{m-1}...γ_0⟩\).

Since \((π, j) \models^{B_j} X^a ϕ_t\) for every \(1 ≤ t ≤ n\), then, \((π, j + 1) \models^{B_j} X^a ϕ_t\) for every \(1 ≤ t ≤ n\). Therefore, the property holds for this case.

- If \(〈p_j, ω_j⟩ \Rightarrow_p 〈p_{j+1}, ω_{j+1}⟩\) corresponds to a simple statement. Let \(〈p_j, γ⟩ \xrightarrow{\text{int}} 〈p_{j+1}, ω⟩\) be the rule associated with the transition \(〈p_j, ω_j⟩ \Rightarrow_p 〈p_{j+1}, ω_{j+1}⟩\). Then, we apply the rules in \((β_{9.3})\) and we select \(π'(i + 1) = 〈⟨p_{j+1}, \{ϕ_1...ϕ_n\}, \varphi_1...\varphi_m⟩, l', B_j⟩, ω_{γ_{m-1}...γ_0}⟩\). For this case, the rule corresponds to a simple statement, then \((π, j) \models^{B_j} X^a ϕ_t\) for every \(1 ≤ t ≤ n\) and \((π, j) \models^{B_j} X^a ϕ_x\) for every \(1 ≤ x ≤ m\).

Thus, \((π, j + 1) \models^{B_j} X^a ϕ_t\) for every \(1 ≤ t ≤ n\). Also, since \((π, j + 1)\) is the abstract-successor of \((π, j)\) in this case, we have \((π, j + 1) \models^{B_j} \varphi_x\) for every \(1 ≤ x ≤ m\). Thus, we obtain \((π, j + 1) \models^{B_j} ϕ_t\) for every \(1 ≤ t ≤ n\) and \((π, j + 1) \models^{B_j} ϕ_x\) for every \(1 ≤ x ≤ m\). Therefore, the property holds for this case.

2. \(γ'_m = 〈γ_m, \Phi'', l''⟩\), then, \(π'(i) = 〈⟨p_j, \Phi_j, l_j⟩, B_j⟩, 〈γ_m, \Phi'', l''⟩, γ_{m-1}...γ_0⟩\)

Note that this case only occurs at return-points. Let \(π(u)\) be the
corresponding call of this return. From the transition rules of call statements in $\beta_{0.1}$, we get that $l''$ is the label expressing whether the execution of the procedure $\mathcal{P}(\pi(u))$ can be terminated or not; and for every $\phi \in \Phi''$, $pr((\pi', i + 1))$ satisfies $\phi$.

Let $l' = l_j$. In this case, we will use the transition rules in $\beta_{0.2.2}$ to compute $\pi(k + 1)$. Firstly, we show that the required conditions in $\beta_{0.2.2}$ are satisfied. In other words, we need to prove that $l_j = l''$ where $l''$ is the label of the caller of $\mathcal{P}(\langle p_j, \gamma_{m-1}\ldots\gamma_0 \rangle)$. This is always satisfied because a call and its corresponding return point always belong to a same procedure. Therefore, we always obtain that $l_j = l''$.

Therefore, if $\pi'(i)$ is of the form $((\langle p_j, \Phi_j, l_j \rangle, B_j), \langle \gamma_m, \Phi'', l'' \gamma_{m-1}\ldots\gamma_0 \rangle)$, we apply the transition rules in $\beta_{0.2.2}$ and select $\pi'(i + 1) = ((\langle p_j, \Phi_j \cup \Phi'', l_j \rangle, B_j), \gamma_m \gamma_{m-1}\ldots\gamma_0)$. From the way we select $\pi'(i + 1)$, it is obvious to see that $pr((\pi', i)) = pr((\pi', i + 1))$.

Also, by applying the induction hypothesis, we obtain that for every $\phi \in \Phi_j$, $pr((\pi', i))$ satisfies $\phi$ $\Longrightarrow$ for every $\phi \in \Phi_j$, $pr((\pi', i + 1))$ satisfies $\phi$ (since $pr((\pi', i)) = pr((\pi', i + 1))$) $\Longrightarrow$ for every $\phi \in \Phi_j \cup \Phi''$ $pr((\pi', i + 1))$ satisfies $\phi$ $\Longrightarrow$ the property related to the set of formulas $\Phi_j \cup \Phi''$ is satisfied. In addition, since $l' = l_j$, we get that $l'$ is the label expressing whether the execution of the procedure $\mathcal{P}(\langle p_j, \gamma_{m-1}\ldots\gamma_0 \rangle)$ from $\langle p_j, \gamma_{m-1}\ldots\gamma_0 \rangle$ terminates or not. Therefore, the property holds for this case.

By this computation, we obtain an infinite run $\pi'$ of $\mathcal{BP}_\psi$ corresponding to a run of $\mathcal{P}$.

Now, we prove that the computed run $\pi'$ is an accepting run of $\mathcal{BP}_\psi$. To do that, we prove that each set of the Büchi accepting condition of $\mathcal{BP}_\psi$ is visited infinitely often by $\pi'$. Suppose that this is not the case, then there exists a set $F_{\phi_1U^v\phi_2}$ where $v \in \{g, a\}$ such that $\pi'$ does not visit infinitely often any control location in $P \times F_{\phi_1U^v\phi_2}$. This means that there exists $k$ where $pr((\pi', i + 1))$ the suffix of $\pi'$ starting from $\pi'(k)$ (denoted $\pi'(k)$) does not visit any control location in $P \times F_{\phi_1U^v\phi_2}$. It implies that for every $t \geq k$, where $\pi'(t) = (\langle p_u, \Phi_u \rangle, \omega_u)$, we must have $\phi_2 \not\in \Phi_u$ and $\phi_1U^v\phi_2 \in \Phi_u$ (otherwise, $\langle p_u, \Phi_u \rangle$ belongs to $P \times F_{\phi_1U^v\phi_2}$).

- $\phi_1U^v\phi_2 \in \Phi_u \Longrightarrow pr((\pi, u)) \models_{\pi} \phi_1U^v\phi_2$. $pr((\pi, u)) \models_{\pi} \phi_1U^v\phi_2$ implies that $\phi_2$ eventually holds.

- $\phi_2 \not\in \Phi_u \Longrightarrow pr((\pi, u)) \not\models_{\pi} \phi_2$. 
Note that the second fact that $\phi_2$ never happens contradicts with the first fact that $\phi_2$ eventually holds. Thus, this cannot be the case. Consequently, the run $\pi'$ visits infinitely often some control locations in $P \times F_{\phi_1 \cup \phi_2}$. $\pi'$ visits infinitely often each set of the Büchi accepting condition of $BP_\psi$. Thus, this cannot be the case. Consequently, $\pi'$ is an accepting run of $BP_\psi$.

In conclusion, from a run $\pi = \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \ldots$ of $P$ such that $\pi \models_B \psi$, we can always obtain an accepting run $\pi'$ of $BP_\psi$ starting from $\langle (\{p_0\}, \{\psi\}, \text{unexit}), B, \omega_0 \rangle$.

Remark 3. Note that the above procedure could not be applied if we consider the operators $X^c$, $U^c$ and $R^c$. Indeed, the key point of our construction is that we use the symbolic transitions of SBPDS to express different possible values of variables which allow us to obtain a better complexity. To do that, our construction is based on the fact that the satisfiability of a given formula at a certain state is ensured by the satisfiability of several formulas at the successor state. However, if we allow $X^c$, this does not hold anymore. Let us consider Figure 3.2 to illustrate this. For instance, we want to determine whether $\langle p_0, \omega_0 \rangle | B^\lambda X^c \phi$. This holds iff $\langle p_1, \omega_1 \rangle | B^\lambda X^c \phi$ (this is expressed by rules $(\beta_{5.3})$), iff $\langle p_2, \omega_2 \rangle | B^\lambda X^c \phi$, iff $\langle p_3, \omega_3 \rangle | B^\lambda X^c \phi$, iff $\langle p_4, \omega_4 \rangle | B^\lambda \phi$. The four first requirements are ensured by rules $(\beta_{5.3})$, since they consider the immediate successor state. However, the last requirement cannot be ensured by a rule like the $(\beta)$ rules above since the caller successor of a state is a predecessor of that state.

Remark 4. One can wonder why we do not apply the approach proposed in Chapter 2 which uses atoms (maximally consistent subsets of $Cl(\psi)$) to deal with all CARET operators (including $X^c$, $U^c$ and $R^c$). The reason is that SPCARET contains variables and quantifiers over variables, which makes it impossible to compute atoms without enumerating all possible values of variables. We could do this, but this has exactly the same complexity as translating the SPCARET formula into CARET and then model checking the CARET formula, since it is based on enumerating all possible values. Thus, we cannot benefit from the symbolic representation of variables using this approach.

Remark 5. Note that the transition rules in $(\beta_{5.2})$ are never applied if there are no free variables in the formula $\psi$. The key point of our construction lies in the case $\exists x \psi' \in \Phi$. Indeed, for this case, our construction have moved the complexity from the formula to the transitions of the symbolic Büchi pushdown system, where all possible values of the variable $x$ are symbolically encoded within the symbolic relation $\theta_x$. Let us take a simple example to illustrate this case. Consider this simple PCARET$^c$ formula $\psi = \exists x \exists y \text{mov}(x, y)$. To model check this formula against Pushdown Systems, we have two approaches:
• Translate \( \psi \) to an equivalent CARET formula and apply the algorithm presented in Chapter 2. We obtain the equivalent CARET formula \( \psi' = \bigvee_{x \in D} \bigvee_{y \in D} \text{mov}(x, y) \). Note that \(|\psi'| = |\psi| \times O(|D|^{|X|})\). Roughly speaking, we will consider all possible combinations of values of the tuple of variables \((x, y)\) which is very large.

• Apply our above algorithm. For the case \( \psi = \exists x \forall y \text{mov}(x, y) \), by applying the transition rules in (\(\beta_{n, 1}\)), the different values of \(x\) are represented by one symbolic relation \(\theta_x\), and the different values of \(y\) are represented by one symbolic relation \(\theta_y\). These relations can be efficiently represented using BDDs as explained in [ES01]. Thus, our algorithms works very well for this case.

The complexity of the above algorithm depends on whether the formula contains universal quantifiers or not:

• If \( \psi \) contains only existential quantifiers and no free variables, then, the number of control locations of \(BP_\psi\) is at most \(|P| \times 2^{O(|\psi|)}\) and the number of transitions is at most \(|\Delta||\Gamma| \times 2^{O(|\psi|)}\). From Theorem 8, the membership problem can be solved in time \(|P|.|\Delta|.|\Gamma|^2.2^{O(|\psi|)}|D|^{3.|X|}\).

• If \( \psi \) contains universal quantifiers or free variables, for the worst case, the number of control locations of \(BP_\psi\) is at most \(|P| \times 2^{O(|\psi|)}|D|^{|X|}\) and the number of transitions is at most \(|\Delta||\Gamma| \times 2^{O(|\psi|)}|D|^{|X|}\). From Theorem 8, the membership problem can be solved in time \(|P|.|\Delta|.|\Gamma|^2.2^{O(|\psi|)}|D|^{|X|}\) \(|D|^{3.|X|}\).

Thus, we get:

**Theorem 10.** Given a PDS \(\mathcal{P} = (P, \Gamma, \Delta)\), a labeling function \(\lambda : P \rightarrow 2^{APP}\) and a PCARET\(^\wedge\)c formula \(\psi\), for every configuration \(\langle p, \omega \rangle\) and every \(B \in B\), whether or not \(\langle p, \omega \rangle\) satisfies \(\psi\) under \(B\) can be solved in time \(|P|.|\Delta|.|\Gamma|^2.2^{O(|\psi|)}|D|^{|X|}) \cdot |D|^{3.|X|}\). Moreover, if \(\psi\) contains only existential quantifiers, and no free variables, then this problem can be solved in time \(|P|.|\Delta|.|\Gamma|^2.2^{O(|\psi|)}|D|^{3.|X|}\).

**Remark 6.** Formulas that describe malicious behaviors do not involve free variables and contain only existential quantifiers since the malware detection problem consists in determining whether there exists a path or a value of a variable for which the malicious behaviour occurs. Thus, for malware detection, our algorithm has a better complexity than translating the PCARET formula into a CARET formula, and then applying the CARET model checking algorithm.
3.4 SPCARET\textsuperscript{c} Model-Checking For PDS

In this section, we discuss how to do SPCARET\textsuperscript{c} model-checking for PDSs. Let then $P$ be a PDS, $\psi$ be a SPCARET\textsuperscript{c} formula, and $V$ be the set of RVEs occurring in $\psi$. We follow the idea of [ST13a] and use Extended Finite Automata to represent RVEs.

**Definition 13.** Let $P = (P, \Gamma, \Delta)$ be a PDS, let $\omega = \{\alpha, \neg \alpha \mid \alpha \in \Gamma \cup X\}$, an Extended Finite Automaton (EFA) $K$ is a tuple $(Q, \Lambda, q_0, Q_f)$ s.t $Q$ is a finite set of states, $q_0$ is the initial state, $Q_f \subseteq Q$ is a finite set of final states, $\Gamma$ is a finite set of letters, $\Lambda$ is a finite set of transition rules in the form $q_1 \rightarrow \rightarrow q_2$ where $q_1, q_2 \in Q, l \subseteq \omega$.

Let $B \in B$ be an environment, $\gamma \in \Gamma$ be the input letter, $t = q_1 \rightarrow \rightarrow q_2$ be a transition rule in $\Lambda$, assume that $K$ is at state $q_1$, then, $K$ can move to the state $q_2$ under $B$ (denoted $q_1 \rightarrow \rightarrow_B q_2$) iff for every $\alpha \in l$, $B(\alpha) = \gamma$ and for every $\neg \alpha \in l$, $B(\alpha) \neq \gamma$. $K$ accepts a word $\gamma_0 ... \gamma_n$ under $B$ iff $K$ has a run $q_0 \rightarrow \rightarrow_B q_1 ... q_n \rightarrow \rightarrow_B q_{n+1}$ where $q_{n+1} \in Q_f$. Let $L(K)$ be the set of all configurations $(\langle p, \omega \rangle, B) \in P \times \Gamma^* \times B$ s.t $K$ accepts $\omega$ under $B$.

**Proposition 5.** [ST13a] For every RVE $e \in V$, we can compute in polynomial time an EFA $K_e$ s.t $L(e) = L(K_e)$.

To do SPCARET\textsuperscript{c} model-checking for PDSs, we first need to represent each RVE $e \in V$ by an EFA $K$ s.t $K$ recognizes all the configurations $(\langle p, \omega \rangle, B) \in P \times \Gamma^* \times B$ in $L(e)$. Then, we compute a Symbolic Pushdown System (SPDS) $P_0$ which is a kind of product between $P$ and $K^1, ..., K^n$ which allows us to determine whether the stack predicates hold or not only by looking at the top of the stack of $P_0$. Computing the SPDS can be done by adapting the construction of [ST13a]. Then, SPCARET model checking for $P$ is reduced to PCARET model-checking for $P_0$. We adapt then the algorithms in Section 3.3.2 to deal with Symbolic PDSs. Thus, we get that:

**Theorem 11.** Given a PDS $P = (P, \Gamma, \Delta)$, a labeling function $\lambda : P \rightarrow 2^{AP^o}$, and a SPCARET\textsuperscript{c} formula $\psi$, we can construct a GSBPDS $BP_\psi = (P', \Gamma', \Delta', F)$ such that for every configuration $\langle p, \omega \rangle \in P \times \Gamma^*$ and every $B \in B$, $\langle p, \omega \rangle$ satisfies $\psi$ under $B$ iff $(\langle \langle p, \{\psi\} \rangle, B, \omega' \rangle \in L(BP_\psi) \text{ where } \omega' \text{ is obtained by performing the product between } \omega \text{ and the EFAs } K^1, ..., K^n$.

3.5 Experiments

We implemented our algorithms in a tool for malware detection. We use IDA Pro [IDA], Jakstab [KV08] and the translation of [ST12a] to obtain PDSs
from binary code or assembly programs. We use Moped [ES01] to check the emptiness of SBPDSs. A program is considered to be a malware if it satisfies one of the SPCARET formulas presented previously, otherwise, it is a benign program. Our tool was able to detect several malwares and to determine that benign programs are benign as reported in Table 3.1. The \#LOC column shows the number of instructions of the assembly program. The result *Yes* expresses that the binary program is detected as a malware, *No* means the program is found as benign.

Moreover, we compared the performance of our algorithms against the approach that consists in translating SPCARET into CARET with regular valuations, and applying the model checking algorithm for CARET. We set the time limit to 20 minutes. You can see in Table 3.1 that we perform much better in all cases, and that in most cases, the approach that consists in translating SPCARET to CARET timeout.

### 3.6 Conclusion

In this chapter, we introduce the logics PCARET and SPCARET and show how they can precisely and succinctly describe several malicious behaviors that cannot be expressed by other existing specification formalisms. We define the sublogics PCARET^c and SPCARET^c, which are subclasses of PCARET and SPCARET respectively where the time operators on caller paths are removed. We then propose an efficient algorithm for PCARET^c model-checking for PDSs and we show that SPCARET^c model-checking for PDSs can be reduced to PCARET^c model-checking for PDSs. Our algorithms are based on reducing the model checking problem to the emptiness problem of Symbolic Büchi Pushdown Systems. The techniques are implemented in a tool for malware detection. We compared the performance of our new SPCARET tool against our CARET model checking tool. Our new tool behaves much better.
### Table 3.1: Detection of real malwares

<table>
<thead>
<tr>
<th>Malware</th>
<th>#LOC</th>
<th>SPCARET Times(s)</th>
<th>SPCARET Result</th>
<th>CARET Times(s)</th>
<th>CARET Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Backdoor.Win32.Small.dm</td>
<td>433</td>
<td>60.3</td>
<td>Yes</td>
<td>724.6</td>
<td>Yes</td>
</tr>
<tr>
<td>Trojan-Downloader.Win32.Agent.cv</td>
<td>3656</td>
<td>28.7</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Email-Worm.Win32.Bater.a</td>
<td>662</td>
<td>15.3</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Backdoor.Win32.Cocoazul.e</td>
<td>784</td>
<td>19.3</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Backdoor.Win32.Agent.jp</td>
<td>10293</td>
<td>43.4</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>HackTool.Win32.Tups.122</td>
<td>18713</td>
<td>24.6</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Email-Worm.Win32.Bloored.a</td>
<td>9705</td>
<td>17.9</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Email-Worm.Win32.Klez.b</td>
<td>10094</td>
<td>20.9</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Email-Worm.Win32.Krynos.b</td>
<td>18327</td>
<td>16.4</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Email-Worm.Win32.Lohack.b</td>
<td>4887</td>
<td>21.6</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Email-Worm.Win32.Lohack.c</td>
<td>5322</td>
<td>22.2</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Trojan-Downloader.Win32.Agent.dh</td>
<td>35880</td>
<td>29.7</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Email-Worm.Win32.Seliz</td>
<td>5281</td>
<td>32.4</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Email-Worm.Win32.Mydoom.az</td>
<td>3457</td>
<td>29.9</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Trojan-Downloader.Win32.Dadobra.z</td>
<td>19904</td>
<td>37.4</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Trojan-Downloader.Win32.Dyfuca ae</td>
<td>11096</td>
<td>45.9</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Backdoor.Win32.IRCBot.jm</td>
<td>7532</td>
<td>43.3</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Backdoor.Win32.Lithium.10.b5</td>
<td>5164</td>
<td>38.2</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Backdoor.Win32.SdBot.as</td>
<td>6450</td>
<td>45.2</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Backdoor.Win32.SdBot.v</td>
<td>6451</td>
<td>49.5</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Backdoor.Win32.Wisdoor.at</td>
<td>6490</td>
<td>41.1</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Email-Worm.Win32.Ardurk.d</td>
<td>1516</td>
<td>29.5</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Email-Worm.Win32.Cholera</td>
<td>5058</td>
<td>58.3</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Trojan-Downloader.Win32.Apropo.al</td>
<td>17870</td>
<td>41.3</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Trojan-Downloader.Win32.Apropo.bb</td>
<td>17785</td>
<td>32.4</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Virus.Win32.NGVCK.1095</td>
<td>3137</td>
<td>54.2</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Virus.Win32.Redemption.b</td>
<td>1486</td>
<td>23.1</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Email-Worm.Win32.Lohack.a</td>
<td>4887</td>
<td>24.9</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Email-Worm.Win32.Scaline.a</td>
<td>8207</td>
<td>89.9</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>IRC-Worm.Win32.Azrael</td>
<td>3302</td>
<td>58.7</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Net-Worm.Win32.Nimda</td>
<td>8670</td>
<td>60.1</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Trojan-Downloader.Win32.Apropo.bd</td>
<td>27071</td>
<td>53.6</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Trojan-Downloader.Win32.Apropo.s</td>
<td>17902</td>
<td>37.1</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Trojan-Downloader.Win32.Apropo.x</td>
<td>45360</td>
<td>68.9</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Trojan-Downloader.Win32.Dyfuca.ab</td>
<td>11041</td>
<td>113.1</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Email-Worm.Win32.NetSky.a</td>
<td>6352</td>
<td>90.3</td>
<td>Yes</td>
<td>874.2</td>
<td>Yes</td>
</tr>
<tr>
<td>Email-Worm.Win32.Klez.e</td>
<td>14562</td>
<td>70.4</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
<tr>
<td>Email-Worm.Win32.Klez.f</td>
<td>14570</td>
<td>75.8</td>
<td>Yes</td>
<td>timeout</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Benign</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>cmd.exe</td>
<td>35856</td>
<td>112.4</td>
<td>No</td>
<td>timeout</td>
</tr>
<tr>
<td>ipv6.exe</td>
<td>12795</td>
<td>95.3</td>
<td>No</td>
<td>timeout</td>
</tr>
<tr>
<td>shutdown.exe</td>
<td>2525</td>
<td>41.2</td>
<td>No</td>
<td>timeout</td>
</tr>
<tr>
<td>ping.exe</td>
<td>1855</td>
<td>38.2</td>
<td>No</td>
<td>timeout</td>
</tr>
<tr>
<td>print.exe</td>
<td>862</td>
<td>25.5</td>
<td>No</td>
<td>354.35</td>
</tr>
<tr>
<td>dplaysrv.exe</td>
<td>6820</td>
<td>41.5</td>
<td>No</td>
<td>timeout</td>
</tr>
<tr>
<td>regedt.exe</td>
<td>60</td>
<td>1.3</td>
<td>No</td>
<td>2.4</td>
</tr>
<tr>
<td>blastcln.exe</td>
<td>13768</td>
<td>125.6</td>
<td>No</td>
<td>timeout</td>
</tr>
</tbody>
</table>
In this chapter, we define the Branching temporal logic of CALLs and RETurns (BCARET) that allows to write branching temporal formulas while taking into account the matching between calls and returns. We consider the model-checking problem of PDSs against BCARET formulas with "standard" valuations (where an atomic proposition holds at a configuration $c$ or not depends only on the control state of $c$, not on its stack) as well as regular valuations (where the set of configurations in which an atomic proposition holds is regular). We show that these problems can be effectively solved by a reduction to the emptiness problem of Alternating Büchi Pushdown Systems (ABPDSs). We show that our results can be applied for malware detection.

Outline. In Section 4.1, we define the logic BCARET. Section 4.2 shows how BCARET can be used to specify branching-time malicious behaviors. Our algorithm to reduce BCARET model-checking to the membership problem of ABPDSs is presented in Section 4.3. Section 4.4 presents the model-checking problem for PDSs against BCARET formulas with regular valuations. Finally, we conclude in Section 4.5.

4.1 Branching Temporal Logic of Calls and Returns - BCARET

In this section, we define the Branching temporal logic of CALLs and RETurns BCARET. For technical reasons, we assume w.l.o.g. that BCARET formulas are given in positive normal form, i.e. negations are applied only to atomic propositions. To do that, we use the release operator $R$ as a dual of the until operator $U$.

Definition 14. Syntax of BCARET

Let $AP$ be a finite set of atomic propositions, a BCARET formula $\varphi$ is defined as follows, where $b \in \{g,a\}$, $e \in AP$:
Chapter 4. Branching Temporal Logic of Calls and Returns for Pushdown Systems

Let $\mathcal{P} = (\mathcal{P}, \Gamma, \Delta)$ be a PDS, $\lambda : AP \to 2^{\mathcal{P} \times \Gamma^*}$ be a labelling function that assigns to each atomic proposition $e \in AP$ a set of configurations of $\mathcal{P}$. The satisfiability relation of a BCAREt formula $\varphi$ at a configuration $\langle p_0, \omega_0 \rangle$ w.r.t. the labelling function $\lambda$, denoted by $\langle p_0, \omega_0 \rangle \Vdash_\lambda \varphi$, is defined inductively as follows:

- $\langle p_0, \omega_0 \rangle \models_\lambda true$ for every $\langle p_0, \omega_0 \rangle$
- $\langle p_0, \omega_0 \rangle \not\models_\lambda false$ for every $\langle p_0, \omega_0 \rangle$
- $\langle p_0, \omega_0 \rangle \models_\lambda e (e \in AP)$ iff $\langle p_0, \omega_0 \rangle \in \lambda(e)$
- $\langle p_0, \omega_0 \rangle \models_\lambda \neg e (e \in AP)$ iff $\langle p_0, \omega_0 \rangle \not\in \lambda(e)$
- $\langle p_0, \omega_0 \rangle \models_\lambda \varphi_1 \lor \varphi_2$ iff ($\langle p_0, \omega_0 \rangle \models_\lambda \varphi_1$ or $\langle p_0, \omega_0 \rangle \models_\lambda \varphi_2$)
- $\langle p_0, \omega_0 \rangle \models_\lambda \varphi_1 \land \varphi_2$ iff ($\langle p_0, \omega_0 \rangle \models_\lambda \varphi_1$ and $\langle p_0, \omega_0 \rangle \models_\lambda \varphi_2$)
- $\langle p_0, \omega_0 \rangle \models_\lambda EX^g \varphi$ iff there exists a global-successor $\langle p', \omega' \rangle$ of $\langle p_0, \omega_0 \rangle$ such that $\langle p', \omega' \rangle \models_\lambda \varphi$
- $\langle p_0, \omega_0 \rangle \models_\lambda AX^g \varphi$ iff $\langle p', \omega' \rangle \models_\lambda \varphi$ for every global-successor $\langle p', \omega' \rangle$ of $\langle p_0, \omega_0 \rangle$
- $\langle p_0, \omega_0 \rangle \models_\lambda E[\varphi_1 U^g \varphi_2]$ iff there exists a global-path $\pi = \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \langle p_2, \omega_2 \rangle \ldots$ of $\mathcal{P}$ starting from $\langle p_0, \omega_0 \rangle$ s.t. $\exists i \geq 0, \langle p_i, \omega_i \rangle \models_\lambda \varphi_2$ and for every $0 \leq j < i$, $\langle p_j, \omega_j \rangle \models_\lambda \varphi_1$
- $\langle p_0, \omega_0 \rangle \models_\lambda A[\varphi_1 U^g \varphi_2]$ iff for every global-path $\pi = \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \langle p_2, \omega_2 \rangle \ldots$ of $\mathcal{P}$ starting from $\langle p_0, \omega_0 \rangle$, $\exists i \geq 0, \langle p_i, \omega_i \rangle \models_\lambda \varphi_2$ and for every $0 \leq j < i$, $\langle p_j, \omega_j \rangle \models_\lambda \varphi_1$
- $\langle p_0, \omega_0 \rangle \models_\lambda E[\varphi_1 R^g \varphi_2]$ iff there exists a global-path $\pi = \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \langle p_2, \omega_2 \rangle \ldots$ of $\mathcal{P}$ starting from $\langle p_0, \omega_0 \rangle$, for every $i \geq 0$, if $\langle p_i, \omega_i \rangle \models_\lambda \varphi_2$ then there exists $0 \leq j < i$ s.t. $\langle p_j, \omega_j \rangle \models_\lambda \varphi_1$
- $\langle p_0, \omega_0 \rangle \models_\lambda A[\varphi_1 R^g \varphi_2]$ iff for every global-path $\pi = \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \langle p_2, \omega_2 \rangle \ldots$ of $\mathcal{P}$ starting from $\langle p_0, \omega_0 \rangle$, for every $i \geq 0$, if $\langle p_i, \omega_i \rangle \models_\lambda \varphi_2$ then there exists $0 \leq j < i$ s.t. $\langle p_j, \omega_j \rangle \models_\lambda \varphi_1$
- $\langle p_0, \omega_0 \rangle \models_\lambda EX^a \varphi$ iff there exists an abstract-successor $\langle p', \omega' \rangle$ of $\langle p_0, \omega_0 \rangle$ such that $\langle p', \omega' \rangle \models_\lambda \varphi$
4.2. Application

4.2. Application

- \( (p_0, \omega_0) \models_\lambda AX^a \varphi \) iff \( (p', \omega') \models_\lambda \varphi \) for every abstract-successor \( (p', \omega') \) of \( (p_0, \omega_0) \)

- \( (p_0, \omega_0) \models_\lambda E[\varphi U^a \varphi_2] \) iff there exists an abstract-path \( \pi = (p_0, \omega_0)(p_1, \omega_1)(p_2, \omega_2) \ldots \) of \( P \) starting from \( (p_0, \omega_0) \) s.t. \( \exists i \geq 0, (p_i, \omega_i) \models_\lambda \varphi_2 \) and for every \( 0 \leq j < i, (p_j, \omega_j) \models_\lambda \varphi_1 \)

- \( (p_0, \omega_0) \models_\lambda A[\varphi U^a \varphi_2] \) iff for every abstract-path \( \pi = (p_0, \omega_0)(p_1, \omega_1)(p_2, \omega_2) \ldots \) of \( P \), \( \exists i \geq 0, (p_i, \omega_i) \models_\lambda \varphi_2 \) and for every \( 0 \leq j < i, (p_j, \omega_j) \models_\lambda \varphi_1 \)

- \( (p_0, \omega_0) \models_\lambda E[\varphi R^a \varphi_2] \) iff there exists an abstract-path \( \pi = (p_0, \omega_0)(p_1, \omega_1)(p_2, \omega_2) \ldots \) of \( P \) starting from \( (p_0, \omega_0) \) s.t. for every \( i \geq 0 \), if \( (p_i, \omega_i) \not\models_\lambda \varphi_2 \) then there exists \( 0 \leq j < i \) s.t. \( (p_j, \omega_j) \models_\lambda \varphi_1 \)

- \( (p_0, \omega_0) \models_\lambda A[\varphi R^a \varphi_2] \) iff for every abstract-path \( \pi = (p_0, \omega_0)(p_1, \omega_1)(p_2, \omega_2) \ldots \) of \( P \) starting from \( (p_0, \omega_0) \), for every \( i \geq 0 \), if \( (p_i, \omega_i) \not\models_\lambda \varphi_2 \) then there exists \( 0 \leq j < i \) s.t. \( (p_j, \omega_j) \models_\lambda \varphi_1 \)

Other BCARET operators can be expressed by the above operators: \( EF^g \varphi = E[true U^g \varphi], \ EF^a \varphi = E[true U^a \varphi], \ AF^g \varphi = A[true U^g \varphi], \ AF^a \varphi = A[true U^a \varphi], \ldots \)

Closure. Given a BCARET formula \( \varphi \), the closure \( Cl(\varphi) \) is the set of all subformulae of \( \varphi \), including \( \varphi \).

Regular Valuations. As previously, we talk about regular valuations when for every \( e \in AP, \lambda(e) \) is a regular language.

Remark 7. CTL can be seen as the subclass of BCARET where the operators \( EX^a \varphi, AX^a \varphi, E[\varphi U^a \varphi], A[\varphi U^a \varphi], E[\varphi R^a \varphi], A[\varphi R^a \varphi] \) are not considered.

4.2 Application

In this section, we show how BCARET can be used to describe branching-time malicious behaviors.

Spyware Behavior. The typical behaviour of a spyware is hunting for personal information (emails, bank account information,...) on local drives by searching files matching certain conditions. To do that, it has to search directories of the host to look for interesting files whose names match a specific condition. When a file is found, the spyware will invoke a payload to steal the information, then continue looking for the remaining matching files. When a folder is found, it will enter the folder path and continue scanning.
that folder recursively. To achieve this behavior, the spyware first calls the API function `FindFirstFileA` to search for the first matching file in a given folder path. After that, it has to check whether the call to the API function `FindFirstFileA` succeeds or not. If the function call fails, the spyware will call the function `GetLastError`. Otherwise, if the function call is successful, `FindFirstFileA` will return a search handle `h`. There are two possibilities in this case. If the returned result is a folder, it will call the API function `FindFirstFileA` again to search for matching results in the found folder. If the returned result is a file, it will call the API function `FindNextFileA` using `h` as first parameter to look for the remaining matching files. This behavior cannot be expressed by LTL or CTL because it requires to express that the return value of the function `FindFirstFileA` should be used as input to the API function `FindNextFileA`. It cannot be described by CARET neither (because this is a branching-time property). Using BCARET, the above behavior can be expressed by the following formula:

\[
\varphi_{sb} = \bigvee_{d \in D} EF^g \left( call(FindFirstFileA) \land EX^a(eax = d) \land AF^a \left( call(GetLastError) \lor call(FindFirstFileA) \lor \left( call(FindNextFileA) \land d \Gamma^* \right) \right) \right)
\]

where the \( \bigvee \) is taken over all possible memory addresses \( d \) which contain the values of search handles \( h \) in the program, \( EX^a \) is a BCARET operator that means "next in some run, in the same procedural context"; \( EF^g \) is the standard CTL \( EF \) operator (eventually in some run), while \( AF^a \) is a BCARET operator that means "eventually in all runs, in the same procedural context".

As mentioned previously, in binary codes and assembly programs, the return value of an API function is put in the register `eax`. Thus, the return value of `FindFirstFileA` is the value of `eax` at its corresponding return-point. Then, the subformula \( call(FindFirstFileA) \land EX^a(eax = d) \) states that there is a call to the API `FindFirstFileA` and the return value of this function is \( d \) (the abstract successor of a call is its corresponding return-point). When `FindNextFileA` is invoked, it requires a search handle as parameter and this search handle must be put on top of the program stack (since parameters are passed through the stack in assembly). The requirement that \( d \) is on top of the program stack is expressed by the regular expression \( d \Gamma^* \). Thus, the subformula \( [call(FindNextFileA) \land d \Gamma^*] \) expresses that `FindNextFileA` is called with \( d \) as parameter (\( d \) stores the information of the search handle).
Therefore, $\varphi_{sb}$ expresses that there is a call to the API $\text{FindFirstFileA}$ with the return value $d$ (the search handle), then, in all runs starting from that call, there will be either a call to the API function $\text{GetLastError}$ or a call to the function $\text{FindFirstFileA}$ or a call to the function $\text{FindNextFileA}$ in which $d$ is used as a parameter. Note that this specification of spyware is more precise than the one described in the introduction since the description in the introduction does not deal with the case when the returned result of $\text{FindFirstFileA}$ is a folder or an error. BCARET can deal with it because BCARET is a branching-time temporal logic. For example, $AF^a$ allows us to take into account all possible abstract-paths from a certain state in the computation tree. By using $AF^a$, $\varphi_{sb}$ can deal with different returned values of $\text{FindFirstFileA}$ as presented above.

4.3 BCARET Model-Checking for Pushdown Systems

In this section, we consider "standard" BCARET model-checking for pushdown systems where an atomic proposition holds at a configuration $c$ or not depends only on the control state of $c$, not on its stack.

4.3.1 Alternating Büchi Pushdown Systems (ABPDSs).

Definition 15. An Alternating Büchi Pushdown System (ABPDS) is a tuple $\mathcal{BP} = (P, \Gamma, \Delta, F)$, where $P$ is a set of control locations, $\Gamma$ is the stack alphabet, $F \subseteq P$ is a set of accepting control locations and $\Delta$ is a transition function that maps each element of $P \times \Gamma$ with a positive boolean formula over $P \times \Gamma^*$.

A configuration of $\mathcal{BP}$ is a pair $\langle p, \omega \rangle$, where $p \in P$ is the current control location and $\omega \in \Gamma^*$ is the current stack content. Without loss of generality, we suppose that the boolean formulas of ABPDSs are in disjunctive normal form $\bigvee_{j=1}^{n} \bigwedge_{i=1}^{m_j} \langle p_j^i, \omega_j^i \rangle$. Then, we can see $\Delta$ as a subset of $(P \times \Gamma) \times 2^{P \times \Gamma^*}$ by rewriting the rules of $\Delta$ in the form $\langle p, \gamma \rangle \rightarrow \bigvee_{j=1}^{n} \bigwedge_{i=1}^{m_j} \langle p_j^i, \omega_j^i \rangle$ as $n$ rules of the form $\langle p, \gamma \rangle \rightarrow \{\langle p_1^i, \omega_1^i \rangle, \ldots, \langle p_n^i, \omega_n^i \rangle\}$, where $1 \leq j \leq n$. Let $\langle p, \gamma \rangle \rightarrow \{\langle p_1^1, \omega_1^1 \rangle, \ldots, \langle p_n^1, \omega_n^1 \rangle\}$ be a rule of $\Delta$, then, for every $\omega \in \Gamma^*$, the configuration $\langle p, \omega \rangle$ (resp. $\{\langle p_1^1, \omega_1^1 \omega \rangle, \ldots, \langle p_n^1, \omega_n^1 \omega \rangle\}$) is an immediate predecessor (resp. successor) of $\{\langle p_1^1, \omega_1^2 \omega \rangle, \ldots, \langle p_n^1, \omega_n^2 \omega \rangle\}$ (resp. $\langle p, \gamma \omega \rangle$).

A run $\rho$ of $\mathcal{BP}$ starting form an initial configuration $\langle p_0, \omega_0 \rangle$ is a tree whose root is labelled by $\langle p_0, \omega_0 \rangle$, and whose other nodes are labelled by elements in $P \times \Gamma^*$. If a node of $\rho$ is labelled by a configuration $\langle p, \omega \rangle$ and has $n$ children labelled by $\langle p_1, \omega_1 \rangle, \ldots, \langle p_n, \omega_n \rangle$ respectively, then, $\langle p, \omega \rangle$ must be a predecessor
of \{⟨p_1, \omega_1⟩, ..., ⟨p_n, \omega_n⟩\} in \mathcal{BP}. A path of a run ρ is an infinite sequence of configurations \(c_0 c_1 c_2 \ldots\). s.t. \(c_0\) is the root of \(ρ\) and \(c_{i+1}\) is one of the children of \(c_i\) for every \(i \geq 0\). A path is accepting iff it visits infinitely often configurations with control locations in \(F\). A run \(ρ\) is accepting iff every path of \(ρ\) is accepting.

The language of \(\mathcal{BP}, \mathcal{L}(\mathcal{BP})\), is the set of configurations \(c\) s.t. \(\mathcal{BP}\) has an accepting run starting from \(c\).

\(\mathcal{BP}\) defines the reachability relation \(\Rightarrow_{\mathcal{BP}}\) as follows: (1) \(c \Rightarrow_{\mathcal{BP}} \{c\}\) for every \(c \in P \times \Gamma^*\), (2) \(c \Rightarrow_{\mathcal{BP}} C\) if \(C\) is an immediate successor of \(c\); (3) if \(c \Rightarrow_{\mathcal{BP}} \{c_1, c_2, \ldots, c_n\}\) and \(c_i \Rightarrow_{\mathcal{BP}} \tilde{C}_i\) for every \(1 \leq i \leq n\), then \(c \Rightarrow_{\mathcal{BP}} \bigcup_{i=1}^n \tilde{C}_i\).

Given \(c_0 \Rightarrow_{\mathcal{BP}} C\), then, \(\mathcal{BP}\) has an accepting run from \(c_0\) iff \(\mathcal{BP}\) has an accepting run from \(c'\) for every \(c' \in C'\).

**Theorem 12.** [ST11] Given an ABPDS \(\mathcal{BP} = (P, \Gamma, \Delta, F)\), for every configuration \((p, \omega) \in P \times \Gamma^*\), whether or not \((p, \omega) \in \mathcal{L}(\mathcal{BP})\) can be decided in time \(O(|P|^2, |\Gamma|, (|\Delta|2^{5|P|} + 2^{|P|}|\omega|))\).

### 4.3.2 From BCARET model checking of PDSs to the membership problem in ABPDSs

Let \(\mathcal{P} = (P, \Gamma, \Delta)\) be a pushdown system with an initial configuration \(c_0\). Given a set of atomic propositions \(AP\), let \(\varphi\) be a BCARET formula. Let \(f : AP \rightarrow 2^P\) be a function that associates each atomic proposition with a set of control states, and \(\lambda_f : AP \rightarrow 2^{P \times \Gamma^*}\) be a labelling function s.t. for every \(e \in AP\), \(\lambda_f(e) = \{(p, \omega) \mid p \in f(e), \omega \in \Gamma^*\}\). In this section, we propose an algorithm to check whether \(c_0 \models_{\lambda_f} \varphi\). Intuitively, we construct an Alternating Büchi Pushdown System \(\mathcal{BP}_\varphi\) which recognizes a configuration \(c\) iff \(c \models_{\lambda_f} \varphi\). Then to check whether \(c_0 \models_{\lambda_f} \varphi\), we will check if \(c_0 \in \mathcal{L}(\mathcal{BP}_\varphi)\).

The membership problem of an ABPDS can be solved effectively by Theorem 12.

Let \(\mathcal{BP}_\varphi = (P', \Gamma', \Delta', F)\) be the ABPDS defined as follows:

- \(P' = P \cup (P \times Cl(\varphi)) \cup \{p_\bot\}\)
- \(\Gamma' = \Gamma \cup (\Gamma \times Cl(\varphi)) \cup \{\gamma_\bot\}\)
- \(F = F_1 \cup F_2 \cup F_3\) where
  
  - \(F_1 = \{(p, e) \mid e \in Cl(\varphi), e \in AP\text{ and } p \in f(e)\}\)
  
  - \(F_2 = \{(p, \neg e) \mid \neg e \in Cl(\varphi), e \in AP\text{ and } p \notin f(e)\}\)
  
  - \(F_3 = \{P \times Cl_{R}(\varphi)\}\text{ where } Cl_{R}(\varphi)\) is the set of formulas of \(Cl(\varphi)\) in the form \(E[\varphi_1 R^b \varphi_2]\text{ or } A[\varphi_1 R^b \varphi_2] \text{ (}b \in \{g, a\}\)
4.3. BCARET Model-Checking for Pushdown Systems 87

The transition relation $\Delta'$ is the smallest set of transition rules defined as follows: $\Delta \subseteq \Delta'$ and for every $p \in P$, $\phi \in Cl(\varphi)$, $\gamma \in \Gamma$, $b \in \{g, a\}$ and $t \in \{\text{call, ret, int}\}$:

(a1) If $\phi = e$, $e \in AP$ and $p \in f(e)$, then, $\langle \langle p, \phi \rangle, \gamma \rangle \rightarrow \langle \langle p, \phi \rangle, \gamma \rangle \in \Delta'$

(a2) If $\phi = \neg e$, $e \in AP$ and $p \notin f(e)$, then, $\langle \langle p, \phi \rangle, \gamma \rangle \rightarrow \langle \langle p, \phi \rangle, \gamma \rangle \in \Delta'$

(a3) If $\phi = \phi_1 \land \phi_2$, then, $\langle \langle p, \phi \rangle, \gamma \rangle \rightarrow \langle \langle p, \phi_1 \rangle, \gamma \rangle \land \langle \langle p, \phi_2 \rangle, \gamma \rangle \in \Delta'$

(a4) If $\phi = \phi_1 \lor \phi_2$, then, $\langle \langle p, \phi \rangle, \gamma \rangle \rightarrow \langle \langle p, \phi_1 \rangle, \gamma \rangle \lor \langle \langle p, \phi_2 \rangle, \gamma \rangle \in \Delta'$

(a5) If $\phi = EX^g\phi_1$, then $\langle \langle p, \phi \rangle, \gamma \rangle \rightarrow \bigvee_{(p, q) \in \Delta}(\langle q, \phi_1 \rangle, \omega) \in \Delta'$ where $t \in \{\text{call, int, ret}\}$

(a6) If $\phi = AX^g\phi_1$, then $\langle \langle p, \phi \rangle, \gamma \rangle \rightarrow h_1 \lor h_2 \lor h_3 \in \Delta'$, where

- $h_1 = \bigvee_{(p, q) \in (q, \gamma \gamma \gamma \gamma) \in \Delta}(q, \gamma_1 \gamma_2, \phi_1)$
- $h_2 = \bigvee_{(p, q) \in (q, \gamma \gamma \gamma \gamma) \in \Delta}(q, \phi_1 \gamma)$
- $h_3 = \bigvee_{(p, q) \in (p, \gamma \gamma \gamma \gamma) \in \Delta}(p, p)$

(a7) If $\phi = AX^a\phi_1$, then $\langle \langle p, \phi \rangle, \gamma \rangle \rightarrow h_1 \land h_2 \land h_3 \in \Delta'$, where

- $h_1 = \bigwedge_{(p, q) \in (q, \gamma \gamma \gamma \gamma) \in \Delta}(q, \gamma_1 \gamma_2, \phi_1)$
- $h_2 = \bigwedge_{(p, q) \in (q, \gamma \gamma \gamma \gamma) \in \Delta}(q, \phi_1 \gamma)$
- $h_3 = \bigwedge_{(p, q) \in (q, \gamma \gamma \gamma \gamma) \in \Delta}(p, p)$

(a8) If $\phi = EX^a\phi_1$, then $\langle \langle p, \phi \rangle, \gamma \rangle \rightarrow h_1 \lor h_2 \lor h_3 \in \Delta'$, where

- $h_1 = \bigwedge_{(p, q) \in (q, \gamma \gamma \gamma \gamma) \in \Delta}(q, \gamma_1 \gamma_2, \phi_1)$
- $h_2 = \bigwedge_{(p, q) \in (q, \gamma \gamma \gamma \gamma) \in \Delta}(q, \phi_1 \gamma)$
- $h_3 = \bigwedge_{(p, q) \in (q, \gamma \gamma \gamma \gamma) \in \Delta}(p, p)$

(a9) If $\phi = E[\phi_1 U^g\phi_2]$, then,

$\langle \langle p, \phi \rangle, \gamma \rangle \rightarrow \langle \langle p, \phi_2 \rangle, \gamma \rangle \lor \bigvee_{(p, q) \in (q, \gamma \gamma \gamma \gamma) \in \Delta}(\langle q, \phi_1 \rangle, \gamma) \land \langle q, \phi_1 \rangle, \gamma \rangle \in \Delta'$

(a10) If $\phi = E[\phi_1 U^a\phi_2]$, then, $\langle \langle p, \phi \rangle, \gamma \rangle \rightarrow \langle \langle p, \phi_2 \rangle, \gamma \rangle \lor h_1 \lor h_2 \lor h_3 \in \Delta'$, where

- $h_1 = \bigvee_{(p, q) \in (q, \gamma \gamma \gamma \gamma) \in \Delta}(q, \gamma_1 \gamma_2, \phi_1)$
- $h_2 = \bigvee_{(p, q) \in (q, \gamma \gamma \gamma \gamma) \in \Delta}(q, \phi_1 \gamma)$
- $h_3 = \bigvee_{(p, q) \in (q, \gamma \gamma \gamma \gamma) \in \Delta}(p, p)$
Chapter 4. Branching Temporal Logic of Calls and Returns for Pushdown Systems

(a11) If $\phi = A[\phi_1 U^g \phi_2]$, then,

$$\langle \langle p, \phi \rangle, \gamma \rangle \rightarrow \langle \langle p, \phi_2 \rangle, \gamma \rangle \lor \bigwedge_{(p, \gamma) \rightarrow (q, \omega) \in \Delta} (\langle \langle p, \phi_1 \rangle, \gamma \rangle \land \langle \langle q, \phi \rangle, \omega \rangle) \in \Delta'$$

(a12) If $\phi = A[\phi_1 U^a \phi_2]$, then,

$$\langle \langle p, \phi \rangle, \gamma \rangle \rightarrow \langle \langle p, \phi_2 \rangle, \gamma \rangle \lor (h_1 \land h_2 \land h_3) \in \Delta', \text{ where}$$

- $h_1 = \bigwedge_{(p, \gamma) \xrightarrow{\text{call}} (q, \gamma_1 \gamma_2) \in \Delta} (\langle \langle p, \phi_1 \rangle, \gamma \rangle \land \langle q, \gamma_1 \langle \gamma_2, \phi \rangle \rangle)$
- $h_2 = \bigwedge_{(p, \gamma) \xrightarrow{\text{int}} (q, \omega) \in \Delta} (\langle \langle p, \phi_1 \rangle, \gamma \rangle \land \langle \langle q, \phi \rangle, \omega \rangle)$
- $h_3 = \bigwedge_{(p, \gamma) \xrightarrow{\text{ret}} (q, \varepsilon) \in \Delta} (p_\perp, \gamma_\perp)$

(a13) If $\phi = E[\phi_1 R^g \phi_2]$, then, we add to $\Delta'$ the rule:

$$\langle \langle p, \phi \rangle, \gamma \rangle \rightarrow ((\langle \langle p, \phi_2 \rangle, \gamma \rangle \land \langle \langle p, \phi_1 \rangle, \gamma \rangle) \lor \bigvee_{(p, \gamma) \xrightarrow{\text{call}} (q, \omega) \in \Delta} ((\langle \langle p, \phi_2 \rangle, \gamma \rangle \land \langle \langle q, \phi \rangle, \omega \rangle))$$

(a14) If $\phi = A[\phi_1 R^a \phi_2]$, then, we add to $\Delta'$ the rule:

$$\langle \langle p, \phi \rangle, \gamma \rangle \rightarrow ((\langle \langle p, \phi_2 \rangle, \gamma \rangle \land \langle \langle p, \phi_1 \rangle, \gamma \rangle) \lor \bigwedge_{(p, \gamma) \xrightarrow{\text{call}} (q, \omega) \in \Delta} ((\langle \langle p, \phi_2 \rangle, \gamma \rangle \land \langle \langle q, \phi \rangle, \omega \rangle))$$

(a15) If $\phi = E[\phi_1 R^a \phi_2]$: $\langle \langle p, \phi \rangle, \gamma \rangle \rightarrow ((\langle \langle p, \phi_2 \rangle, \gamma \rangle \land \langle \langle p, \phi_1 \rangle, \gamma \rangle) \lor h_1 \lor h_2 \lor h_3 \in \Delta'$, where

- $h_1 = \bigvee_{(p, \gamma) \xrightarrow{\text{call}} (q, \gamma_1 \gamma_2) \in \Delta} (\langle \langle p, \phi_2 \rangle, \gamma \rangle \land \langle q, \gamma_1 \langle \gamma_2, \phi \rangle \rangle)$
- $h_2 = \bigvee_{(p, \gamma) \xrightarrow{\text{int}} (q, \omega) \in \Delta} (\langle \langle p, \phi_2 \rangle, \gamma \rangle \land \langle \langle q, \phi \rangle, \omega \rangle)$
- $h_3 = \bigvee_{(p, \gamma) \xrightarrow{\text{ret}} (q, \varepsilon) \in \Delta} (p_\perp, \gamma_\perp)$

(a16) If $\phi = A[\phi_1 R^a \phi_2]$, $\langle \langle p, \phi \rangle, \gamma \rangle \rightarrow ((\langle \langle p, \phi_2 \rangle, \gamma \rangle \land \langle \langle p, \phi_1 \rangle, \gamma \rangle) \lor (h_1 \land h_2 \land h_3) \in \Delta'$, where

- $h_1 = \bigwedge_{(p, \gamma) \xrightarrow{\text{call}} (q, \gamma_1 \gamma_2) \in \Delta} (\langle \langle p, \phi_2 \rangle, \gamma \rangle \land \langle q, \gamma_1 \langle \gamma_2, \phi \rangle \rangle)$
- $h_2 = \bigwedge_{(p, \gamma) \xrightarrow{\text{int}} (q, \omega) \in \Delta} (\langle \langle p, \phi_2 \rangle, \gamma \rangle \land \langle \langle q, \phi \rangle, \omega \rangle)$
- $h_3 = \bigwedge_{(p, \gamma) \xrightarrow{\text{ret}} (q, \varepsilon) \in \Delta} (p_\perp, \gamma_\perp)$

(a17) for every $\langle p, \gamma \rangle \xrightarrow{\text{ret}} \langle q, \varepsilon \rangle \in \Delta$:

- $\langle q, \langle \gamma'', \phi_1 \rangle \rangle \rightarrow \langle \langle q, \phi_1 \rangle, \gamma'' \rangle \in \Delta'$ for every $\gamma'' \in \Gamma, \phi_1 \in CL(\varphi)$

(a18) $\langle p_\perp, \gamma_\perp \rangle \rightarrow \langle p_\perp, \gamma_\perp \rangle \in \Delta'$
Roughly speaking, the ABPDS $\mathcal{BP}_\varphi$ is a kind of product between $\mathcal{P}$ and the BCARET formula $\varphi$ which ensures that $\mathcal{BP}_\varphi$ has an accepting run from $\langle \langle p, \varphi \rangle, \omega \rangle$ iff the configuration $\langle p, \omega \rangle$ satisfies $\varphi$. The form of the control locations of $\mathcal{BP}_\varphi$ is $\langle p, \phi \rangle$ where $\phi \in Cl(\varphi)$. Let us explain the intuition behind our construction:

- If $\phi = e \in AP$, then, for every $\omega \in \Gamma^*$, $\langle p, \omega \rangle \models \lambda_e \phi$ iff $p \in f(e)$. In other words, $\mathcal{BP}_\varphi$ should have an accepting run from $\langle \langle p, e \rangle, \omega \rangle$ iff $p \in f(e)$. This is ensured by the transition rules in (a1) which add a loop at $\langle \langle p, e \rangle, \omega \rangle$ where $p \in f(e)$ and the fact that $\langle p, e \rangle \in F$.

- If $\phi = \neg e (e \in AP)$, then, for every $\omega \in \Gamma^*$, $\langle p, \omega \rangle \models \lambda_e \phi$ iff $p \notin f(e)$. In other words, $\mathcal{BP}_\varphi$ should have an accepting run from $\langle \langle p, \neg e \rangle, \omega \rangle$ iff $p \notin f(e)$. This is ensured by the transition rules in (a2) which add a loop at $\langle \langle p, \neg e \rangle, \omega \rangle$ where $p \notin f(e)$ and the fact that $\langle p, \neg e \rangle \in F$.

- If $\phi = \phi_1 \land \phi_2$, then, for every $\omega \in \Gamma^*$, $\langle p, \omega \rangle \models \lambda_j \phi$ iff $\langle p, \omega \rangle \models \lambda_j \phi_1$ and $\langle p, \omega \rangle \models \lambda_j \phi_2$. This is ensured by the transition rules in (a3) stating that $\mathcal{BP}_\varphi$ has an accepting run from $\langle \langle p, \phi_1 \land \phi_2 \rangle, \omega \rangle$ iff $\mathcal{BP}_\varphi$ has an accepting run from both $\langle \langle p, \phi_1 \rangle, \omega \rangle$ and $\langle \langle p, \phi_2 \rangle, \omega \rangle$. (a4) is similar to (a3).

- If $\phi = E[\phi_1 U^g \phi_2]$, then, for every $\omega \in \Gamma^*$, $\langle p, \omega \rangle \models \lambda_j \phi$ iff $\langle p, \omega \rangle \models \lambda_j \phi_2$ or $\langle p, \omega \rangle \models \lambda_j \phi_1$ and there exists an immediate successor $\langle p', \omega' \rangle$ of $\langle p, \omega \rangle$ s.t. $\langle p', \omega' \rangle \models \lambda_j \phi$. This is ensured by the transition rules in (a9) stating that $\mathcal{BP}_\varphi$ has an accepting run from $\langle \langle p, E[\phi_1 U^g \phi_2] \rangle, \omega \rangle$ iff $\mathcal{BP}_\varphi$ has an accepting run from $\langle \langle p, \phi_2 \rangle, \omega \rangle$ or $\mathcal{BP}_\varphi$ has an accepting run from both $\langle \langle p, \phi_1 \rangle, \omega \rangle$ and $\langle \langle p', \phi_1 \rangle, \omega' \rangle$ where $\langle p', \omega' \rangle$ is an immediate successor of $\langle p, \omega \rangle$. (a11) is similar to (a9).

- If $\phi = E[\phi_1 R^g \phi_2]$, then, for every $\omega \in \Gamma^*$, $\langle p, \omega \rangle \models \lambda_j \phi$ iff $\langle p, \omega \rangle \models \lambda_j \phi_2$ and $\langle p, \omega \rangle \models \lambda_j \phi_1$ or $\langle p, \omega \rangle \models \lambda_j \phi_2$ and there exists an immediate successor $\langle p', \omega' \rangle$ of $\langle p, \omega \rangle$ s.t. $\langle p', \omega' \rangle \models \lambda_j \phi$. This is ensured by the transition rules in (a13) stating that $\mathcal{BP}_\varphi$ has an accepting run from $\langle \langle p, E[\phi_1 R^g \phi_2] \rangle, \omega \rangle$ iff $\mathcal{BP}_\varphi$ has an accepting run from both $\langle \langle p, \phi_2 \rangle, \omega \rangle$ and $\langle \langle p, \phi_1 \rangle, \omega \rangle$; or $\mathcal{BP}_\varphi$ has an accepting run from both $\langle \langle p, \phi_2 \rangle, \omega \rangle$ and $\langle \langle p', \phi_1 \rangle, \omega' \rangle$ where $\langle p', \omega' \rangle$ is an immediate successor of $\langle p, \omega \rangle$. In addition, for $R^g$ formulas, the stop condition is not required, i.e, for a formula $\phi_1 R^g \phi_2$ that is applied to a specific run, we don’t require that $\phi_1$ must eventually hold. To ensure that the runs on which $\phi_2$ always holds are accepted, we add $\langle p, \phi \rangle$ to the Büchi accepting condition $F$ (via the subset $F_3$ of $F$). (a14) is similar to (a13).
Chapter 4. Branching Temporal Logic of Calls and Returns for Pushdown Systems

- If $\phi = EX^a \phi_1$, then, for every $\omega \in \Gamma^*$, $\langle p, \omega \rangle \models \lambda_f \phi$ iff there exists an immediate successor $\langle p', \omega' \rangle$ of $\langle p, \omega \rangle$ s.t. $\langle p', \omega' \rangle \models \lambda_f \phi_1$. This is ensured by the transition rules in (a5) stating that $BP_{\varphi}$ has an accepting run from $\langle \langle p, EX^a \phi_1 \rangle, \omega \rangle$ iff there exists an immediate successor $\langle p', \omega' \rangle$ of $\langle p, \omega \rangle$ s.t. $BP_{\varphi}$ has an accepting run from $\langle \langle p', \phi_1 \rangle, \omega' \rangle$. (a6) is similar to (a5).

![Diagram](image)

Figure 4.1: $\langle p, \omega \rangle \Rightarrow_{P} \langle p', \omega' \rangle$ corresponds to a call statement

- If $\phi = EX^a \phi_1$, then, for every $\omega \in \Gamma^*$, $\langle p, \omega \rangle \models \lambda_f \phi$ iff there exists an abstract-successor $\langle p_k, \omega_k \rangle$ of $\langle p, \omega \rangle$ s.t. $\langle p_k, \omega_k \rangle \models \lambda_f \phi_1$ (A1). Let $\pi \in Traces(\langle p, \omega \rangle)$ be a run starting from $\langle p, \omega \rangle$ on which $\langle p_k, \omega_k \rangle$ is the abstract-successor of $\langle p, \omega \rangle$. Over $\pi$, let $\langle p', \omega' \rangle$ be the immediate successor of $\langle p, \omega \rangle$. In what follows, we explain how we can ensure (A1).

1. Firstly, we show that for every abstract-successor $\langle p_k, \omega_k \rangle \neq \bot$ of $\langle p, \omega \rangle$, $\langle \langle p, EX^a \phi_1 \rangle, \omega \rangle \Rightarrow_{BP_{\varphi}} \langle \langle p_k, \phi_1 \rangle, \omega_k \rangle$. There are two possibilities:

   - If $\langle p, \omega \rangle \Rightarrow_{P} \langle p', \omega' \rangle$ corresponds to a call statement. Let us consider Figure 4.1 to explain this case. $\langle \langle p, \phi \rangle, \omega \rangle \Rightarrow_{BP_{\varphi}} \langle \langle p_k, \phi_1 \rangle, \omega_k \rangle$ is ensured by rules corresponding to $h_1$ in (a7), the rules in $\Delta \subseteq \Delta'$ and the rules in (a17) as follows: rules corresponding to $h_1$ in (a7) allow to record $\phi_1$ in the return point of the call, rules in $\Delta \subseteq \Delta'$ allow to mimic the run of the PDS $P$ and rules in (a17) allow to extract and put back $\phi_1$ when the return-point is reached. In what follows, we show in more details how this works:

     Let $\langle p, \gamma \rangle \xrightarrow{call} \langle p', \gamma' \gamma'' \rangle$ be the rule associated with the transition $\langle p, \omega \rangle \Rightarrow_{P} \langle p', \omega' \rangle$, then we have $\omega = \gamma \omega''$ and $\omega' = \gamma' \gamma'' \omega''$. Let $\langle p_{k-1}, \omega_{k-1} \rangle \Rightarrow_{P} \langle p_k, \omega_k \rangle$ be the transition that corresponds to the $ret$ statement of this call on $\pi$. Let then $\langle p_{k-1}, \beta \rangle \xrightarrow{ret} \langle p_k, \varepsilon \rangle \in \Delta$ be the corresponding return rule. Then, we have necessarily $\omega_{k-1} = \beta \gamma'' \omega''$, since as explained in Section 2.2.1, $\gamma''$ is the return address of the call. After applying this rule, $\omega_k = \gamma'' \omega''$. In other words, $\gamma''$ will be the topmost stack symbol at the corresponding
To do this, we exploit additional trap configurations. We use now, let us consider the case where \( p \in P \) where the topmost stack symbol. More specifically, the following sequence of \( P \): \( \langle p', \gamma' \rangle \Rightarrow \langle p'' \rangle \Rightarrow \langle p_k, \gamma'' \rangle \) will be mimicked by the following sequence of \( BP_\varphi \): \( \langle p', \gamma \rangle \Rightarrow \langle p_k, \gamma'' \rangle \Rightarrow \langle p_k, \gamma'' \rangle \) by \( \langle p_k, \gamma''' \rangle \) and \( \langle p_k, \gamma''' \rangle \) by \( \langle p_k, \gamma''' \rangle \). Therefore, we obtain that \( \langle p', \gamma' \rangle \Rightarrow \langle p_k, \gamma''' \rangle \). The property holds for this case.

If \( \langle p, \omega \rangle \Rightarrow P \langle p', \omega' \rangle \) corresponds to a simple statement. Then, the abstract successor of \( p \) is its immediate successor \( p' \). Thus, we get that \( p_k, \omega_k \rangle = \langle p, \omega' \rangle \). From the transition rules corresponding to \( h_2 \) in (a7), we get that \( \langle p, \gamma'' \rangle \Rightarrow \langle p_k, \gamma''' \rangle \). Therefore, \( \langle p, \gamma''' \rangle \Rightarrow \langle p_k, \gamma''' \rangle \). The property holds for this case.

2. Now, let us consider the case where \( p_k, \omega_k \rangle \), the abstract successor of \( p \), is \( \perp \). This case occurs when \( p \), \( \omega \rangle \Rightarrow P \langle p', \omega' \rangle \) corresponds to a return statement. Then, one abstract successor of \( p, \omega \rangle \) is \( \perp \). Note that \( \perp \) does not satisfy any formula, i.e., \( \perp \) does not satisfy \( \phi_1 \). Therefore, from \( \langle p, \gamma'' \rangle \), we need to ensure that the path of \( BP_\varphi \) reflecting the possibility in (A1) that \( p_k, \omega_k \rangle \Rightarrow \langle p, \omega' \rangle \), then we have \( \omega = \gamma'' \) and \( \omega' = \gamma''' \). We add the transition rule corresponding to \( h_3 \) in (a7) to allow \( \langle p, \gamma'' \rangle \Rightarrow \langle p_k, \gamma''' \rangle \). Since a run of \( BP_\varphi \) includes only infinite paths, we equip these trap configurations with self-loops by the transition rules in (a18), i.e., \( p_k, \gamma'' \rangle \Rightarrow \langle p_k, \gamma''' \rangle \). As a result, we obtain a corresponding path in \( BP_\varphi \): \( \langle p, \gamma'' \rangle \Rightarrow \langle p_k, \gamma''' \rangle \). Note that this path is not accepted by \( BP_\varphi \) because \( p_k \notin F \).

In summary, for every abstract-successor \( \langle p_k, \omega_k \rangle \) of \( p, \omega \), if
Chapter 4. Branching Temporal Logic of Calls and Returns for

Pushdown Systems

\( \langle p_k, \omega_k \rangle \neq \bot \), then, \( \langle \langle p, EX^a \phi_1 \rangle, \omega \rangle \Rightarrow BP_\varphi \langle \langle p_k, \phi_1 \rangle, \omega_k \rangle \); otherwise \( \langle \langle p, EX^a \phi_1 \rangle, \omega \rangle \Rightarrow BP_\varphi \langle \eta, \gamma_1 \omega'' \rangle \Rightarrow BP_\varphi \langle \eta, \gamma_1 \omega'' \rangle \) which is not accepted by \( BP_\varphi \). Therefore, (A1) is ensured by the transition rules in (a7) stating that \( BP_\varphi \) has an accepting run from \( \langle \langle p, EX^a \phi_1 \rangle, \omega \rangle \) iff there exists an abstract successor \( \langle p_k, \omega_k \rangle \) of \( \langle p, \omega \rangle \) s.t. \( BP_\varphi \) has an accepting run from \( \langle \langle p_k, \phi_1 \rangle, \omega_k \rangle \).

As a result, we get that:

**Lemma 6.** Let \( \langle p_k, \omega_k \rangle \) be an abstract-successor of \( \langle p, \omega \rangle \) on \( \mathcal{P} \). For every \( \phi = EX^a \phi_1 \in Cl(\varphi) \), \( \langle \langle p, \phi \rangle, \omega \rangle \Rightarrow BP_\varphi \langle \langle p_k, \phi_1 \rangle, \omega_k \rangle \).

- If \( \phi = AX^a \phi_1 \): this case is ensured by the transition rules in (a8) together with (a17) and \( \Delta \subseteq \Delta' \). The intuition of (a8) is similar to that of (a7).

- If \( \phi = E[\phi_1 U^a \phi_2] \), then, for every \( \omega \in \Gamma^* \), \( \langle p, \omega \rangle \models_{M} \phi \) iff \( \langle p, \omega \rangle \models_{M} \phi_2 \) or \( \langle p, \omega \rangle \models_{M} \phi_1 \) and there exists an abstract successor \( \langle p_k, \omega_k \rangle \) of \( \langle p, \omega \rangle \) s.t. \( \langle p_k, \omega_k \rangle \models_{M} \phi \) (A2). Let \( \pi \in Traces(\langle p, \omega \rangle) \) be a run starting from \( \langle p, \omega \rangle \) on which \( \langle p_k, \omega_k \rangle \) is the abstract-successor of \( \langle p, \omega \rangle \). Over \( \pi \), let \( \langle p', \omega' \rangle \) be the immediate successor of \( \langle p, \omega \rangle \).

1. Firstly, we show that for every abstract-successor \( \langle p_k, \omega_k \rangle \neq \bot \) of \( \langle p, \omega \rangle \), \( \langle \langle p, \phi \rangle, \omega \rangle \Rightarrow BP_\varphi \{ \langle \langle p, \phi_1 \rangle, \omega \rangle, \langle \langle p_k, \phi_1 \rangle, \omega_k \rangle \} \). There are two possibilities:

   - If \( \langle p, \omega \rangle \Rightarrow_{\mathcal{P}} \langle p', \omega' \rangle \) corresponds to a call statement. From the rules corresponding to \( h_1 \) in (a10), we get that \( \langle \langle p, \phi \rangle, \omega \rangle \Rightarrow BP_\varphi \{ \langle \langle p, \phi_1 \rangle, \omega \rangle, \langle p', \omega' \rangle \} \) where \( \langle p', \omega' \rangle \) is the immediate successor of \( \langle p, \omega \rangle \). Thus, to ensure that \( \langle \langle p, \phi \rangle, \omega \rangle \Rightarrow BP_\varphi \{ \langle \langle p, \phi_1 \rangle, \omega \rangle, \langle \langle p_k, \phi_1 \rangle, \omega_k \rangle \} \), we only need to ensure that \( \langle p', \omega' \rangle \Rightarrow BP_\varphi \langle \langle p_k, \phi_1 \rangle, \omega_k \rangle \). As for the case \( \phi = EX^a \phi_1 \), \( \langle p', \omega' \rangle \Rightarrow BP_\varphi \langle \langle p_k, \phi_1 \rangle, \omega_k \rangle \) is ensured by the rules in \( \Delta \subseteq \Delta' \) and the rules in (a17): rules in \( \Delta \subseteq \Delta' \) allow to mimic the run of the PDS \( \mathcal{P} \) before the return and rules in (a17) allow to extract and put back \( \phi_1 \) when the return-point is reached.

   - If \( \langle p, \omega \rangle \Rightarrow_{\mathcal{P}} \langle p', \omega' \rangle \) corresponds to a simple statement. Then, the abstract successor of \( \langle p, \omega \rangle \) is its immediate successor \( \langle p', \omega' \rangle \). Thus, we get that \( \langle p_k, \omega_k \rangle = \langle p', \omega' \rangle \). From the transition rules corresponding to \( h_2 \) in (a10), we get that \( \langle \langle p, E[\phi_1 U^a \phi_2] \rangle, \omega \rangle \Rightarrow BP_\varphi \{ \langle \langle p, \phi_1 \rangle, \omega \rangle, \langle p', \phi_1 \rangle, \omega' \rangle \} \). Therefore, \( \langle \langle p, E[\phi_1 U^a \phi_2] \rangle, \omega \rangle \Rightarrow BP_\varphi \{ \langle \langle p, \phi_1 \rangle, \omega \rangle, \langle \langle p_k, \phi_1 \rangle, \omega_k \rangle \} \). In other words, \( BP_\varphi \) has an accepting run from both \( \langle \langle p, \phi_1 \rangle, \omega \rangle \) and \( \langle \langle p_k, \phi_1 \rangle, \omega_k \rangle \) where \( \langle p_k, \omega_k \rangle \) is an abstract successor of \( \langle p, \omega \rangle \). The property holds for this case.
4.3. BCARET Model-Checking for Pushdown Systems

As a result, we get that:

**Lemma 7.** Let \( \langle p_k, \omega_k \rangle \) be an abstract-successor of \( \langle p, \omega \rangle \) on \( \mathcal{P} \). For every \( \phi = E[\phi_1 U^n \phi_2] \in \text{Cl}(\varphi) \), \( \langle \langle p, \phi \rangle, \omega \rangle \Rightarrow_{\mathcal{BP}_\varphi} \{ \langle \langle p, \phi_1 \rangle, \omega \rangle, \langle \langle p_k, \phi \rangle, \omega_k \rangle \} \).

2. Now, let us consider the case where \( \langle p_k, \omega_k \rangle = \perp \). As explained previously, this case occurs when \( \langle p, \omega \rangle \Rightarrow \langle p', \omega' \rangle \) corresponds to a return statement. Then, the abstract successor of \( \langle p, \omega \rangle \) is \( \perp \). Note that \( \perp \) does not satisfy any formula, i.e., \( \perp \) does not satisfy \( \phi \). Therefore, from \( \langle \langle p, E[\phi_1 U^n \phi_2] \rangle, \omega \rangle \), we need to ensure that the path reflecting the possibility in \( \text{(A2)} \) that \( \langle (p, \omega) \rangle \not\models_{\lambda_f} \phi_1 \) and \( \langle p_k, \omega_k \rangle \not\models_{\lambda_f} \phi \) is not accepted by \( \mathcal{BP}_\varphi \). This is ensured as for the case \( \phi = \text{EX}^a \phi_1 \) by the transition rules corresponding to \( h_3 \) in \( \text{(a10)} \).

In summary, for every abstract-successor \( \langle p_k, \omega_k \rangle \) of \( \langle p, \omega \rangle \), if \( \langle p_k, \omega_k \rangle \not\models \perp \), then, \( \langle \langle p, E[\phi_1 U^n \phi_2] \rangle, \omega \rangle \Rightarrow_{\mathcal{BP}_\varphi} \{ \langle \langle p, \phi_1 \rangle, \omega \rangle, \langle \langle p_k, E[\phi_1 U^n \phi_2] \rangle, \omega_k \rangle \} \); otherwise \( \langle \langle p, E[\phi_1 U^n \phi_2] \rangle, \omega \rangle \Rightarrow_{\mathcal{BP}_\varphi} \langle \langle p, \gamma \omega'' \rangle \Rightarrow_{\mathcal{BP}_\varphi} \langle \langle p, \gamma \omega'' \rangle \rangle \) which is not accepted by \( \mathcal{BP}_\varphi \). Therefore, \( \text{(a2)} \) is ensured by the transition rules in \( \text{(a10)} \) stating that \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle \langle p, E[\phi_1 U^n \phi_2] \rangle, \omega \rangle \) if \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle \langle p, \phi_2 \rangle, \omega \rangle \); or \( \mathcal{BP}_\varphi \) has an accepting run from both \( \langle \langle p, \phi_1 \rangle, \omega \rangle \) and \( \langle \langle p_k, E[\phi_1 U^n \phi_2] \rangle, \omega_k \rangle \) where \( \langle p_k, \omega_k \rangle \) is an abstract successor of \( \langle p, \omega \rangle \).

- If \( \phi = E[\phi_1 R^n \phi_2] \), then, for every \( \omega \in \Gamma^* \), \( \langle p, \omega \rangle \models_{\lambda_f} \phi \) iff \( \langle \langle p, \omega \rangle \models_{\lambda_f} \phi_1 \) and \( \langle p, \omega \rangle \models_{\lambda_f} \phi_2 \) or \( \langle \langle p, \omega \rangle \models_{\lambda_f} \phi_1 \) and there exists an abstract successor \( \langle p_k, \omega_k \rangle \) of \( \langle p, \omega \rangle \) s.t. \( \langle p_k, \omega_k \rangle \models_{\lambda_f} \phi \) (A3). Let \( \pi \in \text{Traces}(\langle p, \omega \rangle) \) be a run starting from \( \langle p, \omega \rangle \) on which \( \langle p_k, \omega_k \rangle \) is the abstract-successor of \( \langle p, \omega \rangle \). Over \( \pi \), let \( \langle p', \omega' \rangle \) be the immediate successor of \( \langle p, \omega \rangle \).

1. Firstly, we show that for every abstract-successor \( \langle p_k, \omega_k \rangle \not\models \perp \) of \( \langle p, \omega \rangle \), \( \langle \langle p, \phi \rangle, \omega \rangle \Rightarrow_{\mathcal{BP}_\varphi} \{ \langle \langle p, \phi_1 \rangle, \omega \rangle, \langle \langle p_k, \phi \rangle, \omega_k \rangle \} \). There are two possibilities:

   - If \( \langle p, \omega \rangle \Rightarrow \langle p', \omega' \rangle \) corresponds to a call statement. From the rules corresponding to \( h_1 \) in \( \text{(a15)} \), we get that \( \langle \langle p, \phi \rangle, \omega \rangle \Rightarrow_{\mathcal{BP}_\varphi} \{ \langle \langle p, \phi_2 \rangle, \omega \rangle, \langle p', \omega' \rangle \} \) where \( \langle p', \omega' \rangle \) is the immediate successor of \( \langle p, \omega \rangle \). Thus, to ensure that \( \langle \langle p, \phi \rangle, \omega \rangle \Rightarrow_{\mathcal{BP}_\varphi} \{ \langle \langle p, \phi_2 \rangle, \omega \rangle, \langle \langle p_k, \phi \rangle, \omega_k \rangle \} \), we only need to ensure that \( \langle p', \omega' \rangle \Rightarrow_{\mathcal{BP}_\varphi} \{ \langle p_k, \phi \rangle, \omega_k \} \). As for the case \( \phi = \text{EX}^a \phi_1 \), \( \langle p', \omega' \rangle \Rightarrow_{\mathcal{BP}_\varphi} \{ \langle p_k, \phi \rangle, \omega_k \} \) is ensured by the rules in \( \Delta \subseteq \Delta' \) and the rules in \( \text{(a17)} \): rules in \( \Delta \subseteq \Delta' \) allow to mimic the run of the PDS \( \mathcal{P} \) before the return and rules in \( \text{(a17)} \) allow to extract and put back \( \phi_1 \) when the return-point is reached.
Chapter 4. Branching Temporal Logic of Calls and Returns for Pushdown Systems

- If \( (p, \omega) \Rightarrow \varphi (p', \omega') \) corresponds to a simple statement. Then, the abstract successor of \( (p, \omega) \) is its immediate successor \( (p', \omega') \). Thus, we get that \( (p_k, \omega_k) = (p', \omega') \). From the transition rules corresponding to \( h_2 \) in (a15), we get that \( \langle (p, E[\phi_1 R^a \phi_2]), \omega \rangle \Rightarrow_{BP_{\varphi}} \{ \langle (p, \phi_2), \omega \rangle, \langle (p', \varphi), \omega' \rangle \} \). Therefore, \( \langle (p, E[\phi_1 R^a \phi_2]), \omega \rangle \Rightarrow_{BP_{\varphi}} \{ \langle (p, \phi_2), \omega \rangle, \langle (p_k, \varphi), \omega_k \rangle \} \).

2. Now, let us consider the case where \( (p_k, \omega_k) = \bot \). As explained previously, this case occurs when \( (p, \omega) \Rightarrow \varphi (p', \omega') \) corresponds to a return statement. Then, the abstract successor of \( (p, \omega) \) is \( \bot \). Note that \( \bot \) does not satisfy any formula, i.e., \( \bot \) does not satisfy \( \varphi \). Therefore, from \( \{ \langle (p, E[\phi_1 R^a \phi_2]), \omega \rangle \} \), we need to ensure that the path reflecting the possibility in (A3) that \( (p, \omega) \vdash_{\lambda_f} \phi_2 \) and \( (p_k, \omega_k) \vdash_{\lambda_f} \varphi \) is not accepted by \( BP_{\varphi} \). This is ensured as for the case \( \varphi = EX^a \phi_1 \) by the transition rules corresponding to \( h_3 \) in (a15).

In summary, for every abstract-successor \( (p_k, \omega_k) \) of \( (p, \omega) \), if \( (p_k, \omega_k) \neq \bot \), then, \( \langle (p, E[\phi_1 R^a \phi_2]), \omega \rangle \Rightarrow_{BP_{\varphi}} \{ \langle (p, \phi_2), \omega \rangle, \langle (p_k, E[\phi_1 R^a \phi_2]), \omega_k \rangle \} \); otherwise \( \langle (p, E[\phi_1 R^a \phi_2]), \omega \rangle \Rightarrow_{BP_{\varphi}} \{ \langle p, \varphi, \omega'' \rangle, \langle (p_k, \varphi), \omega_k \rangle \} \). Thus, (A3) is ensured by the transition rules in (a15) stating that \( BP_{\varphi} \) has an accepting run from \( \langle (p, E[\phi_1 R^a \phi_2]), \omega \rangle \) iff \( BP_{\varphi} \) has an accepting run from both \( \langle (p, \phi_1), \omega \rangle \) and \( \langle (p, \phi_2), \omega \rangle \); or \( BP_{\varphi} \) has an accepting run from both \( \langle (p, \phi_2), \omega \rangle \) and \( \langle (p_k, E[\phi_1 R^a \phi_2]), \omega_k \rangle \) where \( (p_k, \omega_k) \) is an abstract successor of \( (p, \omega) \).

As a result, we get that:

**Lemma 8.** Let \( (p_k, \omega_k) \) be an abstract-successor of \( (p, \omega) \) on \( \mathcal{P} \). For every \( \varphi = E[\phi_1 R^a \phi_2] \in CI(\varphi) \), \( \langle (p, \varphi), \omega \rangle \Rightarrow_{BP_{\varphi}} \{ \langle (p, \phi_2), \omega \rangle, \langle (p_k, \varphi), \omega_k \rangle \} \).

- The intuition behind the rules corresponding to the cases \( \varphi = A[\phi_1 U^a \phi_2] \), \( \varphi = A[\phi_1 R^a \phi_2] \) are similar to the previous case.

**The Büchi accepting condition.** The elements of the Büchi accepting condition set \( F \) of \( BP_{\varphi} \) ensure the liveness requirements of until-formulas on infinite global paths, infinite abstract paths as well as on finite abstract paths.

- With regards to infinite global paths, the fact that the liveness requirement \( \phi_2 \) in \( E[\phi_1 U^a \phi_2] \) is eventually satisfied in \( \mathcal{P} \) is ensured by the fact that \( (p, E[\phi_1 U^a \phi_2]) \) does not belong to \( F \). Note that \( (p, \omega) \vdash_{\lambda_f} E[\phi_1 U^a \phi_2] \) iff \( (p, \omega) \vdash_{\lambda_f} \phi_2 \) or there exists a global-successor \( (p', \omega') \) s.t. \( (p, \omega) \vdash_{\lambda_f} \phi_1 \) and \( (p', \omega') \vdash_{\lambda_f} E[\phi_1 U^a \phi_2] \). Because \( \phi_2 \) should hold eventually, to
avoid the case where a run of $BP_{\phi}$ always carries $E[\phi_1U^a\phi_2]$ and never reaches $\phi_2$, we don’t set $\langle p, E[\phi_1U^a\phi_2]\rangle$ as an element of the Büchi accepting condition set. This guarantees that the accepting run of $BP_{\phi}$ must visit some control locations in $\langle p, \phi_2\rangle$ which ensures that $\phi_2$ will eventually hold. The liveness requirements of $A[\phi_1U^a\phi_2]$ are ensured as for the case of $E[\phi_1U^a\phi_2]$.

- With regards to infinite abstract paths, the fact that the liveness requirement $\phi_2$ in $E[\phi_1U^a\phi_2]$ is eventually satisfied in $P$ is ensured by the fact that $\langle p, E[\phi_1U^a\phi_2]\rangle$ doesn’t belong to $F$. The intuition behind this case is similar to the intuition of $E[\phi_1U^a\phi_2]$. The liveness requirements of $A[\phi_1U^a\phi_2]$ are ensured as for the case of $E[\phi_1U^a\phi_2]$.

- With regards to finite abstract paths $\langle p_0, \omega_0\rangle\langle p_1, \omega_1\rangle\ldots\langle p_m, \omega_m\rangle$ where $\langle p_m, \omega_m\rangle \Rightarrow P \langle p_{m+1}, \omega_{m+1}\rangle$ corresponds to a return statement, the fact that the liveness requirement $\phi_2$ in $E[\phi_1U^a\phi_2]$ is eventually satisfied in $P$ is ensured by the fact that $p_\bot$ doesn’t belong to $F$. Look at Figure 4.2 for an illustration. In this figure, for every $i + 1 \leq u \leq k - 1$, the abstract path starting from $\langle p_u, \omega_u\rangle$ is finite because the abstract successor of $\langle p_{k-1}, \omega_{k-1}\rangle$ is $\bot$ since $\langle p_{k-1}, \omega_{k-1}\rangle \Rightarrow P \langle p_k, \omega_k\rangle$ corresponds to a return statement. Suppose that we want to check whether $\langle p_{k-1}, \omega_{k-1}\rangle \models F, E[\phi_1U^a\phi_2]$, then, we get that $\langle p_{k-1}, \omega_{k-1}\rangle \models F, E[\phi_1U^a\phi_2]$ iff $\langle p_{k-1}, \omega_{k-1}\rangle \models F, \phi_2$ or there exists an abstract-successor $\langle p', \omega'\rangle$ s.t. $\langle p_{k-1}, \omega_{k-1}\rangle \models F, \phi_1$ and $\langle p', \omega'\rangle \models F, E[\phi_1U^a\phi_2]$. Since $\phi_2$ should eventually hold, $\phi_2$ should hold at $\langle p_{k-1}, \omega_{k-1}\rangle$ because the
abstract-successor of \( \langle p_{k-1}, \omega_{k-1} \rangle \) on this abstract-path is \( \bot \). To ensure this, we move \( \langle p_{k-1}, \omega_{k-1} \rangle \) to the trap configuration \( \langle p_\bot, \gamma_\bot \rangle \) and add a loop here by the transition rule \((\alpha 18)\). In addition, we don’t set \( p_\bot \) as an element of the Büchi accepting condition set, which means that \( \langle p_{k-1}, \omega_{k-1} \rangle \models_{\lambda_f} E[\phi_1 U^a \phi_2] \) iff \( \langle p_{k-1}, \omega_{k-1} \rangle \models_{\lambda_f} \phi_2 \) by the transition rules in \((\alpha 10)\). This ensures the liveness requirement \( \phi_2 \) in \( E[\phi_1 U^a \phi_2] \) is eventually satisfied.

• With regards to finite abstract paths \( \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \ldots \langle p_m, \omega_m \rangle \) where \( \langle p_m, \omega_m \rangle \Rightarrow \langle p_{m+1}, \omega_{m+1} \rangle \) corresponds to a call statement but this call never reaches its corresponding return-point, the fact that the liveness requirement \( \phi_2 \) in \( E[\phi_1 U^a \phi_2] \) is eventually satisfied in \( \mathcal{P} \) is ensured by the fact that \( p \notin F \). Look at Figure 4.3 where the procedure \( \text{proc} \) never terminates. In this figure, for every \( 0 \leq u \leq i \), the abstract path starting from \( \langle p_u, \omega_u \rangle \) is finite. Suppose that we want to check whether \( \langle p_i, \omega_i \rangle \models_{\lambda_f} E[\phi_1 U^a \phi_2] \), then we get that \( \langle p_i, \omega_i \rangle \models_{\lambda_f} E[\phi_1 U^a \phi_2] \) iff \( \langle p_i, \omega_i \rangle \models_{\lambda_f} \phi_2 \) or there exists an abstract-successor \( \langle p', \omega' \rangle \) s.t. \( \langle p_i, \omega_i \rangle \models_{\lambda_f} \phi_1 \) and \( \langle p', \omega' \rangle \models_{\lambda_f} E[\phi_1 U^a \phi_2] \). Since \( \phi_2 \) should eventually hold, \( \phi_2 \) should hold at \( \langle p_i, \omega_i \rangle \) because the abstract-successor of \( \langle p_i, \omega_i \rangle \) on this abstract-path is \( \bot \). As explained above, at \( \langle p_i, \omega_i \rangle \), we will encode the formula \( E[\phi_1 U^a \phi_2] \) into the stack and mimic the run of \( \mathcal{P} \) on \( \mathcal{BP}_\varphi \) until it reaches the corresponding return-point of the call. In other words, if the call is never reached, the run of \( \mathcal{BP}_\varphi \) will infinitely visit the control locations of \( \mathcal{P} \). To ensure this path unaccepted, we don’t set \( p \in \mathcal{P} \) as an element of the Büchi accepting condition set, which means that \( \langle p_i, \omega_i \rangle \models_{\lambda_f} E[\phi_1 U^a \phi_2] \) iff \( \langle p_i, \omega_i \rangle \models_{\lambda_f} \phi_2 \) by the transition rules in \((\alpha 10)\). This ensures the liveness requirement \( \phi_2 \) in \( E[\phi_1 U^a \phi_2] \) is eventually satisfied.

Thus, we can show that:

**Theorem 13.** Given a PDS \( \mathcal{P} = (P, \Gamma, \Delta) \), a set of atomic propositions \( \mathcal{AP} \), a labelling function \( f : \mathcal{AP} \to 2^P \) and a BCARET formula \( \varphi \), we can compute an ABPDS \( \mathcal{BP}_\varphi \) such that for every configuration \( \langle p, \omega \rangle \), \( \langle p, \omega \rangle \models_{\lambda_f} \varphi \) iff \( \mathcal{BP}_\varphi \) has an accepting run from the configuration \( \langle \langle p, \varphi \rangle, \omega \rangle \).

**Formal proof:** Given \( c_0 \Rightarrow_{\mathcal{BP}_\varphi} \{c_1, c_2, \ldots, c_n\} \) where for every \( 0 \leq i \leq n \), \( c_i \) is a configuration of the ABPDS \( \mathcal{BP}_\varphi \). For presentation reasons, we also write \( c_0 \Rightarrow_{\mathcal{BP}_\varphi} c_1 \land c_2 \land \ldots \land c_n \). We prove the following two directions:

\[ (\Rightarrow) \] Assume that \( \langle p, \omega \rangle \models_{\lambda_f} \varphi \), we need to prove that \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle \langle p, \varphi \rangle, \omega \rangle \). In what follows, we show how this is ensured by induction on the structure for the BCARET formula \( \varphi \).
Proof.

• Base case:
  - $\varphi = e$ ($e \in AP$): $\langle p, \omega \rangle \models_{\lambda_f} \varphi \implies p \in f(e)$. According to the transition rule in (a1), we get $\langle \langle p, e \rangle, \omega \rangle \models_{BP_{\varphi}} \langle \langle p, e \rangle, \omega \rangle$. In addition, we get that $\langle p, e \rangle \in F$ for every $p \in f(e)$. Therefore, $BP_{\varphi}$ has an accepting run from $\langle \langle p, e \rangle, \omega \rangle$. In other words, $BP_{\varphi}$ has an accepting run from $\langle \langle p, \varphi \rangle, \omega \rangle$. The property holds for this case.
  
  - $\varphi = \neg e$ ($e \in AP$): $\langle p, \omega \rangle \models_{\lambda_f} \varphi \implies p \notin f(e)$. According to the transition rule in (a2), we get $\langle \langle p, \neg e \rangle, \omega \rangle \models_{BP_{\varphi}} \langle \langle p, \neg e \rangle, \omega \rangle$. In addition, we get that $\langle p, \neg e \rangle \in F$ for every $p \notin f(e)$. Therefore, $BP_{\varphi}$ has an accepting run from $\langle \langle p, \neg e \rangle, \omega \rangle$. In other words, $BP_{\varphi}$ has an accepting run from $\langle \langle p, \varphi \rangle, \omega \rangle$. The property holds for this case.

• Induction Step:
  
  - Case $\varphi = \varphi_1 \lor \varphi_2$:
    
    * Since $\langle p, \omega \rangle \models_{\lambda_f} \varphi_1$ or $\langle p, \omega \rangle \models_{\lambda_f} \varphi_2$, we obtain that $\langle p, \omega \rangle \models_{\lambda_f} \varphi_1$ or $\langle p, \omega \rangle \models_{\lambda_f} \varphi_2$. By applying the induction hypothesis, we get that $BP_{\varphi}$ has an accepting run from $\langle \langle p, \varphi_1 \rangle, \omega \rangle$ or $\langle \langle p, \varphi_2 \rangle, \omega \rangle$ (1).
    
    * According to the transition rule in (a4), we obtain $\langle \langle p, \varphi \rangle, \omega \rangle \models_{BP_{\varphi}} \{\langle \langle p, \varphi_1 \rangle, \omega \rangle\}, \langle \langle p, \varphi \rangle, \omega \rangle \models_{BP_{\varphi}} \{\langle \langle p, \varphi_2 \rangle, \omega \rangle\}$ (2).

  From (1) and (2), we get that $BP_{\varphi}$ has an accepting run from the configuration $\langle \langle p, \varphi \rangle, \omega \rangle$.

  - Case $\varphi = \varphi_1 \land \varphi_2$:
    
    * Since $\langle p, \omega \rangle \models_{\lambda_f} \varphi_1$ and $\langle p, \omega \rangle \models_{\lambda_f} \varphi_2$, we obtain that $\langle p, \omega \rangle \models_{\lambda_f} \varphi_1$ and $\langle p, \omega \rangle \models_{\lambda_f} \varphi_2$. By applying the induction hypothesis, we get that $BP_{\varphi}$ has an accepting run from $\langle \langle p, \varphi_1 \rangle, \omega \rangle$ and $\langle \langle p, \varphi_2 \rangle, \omega \rangle$ (3).
    
    * According to the transition rule in (a3), we obtain $\langle \langle p, \varphi \rangle, \omega \rangle \models_{BP_{\varphi}} \{\langle \langle p, \varphi_1 \rangle, \omega \rangle, \langle \langle p, \varphi_2 \rangle, \omega \rangle\}$ (4).

  From (3) and (4), we get that $BP_{\varphi}$ has an accepting run from the configuration $\langle \langle p, \varphi \rangle, \omega \rangle$.

  - Case $\varphi = EX^{g} \phi_1$:
    
    * Since $\langle p, \omega \rangle \models_{\lambda_f} \varphi$, then, there exists a global successor $\langle p', \omega' \rangle$ of $\langle p, \omega \rangle$ s.t. $\langle p', \omega' \rangle \models_{\lambda_f} \varphi_1$. Note that the global successor of a configuration is its immediate successor. Thus, there exists an immediate successor $\langle p', \omega' \rangle$ of $\langle p, \omega \rangle$ s.t. $\langle p', \omega' \rangle \models_{\lambda_f} \varphi_1$. Therefore, by applying the induction hypothesis, $BP_{\varphi}$ has an accepting run from $\langle \langle p', \varphi_1 \rangle, \omega' \rangle$ (5).
* According to the transition rule in (a5), we obtain
\[ \langle (p, \varphi), \omega \rangle \Rightarrow_{\mathcal{BP}_\varphi} \{ (p', \varphi_1), \omega' \} \] (6).

From (5) and (6), we get that \( \mathcal{BP}_\varphi \) has an accepting run from the configuration \( \langle (p, \varphi), \omega \rangle \).

- Case \( \varphi = EX^a \phi_1 \):
  * Since \( \langle p, \omega \rangle \models_{\lambda_j} \varphi \), then, there exists an abstract successor \( \langle p_k, \omega_k \rangle \) of \( \langle p, \omega \rangle \) s.t. \( \langle p_k, \omega_k \rangle \models_{\lambda_j} \varphi_1 \). Therefore, by applying the induction hypothesis, \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle (p_k, \varphi_1), \omega_k \rangle \) (7).
  * According to Lemma 6, we obtain \( \langle (p, \varphi), \omega \rangle \Rightarrow_{\mathcal{BP}_\varphi} \{ \langle (p_k, \varphi_1), \omega_k \rangle \} \) (8).

From (7) and (8), we get that \( \mathcal{BP}_\varphi \) has an accepting run from the configuration \( \langle (p, \varphi), \omega \rangle \).

- Case \( \varphi = E[\varphi_1 U^g \varphi_2] \): \( \langle p, \omega \rangle \models_{\lambda_j} E[\varphi_1 U^g \varphi_2] \), then, there exists a run \( \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \langle p_2, \omega_2 \rangle \ldots \) starting from \( \langle p, \omega \rangle \) where \( \langle p, \omega \rangle = \langle p_0, \omega_0 \rangle \) on which there exists \( i \geq 0 \) s.t. \( \langle p_i, \omega_i \rangle \models_{\lambda_j} \varphi_2 \) and for every \( 0 \leq j < i \), \( \langle p_j, \omega_j \rangle \models_{\lambda_j} \varphi_1 \) (since the global successor of a configuration is its immediate successor). By applying the induction hypothesis, we get that:
  * \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle (p_i, \varphi_2), \omega_i \rangle \) (9).
  * For every \( 0 \leq j < i \), \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle (p_j, \varphi_1), \omega_j \rangle \) (10).

According to the rules in (a9), we get \( \langle (p_i, \varphi), \omega_i \rangle \Rightarrow_{\mathcal{BP}_\varphi} \langle (p_i, \varphi_2), \omega_i \rangle \) (11). From (9) and (11), \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle (p_i, \varphi), \omega_i \rangle \) (12).

Now, we prove that \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle (p, \varphi), \omega \rangle \). There are two cases:

a) If \( i = 0 \), then \( \langle (p, \varphi), \omega \rangle = \langle (p_i, \varphi), \omega_i \rangle \). Therefore, from (12), \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle (p, \varphi), \omega \rangle \).

b) If \( i > 0 \). Firstly, we show that for every \( 0 \leq j < i \), \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle (p_j, \varphi), \omega_j \rangle \) by induction on \( l = i - j \).

* Basis. \( l = 1 \), which means that \( \langle p_i, \omega_i \rangle \) is an immediate successor of \( \langle p_j, \omega_j \rangle \). \( \langle (p_j, \varphi), \omega_j \rangle \Rightarrow_{\mathcal{BP}_\varphi} \langle (p_j, \varphi_1), \omega_j \rangle \wedge \langle (p_i, \varphi), \omega_i \rangle \) (by the rules in (a9) and the fact that \( \langle p_i, \omega_i \rangle \) is an immediate successor of \( \langle p_j, \omega_j \rangle \)). Also, \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle (p_j, \varphi_1), \omega_j \rangle \) (by (10)) and \( \langle (p_i, \varphi), \omega_i \rangle \) (by (12)). Therefore, \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle (p_j, \varphi), \omega_j \rangle \). The property holds for this case.
* Step. \( l > 1 \), we get that \( \langle p_j, \omega_j \rangle \xrightarrow{a} \langle p_{j+1}, \omega_{j+1} \rangle \xrightarrow{a} \langle p_i, \omega_i \rangle \).

By applying the induction hypothesis on \( l \), we get that \( BP_\varphi \) has an accepting run from \( \langle \langle p_{j+1}, \varphi \rangle, \omega_{j+1} \rangle \) (13).

According to the rules in (a9) and the fact that \( \langle p_{j+1}, \omega_{j+1} \rangle \) is an immediate successor of \( \langle p_j, \omega_j \rangle \), we get that \( \langle \langle p_j, \varphi \rangle, \omega_j \rangle \xrightarrow{a} BP_\varphi \langle \langle p_{j+1}, \varphi \rangle, \omega_{j+1} \rangle \). In addition, \( BP_\varphi \) has an accepting run from \( \langle \langle p_j, \varphi_1 \rangle, \omega_j \rangle \) (by (10)) and \( \langle \langle p_{j+1}, \varphi \rangle, \omega_{j+1} \rangle \) (by (13)). Thus, \( BP_\varphi \) has an accepting run from \( \langle \langle p_j, \varphi \rangle, \omega_j \rangle \).

The property holds for this case.

Note that \( \langle p, \omega \rangle = \langle p_j, \omega_j \rangle \) when \( j = 0 \). As a result, we obtain that \( BP_\varphi \) has an accepting run from \( \langle \langle p, \varphi \rangle, \omega \rangle \). The property holds for this case.

- Case \( \varphi = E[\varphi_1 U^a \varphi_2] \): \( \langle p, \omega \rangle \models_{\lambda_f} E[\varphi_1 U^a \varphi_2] \) implies that there exists an (finite or infinite) abstract path \( \pi_a = \langle p_{z_0}, \omega_{z_0} \rangle \langle p_{z_1}, \omega_{z_1} \rangle \langle p_{z_2}, \omega_{z_2} \rangle \ldots \) starting from \( \langle p, \omega \rangle \) where \( \langle p, \omega \rangle = \langle p_{z_0}, \omega_{z_0} \rangle \) on which there exists \( i \geq 0 \) s.t. \( \langle p_{z_i}, \omega_{z_i} \rangle \models_{\lambda_f} \varphi_2 \) and for every \( 0 \leq j < i \), \( \langle p_{z_j}, \omega_{z_j} \rangle \models_{\lambda_f} \varphi_1 \). By applying the induction hypothesis, we get that:

* \( BP_\varphi \) has an accepting run from \( \langle \langle p_{z_1}, \varphi_2 \rangle, \omega_{z_1} \rangle \) (14).

* For every \( 0 \leq j < i \), \( BP_\varphi \) has an accepting run from \( \langle \langle p_{z_j}, \varphi_1 \rangle, \omega_{z_j} \rangle \) (15).

According to the rules in (a10), we get \( \langle \langle p_{z_1}, \varphi \rangle, \omega_{z_1} \rangle \xrightarrow{a} BP_\varphi \langle \langle p_{z_2}, \varphi \rangle, \omega_{z_2} \rangle \) (16). From (14) and (16), \( BP_\varphi \) has an accepting run from \( \langle \langle p_{z_1}, \varphi \rangle, \omega_{z_1} \rangle \) (17).

Now, we prove that \( BP_\varphi \) has an accepting run from \( \langle \langle p, \varphi \rangle, \omega \rangle \).

There are two cases:

a) If \( i = 0 \); then \( \langle \langle p, \varphi \rangle, \omega \rangle = \langle \langle p_{z_1}, \varphi \rangle, \omega_{z_1} \rangle \). Therefore, \( BP_\varphi \) has an accepting run from \( \langle \langle p, \varphi \rangle, \omega \rangle \).

b) If \( i > 0 \). Firstly, we show that for every \( 0 \leq j < i \), \( BP_\varphi \) has an accepting run from \( \langle \langle p_{z_j}, \varphi \rangle, \omega_{z_j} \rangle \) by induction on \( l = i - j \).

* Basis. \( l = 1 \), which means that \( \langle p_{z_1}, \omega_{z_1} \rangle \) is an abstract successor of \( \langle p_{z_j}, \omega_{z_j} \rangle \). Thus, we obtain that \( \langle \langle p_{z_1}, \varphi \rangle, \omega_{z_1} \rangle \xrightarrow{a} BP_\varphi \{ \langle p_{z_1}, \varphi_1 \rangle, \omega_{z_1} \rangle, \langle p_{z_1}, \varphi \rangle, \omega_{z_1} \} \) (by Lemma 7 and the fact that \( \langle p_{z_1}, \omega_{z_1} \rangle \) is an abstract successor of \( \langle p_{z_j}, \omega_{z_j} \rangle \)). Also, \( BP_\varphi \) has an accepting run from \( \langle \langle p_{z_1}, \varphi_1 \rangle, \omega_{z_1} \rangle \) (by (15)) and \( \langle \langle p_{z_1}, \varphi \rangle, \omega_{z_1} \rangle \) (by (17)). Therefore, \( BP_\varphi \) has an accepting run from \( \langle \langle p_{z_1}, \varphi \rangle, \omega_{z_1} \rangle \). The property holds for this case.

* Step. \( l > 1 \), we get that \( \langle p_{z_1}, \omega_{z_1} \rangle \xrightarrow{a} \langle p_{z_{j+1}}, \omega_{z_{j+1}} \rangle \xrightarrow{a} \langle p_i, \omega_i \rangle \).

By applying the induction hypothesis on \( l \), we get that \( BP_\varphi \) has an accepting run from \( \langle \langle p_{z_{j+1}}, \varphi \rangle, \omega_{z_{j+1}} \rangle \) (13).
Chapter 4. Branching Temporal Logic of Calls and Returns for Pushdown Systems

\[ \langle p_z, \omega_z \rangle. \] By applying the induction hypothesis on \( l \), we get that \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle \langle p_{z_l}, \varphi \rangle, \omega_{z_l} \rangle \) (18).

According to Lemma 7 and the fact that \( \langle p_{z_l}, \omega_{z_l} \rangle \) is an abstract successor of \( \langle p_z, \omega_z \rangle \), we get that \( \langle \langle p_z, \varphi \rangle, \omega_z \rangle \Rightarrow_{\mathcal{BP}_\varphi} \{\langle p_{z_l}, \varphi_1 \rangle, \omega_{z_l} \rangle, \langle p_{z_l+1}, \varphi_1 \rangle, \omega_{z_l+1} \rangle \} \). In addition, \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle \langle p_{z_l}, \varphi_1 \rangle, \omega_{z_l} \rangle \) (by (15)) and \( \langle p_{z_l+1}, \varphi_1 \rangle, \omega_{z_l+1} \rangle \) (by (18)). Thus, \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle \langle p_z, \varphi \rangle, \omega_z \rangle \).

The property holds for this case.

Note that \( \langle p, \omega \rangle = \langle p_{z_l}, \omega_{z_l} \rangle \) when \( j = 0 \). As a result, we obtain that \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle \langle p, \varphi \rangle, \omega \rangle \). The property holds for this case.

- Case \( \varphi = A[\varphi_1 U^g \varphi_2] \): This case is similar to the case \( \varphi = E[\varphi_1 U^g \varphi_2] \).
- Case \( \varphi = E[\varphi_1 R^g \varphi_2] \): Based on the semantic of BCARET, \( \langle p, \omega \rangle \models_{\lambda_f} E[\varphi_1 R^g \varphi_2] \) implies that \( \mathcal{P} \) has a run \( \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \langle p_2, \omega_2 \rangle \ldots \) starting from \( \langle p, \omega \rangle \) (since the global successor of a configuration is its immediate successor) such that:

1. there exists \( i \geq 0 \) s.t. \( \langle p_i, \omega_i \rangle \models_{\lambda_f} \varphi_1 \) and for every \( 0 \leq j \leq i \), \( \langle p_j, \omega_j \rangle \models_{\lambda_f} \varphi_2 \);
2. or for every \( i \geq 0 \), \( \langle p_i, \omega_i \rangle \models_{\lambda_f} \varphi_2 \).

For the first case, we can prove that \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle \langle p, \varphi \rangle, \omega \rangle \) by applying the induction on \( i - j \) similar to the case \( \varphi = E[\varphi_1 U^g \varphi_2] \).

Now let us consider the second case, where \( \mathcal{P} \) has an infinite run \( \pi = \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \langle p_2, \omega_2 \rangle \ldots \) starting from \( \langle p, \omega \rangle \) where \( \langle p, \omega \rangle = \langle p_0, \omega_0 \rangle \) such that \( \langle p_i, \omega_i \rangle \models_{\lambda_f} \varphi_2 \) for every \( i \geq 0 \) (19). We need to show that \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle \langle p, \varphi \rangle, \omega \rangle \).

* From the transition rules in (a13), we get that \( \langle \langle p_i, \varphi \rangle, \omega_i \rangle \Rightarrow_{\mathcal{BP}_\varphi} \langle \langle p_{i+1}, \varphi \rangle, \omega_{i+1} \rangle \) for every \( i \geq 0 \). Therefore, \( \langle \langle p_i, \varphi \rangle, \omega_0 \rangle \Rightarrow_{\mathcal{BP}_\varphi} \bigwedge_{i \geq 0} \langle \langle p_i, \varphi \rangle, \omega_{i+1} \rangle \) (20).
* Since \( \langle p_i, \omega_i \rangle \models_{\lambda_f} \varphi_2 \) (by (19)), we get that \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle \langle p_i, \varphi \rangle, \omega_i \rangle \) for every \( i \geq 0 \) (by the induction hypothesis) (21).
* In addition, for every \( i \geq 0 \), \( \langle p_i, \varphi \rangle \) is an accepting control location, then, the path \( \langle \langle p_0, \varphi \rangle, \omega_0 \rangle \langle p_1, \varphi \rangle, \omega_1 \rangle \langle p_2, \varphi \rangle, \omega_2 \rangle \ldots \) is accepted (22).
From (20), (21), (22), we obtain that $BP_\varphi$ has an accepting run from $\langle \langle p_0, \varphi \rangle, \omega_0 \rangle$. In other words, $BP_\varphi$ has an accepting run from $\langle \langle p, \varphi \rangle, \omega \rangle$. The property holds for this case.

- Case $\varphi = E[\varphi_1 R^a \varphi_2]$: Based on the semantic of BCARET, $\langle p, \omega \rangle \models_{\lambda_f} E[\varphi_1 R^a \varphi_2]$ implies that $P$ has an abstract path $\langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \langle p_2, \omega_2 \rangle ...$ starting from $\langle p, \omega \rangle$ where $\langle p, \omega \rangle = \langle p_0, \omega_0 \rangle$ s.t.

1. there exists $i \geq 0$ s.t. $\langle p_{z_i}, \omega_{z_i} \rangle \models_{\lambda_f} \varphi_1$ and for every $0 \leq j \leq i$, $\langle p_{z_j}, \omega_{z_j} \rangle \models_{\lambda_f} \varphi_2$
2. or for every $i \geq 0$, $\langle p_{z_i}, \omega_{z_i} \rangle \models_{\lambda_f} \varphi_2$

For the first case, we can prove that $BP_\varphi$ has an accepting run from $\langle \langle p, \varphi \rangle, \omega \rangle$ by applying the induction on $i - j$ similar to the case $\varphi = E[\varphi_1 U^a \varphi_2]$ by applying Lemma 8.

Now let us consider the second case, where $P$ has an abstract path $\pi = \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \langle p_2, \omega_2 \rangle ...$ starting from $\langle p, \omega \rangle$ where $\langle p, \omega \rangle = \langle p_0, \omega_0 \rangle$ such that $\langle p_{z_i}, \omega_{z_i} \rangle \models_{\lambda_f} \varphi_2$ for every $i \geq 0$ (23).

We need to show that $BP_\varphi$ has an accepting run from $\langle \langle p, \varphi \rangle, \omega \rangle$.

* According to Lemma 8 and the fact that $\langle p_{z_{i+1}}, \omega_{z_{i+1}} \rangle$ is an abstract-successor of $\langle p_{z_i}, \omega_{z_i} \rangle$, we get that $\langle \langle p_{z_i}, \varphi \rangle, \omega_{z_i} \rangle \Rightarrow_{BP_\varphi} \langle \langle p_{z_i}, \varphi_2 \rangle, \omega_{z_i} \rangle \land \langle \langle p_{z_{i+1}}, \varphi \rangle, \omega_{z_{i+1}} \rangle$ for every $i \geq 0$. Therefore, $\langle \langle p_{z_0}, \varphi \rangle, \omega_{z_0} \rangle \Rightarrow_{BP_\varphi} \land_{i \geq 0} \langle \langle p_{z_i}, \varphi_2 \rangle, \omega_{z_i} \rangle \land_{i \geq 1} \langle \langle p_{z_{i+1}}, \varphi \rangle, \omega_{z_{i+1}} \rangle$ (24)

* Since $\langle p_{z_i}, \omega_{z_i} \rangle \models_{\lambda_f} \varphi_2$ (by (23)), we get that $BP_\varphi$ has an accepting run from $\langle \langle p_{z_i}, \varphi_2 \rangle, \omega_{z_i} \rangle$ for every $i \geq 0$ (by the induction hypothesis) (25).

* In addition, for every $i \geq 0$, $\langle p_{z_i}, \varphi \rangle$ is an accepting control location, then, the path $\langle \langle p_{z_0}, \varphi \rangle, \omega_{z_0} \rangle \langle \langle p_{z_1}, \varphi \rangle, \omega_{z_1} \rangle \langle \langle p_{z_2}, \varphi \rangle, \omega_{z_2} \rangle ...$ is accepted (26).

From (24), (25), (26), we obtain that $BP_\varphi$ has an accepting run from $\langle \langle p, \varphi \rangle, \omega_{z_0} \rangle$. In other words, $BP_\varphi$ has an accepting run from $\langle \langle p, \varphi \rangle, \omega \rangle$. The property holds for this case.

(\iff) Assume that $BP_\varphi$ has an accepting run from the configuration $\langle \langle p, \varphi \rangle, \omega \rangle$, we need to prove that $\langle p, \omega \rangle \models_{\lambda_f} \varphi$. In what follows, we prove this by induction on the structure of $\varphi$.

Proof. 

- Base case:
Chapter 4. Branching Temporal Logic of Calls and Returns for Pushdown Systems

- $\varphi = e(e \in AP)$. $BP_\varphi$ has an accepting run from $\langle \langle p, e \rangle, \omega \rangle$ if $\langle \langle p, e \rangle, \omega \rangle$ must have immediate successors. From all transition rules of $BP_\varphi$, the unique way to have immediate successors of $\langle \langle p, e \rangle, \omega \rangle$ is from the rules in $(a1)$, which means that $\langle \langle p, e \rangle, \omega \rangle \Rightarrow_{BP_\varphi} \langle \langle p, e \rangle, \omega \rangle$. Thus, from the condition in the transition rules in $(a1)$, we obtain that $p \in f(e)$. Therefore, $\langle p, \omega \rangle \models_{\lambda_f} e$. In other words, $\langle p, \omega \rangle \models_{\lambda_f} \varphi$. The property holds for this case.

- $\varphi = \neg e(e \in AP)$. $BP_\varphi$ has an accepting run from $\langle \langle p, \neg e \rangle, \omega \rangle$ if $\langle \langle p, \neg e \rangle, \omega \rangle$ must have immediate successors. From all transition rules of $BP_\varphi$, the unique way to have immediate successors of $\langle \langle p, \neg e \rangle, \omega \rangle$ is from the rules in $(a2)$, which means that $\langle \langle p, \neg e \rangle, \omega \rangle \Rightarrow_{BP_\varphi} \langle \langle p, \neg e \rangle, \omega \rangle$. Thus, from the condition in the transition rules in $(a2)$, we obtain that $p \notin f(e)$. Therefore, $\langle p, \omega \rangle \models_{\lambda_f} \neg e$. In other words, $\langle p, \omega \rangle \models_{\lambda_f} \varphi$. The property holds for this case.

- Induction Step:

- $\varphi = \varphi_1 \lor \varphi_2$

  From the transition rules in $(a4)$, we get that $\langle \langle p, \varphi \rangle, \omega \rangle \Rightarrow_{BP_\varphi} \langle \langle p, \varphi_1 \rangle, \omega \rangle$ and $\langle \langle p, \varphi \rangle, \omega \rangle \Rightarrow_{BP_\varphi} \langle \langle p, \varphi_2 \rangle, \omega \rangle$. Thus, $BP_\varphi$ has an accepting run from $\langle \langle p, \varphi_1 \rangle, \omega \rangle$ or $BP_\varphi$ has an accepting run from $\langle \langle p, \varphi_2 \rangle, \omega \rangle$ (27).

  By applying the induction hypothesis, we obtain that:

  * $BP_\varphi$ has an accepting run from $\langle \langle p, \varphi_1 \rangle, \omega \rangle$ implies that $\langle p, \omega \rangle \models_{\lambda_f} \varphi_1$ (28)

  * $BP_\varphi$ has an accepting run from $\langle \langle p, \varphi_2 \rangle, \omega \rangle$ implies that $\langle p, \omega \rangle \models_{\lambda_f} \varphi_2$ (29)

  From (27), (28), (29), we get that $BP_\varphi$ has an accepting run from $\langle \langle p, \varphi \rangle, \omega \rangle$ implies $\langle p, \omega \rangle \models_{\lambda_f} \varphi_1$ or $\langle p, \omega \rangle \models_{\lambda_f} \varphi_2$. In other words, $BP_\varphi$ has an accepting run from $\langle \langle p, \varphi \rangle, \omega \rangle$ implies $\langle p, \omega \rangle \models_{\lambda_f} \varphi_1 \lor \varphi_2$. The property holds for this case.

- $\varphi = \varphi_1 \land \varphi_2$

  From the transition rules in $(a3)$, we get that $\langle \langle p, \varphi \rangle, \omega \rangle \Rightarrow_{BP_\varphi} \langle \langle p, \varphi_1 \rangle, \omega \rangle \land \langle \langle p, \varphi_2 \rangle, \omega \rangle$. Thus, $BP_\varphi$ has an accepting run from $\langle \langle p, \varphi_1 \rangle, \omega \rangle$ or $BP_\varphi$ has an accepting run from $\langle \langle p, \varphi_2 \rangle, \omega \rangle$ and $BP_\varphi$ has an accepting run from $\langle \langle p, \varphi_2 \rangle, \omega \rangle$ (30).

  By applying the induction hypothesis, we obtain that:

  * $BP_\varphi$ has an accepting run from $\langle \langle p, \varphi_1 \rangle, \omega \rangle$ implies that $\langle p, \omega \rangle \models_{\lambda_f} \varphi_1$ (31)
4.3. BCARET Model-Checking for Pushdown Systems

* $\mathcal{BP}_\varphi$ has an accepting run from $\langle \langle p, \varphi_2 \rangle, \omega \rangle$ implies that $\langle p, \omega \rangle \models_{\lambda_f} \varphi_2 \ (32)$

From (30), (31), (32), we get that $\mathcal{BP}_\varphi$ has an accepting run from $\langle \langle p, \varphi \rangle, \omega \rangle$ implies $\langle p, \omega \rangle \models_{\lambda_f} \varphi_1$ and $\langle p, \omega \rangle \models_{\lambda_f} \varphi_2$. In other words, $\mathcal{BP}_\varphi$ has an accepting run from $\langle \langle p, \varphi \rangle, \omega \rangle$ implies $\langle p, \omega \rangle \models_{\lambda_f} \varphi_1 \land \varphi_2$. The property holds for this case.

- $\varphi = AX^g \varphi_1$

  From the transition rules in (a6), we get that $\langle \langle p, \varphi \rangle, \omega \rangle \Rightarrow_{\mathcal{BP}_\varphi} \{ \langle \langle p_1, \varphi_1 \rangle, \omega_1 \rangle, ..., \langle \langle p_n, \varphi_1 \rangle, \omega_n \rangle \}$ where for every $1 \leq i \leq n$, $\langle p_i, \omega_i \rangle$ is an immediate successor (since the global-successor is the immediate successor) of $\langle p, \omega \rangle$ on $\mathcal{P}$. Thus, $\mathcal{BP}_\varphi$ has an accepting run from $\langle \langle p, \varphi \rangle, \omega \rangle$ iff for every $1 \leq i \leq n$, $\mathcal{BP}_\varphi$ has an accepting run from $\langle \langle p_i, \varphi_1 \rangle, \omega_i \rangle$ (33).

  By applying the induction hypothesis, we obtain that:

  * For every $1 \leq i \leq n$, $\mathcal{BP}_\varphi$ has an accepting run from $\langle \langle p_i, \varphi_1 \rangle, \omega_i \rangle$ implies that $\langle p_i, \omega_i \rangle \models_{\lambda_f} \varphi_1$ (34).

From (33), (34), we get that $\mathcal{BP}_\varphi$ has an accepting run from $\langle \langle p, \varphi \rangle, \omega \rangle$ implies $\langle p_i, \omega_i \rangle \models_{\lambda_f} \varphi_1$ for every $1 \leq i \leq n$. In other words, $\mathcal{BP}_\varphi$ has an accepting run from $\langle \langle p, \varphi \rangle, \omega \rangle$ implies $\langle p, \omega \rangle \models_{\lambda_f} AX^g \varphi_1$ (by the semantic of BCARET). The property holds for this case.

- $\varphi = EX^g \varphi_1$. This case is similar to the case $\varphi = AX^g \varphi_1$.

- $\varphi = EX^a \varphi_1$

  From the transition rules in (a7) and Lemma 6, we get that $\langle \langle p, \varphi \rangle, \omega \rangle \Rightarrow_{\mathcal{BP}_\varphi} \{ \langle \langle p_k, \varphi_1 \rangle, \omega_k \rangle \}$ where $\langle p_k, \omega_k \rangle$ is an abstract successor of $\langle p, \omega \rangle$ on $\mathcal{P}$. Thus, $\mathcal{BP}_\varphi$ has an accepting run from $\langle \langle p, \varphi \rangle, \omega \rangle$ iff there exists an abstract successor $\langle p_k, \omega_k \rangle$ of $\langle p, \omega \rangle$ s.t. $\mathcal{BP}_\varphi$ has an accepting run from $\langle \langle p_k, \varphi_1 \rangle, \omega_k \rangle$ (35).

  By applying the induction hypothesis, we obtain that:

  * $\mathcal{BP}_\varphi$ has an accepting run from $\langle \langle p_k, \varphi_1 \rangle, \omega_k \rangle$ implies that $\langle p_k, \omega_k \rangle \models_{\lambda_f} \varphi_1$ (36).

From (35), (36), we get that $\mathcal{BP}_\varphi$ has an accepting run from $\langle \langle p, \varphi \rangle, \omega \rangle$ implies that there exists an abstract successor $\langle p_k, \omega_k \rangle$ of $\langle p, \omega \rangle$ s.t. $\langle p_k, \omega_k \rangle \models_{\lambda_f} \varphi_1$. Therefore, $\mathcal{BP}_\varphi$ has an accepting run from $\langle \langle p, \varphi \rangle, \omega \rangle$ implies that $\langle p, \omega \rangle \models_{\lambda_f} EX^a \varphi_1$ (by the semantic of BCARET). The property holds for this case.

- $\varphi = AX^g \varphi_1$. This case is similar to the case $\varphi = EX^g \varphi_1$. 

Chapter 4. Branching Temporal Logic of Calls and Returns for Pushdown Systems

- Case $\varphi = E[\varphi_1 U^a \varphi_2]$: Let $\rho$ be an accepting run of $BP_\varphi$ starting from $\langle \langle p, \varphi \rangle, \omega \rangle$ (37). Note that $\rho$ is a tree. We need to show that $\langle p, \omega \rangle \models _\lambda \varphi$.

  From transition rules in (a9), each configuration $\langle \langle p_i, \varphi \rangle, \omega_i \rangle$ in $\rho$ has either (38) two children $\{\langle p_i, \varphi_1 \rangle, \omega_i \}, \{\langle p_{i+1}, \varphi \rangle, \omega_{i+1} \}$ or (39) one child $\{\langle p_i, \varphi_2 \rangle, \omega_i \}$.

  Let’s take Figure 4.4a for an illustration in this case. Let $\pi$ be the run of $P$ corresponding to the run $\rho$ of $BP_\varphi$. Firstly, we show that there must exist a configuration $\langle \langle p_n, \varphi \rangle, \omega_n \rangle$ in $\rho$ s.t. $\langle \langle p_n, \varphi \rangle, \omega_n \rangle$ has only one child $\langle \langle p_n, \varphi_2 \rangle, \omega_n \rangle$. Suppose that this is not the case, then, the path of $\rho \pi = \langle \langle p_0, \varphi \rangle, \omega_0 \rangle \langle \langle p_1, \varphi \rangle, \omega_1 \rangle \langle \langle p_2, \varphi \rangle, \omega_2 \rangle \ldots$ is not accepted because $\langle p_i, \varphi \rangle \notin F$ for every $p \in P$, $\varphi = E[\varphi_1 U^a \varphi_2]$. One path of $\rho$ is not accepted implies that $\rho$ is not an accepting run. This contradicts with (37). Thus, there must exist a configuration $\langle \langle p_n, \varphi \rangle, \omega_n \rangle$ in $\rho$ s.t. $\langle \langle p_n, \varphi \rangle, \omega_n \rangle$ has only one child $\langle \langle p_n, \varphi_2 \rangle, \omega_n \rangle$. Therefore, $\rho$ has a path $\pi' = \langle \langle p_0, \varphi \rangle, \omega_0 \rangle \langle \langle p_1, \varphi \rangle, \omega_1 \rangle \ldots \langle \langle p_n, \varphi \rangle, \omega_n \rangle \langle \langle p_n, \varphi_2 \rangle, \omega_n \rangle \ldots$ where $\langle \langle p_0, \varphi \rangle, \omega_0 \rangle = \langle \langle p, \varphi \rangle, \omega \rangle$ s.t. for every $0 \leq i < n$, $\langle \langle p_i, \varphi \rangle, \omega_i \rangle$ has two children $\{\langle p_i, \varphi_1 \rangle, \omega_i \}, \{\langle p_{i+1}, \varphi \rangle, \omega_{i+1} \}$ and $\langle \langle p_n, \varphi \rangle, \omega_n \rangle$ has only one child $\langle \langle p_n, \varphi_2 \rangle, \omega_n \rangle$ (see Figure 4.4a).

  Since $\rho$ is an accepting run of $BP_\varphi$, we obtain that:

  * for every $0 \leq i < n$, $BP_\varphi$ has an accepting run from $\langle \langle p_i, \varphi_1 \rangle, \omega_i \rangle$ (40)
  * $BP_\varphi$ has an accepting run from $\langle \langle p_n, \varphi_2 \rangle, \omega_n \rangle$ (41).

  By applying the induction hypothesis, we get that:

  * (40) implies that $\langle p_i, \omega_i \rangle \models _\lambda \varphi_1$ for every $0 \leq i < n$ (42)
  * (41) implies that $\langle p_n, \omega_n \rangle \models _\lambda \varphi_2$ (43)

  From (42), (43), we get that $\langle p, \omega \rangle \models _\lambda \varphi$. The property holds for this case.

- Case $\varphi = E[\varphi_1 U^a \varphi_2]$: Let $\rho$ be an accepting run of $BP_\varphi$ starting from $\langle \langle p, \varphi \rangle, \omega \rangle$ (44). Note that $\rho$ is a tree. We need to show that $\langle p, \omega \rangle \models _\lambda \varphi$.

  From transition rules in (a10) and Lemma 7, we get that for each configuration $\langle \langle p_z, \varphi \rangle, \omega_z \rangle$ in $\rho$: either (45) $\langle \langle p_z, \varphi \rangle, \omega_z \rangle \Rightarrow_{BP_\varphi} \{\langle p_{z_1}, \varphi \rangle, \omega_{z_1} \}, \{\langle p_{z_{i+1}}, \varphi \rangle, \omega_{z_{i+1}} \}$ where $p_{z_{i+1}}$ is the abstract successor of $\langle p_z, \omega_z \rangle$ or (46) $\langle \langle p_z, \varphi \rangle, \omega_z \rangle \Rightarrow_{BP_\varphi} \{\langle p_z, \varphi_2 \rangle, \omega_z \}$.

  Let’s take Figure 4.4b for an illustration in this case. Let $\pi$ be the run of $P$ corresponding to $\rho$. Firstly, we show that there must exist a configuration $\langle \langle p_{z_n}, \varphi \rangle, \omega_{z_n} \rangle$ in $\rho$ s.t. $\langle \langle p_{z_n}, \varphi \rangle, \omega_{z_n} \rangle \Rightarrow_{BP_\varphi}$
4.3. BCARET Model-Checking for Pushdown Systems

Figure 4.4: $\mathcal{BP}_\varphi$ has an accepting run $\rho$ from $\langle (p, \varphi), \omega \rangle$
Chapter 4. Branching Temporal Logic of Calls and Returns for
Pushdown Systems

\{\langle p_{z_2}, \varphi_2 \rangle, \omega_{z_2} \}\}. Suppose that this is not the case, then, the path of \( p \) \( \pi' = \langle p_{z_0}, \varphi \rangle \ldots \langle p_{z_2}, \varphi \rangle \ldots \langle p_{z_3}, \varphi \rangle \ldots \) is not accepted because \( \langle p, \varphi \rangle \notin F \) for every \( p \in P \), \( \varphi = E[\varphi_1 U^a \varphi_2] \). One path of \( p \) is not accepted implies that \( p \) is not an accepting run. This contradicts with (44). Thus, there must exist a configuration \( \langle p_{z_n}, \varphi \rangle, \omega_{z_n} \rangle \) in \( p \) s.t. \( \langle p_{z_0}, \varphi \rangle, \omega_{z_0} \rangle \Rightarrow_{BP_\varphi} \{\langle p_{z_1}, \varphi \rangle, \omega_{z_1} \rangle \}. Therefore, \( p \) has a path \( \pi'' = \langle p_{z_0}, \varphi \rangle, \omega_{z_0} \rangle \ldots \langle p_{z_1}, \varphi \rangle, \omega_{z_1} \rangle \ldots \langle p_{z_n}, \varphi \rangle, \omega_{z_n} \rangle \langle p_{z_0}, \varphi \rangle, \omega_{z_0} \rangle \ldots \langle p_{z_2}, \varphi \rangle, \omega_{z_2} \rangle \ldots \langle p_{z_3}, \varphi \rangle, \omega_{z_3} \rangle \ldots \) where \( \langle p_{z_0}, \varphi \rangle, \omega_{z_0} \rangle = \langle p, \varphi \rangle, \omega \rangle \) s.t. for every \( 0 \leq i < n \), \( \langle p_{z_i}, \varphi \rangle, \omega_{z_i} \rangle \Rightarrow_{BP_\varphi} \{\langle p_{z_{i+1}}, \varphi \rangle, \omega_{z_{i+1}} \rangle \} \) and \( \langle p_{z_0}, \varphi \rangle, \omega_{z_0} \rangle \) has only one child \( \langle p_{z_n}, \varphi_2 \rangle, \omega_{z_n} \rangle \) (see Figure 4.4b). Since \( p \) is an accepting run of \( BP_\varphi \), we obtain that:

* for every \( 0 \leq i < n \), \( BP_\varphi \) has an accepting run from \( \langle p_{z_i}, \varphi_1 \rangle, \omega_{z_i} \rangle \) (47)
* \( BP_\varphi \) has an accepting run from \( \langle p_{z_n}, \varphi_2 \rangle, \omega_{z_n} \rangle \) (48)

By applying the induction hypothesis, we get that:

* (47) implies that \( \langle p_{z_i}, \varphi_1 \rangle \models \lambda \varphi_1 \) for every \( 0 \leq i < n \) (49)
* (48) implies that \( \langle p_{z_n}, \varphi_2 \rangle \models \lambda \varphi \) (50)

From (49), (50), we get that \( \langle p_{z_0}, \varphi_0 \rangle \models \lambda \varphi \). In other words, \( \langle p, \varphi \rangle \models \lambda \varphi \). The property holds for this case.

- Case \( \varphi = A[\varphi_1 U^g \varphi_2] \): This case is similar to the case \( \varphi = E[\varphi_1 U^g \varphi_2] \). The property holds.

- Case \( \varphi = E[\varphi_1 R^g \varphi_2] \): Let \( p \) be an accepting run of \( BP_\varphi \) starting from \( \langle p, \varphi \rangle, \omega \rangle \). We need to show that \( \langle p, \varphi \rangle \models \lambda \varphi \).

From transition rules in (a13), each configuration \( \langle p_i, \varphi_i \rangle, \omega_i \rangle \) in \( p \) has two children (51) \{\langle p_i, \varphi_2 \rangle, \omega_1 \rangle, \langle p_{i+1}, \varphi \rangle, \omega_{i+1} \rangle \} or (52) \{\langle p_i, \varphi_2 \rangle, \omega_i \rangle, \langle p_i, \varphi_1 \rangle, \omega_i \rangle \}. There are two possibilities:

1. \( \varphi_1 \) eventually occurs. In other words, there exists a configuration \( \langle p_n, \varphi_1 \rangle, \omega_n \rangle \) in \( p \) whose two children are \( \langle p_n, \varphi_1 \rangle, \omega_n \rangle \) and \( \langle p_n, \varphi_2 \rangle, \omega_n \rangle \). Therefore, \( p \) has a path \( \pi' = \langle p_0, \varphi_1 \rangle, \omega_0 \rangle \langle p_1, \varphi_1 \rangle, \omega_1 \rangle \ldots \langle p_n, \varphi_1 \rangle, \omega_n \rangle \ldots \) starting from \( \langle p, \varphi \rangle, \omega \rangle \) where \( \langle p_i, \varphi_i \rangle, \omega_i \rangle = \langle p_i, \varphi_i \rangle, \omega_i \rangle \) s.t. for every \( 0 \leq i < n \), \( \langle p_i, \varphi_i \rangle, \omega_i \rangle \) has two children \{\langle p_i, \varphi_2 \rangle, \omega_i \rangle, \langle p_{i+1}, \varphi \rangle, \omega_{i+1} \rangle \} and \( \langle p_n, \varphi_1 \rangle, \omega_n \rangle \) has two children \{\langle p_n, \varphi_2 \rangle, \omega_n \rangle, \langle p_n, \varphi_1 \rangle, \omega_n \rangle \}. Since \( p \) is an accepting run of \( BP_\varphi \), we obtain that:
4.3. BCARET Model-Checking for Pushdown Systems

- for every $0 \leq i < n$, $\mathcal{BP}_\varphi$ has an accepting run from $\langle (p_i, \varphi_2), \omega_i \rangle$

(53)

- $\mathcal{BP}_\varphi$ has an accepting run from $\langle (p_n, \varphi_1), \omega_n \rangle$ and $\langle (p_n, \varphi_2), \omega_n \rangle$

(54)

By applying the induction hypothesis, we get that:

- (53) implies that $\langle p_i, \omega_i \rangle \models_{\lambda_f} \varphi_2$ for every $0 \leq i < n$ (55)

- (54) implies that $\langle p_n, \omega_n \rangle \models_{\lambda_f} \varphi_1$ and $\langle p_n, \omega_n \rangle \models_{\lambda_f} \varphi_2$ (56)

From (55), (56), we get that $\langle p_0, \omega_0 \rangle \models_{\lambda_f} \varphi$. In other words, $\langle p, \omega \rangle \models_{\lambda_f} \varphi$. The property holds for this case.

2. $\varphi_1$ never occurs. In other words, every configuration $\langle (p_i, \varphi), \omega_i \rangle$ in $\rho$ has two children $\{\langle (p_i, \varphi_2), \omega_i \rangle, \langle (p_i+1, \varphi), \omega_{i+1} \rangle\}$. Therefore, $\rho$ has a path $\pi' = \langle (p_0, \varphi), \omega_0 \rangle \langle (p_1, \varphi), \omega_1 \rangle \ldots$ starting from $\langle (p, \varphi), \omega \rangle$ where $\langle (p, \varphi), \omega \rangle = \langle (p_0, \varphi), \omega_0 \rangle$ s.t. for every $i \geq 0$, $\langle (p_i, \varphi), \omega_i \rangle$ has two children $\{\langle (p_i, \varphi_2), \omega_i \rangle, \langle (p_i+1, \varphi), \omega_{i+1} \rangle\}$. Since $\rho$ is an accepting run, we get that $\mathcal{BP}_\varphi$ has an accepting run from the configuration $\langle (p_i, \varphi_2), \omega_i \rangle$ for every $i \geq 0$. Then, by applying the induction hypothesis, we obtain $\langle p_i, \omega_i \rangle \models_{\lambda_f} \varphi_1$ for every $i \geq 0$. According to the semantics of BCARET, we get that $\langle p_0, \omega_0 \rangle \models_{\lambda_f} \varphi$. In other words, $\langle p, \omega \rangle \models_{\lambda_f} \varphi$. The property holds for this case.

- Case $\varphi = E[\varphi_1 R^e \varphi_2]$: Let $\rho$ be an accepting run of $\mathcal{BP}_\varphi$ starting from $\langle (p, \varphi), \omega \rangle$. We need to show that $\langle p, \omega \rangle \models_{\lambda_f} \varphi$.

From transition rules in (a15) and Lemma 8, we get that for each configuration $\langle (p_1, \varphi), \omega_2 \rangle$ in $\rho$: either (57) $\langle (p_2, \varphi), \omega_3 \rangle \Rightarrow_{\mathcal{BP}_\varphi} \{\langle (p_2, \varphi_2), \omega_3 \rangle, \langle (p_2, \varphi), \omega_3 \rangle\}$ or (58) $\langle (p_3, \varphi), \omega_3 \rangle \Rightarrow_{\mathcal{BP}_\varphi} \{\langle (p_3, \varphi_2), \omega_3 \rangle, \langle (p_3, \varphi_1), \omega_3 \rangle\}$. There are two possibilities:

1. $\varphi_1$ eventually occurs. In other words, there exists a configuration $\langle (p_{z_i}, \varphi), \omega_{z_i} \rangle$ in $\rho$ s.t. $\langle (p_{z_i}, \varphi), \omega_{z_i} \rangle \Rightarrow_{\mathcal{BP}_\varphi} \{\langle (p_{z_i}, \varphi_2), \omega_{z_i} \rangle, \langle (p_{z_i}, \varphi_1), \omega_{z_i} \rangle\}$. Therefore, $\rho$ has a path $\pi' = \langle (p_0, \varphi), \omega_0 \rangle \langle (p_1, \varphi), \omega_1 \rangle \ldots$ where $\langle (p_0, \varphi), \omega_0 \rangle = \langle (p, \varphi), \omega \rangle$ s.t. for every $0 \leq i < n$, $\langle (p_i, \varphi), \omega_i \rangle \Rightarrow_{\mathcal{BP}_\varphi} \{\langle (p_{i+1}, \varphi), \omega_{i+1} \rangle, \langle (p_{i+1}, \varphi), \omega_{i+1} \rangle\}$ where $\langle (p_{i+1}, \omega_{i+1} \rangle$ is the abstract successor of $\langle (p_i, \omega_i \rangle$ and $\langle (p_{i+1}, \varphi), \omega_{i+1} \rangle \Rightarrow_{\mathcal{BP}_\varphi} \{\langle (p_{i+1}, \varphi_2), \omega_{i+1} \rangle, \langle (p_{i+1}, \varphi_1), \omega_{i+1} \rangle\}$. Since $\rho$ is an accepting run of $\mathcal{BP}_\varphi$, we obtain that:

- for every $0 \leq i < n$, $\mathcal{BP}_\varphi$ has an accepting run from $\langle (p_{z_i}, \varphi_2), \omega_{z_i} \rangle$ (59)
Chapter 4. Branching Temporal Logic of Calls and Returns for Pushdown Systems

108

Given a PDS

Theorem 14. Given a PDS \( P = (P, \Gamma, \Delta) \), a set of atomic propositions \( AP \), a labelling function \( f : AP \to 2^P \) and a BCARET formula \( \varphi \), for every configuration \( \langle p, \omega \rangle \in P \times \Gamma^* \), whether or not \( \langle p, \omega \rangle \) satisfies \( \varphi \) can be solved in time \( O(|P|^2|\varphi|\cdot|\Gamma||\Delta|\cdot|\varphi|\cdot2^{2|P|}\cdot2^{2|\varphi|} + 2|P||\varphi|\cdot|\omega|) \)

4.4 BCARET model-checking for PDSs with regular valuations

Up to now, we have considered the standard model-checking problem for BCARET, where the validity of an atomic proposition depends only on the
control state, not on the stack. In this section, we go further and consider model-checking with regular valuations where the set of configurations in which an atomic proposition holds is a regular set of configurations (see Sections 2.4 and 4.1 for a formal definition of regular valuations).

### 4.4.1 Multi Automata

**Definition 16.** [BEM97] Let \( \mathcal{P} = (P, \Gamma, \Delta) \) be a PDS. A \( \mathcal{P} \)-Multi-Automaton (MA for short) is a tuple \( A = (Q, \Gamma, \delta, I, Q_f) \), where \( Q \) is a finite set of states, \( \delta \subseteq Q \times \Gamma \times Q \) is a finite set of transition rules, \( I = P \subseteq Q \) is a set of initial states, \( Q_f \subseteq Q \) is a set of final states.

The transition relation \( \rightarrow_\delta \subseteq Q \times \Gamma^* \times Q \) is defined as follows:

- \( q \xrightarrow{\varepsilon}_\delta q \) for every \( q \in Q \)
- \( q \xrightarrow{\gamma}_\delta q' \) if \( (q, \gamma, q') \in \delta \)
- if \( q \xrightarrow{\omega}_\delta q' \) and \( q' \xrightarrow{\gamma}_\delta q'' \), then \( q \xrightarrow{\omega \gamma}_\delta q'' \)

A recognizes a configuration \( \langle p, \omega \rangle \) where \( p \in P \), \( \omega \in \Gamma^* \) iff \( p \xrightarrow{\omega \gamma}_\delta q \) for some \( q \in Q_f \). The language of \( A \), \( L(A) \), is the set of all configurations which are recognized by \( A \). A set of configurations is regular if it is recognized by some Multi-Automaton.

### 4.4.2 From BCARET model checking of PDSs with regular valuations to the membership problem in ABPDSs

Let \( \mathcal{P} = (P, \Gamma, \Delta) \) be a PDS. We suppose w.l.o.g. that \( \mathcal{P} \) has a bottom stack symbol \( \sharp \) that is never popped from the stack. Let \( AP \) be a set of atomic propositions. Let \( \varphi \) be a BCARET formula over \( AP \), \( \lambda : AP \rightarrow 2^{P \times \Gamma^*} \) be a labelling function s.t. for every \( e \in AP \), \( \lambda(e) \) is a regular set of configurations. Given a configuration \( c_0 \), we propose in this section an algorithm to check whether \( c_0 \models_\lambda \varphi \). Intuitively, we compute an ABPDS \( \mathcal{BP}_\varphi \) s.t. \( \mathcal{BP}_\varphi \) recognizes a configuration \( c \) of \( \mathcal{P} \) iff \( c \models_\lambda \varphi \). Then, to check if \( c_0 \) satisfies \( \varphi \), we will check whether \( \mathcal{BP}_\varphi \) recognizes \( c_0 \).

For every \( e \in AP \), since \( \lambda(e) \) is a regular set of configurations, let \( M_e = (Q_e, \Gamma, \delta_e, I_e, F_e) \) be a multi-automaton s.t. \( L(M_e) = \lambda(e) \). \( M_{\neg e} = (Q_{\neg e}, \Gamma, \delta_{\neg e}, I_{\neg e}, F_{\neg e}) \) be a multi-automaton s.t. \( L(M_{\neg e}) = P \times \Gamma^* \setminus \lambda(e) \), which means \( M_{\neg e} \) will recognize the complement of \( \lambda(e) \) that is the set of configurations in which \( e \) doesn’t hold. Note that for every \( e \in AP \), the initial
states of $M_e$ and $M_{l_e}$ are the control locations $p \in \mathcal{P}$. Thus, to distinguish between the initial states of these two automata, we will denote the initial state corresponding to the control location $p$ in $M_e$ (resp. $M_{l_e}$) by $p_e$ (resp. $p_{l_e}$). Let $AP^+(\varphi) = \{ e \in AP \mid e \in Cl(\varphi) \}$ and $AP^-(\varphi) = \{ e \in AP \mid \neg e \in Cl(\varphi) \}$.

Let $BP'_{\varphi} = (P'', \Gamma'', \Delta'', F')$ be the ABPDS defined as follows:

- $P'' = P \cup P \times Cl(\varphi) \cup \{ p_1 \} \cup \bigcup_{e \in AP^+(\varphi)} Q_e \cup \bigcup_{e \in AP^-(\varphi)} Q_{\neg e}$
- $\Gamma'' = \Gamma \cup (\Gamma \times Cl(\varphi)) \cup \{ \gamma_1 \}$
- $F' = F_1 \cup F_2 \cup F_3$ where
  - $F_1 = \bigcup_{e \in AP^+(\varphi)} F_e$
  - $F_2 = \bigcup_{e \in AP^-(\varphi)} F_{\neg e}$
  - $F_3 = \{ P \times Cl_R(\varphi) \}$ where $Cl_R(\varphi)$ is the set of formulas of $Cl(\varphi)$ in the form $E[\varphi_1 R^b \varphi_2]$ or $A[\varphi_1 R^b \varphi_2]$ ($b \in \{ g, a \}$)

The transition relation $\Delta''$ is the smallest set of transition rules defined as follows: $\Delta \subseteq \Delta''$, $\Delta'_0 \subseteq \Delta''$ where $\Delta'_0$ is the transitions of $\Delta'$ that are created by the rules from $(\alpha 3)$ to $(\alpha 17)$ and such that:

$(\beta 1)$ for every $p \in P$, $e \in AP^+(\varphi)$, $\gamma \in \Gamma$: $\langle \langle p, e \rangle, \gamma \rangle \rightarrow \langle p_e, \gamma \rangle \in \Delta''$

$(\beta 2)$ for every $p \in P$, $e \in AP^-(\varphi)$, $\gamma \in \Gamma$: $\langle \langle p, \neg e \rangle, \gamma \rangle \rightarrow \langle p_{\neg e}, \gamma \rangle \in \Delta''$

$(\beta 3)$ for every $(q_1, \gamma, q_2) \in (\bigcup_{e \in AP^+(\varphi)} \delta_e) \cup (\bigcup_{e \in AP^-(\varphi)} \delta_{\neg e})$: $\langle q_1, \gamma \rangle \rightarrow \langle q_2, \epsilon \rangle \in \Delta''$

$(\beta 4)$ for every $q \in (\bigcup_{e \in AP^+(\varphi)} F_e) \cup (\bigcup_{e \in AP^-(\varphi)} F_{\neg e})$: $\langle q, \# \rangle \rightarrow \langle q, \# \rangle \in \Delta''$

Intuitively, we compute the ABPDS $BP'_{\varphi}$ such that $BP'_{\varphi}$ has an accepting run from $\langle \langle p, \phi \rangle, \omega \rangle$ iff the configuration $\langle p, \omega \rangle$ satisfies $\phi$ according to the regular labellings $M_e$ for every $e \in AP$. The only difference with the previous case of standard valuations, where an atomic proposition holds at a configuration depends only on the control location of that configuration, not on its stack, comes from the interpretation of the atomic proposition $e$. This is why $\Delta''$ contains $\Delta$ and $\Delta'_0$ (which are the transitions of $BP_{\varphi}$ that don’t consider the atomic propositions). Here the rules $(\beta_1)$ – $(\beta_4)$ deal with the cases $e$, $\neg e$ ($e \in AP$). Given $p \in P$, $\phi = e \in AP$, $\omega \in \Gamma^*$, we get that the ABPDS $BP'_{\varphi}$ should accept $\langle \langle p, e \rangle, \omega \rangle$ iff $\langle p, \omega \rangle \in L(M_e)$. To check whether $\langle p, \omega \rangle \in L(M_e)$, we let $BP'_{\varphi}$ go to state $p_e$, the initial state corresponding to $p$ in $M_e$ by adding rules in $(\beta 1)$; and then, from this state, we will check whether $\omega$ is accepted by $M_e$. This is ensured by the transition rules in $(\beta 3)$ and $(\beta 4)$. $(\beta 3)$ lets
4.4. BCARET model-checking for PDSs with regular valuations

$BP'_\varphi$ mimic a run of $M_e$ on $\omega$, i.e., if $BP'_\varphi$ is in a state $q_1$ with $\gamma$ on the top of the stack, and if $(q_1, \gamma, q_2)$ is a transition rule in $M_e$, then, $BP'_\varphi$ will move to state $q_2$ and pop $\gamma$ from its stack. Note that popping $\gamma$ allows us to check the rest of the word. In $M_e$, a configuration is accepted if the run with the word $\omega$ reaches the final state in $F_e$; i.e., if $BP'_\varphi$ reaches a state $q_1' \in F_e$ with an empty stack, i.e., with a stack containing the bottom stack symbol $\sharp$. Thus, we add $F_e$ as a set of accepting control locations in $BP'_\varphi$. Since $BP'_\varphi$ only recognizes infinite paths, $(\beta 4)$ adds a loop on every configuration $\langle q, \sharp \rangle$ where $q \in F_e$. The intuition behind the transition rules in $(\beta 2)$ is similar to that of $(\beta 1)$. They correspond to the case where $\phi = \neg e$.

**Theorem 15.** Given a PDS $\mathcal{P} = (P, \Gamma, \Delta)$, a set of atomic propositions $AP$, a regular labelling function $\lambda : AP \rightarrow 2^{P \times \Gamma^*}$ and a BCARET formula $\varphi$, we can compute an ABPDS $BP'_\varphi$ such that for every configuration $\langle p, \omega \rangle$, $\langle p, \omega \rangle \models_\lambda \varphi$ iff $BP'_\varphi$ has an accepting run from the configuration $\langle \langle p, \varphi \rangle, \omega \rangle$.

**Formal proof.** The only difference with the previous case of standard valuations, where an atomic proposition holds at a configuration depends only on the control location of that configuration, not on its stack, comes from the interpretation of the atomic proposition $e$. We prove the following two directions.

$\implies$ Assume that $\langle p, \omega \rangle \models_\lambda \varphi$, we need to prove that $BP'_\varphi$ has an accepting run from $\langle \langle p, \varphi \rangle, \omega \rangle$. In what follows, we show how this is ensured by induction on the structure of the BCARET formula $\varphi$.

**Proof.**

- **Base case:**
  
  $\varphi = e(e \in AP)$: $\langle p, \omega \rangle \models_\lambda \varphi \implies \langle p, \omega \rangle \in \lambda(e) \implies M_e$ has an accepting run from the corresponding initial state $p_e \implies p_e \xrightarrow{\delta} q_f$ where $q_f \in F_e$. We show that $BP'_\varphi$ has an accepting run from $\langle p_e, \omega \rangle$ by induction on the length $m$ of the stack $\omega$, where $m \geq 0$. Note that the bottom stack symbol $\sharp$ is not counted in the length of $\omega$.

  * $m = 1$ (note that $\sharp$ will never be popped), then $\omega = \sharp$. Therefore $p_e \xrightarrow{\delta} q_f$. We obtain that $\langle p_e, \sharp \rangle \Rightarrow_{BP'_\varphi} \langle q_f, \sharp \rangle \Rightarrow_{BP'_\varphi} \langle q_f, \sharp \rangle$. As $q_f$ is an accepting control location (by $F_1$), we get that $BP'_\varphi$ has an accepting run from $\langle p_e, \sharp \rangle$. In other words, $BP'_\varphi$ has an accepting run from $\langle p_e, \omega \rangle$. The property holds for this case.

  * Step. $m >= 2$, then, there exists $\gamma \in \Gamma, u \in \Gamma^*, q \in Q_e$ s.t. $\omega = \gamma u$ and
Chapter 4. Branching Temporal Logic of Calls and Returns for Pushdown Systems

\[ p_e \xrightarrow{\gamma_e} q \xrightarrow{u} q_f \text{ in } M_e \]

By applying the induction hypothesis on \( m \), we get that \( \mathcal{BP}_\varphi' \) has an accepting run from \( \langle q, u \rangle \). Also, we have \( \langle p_e, \gamma u \rangle \Rightarrow_{\mathcal{BP}_\varphi'} \langle q, u \rangle \) (by the transition rules in (\( \beta 3 \))). Therefore, \( \mathcal{BP}_\varphi' \) has an accepting run from \( \langle p_e, \omega \rangle \). The property holds for this case.

Since \( \mathcal{BP}_\varphi' \) has an accepting run from \( \langle p_e, \omega \rangle \) and the fact that \( \langle \langle p, e \rangle, \omega \rangle \Rightarrow_{\mathcal{BP}_\varphi'} \langle p_e, \omega \rangle \) (by transition rules in (\( \alpha 1 \))), we obtain that \( \mathcal{BP}_\varphi' \) has an accepting run from \( \langle \langle p, e \rangle, \omega \rangle \). In other words, \( \mathcal{BP}_\varphi' \) has an accepting run from \( \langle \langle p, \varphi \rangle, \omega \rangle \). The property holds for this case.

\( \varphi = \neg e (e \in AP) \). This case is similar to the case \( \varphi = e (e \in AP) \).

The property holds for this case.

• Induction Step: The proof is similar to the previous case of standard valuations. This is why \( \Delta'' \) contains \( \Delta \) and \( \Delta'_0 \) (which are the transitions of \( \mathcal{BP}_\varphi \) that don’t consider the atomic propositions).

\( \left( \iff \right) \) Assume that \( \mathcal{BP}_\varphi' \) has an accepting run from the configuration \( \langle \langle p, \varphi \rangle, \omega \rangle \), we need to prove that \( \langle p, \omega \rangle \models_\lambda \varphi \). In what follows, we prove this by induction on the structure of \( \varphi \).

Proof. • Base case:

\( \varphi = e (e \in AP) \).

Firstly, we will prove that for every \( q \in Q_e \), \( \mathcal{BP}_\varphi' \) has an accepting run from \( \langle q, \omega \rangle \) implies that \( q \xrightarrow{\omega \delta_e} q_f \) where \( q_f \in F_e \). We show this by induction on the length \( m \) of the stack \( \omega \).

* Basis. \( m = 1 \) (note that \( \sharp \) will never be popped), then, \( \omega = \sharp \).

Since \( \mathcal{BP}_\varphi' \) has an accepting run from \( \langle q, \sharp \rangle \), we get that \( \langle q, \sharp \rangle \) must have immediate successors. From all transition rules of \( \mathcal{BP}_\varphi' \), the unique way to have immediate successors of \( \langle q, \sharp \rangle \) is from the rules in (\( \beta 4 \)), which means that \( \langle q, \sharp \rangle \Rightarrow_{\mathcal{BP}_\varphi'} \langle q_f \rangle \). By the condition in (\( \beta 4 \)), we must have \( q \in F_e \implies q \xrightarrow{\delta_e} q_f \). The property holds in this case.

* Step. \( m \geq 2 \), then, there exists \( \gamma \in \Gamma, u \in \Gamma^* \), \( q \in Q_e \) s.t. \( \omega = \gamma u \). We need to show that \( q \xrightarrow{\gamma u \delta_e} q_f \) where \( q_f \in F_e \).
4.5. Conclusion

Theorem 16. Given a PDS $\mathcal{P} = (P, \Gamma, \Delta)$, a set of atomic propositions $AP$, a regular labelling function $\lambda : AP \rightarrow 2^{P \times \Gamma^*}$ and a BCARET formula $\varphi$, for every configuration $(p, \omega) \in P \times \Gamma^*$, whether or not $(p, \omega)$ satisfies $\varphi$ can be solved in time $O((|P||\varphi| + k)^2, |\Gamma||\varphi|(|P||\Gamma||\Delta||\varphi| + d) + 2^{|P||\varphi|+k} + 2^{|P||\varphi|+k} + |\omega|))$

4.5 Conclusion

In this chapter, we introduce the Branching temporal logic of CAlls and RETurns BCARET and show how it can be used to describe malicious behaviors
that CARET and other specification formalisms cannot. We present an algorithm for "standard" BCARET model checking for PDSs where whether a configuration of a PDS satisfies an atomic proposition or not depends only on the control location of that configuration. Moreover, we consider BCARET model-checking for PDSs with regular valuations where the set of configurations on which an atomic proposition holds is a regular language. Our approach is based on reducing these problems to the emptiness problem of Alternating Büchi Pushdown Systems.
In this chapter, we propose to use the Branching Temporal Logic of Calls and Returns BCARET for malicious behavior specification. Since BCARET formulas for malicious behaviors are huge, we propose to extend BCARET with variables, quantifiers and predicates over the stack. Our new logic is called Stack Branching temporal Predicate logic of Calls and RETurns (SBPCARET). We reduce the malware detection problem to the model checking problem of Pushdown Systems (PDSs) against SBPCARET formulas, and we propose an efficient algorithm to model check SBPCARET formulas for PDSs.

**Outline.** In Section 5.1, we introduce the new logic SBPCARET and shows how SBPCARET can be used to succinctly specify branching-time malicious behaviors. Section 5.2 shows how to model-check SBPCARET formulas against PDSs. Finally, we conclude in Section 5.3.

### 5.1 Stack Branching temporal Predicate logic of Calls and RETurns

In this section, we define the Stack Branching temporal Predicate logic of Calls and RETurns (SBPCARET) as an extension of BCARET (as presented in Chapter 4) with variables and regular predicates over the stack contents. The predicates contain variables that can be quantified existentially or universally. Regular predicates are expressed by regular variable expressions and are used to describe the stack content of PDSs.

#### 5.1.1 Environments, Predicates and Regular Variable Expressions

As in Chapter 3, let $\mathcal{X} = \{x_1, ..., x_n\}$ be a finite set of variables over a finite domain $\mathcal{D}$. Let $B : \mathcal{X} \cup \mathcal{D} \rightarrow \mathcal{D}$ be an environment that associates each variable $x \in \mathcal{X}$ with a value $d \in \mathcal{D}$ s.t $B(d) = d$ for every $d \in \mathcal{D}$. Let $B[x \leftarrow d]$ be an environment obtained from $B$ such that $B[x \leftarrow d](x) = d$ and
A SBPCARET formula is a BCARET formula where predicates and RVEs are fixed. Let $\mathcal{B}$ be the set of all environments.

Let $AP = \{a, b, c, \ldots\}$ be a finite set of atomic propositions. Let $AP_D$ be a finite set of atomic predicates of the form $b(\alpha_1, \ldots, \alpha_m)$ such that $b \in AP$ and $\alpha_i \in D$ for every $1 \leq i \leq m$. Let $AP_X$ be a finite set of atomic predicates $b(\alpha_1, \ldots, \alpha_n)$ such that $b \in AP$ and $\alpha_i \in \mathcal{X} \cup D$ for every $1 \leq i \leq n$.

Let $\mathcal{P} = (P, \Gamma, \Delta)$ be a Labelled PDS. Regular Variable Expressions (RVEs) over $\mathcal{X} \cup \Gamma$ are as defined in Section 3.1.1.

5.1.2 The Stack Branching temporal Predicate logic of CAlls and RETurns - SBPCARET

A SBPCARET formula is a BCARET formula where predicates and RVEs are used as atomic propositions and where quantifiers are applied to variables. For technical reasons, we assume w.l.o.g. that formulas are written in positive normal form, where negations are applied only to atomic predicates, and we use the release operator $R$ as the dual of the until operator $U$. From now on, we fix a finite set of variables $\mathcal{X}$, a finite set of atomic propositions $AP$, a finite domain $D$, and a finite set of RVEs $\mathcal{V}$. A SBPCARET formula is defined as follows, where $v \in \{g, a\}$, $x \in \mathcal{X}$, $e \in \mathcal{V}$, $b(\alpha_1, \ldots, \alpha_n) \in AP_X$:

$$\varphi := true \mid false \mid b(\alpha_1, \ldots, \alpha_n) \mid \neg b(\alpha_1, \ldots, \alpha_n) \mid e \mid \neg e \mid \varphi \lor \varphi \mid \varphi \land \varphi \mid \forall x \varphi \mid \exists x \varphi \mid EX^v \varphi \mid AX^v \varphi \mid E[\varphi U^v \varphi] \mid A[\varphi U^v \varphi] \mid E[\varphi R^v \varphi] \mid A[\varphi R^v \varphi]$$

Let $\lambda : P \rightarrow 2^{AP_D}$ be a labelling function which associates each control location to a set of atomic predicates. Let $\varphi$ be a SBPCARET formula over $AP$. Let $\langle p, \omega \rangle$ be a configuration of $\mathcal{P}$. Then we say that $\mathcal{P}$ satisfies $\varphi$ at $\langle p, \omega \rangle$ (denoted by $\langle p, \omega \rangle \models_{\lambda} \varphi$) iff there exists an environment $B \in \mathcal{B}$ such that $\langle p, \omega \rangle$ satisfies $\varphi$ under $B$ (denoted by $\langle p, \omega \rangle \models_{B} \varphi$). The satisfiability relation of a SBPCARET formula $\varphi$ at a configuration $\langle p_0, \omega_0 \rangle$ under the environment $B$ w.r.t. the labelling function $\lambda$, denoted by $\langle p_0, \omega_0 \rangle \models_{\lambda} B \varphi$, is defined inductively as follows:

- $\langle p_0, \omega_0 \rangle \models_{\lambda} B true$ for every $\langle p_0, \omega_0 \rangle$
- $\langle p_0, \omega_0 \rangle \not\models_{\lambda} B false$ for every $\langle p_0, \omega_0 \rangle$
- $\langle p_0, \omega_0 \rangle \models_{\lambda} B b(\alpha_1, \ldots, \alpha_n)$, iff $b(B(\alpha_1), \ldots, B(\alpha_n)) \in \lambda(p_0)$
- $\langle p_0, \omega_0 \rangle \not\models_{\lambda} B \neg b(\alpha_1, \ldots, \alpha_n)$, iff $b(B(\alpha_1), \ldots, B(\alpha_n)) \notin \lambda(p_0)$
- $\langle p_0, \omega_0 \rangle \models_{\lambda} B e$ iff $\langle (p_0, \omega_0), B \rangle \in L(e)$
5.1. Stack Branching temporal Predicate logic of CAlls and RET urns

- \( \langle p_0, \omega_0 \rangle \models^B_\lambda \neg \phi \iff (\langle p_0, \omega_0 \rangle, B) \notin L(e) \)
- \( \langle p_0, \omega_0 \rangle \models^B_\lambda \phi_1 \lor \phi_2 \iff (\langle p_0, \omega_0 \rangle \models^B_\lambda \phi_1 \text{ or } \langle p_0, \omega_0 \rangle \models^B_\lambda \phi_2) \)
- \( \langle p_0, \omega_0 \rangle \models^B_\lambda \phi_1 \land \phi_2 \iff (\langle p_0, \omega_0 \rangle \models^B_\lambda \phi_1 \text{ and } \langle p_0, \omega_0 \rangle \models^B_\lambda \phi_2) \)
- \( \langle p_0, \omega_0 \rangle \models^B_\lambda \forall x \phi \iff \text{for every } d \in D, \langle p_0, \omega_0 \rangle \models^B_{[x=d]} \phi \)
- \( \langle p_0, \omega_0 \rangle \models^B_\lambda \exists x \phi \iff \text{there exists } d \in D, \langle p_0, \omega_0 \rangle \models^B_{[x=d]} \phi \)
- \( \langle p_0, \omega_0 \rangle \models^B_\lambda \mathcal{E} \mathcal{X}^a \phi \iff \text{there exists a global-successor } \langle p', \omega' \rangle \text{ of } \langle p_0, \omega_0 \rangle \)
  such that \( \langle p', \omega' \rangle \models^B_\lambda \phi \)
- \( \langle p_0, \omega_0 \rangle \models^B_\lambda \mathcal{A} \mathcal{X}^a \phi \iff \langle p', \omega' \rangle \models^B_\lambda \phi \text{ for every global-successor } \langle p', \omega' \rangle \text{ of } \langle p_0, \omega_0 \rangle \)
- \( \langle p_0, \omega_0 \rangle \models^B_\lambda \mathcal{E}[\phi_1 U^a \phi_2] \iff \text{there exists a global-path } \pi = \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \langle p_2, \omega_2 \rangle \ldots \text{ of } \mathcal{P} \text{ starting from } \langle p_0, \omega_0 \rangle \text{ s.t. } \exists i \geq 0, \langle p_i, \omega_i \rangle \models^B_\lambda \phi_2 \text{ and for every } 0 \leq j < i, \langle p_j, \omega_j \rangle \models^B_\lambda \phi_1 \)
- \( \langle p_0, \omega_0 \rangle \models^B_\lambda \mathcal{A}[\phi_1 U^a \phi_2] \iff \text{for every global-path } \pi = \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \langle p_2, \omega_2 \rangle \ldots \text{ of } \mathcal{P} \text{ starting from } \langle p_0, \omega_0 \rangle, \exists i \geq 0, \langle p_i, \omega_i \rangle \models^B_\lambda \phi_2 \text{ and for every } 0 \leq j < i, \langle p_j, \omega_j \rangle \models^B_\lambda \phi_1 \)
- \( \langle p_0, \omega_0 \rangle \models^B_\lambda \mathcal{E}[\phi_1 R^a \phi_2] \iff \text{there exists a global-path } \pi = \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \langle p_2, \omega_2 \rangle \ldots \text{ of } \mathcal{P} \text{ starting from } \langle p_0, \omega_0 \rangle \text{ s.t. for every } i \geq 0, \langle p_i, \omega_i \rangle \models^B_\lambda \phi_2 \text{ then there exists } 0 \leq j < i \text{ s.t. } \langle p_j, \omega_j \rangle \models^B_\lambda \phi_1 \)
- \( \langle p_0, \omega_0 \rangle \models^B_\lambda \mathcal{A}[\phi_1 R^a \phi_2] \iff \text{for every global-path } \pi = \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \langle p_2, \omega_2 \rangle \ldots \text{ of } \mathcal{P} \text{ starting from } \langle p_0, \omega_0 \rangle, \text{ for every } i \geq 0, \langle p_i, \omega_i \rangle \models^B_\lambda \phi_2 \text{ then there exists } 0 \leq j < i \text{ s.t. } \langle p_j, \omega_j \rangle \models^B_\lambda \phi_1 \)
- \( \langle p_0, \omega_0 \rangle \models^B_\lambda \mathcal{E} \mathcal{X}^a \phi \iff \text{there exists an abstract-successor } \langle p', \omega' \rangle \text{ of } \langle p_0, \omega_0 \rangle \)
  such that \( \langle p', \omega' \rangle \models^B_\lambda \phi \)
- \( \langle p_0, \omega_0 \rangle \models^B_\lambda \mathcal{A} \mathcal{X}^a \phi \iff \langle p', \omega' \rangle \models^B_\lambda \phi \text{ for every abstract-successor } \langle p', \omega' \rangle \text{ of } \langle p_0, \omega_0 \rangle \)
- \( \langle p_0, \omega_0 \rangle \models^B_\lambda \mathcal{E}[\phi_1 U^a \phi_2] \iff \text{there exists an abstract-path } \pi = \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \langle p_2, \omega_2 \rangle \ldots \text{ of } \mathcal{P} \text{ starting from } \langle p_0, \omega_0 \rangle \text{ s.t. } \exists i \geq 0, \langle p_i, \omega_i \rangle \models^B_\lambda \phi_2 \text{ and for every } 0 \leq j < i, \langle p_j, \omega_j \rangle \models^B_\lambda \phi_1 \)
- \( \langle p_0, \omega_0 \rangle \models^B_\lambda \mathcal{A}[\phi_1 U^a \phi_2] \iff \text{for every abstract-path } \pi = \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \langle p_2, \omega_2 \rangle \ldots \text{ of } \mathcal{P}, \exists i \geq 0, \langle p_i, \omega_i \rangle \models^B_\lambda \phi_2 \text{ and for every } 0 \leq j < i, \langle p_j, \omega_j \rangle \models^B_\lambda \phi_1 \)
• $\langle p_0, \omega_0 \rangle \models^E \lambda [\phi_1 R^a \phi_2]$ iff there exists an abstract-path $\pi = \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \langle p_2, \omega_2 \rangle \ldots$ of $P$ starting from $\langle p_0, \omega_0 \rangle$, for every $i \geq 0$, if $\langle p_i, \omega_i \rangle \not\models^B \phi_2$ then there exists $0 \leq j < i$ s.t. $\langle p_j, \omega_j \rangle \models^B \phi_1$

• $\langle p_0, \omega_0 \rangle \models^B \lambda [\phi_1 R^a \phi_2]$ iff for every abstract-path $\pi = \langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \langle p_2, \omega_2 \rangle \ldots$ of $P$ starting from $\langle p_0, \omega_0 \rangle$, for every $i \geq 0$, if $\langle p_i, \omega_i \rangle \not\models^B \phi_2$ then there exists $0 \leq j < i$ s.t. $\langle p_j, \omega_j \rangle \models^B \phi_1$

Other SBPCARET operators can be expressed by the above operators: $EF^g \phi = E[true U^g \phi]$, $EF^a \phi = E[true U^a \phi]$, $AF^g \phi = A[true U^g \phi]$, $AF^a \phi = A[true U^a \phi]$, ...

### Closure
Given a SBPCARET formula $\phi$, the closure $Cl(\phi)$ is the set of all subformulae of $\phi$, including $\phi$. Let $AP^+(\phi) = \{b(\alpha_1, ..., \alpha_n) \in AP_X \mid b(\alpha_1, ..., \alpha_n) \in Cl(\phi)\}$; $AP^-(\phi) = \{b(\alpha_1, ..., \alpha_n) \in AP_X \mid \neg b(\alpha_1, ..., \alpha_n) \in Cl(\phi)\}$, $Reg^+(\phi) = \{e \in V \mid e \in Cl(\phi)\}$, $Reg^-(\phi) = \{e \in V \mid \neg e \in Cl(\phi)\}$

### 5.1.3 Modelling Malicious Behaviours Using SBPCARET

In this section, we show how SBPCARET can be used to succinctly specify the malicious behavior presented in Section 4.2.

**Spyware Behavior.** The BCARET formula $\varphi_{sb}$ described in Section 4.2 can be represented by the SBPCARET formula:

$$\varphi'_{sb} = \exists x EF^g \left( call(FindFirstFileA) \land EX^a (eax = x) \land AF^a \left( call(GetLastError) \lor call(FindFirstFileA) \lor \left( call(FindNextFileA) \land x T^a \right) \right) \right)$$

This formula states that there is a call to the API $FindFirstFileA$ with the return value $x$ (the search handle), then, in all runs starting from that call, there will be either a call to the API function $GetLastError$ or a call to the function $FindFirstFileA$ or a call to the function $FindNextFileA$ in which $x$ is used as a parameter. Note that in this case, $x$ is the memory address containing the values of search handles. It can be seen that $\varphi'_{sb}$ is much more compact than $\varphi_{sb}$. 
5.2 SBPCARET Model-Checking for Pushdown Systems

In this section, we show how to do SBPCARET model-checking for PDSs. Let then $P$ be a PDS, $\varphi$ be a SBPCARET formula, and $\mathcal{V}$ be the set of RVEs occurring in $\varphi$. We follow the idea of [ST12b] and use Variable Automata to represent RVEs.

5.2.1 Variable Automata

Given a PDS $P = (P, \Gamma, \Delta)$ s.t. $\Gamma \subseteq \mathcal{D}$, a Variable Automaton (VA) [ST12b] is a tuple $(Q, \Gamma, \delta, s, F)$, where $Q$ is a finite set of states, $\Gamma$ is the input alphabet, $s \in Q$ is an initial state; $F \subseteq Q$ is a finite set of accepting states; and $\delta$ is a finite set of transition rules of the form $p \xrightarrow{\alpha} \{q_1, \ldots, q_n\}$ where $\alpha$ can be $x$, $\neg x$, or $\gamma$, for any $x \in \mathcal{X}$ and $\gamma \in \Gamma$.

Let $B \in \mathcal{B}$. A run of VA on a word $\gamma_1, \ldots, \gamma_m$ under $B$ is a tree of height $m$ whose root is labelled by the initial state $s$, and each node at depth $k$ labelled by a state $q$ has $h$ children labelled by $p_1, \ldots, p_h$ respectively, such that:

- either $q \xrightarrow{\gamma_k} \{p_1, \ldots, p_h\} \in \delta$ and $\gamma_k \in \Gamma$;
- or $q \xrightarrow{x} \{p_1, \ldots, p_h\} \in \delta$, $x \in \mathcal{X}$ and $B(x) = \gamma_k$;
- or $q \xrightarrow{\neg x} \{p_1, \ldots, p_h\} \in \delta$, $x \in \mathcal{X}$ and $B(x) \neq \gamma_k$.

A branch of the tree is accepting iff the leaf of the branch is an accepting state. A run is accepting iff all its branches are accepting. A word $\omega \in \Gamma^*$ is accepted by a VA under an environment $B \in \mathcal{B}$ iff the VA has an accepting run on the word $\omega$ under the environment $B$.

The language of a VA $M$, denoted by $L(M)$, is a subset of $(P \times \Gamma^*) \times \mathcal{B}$. $((p, \omega), B) \in L(M)$ iff $M$ accepts the word $\omega$ under the environment $B$.

Theorem 17. [ST12b] For every regular expression $e \in \mathcal{V}$, we can compute in polynomial time a Variable Automaton $M$ s.t. $L(M) = L(e)$.

Theorem 18. [ST12b] VAs are closed under boolean operations.

5.2.2 Symbolic Alternating Büchi Pushdown Systems (SABPDSs).

Definition 17. A Symbolic Alternating Büchi Pushdown System (SABPDS) is a tuple $BP = (P, \Gamma, \Delta, F)$, where $P$ is a set of control locations, $\Gamma \subseteq \mathcal{D}$ is stack
alphabet, $F \subseteq P \times 2^E$ is a set of accepting control locations and $\Delta$ is a finite set of transitions of the form $(p, \gamma) \mapsto \{\langle p_1, \omega_1 \rangle, ..., \langle p_n, \omega_n \rangle\}$ where $p \in P$, $\gamma \in \Gamma$, for every $1 \leq i \leq n$: $p_i \in P$, $\omega_i \in \Gamma^*$; and $\mathbb{R} : (B)^n \to 2^E$ is a function that maps a tuple of environments $(B_1, ..., B_n)$ to a set of environments.

A configuration of a SABPDS $BP$ is a tuple $(\langle p, B \rangle, \omega)$, where $p \in P$ is the current control location, $B \in \mathcal{B}$ is an environment and $\omega \in \Gamma^*$ is the current stack content. Let $(p, \gamma) \mapsto \{\langle p_1, \omega_1 \rangle, ..., \langle p_n, \omega_n \rangle\}$ be a rule of $\Delta$, then, for every $\omega \in \Gamma^*$, $B, B_1, ..., B_n \in B$, if $B \in \mathbb{R}(B_1, ..., B_n)$, then the configuration $(\langle p, B \rangle, \gamma\omega)$ (resp. $\{\langle p_1, B_1 \rangle, \omega_1\omega \}, ..., \langle p_n, B_n \rangle, \omega_n\omega\})$ is an immediate predecessor (resp. successor) of $(\langle p_1, B_1 \rangle, \omega_1\omega \}, ..., \langle p_n, B_n \rangle, \omega_n\omega\})$.

A run $\rho$ of a SABPDS $BP$ starting from an initial configuration $(\langle p_0, B_0 \rangle, \omega_0)$ is a tree whose root is labelled by $(\langle p_0, B_0 \rangle, \omega_0)$, and whose other nodes are labelled by elements in $P \times \mathcal{B} \times \Gamma^*$. If a node of $\rho$ is labelled by a configuration $(\langle p, B \rangle, \omega)$ and has $n$ children labelled by $(\langle p_1, B_1 \rangle, \omega_1), ..., (\langle p_n, B_n \rangle, \omega_n)$ respectively, then, $(\langle p, B \rangle, \omega)$ must be a predecessor of $(\langle p_1, B_1 \rangle, \omega_1), ..., (\langle p_n, B_n \rangle, \omega_n)$ in $BP$. A path of a run $\rho$ is an infinite sequence of configurations $c_0c_1c_2...$ s.t. $c_0$ is the root of $\rho$ and $c_{i+1}$ is one of the children of $c_i$ for every $i \geq 0$. A path is accepting iff it visits infinitely often configurations with control locations in $F$. A run $\rho$ is accepting iff every path of $\rho$ is accepting. The language of $BP$, $L(BP)$, is the set of configurations $c$ s.t. $BP$ has an accepting run starting from $c$.

$BP$ defines the reachability relation $\Rightarrow_{BP}$: $2^{(P \times \mathcal{B}) \times \Gamma^*} \to 2^{(P \times \mathcal{B}) \times \Gamma^*}$ as follows:
1. $c \Rightarrow_{BP} \{c\}$ for every $c \in P \times \mathcal{B} \times \Gamma^*$,
2. $c \Rightarrow_{BP} C$ if $C$ is an immediate successor of $c$,
3. if $c \Rightarrow_{BP} \{c_1, c_2, ..., c_n\}$ and $c_i \Rightarrow_{BP} C_i$ for every $1 \leq i \leq n$, then $c \Rightarrow_{BP} \bigcup_{i=1}^n C_i$.

Given $c_0 \Rightarrow_{BP} C'$, then, $BP$ has an accepting run from $c_0$ if $BP$ has an accepting run from $c'$ for every $c' \in C'$.

**Theorem 19.** [ST12b] The membership problem of SABPDS can be solved effectively.

**Functions of $\mathbb{R}$.** In what follows, we define several functions of $\mathbb{R}$ which will be used in the next sections. These functions were first defined in [ST12b].

1. $id(B) = \{B\}$. This is the identity function.
2. $\text{equal}(B_1, ..., B_n) = \begin{cases} \{B_i\} & \text{if } B_i = B_j \text{ for every } 1 \leq i, j \leq n, \\ \emptyset & \text{otherwise} \end{cases}$
This function checks whether all the environments are equal and returns \{B_1\} (which is also equal to B_i for every i). Otherwise, it returns the emptyset.

\[ \text{meet}_x^{c_1,\ldots,c_n}(B_1,\ldots,B_n) = \begin{cases} \text{Abs}_x(B_1) & \text{if } B_i(x) = c_i \text{ for } 1 \leq i \leq n, \\
& \text{and } B_i(y) = B_j(y) \text{ for } y \neq x, 1 \leq i, j \leq n; \\
\emptyset & \text{otherwise} \end{cases} \]

This function checks whether (1) \(B_i(x) = c_i\) for every \(1 \leq i \leq n\) (2) for every \(y \neq x\); every \(1 \leq i, j \leq n\) \(B_i(y) = B_j(y)\). If the conditions are satisfied, it returns \(\text{Abs}_x(B_1)^1\), otherwise it returns the emptyset.

\[ \text{join}_x^c(B_1,\ldots,B_n) = \begin{cases} B_1 & \text{if } B_i(x) = c \text{ for } 1 \leq i \leq n \\
& \text{and } B_i = B_j \text{ for } 1 \leq i, j \leq n; \\
\emptyset & \text{otherwise} \end{cases} \]

This function checks whether \(B_i(x) = c\) for every \(i\). If this condition is satisfied, \(\text{equal}(B_1,\ldots,B_n)\) is returned, otherwise, the emptyset is returned.

\[ \text{join}_c^{-x}(B_1,\ldots,B_n) = \begin{cases} B_1 & \text{if } B_i(x) \neq c \text{ for } 1 \leq i \leq n \\
& \text{and } B_i = B_j \text{ for } 1 \leq i, j \leq n; \\
\emptyset & \text{otherwise} \end{cases} \]

This function checks whether \(B_i(x) \neq c\) for every \(i\). If this condition is satisfied, \(\text{equal}(B_1,\ldots,B_n)\) is returned, otherwise, the emptyset is returned.

\(^1\text{Abs}_x(B_1)\) is as defined in Section 5.1.1
5.2.3 From SBPCARET model checking of PDSs to the membership problem in SABPDSs

Let \( \mathcal{P} = (P, \Gamma, \Delta) \) be a PDS. We suppose w.l.o.g. that \( \mathcal{P} \) has a bottom stack symbol \( \sharp \) that is never popped from the stack. Let AP be a set of atomic propositions. Let \( \varphi \) be a SBPCARET formula over \( AP, \lambda : P \rightarrow 2^{AP} \) be a labelling function. Given a configuration \( \langle p_0, \omega_0 \rangle \), we propose in this section an algorithm to check whether \( \langle p_0, \omega_0 \rangle \Vdash_\lambda \varphi \), i.e., whether there exists an environment \( B \) s.t. \( \langle p_0, \omega_0 \rangle \Vdash^B_\lambda \varphi \). Intuitively, we compute an SABPDS \( BP_\varphi \) s.t. \( \langle p, \omega \rangle \Vdash^B_\lambda \varphi \) iff \( \langle \langle p, \varphi \rangle, B \rangle, \omega \rangle \in \mathcal{L}(BP_\varphi) \) for every \( p \in P, \omega \in \Gamma^*, B \in \mathcal{B} \).

Then, to check if \( \langle p_0, \omega_0 \rangle \Vdash_\lambda \varphi \), we will check whether there exists a \( B \in \mathcal{B} \) s.t. \( \langle \langle p_0, \varphi \rangle, B \rangle, \omega_0 \rangle \in \mathcal{L}(BP_\varphi) \).

Let \( Reg^+(\varphi) = \{e_1, ..., e_k\} \) and \( Reg^- (\varphi) = \{e_{k+1}, ..., e_m\} \). Using Theorems 17 and 18; for every \( 1 \leq i \leq k \), we can compute a VA \( M_{e_i} = (Q_{e_i}, \Gamma, \delta_{e_i}, s_{e_i}, F_{e_i}) \) s.t. \( L(M_{e_i}) = L(e_i) \). In addition, for every \( k + 1 \leq j \leq m \), we can compute a VA \( M_{e_j} = (Q_{e_j}, \Gamma, \delta_{e_j}, s_{e_j}, F_{e_j}) \) s.t. \( L(M_{e_j}) = (P \times \Gamma^*) \times \mathcal{B} \setminus L(e_j) \). Let \( \mathcal{M} \) be the union of all these automata, \( \mathcal{S} \) and \( \mathcal{F} \) be respectively the union of all states and final states of these automata.

Let \( BP_\varphi = (P', \Gamma', \Delta', \mathcal{F}) \) be the SABPDS defined as follows:

- \( P' = P \cup (P \times Cl(\varphi)) \cup \mathcal{S} \cup \{p_\perp\} \)
- \( \Gamma' = \Gamma \cup (\Gamma \times Cl(\varphi)) \cup \{\gamma_\perp\} \)
- \( \mathcal{F} = F_1 \cup F_2 \cup F_3 \cup F_4 \) where
  - \( F_1 = \{\langle \langle p, b(\alpha_1, ..., \alpha_n) \rangle, B \rangle \mid b(\alpha_1, ..., \alpha_n) \in AP^+(\varphi), \text{ and } B \in \mathcal{B} \text{ s.t. } b(B(\alpha_1), ..., B(\alpha_n)) \in \lambda(p)\} \)
  - \( F_2 = \{\langle \langle p, \neg b(\alpha_1, ..., \alpha_n) \rangle, B \rangle \mid b(\alpha_1, ..., \alpha_n) \in AP^-(\varphi), \text{ and } B \in \mathcal{B} \text{ s.t. } b(B(\alpha_1), ..., B(\alpha_n)) \notin \lambda(p)\} \)
  - \( F_3 = P \times Cl_R(\varphi) \times \mathcal{B} \) where \( Cl_R(\varphi) \) is the set of formulas of \( Cl(\varphi) \) in the form \( E[\varphi_1 R^n \varphi_2] \) or \( A[\varphi_1 R^n \varphi_2] \) \((v \in \{g, a\})\)
  - \( F_4 = \mathcal{F} \times \mathcal{B} \)

The transition relation \( \Delta' \) is the smallest set of transition rules defined as follows: For every \( p \in P, \varphi \in Cl(\varphi), \gamma \in \Gamma \) and \( t \in \{call, ret, int\} \):

\( (h1) \) If \( \varphi = b(\alpha_1, ..., \alpha_n) \), then, \( \langle \langle p, \varphi \rangle, \gamma \rangle \xrightarrow{id} \langle \langle p, \varphi \rangle, \gamma \rangle \in \Delta' \)

\( (h2) \) If \( \varphi = \neg b(\alpha_1, ..., \alpha_n) \), then, \( \langle \langle p, \varphi \rangle, \gamma \rangle \xrightarrow{id} \langle \langle p, \varphi \rangle, \gamma \rangle \in \Delta' \)

\( (h3) \) If \( \varphi = \phi_1 \land \phi_2 \), then, \( \langle \langle p, \varphi \rangle, \gamma \rangle \xrightarrow{eq\text{al}} \langle \langle p, \phi_1 \rangle, \gamma \rangle \cup \langle \langle p, \phi_2 \rangle, \gamma \rangle \in \Delta' \)
(h4) If $\phi = \phi_1 \lor \phi_2$, then, $\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{id} \langle \langle p, \phi_1 \rangle, \gamma \rangle \in \Delta'$ and $\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{id} \langle \langle p, \phi_2 \rangle, \gamma \rangle \in \Delta'$

(h5) If $\phi = \exists x \phi_1$, then, $\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{\text{meet}_{\exists}^c} \langle \langle p, \phi_1 \rangle, \gamma \rangle \in \Delta'$ for every $c \in \mathcal{D}$

(h6) If $\phi = \forall x \phi_1$, then, $\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{\text{meet}_{\forall}^c} \left[ \langle \langle p, \phi_1 \rangle, \gamma \rangle, \ldots, \langle \langle p, \phi_1 \rangle, \gamma \rangle \right] \in \Delta'$ where $\langle \langle p, \phi_1 \rangle, \gamma \rangle$ is repeated $m$ times in the right-hand side, where $m$ is the number of elements in $\mathcal{D}$

(h7) If $\phi = EX^g \phi_1$, then,
\[
\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{id} \langle \langle q, \phi_1 \rangle, \omega \rangle \in \Delta' \text{ for every } \langle p, \gamma \rangle \xrightarrow{t} \langle q, \omega \rangle \in \Delta
\]

(h8) If $\phi = AX^g \phi_1$, then,
\[
\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{\text{equal}} \left[ \langle \langle q_1, \phi_1 \rangle, \omega_1 \rangle, \ldots, \langle \langle q_n, \phi_1 \rangle, \omega_n \rangle \right] \in \Delta', \text{ where for every } 1 \leq i \leq n, \langle p, \gamma \rangle \xrightarrow{t} \langle q_i, \omega_i \rangle \in \Delta \text{ and these transitions are all the transitions of } \Delta \text{ that are in the form } \langle p, \gamma \rangle \xrightarrow{t} \langle q, \omega \rangle \text{ that have } \langle p, \gamma \rangle \text{ on the left hand side.}
\]

(h9) If $\phi = EX^g \phi_1$, then,
\[
\begin{align*}
(a) & \quad \langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{id} \langle q, \gamma \langle \gamma', \phi_1 \rangle \rangle \in \Delta' \text{ for every } \langle p, \gamma \rangle \xrightarrow{\text{call}} \langle q, \gamma' \gamma'' \rangle \in \Delta \\
(b) & \quad \langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{id} \langle \langle q, \phi_1 \rangle, \omega \rangle \in \Delta' \text{ for every } \langle p, \gamma \rangle \xrightarrow{\text{int}} \langle q, \omega \rangle \in \Delta \\
(c) & \quad \langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{id} \langle \langle p_{\perp}, \gamma_{\perp} \rangle \rangle \in \Delta' \text{ for every } \langle p, \gamma \rangle \xrightarrow{\text{rel}} \langle q', \varepsilon \rangle \in \Delta
\end{align*}
\]

(h10) If $\phi = AX^g \phi_1$, then,
\[
\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{\text{equal}} \left[ \langle \langle q_1, \gamma_1 \langle \gamma_1', \phi_1 \rangle \rangle, \ldots, \langle \langle q_n, \gamma_n \langle \gamma_n', \phi_1 \rangle \rangle \right] \in \Delta', \text{ where } \langle p_{\perp}, \gamma_{\perp} \rangle \text{ is repeated } k \text{ times in the right-hand side s.t.}:
\]
\[
\begin{align*}
(a) & \quad \text{for every } 1 \leq i \leq m, \langle p, \gamma \rangle \xrightarrow{\text{call}} \langle p_i, \gamma_i \gamma'' \rangle \in \Delta \text{ and these transitions are all the transitions of } \Delta \text{ that are in the form } \langle p, \gamma \rangle \xrightarrow{\text{call}} \langle q, \gamma' \gamma'' \rangle \text{ that have } \langle p, \gamma \rangle \text{ on the left hand side.} \\
(b) & \quad \text{for every } 1 \leq i \leq n, \langle p, \gamma \rangle \xrightarrow{\text{int}} \langle q_i, \omega_i \rangle \in \Delta \text{ and these transitions are all the transitions of } \Delta \text{ that are in the form } \langle p, \gamma \rangle \xrightarrow{\text{int}} \langle q, \omega \rangle \text{ that have } \langle p, \gamma \rangle \text{ on the left hand side.} \\
(c) & \quad \text{for every } 1 \leq i \leq k, \langle p, \gamma \rangle \xrightarrow{\text{rel}} \langle q_i', \varepsilon \rangle \in \Delta \text{ and these transitions are all the transitions of } \Delta \text{ that are in the form } \langle p, \gamma \rangle \xrightarrow{\text{rel}} \langle q', \varepsilon \rangle \text{ that have } \langle p, \gamma \rangle \text{ on the left hand side.}
\end{align*}
\]
Chapter 5. BCARET Model Checking for Malware Detection

(h11) If $\phi = E[\phi_1 U^g \phi_2]$, then,

(a) $\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{id} \langle \langle p, \phi_2 \rangle, \gamma \rangle \in \Delta'$

(b) $\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{\text{equal}} [\langle \langle p, \phi_1 \rangle, \gamma \rangle, \langle q, \phi, \omega \rangle] \in \Delta'$ for every $\langle p, \gamma \rangle \xrightarrow{t} \langle q, \omega \rangle \in \Delta$

(h12) If $\phi = E[\phi_1 U^n \phi_2]$, then,

(a) $\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{id} \langle \langle p, \phi_2 \rangle, \gamma \rangle \in \Delta'$

(b) $\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{\text{equal}} [\langle \langle p, \phi_1 \rangle, \gamma \rangle, \langle q, \gamma' \gamma'' \phi \rangle] \in \Delta'$ for every $\langle p, \gamma \rangle \xrightarrow{\text{call}} \langle q, \gamma' \gamma'' \phi \rangle \in \Delta$

(c) $\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{\text{equal}} [\langle \langle p, \phi_1 \rangle, \gamma \rangle, \langle q, \phi, \omega \rangle] \in \Delta'$ for every $\langle p, \gamma \rangle \xrightarrow{\text{int}} \langle q, \omega \rangle \in \Delta$

(d) $\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{id} \langle p_\bot, \gamma_\bot \rangle \in \Delta'$ for every $\langle p, \gamma \rangle \xrightarrow{\text{ret}} \langle q', \varepsilon \rangle \in \Delta$

(h13) If $\phi = A[\phi_1 U^g \phi_2]$, then,

(a) $\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{id} \langle \langle p, \phi_2 \rangle, \gamma \rangle \in \Delta'$

(b) $\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{\text{equal}} [\langle \langle p, \phi_1 \rangle, \gamma \rangle; \langle q_1, \phi, \omega_1 \rangle, ..., \langle q_n, \phi, \omega_n \rangle] \in \Delta'$

where for every $1 \leq i \leq n, \langle p, \gamma \rangle \xrightarrow{t} \langle q_i, \omega_i \rangle \in \Delta$ and these transitions are all the transitions of $\Delta$ that are in the form $\langle p, \gamma \rangle \xrightarrow{t} \langle q, \omega \rangle$ that have $\langle p, \gamma \rangle$ on the left hand side.

(h14) If $\phi = A[\phi_1 U^n \phi_2]$, then,

(a) $\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{id} \langle \langle p, \phi_2 \rangle, \gamma \rangle \in \Delta'$

(b) $\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{\text{equal}} [\langle \langle p, \phi_1 \rangle, \gamma \rangle; \langle q_1, \gamma' \gamma'' \phi \rangle, ..., \langle q_m, \gamma' \gamma'' \phi \rangle; \langle q_1, \phi, \omega_1 \rangle, ..., \langle q_n, \phi, \omega_n \rangle; \langle p_\bot, \gamma_\bot \rangle, ..., \langle p_\bot, \gamma_\bot \rangle] \in \Delta'$, where $\langle p_\bot, \gamma_\bot \rangle$ is repeated $k$ times in the right-hand side s.t.:

- for every $1 \leq i \leq m, \langle p, \gamma \rangle \xrightarrow{\text{call}} \langle q_i, \gamma' \gamma'' \phi \rangle \in \Delta$ and these transitions are all the transitions of $\Delta$ that are in the form $\langle p, \gamma \rangle \xrightarrow{\text{call}} \langle q, \gamma' \gamma'' \phi \rangle$ that have $\langle p, \gamma \rangle$ on the left hand side.
- for every $1 \leq i \leq n, \langle p, \gamma \rangle \xrightarrow{\text{int}} \langle q_i, \omega_i \rangle \in \Delta$ and these transitions are all the transitions of $\Delta$ that are in the form $\langle p, \gamma \rangle \xrightarrow{\text{int}} \langle q, \omega \rangle$ that have $\langle p, \gamma \rangle$ on the left hand side.
• for every $1 \leq i \leq k$, $\langle p, \gamma \rangle \xrightarrow{\text{ret}} \langle q'_i, \varepsilon \rangle \in \Delta$ and these transitions are all the transitions of $\Delta$ that are in the form $\langle p, \gamma \rangle \xrightarrow{\text{ret}} \langle q', \varepsilon \rangle$ that have $\langle p, \gamma \rangle$ on the left hand side.

(h15) If $\phi = E[\phi_1 R^g \phi_2]$, then, we add to $\Delta'$ the rule:

(a) $\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{\text{equal}} \left[ \langle \langle p, \phi_2 \rangle, \gamma \rangle, \langle p, \phi_1 \rangle, \gamma \rangle \right] \in \Delta'$

(b) $\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{\text{equal}} \left[ \langle \langle p, \phi_2 \rangle, \gamma \rangle, \langle q, \phi \rangle, \omega \rangle \right] \in \Delta'$ for every $\langle p, \gamma \rangle \xrightarrow{t} \langle q, \omega \rangle \in \Delta$

(h16) If $\phi = A[\phi_1 R^g \phi_2]$, then, we add to $\Delta'$ the rule:

(a) $\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{\text{equal}} \left[ \langle \langle p, \phi_2 \rangle, \gamma \rangle, \langle p, \phi_1 \rangle, \gamma \rangle \right] \in \Delta'$

(b) $\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{\text{equal}} \left[ \langle \langle p, \phi_2 \rangle, \gamma \rangle, \langle q_1, \phi_1, \omega_1 \rangle, ..., \langle q_n, \phi_n, \omega_n \rangle \right] \in \Delta'$

where for every $1 \leq i \leq n$, $\langle p, \gamma \rangle \xrightarrow{t} \langle q_i, \omega_i \rangle \in \Delta$ and these transitions are all the transitions of $\Delta$ that are in the form $\langle p, \gamma \rangle \xrightarrow{t} \langle q, \omega \rangle$ that have $\langle p, \gamma \rangle$ on the left hand side.

(h17) If $\phi = E[\phi_1 R^g \phi_2]$, then,

(a) $\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{\text{equal}} \left[ \langle \langle p, \phi_2 \rangle, \gamma \rangle, \langle p, \phi_1 \rangle, \gamma \rangle \right] \in \Delta'$

(b) $\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{\text{equal}} \left[ \langle \langle p, \phi_2 \rangle, \gamma \rangle, \langle q, \gamma' \gamma'' \phi \rangle \right] \in \Delta'$ for every $\langle p, \gamma \rangle \xrightarrow{\text{call}} \langle q, \gamma' \gamma'' \rangle \in \Delta$

(c) $\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{\text{equal}} \left[ \langle \langle p, \phi_2 \rangle, \gamma \rangle, \langle q, \phi \rangle, \omega \rangle \right] \in \Delta'$ for every $\langle p, \gamma \rangle \xrightarrow{\text{int}} \langle q, \omega \rangle \in \Delta$

(d) $\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{\text{id}} \langle p_\perp, \gamma_\perp \rangle \in \Delta'$ for every $\langle p, \gamma \rangle \xrightarrow{\text{ret}} \langle q', \varepsilon \rangle \in \Delta$

(h18) If $\phi = A[\phi_1 R^g \phi_2]$, then,

(a) $\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{\text{equal}} \left[ \langle \langle p, \phi_2 \rangle, \gamma \rangle, \langle p, \phi_1 \rangle, \gamma \rangle \right] \in \Delta'$

(b) $\langle \langle p, \phi \rangle, \gamma \rangle \xrightarrow{\text{equal}} \left[ \langle \langle p, \phi_2 \rangle, \gamma \rangle, \langle p_1, \gamma_1 \gamma''_1 \phi \rangle, ..., \langle p_m, \gamma_m \gamma''_m \phi \rangle \rangle, \langle q_1, \phi_1, \omega_1 \rangle, ..., \langle q_n, \phi_n, \omega_n \rangle, \langle p_\perp, \gamma_\perp \rangle, ..., \langle p_\perp, \gamma_\perp \rangle \right] \in \Delta'$, where $\langle p_\perp, \gamma_\perp \rangle$ is repeated $k$ times in the right-hand side s.t.:

• for every $1 \leq i \leq m$, $\langle p, \gamma \rangle \xrightarrow{\text{call}} \langle p_i, \gamma' \gamma'' \rangle \in \Delta$ and these transitions are all the transitions of $\Delta$ that are in the form $\langle p, \gamma \rangle \xrightarrow{\text{call}} \langle q, \gamma' \gamma'' \rangle$ that have $\langle p, \gamma \rangle$ on the left hand side.
for every \( 1 \leq i \leq n, (p, \gamma) \xrightarrow{\text{id}} (q_i, \omega_i) \in \Delta \) and these transitions are all the transitions of \( \Delta \) that are in the form \( (p, \gamma) \xrightarrow{\text{id}} (q, \omega) \) that have \( (p, \gamma) \) on the left hand side.

- for every \( 1 \leq i \leq k, (p, \gamma) \xrightarrow{\text{ret}} (q', \varepsilon) \in \Delta \) and these transitions are all the transitions of \( \Delta \) that are in the form \( (p, \gamma) \xrightarrow{\text{ret}} (q', \varepsilon) \) that have \( (p, \gamma) \) on the left hand side.

\[(h19)\] for every \( (p, \gamma) \xrightarrow{\text{ret}} (q, \varepsilon) \in \Delta :\]

- \( (q, (\gamma'', \phi_1)) \xleftrightarrow{\text{id}} (\langle q, \phi_1 \rangle, \gamma'') \in \Delta' \) for every \( \gamma'' \in \Gamma, \phi_1 \in \text{Cl}(\varphi) \)

\[(h20)\] \( (p, \gamma, \gamma_{\perp}) \xleftrightarrow{\text{id}} (p, \gamma, \gamma_{\perp}) \in \Delta' \)

\[(h21)\] for every \( (p, \gamma) \xrightarrow{\text{L}} (q, \omega) \in \Delta : \ (p, \gamma) \xleftrightarrow{\text{id}} (q, \omega) \in \Delta' \)

\[(h22)\] If \( \phi = e, e \) is a regular expression, then, \( (\langle p, \phi \rangle, \gamma) \xrightarrow{\text{id}} (s_e, \gamma) \in \Delta' \)

\[(h23)\] If \( \phi = \neg e, e \) is a regular expression, then, \( (\langle p, \phi \rangle, \gamma) \xrightarrow{\text{id}} (s_{\neg e}, \gamma) \in \Delta' \)

\[(h24)\] for every transition \( q \xrightarrow{\Delta} \{q_1, ..., q_n\} \) in \( \mathcal{M} : \ (q, \gamma) \xrightarrow{\text{R}} [(q_1, \varepsilon), ..., (q_n, \varepsilon)] \in \Delta' \), where:

(a) \( \mathbb{R} = \text{equal} \iff \alpha = \gamma \)

(b) \( \mathbb{R} = \text{join}_x \iff \alpha = x \in \mathcal{X} \)

(c) \( \mathbb{R} = \text{join}_{\gamma x} \iff \alpha = \neg x \text{ and } x \in \mathcal{X} \)

\[(h25)\] for every \( q \in \mathcal{F}, (q, \#) \xleftrightarrow{\text{id}} (q, \#) \in \Delta' \)

Roughly speaking, the SABPDS \( \mathcal{BP}_\varphi \) is a kind of product between \( \mathcal{P} \) and the SBPCARET formula \( \varphi \) which ensures that \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle \langle p, \varphi \rangle, B \rangle, \omega \rangle \) iff the configuration \( \langle p, \omega \rangle \) satisfies \( \varphi \) under the environment \( B \). The form of the control locations of \( \mathcal{BP}_\varphi \) is \( \langle \langle p, \phi \rangle, B \rangle \) where \( \phi \in \text{Cl}(\varphi) \), \( B \in \mathcal{B} \). Let us explain the intuition behind our construction:

- If \( \phi = b(\alpha_1, ..., \alpha_n) \), then, for every \( \omega \in \Gamma^* \), \( \langle p, \omega \rangle \models^B \phi \iff b(B(\alpha_1), ..., B(\alpha_n)) \in \lambda(p) \). Thus, for such a \( B, \mathcal{BP}_\varphi \) should have an accepting run from \( \langle \langle p, b(\alpha_1, ..., \alpha_n) \rangle, B \rangle, \omega \rangle \) iff \( b(B(\alpha_1), ..., B(\alpha_n)) \in \lambda(p) \). This is ensured by the transition rules in \( (h1) \) which add a loop at \( \langle \langle p, b(\alpha_1, ..., \alpha_n) \rangle, B \rangle, \omega \rangle \) and the fact that \( \langle \langle p, b(\alpha_1, ..., \alpha_n) \rangle, B \rangle \in F \) (because it is in \( F_1 \)). The function \( \text{id} \) in \( (h1) \) ensures that the environments before and after are the same.
• If $\phi = \neg b(\alpha_1, ..., \alpha_n)$, then, for every $\omega \in \Gamma^*$, $\langle p, \omega \rangle \models_B \phi$ iff $b(B(\alpha_1), ..., B(\alpha_n)) \notin \lambda(p)$. Thus, for such a $B$, $BP_\varphi$ should have an accepting run from $\langle \{p, \neg b(\alpha_1, ..., \alpha_n)\}, B, \omega \rangle$ iff $b(B(\alpha_1), ..., B(\alpha_n)) \notin \lambda(p)$. This is ensured by the transition rules in (h2) which add a loop at $\langle \{p, \neg b(\alpha_1, ..., \alpha_n)\}, B, \omega \rangle$ and the fact that $\langle \{p, \neg b(\alpha_1, ..., \alpha_n)\}, B \rangle \in F$ (because it is in $F_2$). The function $id$ in (h2) ensures that the environments before and after are the same.

• If $\phi = \phi_1 \land \phi_2$, then, for every $\omega \in \Gamma^*$, $\langle p, \omega \rangle \models_B \phi$ iff $\langle p, \omega \rangle \models_B \phi_1$ and $\langle p, \omega \rangle \models_B \phi_2$. This is ensured by the transition rules in (h3) stating that $BP_\varphi$ has an accepting run from $\langle \{p, \phi_1 \land \phi_2\}, B, \omega \rangle$ iff $BP_\varphi$ has an accepting run from both $\langle \{p, \phi_1\}, B, \omega \rangle$ and $\langle \{p, \phi_2\}, B, \omega \rangle$. (h4) is similar to (h3).

• If $\phi = \exists x \phi_1$, then, for every $\omega \in \Gamma^*$, $\langle p, \omega \rangle \models_B \phi$ iff there exists $c \in D$ s.t. $\langle p, \omega \rangle \models_B^{\lambda[x \leftarrow c]} \phi_1$. This is ensured by the transition rules in (h5) stating that $BP_\varphi$ has an accepting run from $\langle \{p, \exists x \phi_1\}, B, \omega \rangle$ iff there exists $c \in D$ s.t. $BP_\varphi$ has an accepting run from $\langle \{p, \phi_1\}, B[x \leftarrow c], \omega \rangle$ since $B \in \text{meet}_F(B[x \leftarrow c])$

• If $\phi = \forall x \phi_1$, then, for every $\omega \in \Gamma^*$, $\langle p, \omega \rangle \models_B \phi$ iff for every $c \in D$, $\langle p, \omega \rangle \models_B^{\lambda[x \leftarrow c]} \phi_1$. This is ensured by the transition rules in (h6) stating that $BP_\varphi$ has an accepting run from $\langle \{p, \forall x \phi_1\}, B, \omega \rangle$ iff for every $c \in D$, $BP_\varphi$ has an accepting run from $\langle \{p, \phi_1\}, B[x \leftarrow c], \omega \rangle$ since if $D = \{c_1, ..., c_m\}$, then, $B \in \text{meet}_F(B[x \leftarrow c_1], ..., B[x \leftarrow c_m])$

• If $\phi = EX^g \phi_1$, then, for every $\omega \in \Gamma^*$, $\langle p, \omega \rangle \models_B \phi$ iff there exists an immediate successor $\langle p', \omega' \rangle$ of $\langle p, \omega \rangle$ s.t. $\langle p', \omega' \rangle \models_B \phi_1$. This is ensured by the transition rules in (h7) stating that $BP_\varphi$ has an accepting run from $\langle \{p, EX^g \phi_1\}, B, \omega \rangle$ iff there exists an immediate successor $\langle p', \omega' \rangle$ of $\langle p, \omega \rangle$ s.t. $BP_\varphi$ has an accepting run from $\langle \{p', \phi_1\}, B, \omega' \rangle$. (h8) is similar to (h7).

• If $\phi = E[\phi_1 U^g \phi_2]$, then, for every $\omega \in \Gamma^*$, $\langle p, \omega \rangle \models_B \phi$ iff $\langle p, \omega \rangle \models_B \phi_2$ or $\langle p, \omega \rangle \models_B \phi_1$ and there exists an immediate successor $\langle p', \omega' \rangle$ of $\langle p, \omega \rangle$ s.t. $\langle p', \omega' \rangle \models_B \phi$. This is ensured by the transition rules in (h11) stating that $BP_\varphi$ has an accepting run from $\langle \{p, E[\phi_1 U^g \phi_2]\}, B, \omega \rangle$ iff $BP_\varphi$ has an accepting run from $\langle \{p, \phi_2\}, B, \omega \rangle$ (by the rules in (h11)(a)) or $BP_\varphi$ has an accepting run from both $\langle \{p, \phi_1\}, B, \omega \rangle$ and $\langle \{p', \phi\}, B, \omega' \rangle$ where $\langle p', \omega' \rangle$ is an immediate successor of $\langle p, \omega \rangle$ (by the rules in (h11)(b)). (h13) is similar to (h11).
• If $\phi = E[\phi_1 R^a \phi_2]$, then, for every $\omega \in \Gamma^*$, $\langle p, \omega \rangle \vdash^G \phi$ iff $(\langle p, \omega \rangle \vdash^G \phi_2$ and $\langle p, \omega \rangle \vdash^G \phi_1$) or $(\langle p, \omega \rangle \vdash^G \phi_2$ and there exists an immediate successor $\langle p', \omega' \rangle$ of $\langle p, \omega \rangle$ s.t. $\langle p', \omega' \rangle \vdash^G \phi$). This is ensured by the transition rules in (h15) stating that $B_\phi$ has an accepting run from $\langle \langle p, E[\phi_1 R^a \phi_2] \rangle, \omega \rangle$ if $B_\phi$ has an accepting run from both $\langle \langle p, \phi_2 \rangle, \omega \rangle$ and $\langle \langle p, \phi_1 \rangle, \omega \rangle$ (by the rules in (h15)(a)); or $B_\phi$ has an accepting run from both $\langle \langle p, \phi_2 \rangle, \omega \rangle$ and $\langle \langle p', \phi \rangle, \omega' \rangle$ where $\langle p', \omega' \rangle$ is an immediate successor of $\langle p, \omega \rangle$ (by the rules in (h15)(b)). In addition, for $R^a$ formulas, the stop condition is not required, i.e., for a formula $\phi_1 R^a \phi_2$ that is applied to a specific run, we don’t require that $\phi_1$ must eventually hold. To ensure that the runs on which $\phi_2$ always holds are accepted, we add $\langle \langle p, \phi \rangle, B \rangle$ to the Büchi accepting condition $F$ (via the subset $F_3$ of $F$). (h16) is similar to (h15).

• If $\phi = EX^a \phi_1$, then, for every $\omega \in \Gamma^*$, $\langle p, \omega \rangle \vdash^G \phi$ iff there exists an abstract-successor $\langle p_k, \omega_k \rangle$ of $\langle p, \omega \rangle$ s.t. $\langle p_k, \omega_k \rangle \vdash^G \phi_1$ (A1). Let $\pi \in Traces(\langle p, \omega \rangle)$ be a run starting from $\langle p, \omega \rangle$ on which $\langle p_k, \omega_k \rangle$ is the abstract-successor of $\langle p, \omega \rangle$. Over $\pi$, let $\langle p', \omega' \rangle$ be the immediate successor of $\langle p, \omega \rangle$. In what follows, we explain how we can ensure this.

1. Firstly, we show that for every abstract-successor $\langle p_k, \omega_k \rangle \neq \bot$ of $\langle p, \omega \rangle$, $\langle \langle p, E X^a \phi_1 \rangle, \omega \rangle \Rightarrow_{B_\phi} \langle \langle p_k, \phi_1 \rangle, B \rangle, \omega_k \rangle$ where $B \in B$. There are two possibilities:

   - If $\langle p, \omega \rangle \Rightarrow_P \langle p', \omega' \rangle$ corresponds to a call statement. $\langle \langle p, \phi \rangle, B \rangle, \omega \rangle \Rightarrow_{B_\phi} \langle \langle p_k, \phi_1 \rangle, B \rangle, \omega_k \rangle$ is ensured by the rules in (h9)(a), the rules in (h21) and the rules in (h19) as follows: rules in (h9)(a) allow to record $\phi_1$ in the return point of the call, rules in (h21) allow to mimic the run of the PDS $P$ and rules in (h19) allow to extract and put back $\phi_1$ when the return-point is reached. The details of how this works are similar to the corresponding case in Section 4.3.2.

   - If $\langle p, \omega \rangle \Rightarrow_P \langle p', \omega' \rangle$ corresponds to a simple statement. Then, the abstract successor of $\langle p, \omega \rangle$ is its immediate successor $\langle p', \omega' \rangle$. Thus, we get that $\langle p_k, \omega_k \rangle = \langle p', \omega' \rangle$. From the transition rules (h9)(b), we get that $\langle \langle p, E X^a \phi_1 \rangle, B \rangle, \omega \rangle \Rightarrow_{B_\phi} \langle \langle p', \phi_1 \rangle, B \rangle, \omega' \rangle$. Therefore, $\langle \langle p, E X^a \phi_1 \rangle, B \rangle, \omega \rangle \Rightarrow_{B_\phi} \langle \langle p_k, \phi_1 \rangle, B \rangle, \omega_k \rangle$. The property holds for this case.

2. Now, let us consider the case where $\langle p_k, \omega_k \rangle$, the abstract successor of $\langle p, \omega \rangle$, is $\bot$. This case occurs when $\langle p, \omega \rangle \Rightarrow_P \langle p', \omega' \rangle$ corresponds to a return statement. Then, one abstract successor of $\langle p, \omega \rangle$ is $\bot$. Note that $\bot$ does not satisfy any formula, i.e., $\bot$ does not satisfy $\phi_1$. Therefore, from
5.2. SBPCARET Model-Checking for Pushdown Systems

We need to ensure that the path of $\mathcal{BP}_\varphi$ reflecting the possibility in (A1) that $\langle p_k, \omega_k \rangle \not\models_\lambda \phi_1$ is not accepted. To do this, we exploit additional trap configurations. We use $p_\perp$ and $\gamma_\bot$ as trap control location and trap stack symbol to obtain these trap configurations. The details of how this works are similar to the corresponding case in Section 4.3.2.

In summary, for every abstract-successor $\langle p_k, \omega_k \rangle$ of $\langle p, \omega \rangle$, if $\langle p_k, \omega_k \rangle \not\models \bot$, then, $\langle \langle p, \text{EX}^a \phi_1 \rangle, \langle B \rangle, \omega \rangle \Rightarrow_{\mathcal{BP}_\varphi} \langle \langle p_k, \phi_1 \rangle, \langle B \rangle, \omega_k \rangle$; otherwise $\langle \langle p, \text{EX}^a \phi_1 \rangle, \langle B \rangle, \omega \rangle \Rightarrow_{\mathcal{BP}_\varphi} \langle \langle p_\perp, \langle B \rangle, \gamma_\bot \omega'' \rangle \Rightarrow_{\mathcal{BP}_\varphi} \langle \langle p_\perp, \langle B \rangle, \gamma_\bot \omega'' \rangle$ which is not accepted by $\mathcal{BP}_\varphi$. Therefore, (A1) is ensured by the transition rules in (h9) stating that $\mathcal{BP}_\varphi$ has an accepting run from $\langle \langle p, \text{EX}^a \phi_1 \rangle, \langle B \rangle, \omega \rangle$ iff there exists an abstract successor $\langle p_k, \omega_k \rangle$ of $\langle p, \omega \rangle$ s.t. $\mathcal{BP}_\varphi$ has an accepting run from $\langle \langle p_k, \phi_1 \rangle, \langle B \rangle, \omega_k \rangle$.

- If $\phi = \text{AX}^a \phi_1$: this case is ensured by the transition rules in (h10) together with (h19) and (h21). The intuition of (h10) is similar to that of (h9).

- If $\phi = E[\phi_1 U^a \phi_2]$, then, for every $\omega \in \Gamma^*$, $\langle p, \omega \rangle \models_\lambda \phi$ iff $\langle p, \omega \rangle \models_\lambda \phi_2$ or $\langle p, \omega \rangle \models_\lambda \phi_1$ and there exists an abstract successor $\langle p_k, \omega_k \rangle$ of $\langle p, \omega \rangle$ s.t. $\langle p_k, \omega_k \rangle \models_\lambda \phi$ (A2). Similar to the case $\phi = E[\phi_1 U^a \phi_2]$ in Section 4.3.2, we get that for every abstract-successor $\langle p_k, \omega_k \rangle$ of $\langle p, \omega \rangle$, if $\langle p_k, \omega_k \rangle \not\models \bot$, then, $\langle \langle p, E[\phi_1 U^a \phi_2] \rangle, \langle B \rangle, \omega \rangle \Rightarrow_{\mathcal{BP}_\varphi} \langle \langle p_k, E[\phi_1 U^a \phi_2] \rangle, \langle B \rangle, \omega_k \rangle$; otherwise $\langle \langle p, E[\phi_1 U^a \phi_2] \rangle, \langle B \rangle, \omega \rangle \Rightarrow_{\mathcal{BP}_\varphi} \langle \langle p_\perp, \langle B \rangle, \gamma_\bot \omega'' \rangle \Rightarrow_{\mathcal{BP}_\varphi} \langle \langle p_\perp, \langle B \rangle, \gamma_\bot \omega'' \rangle$ which is not accepted by $\mathcal{BP}_\varphi$. Therefore, (A2) is ensured by the transition rules in (h12) stating that $\mathcal{BP}_\varphi$ has an accepting run from $\langle \langle p, E[\phi_1 U^a \phi_2] \rangle, \langle B \rangle, \omega \rangle$ iff $\mathcal{BP}_\varphi$ has an accepting run from $\langle \langle p, \phi_2 \rangle, \langle B \rangle, \omega \rangle$ or $\mathcal{BP}_\varphi$ has an accepting run from both $\langle \langle p, \phi_1 \rangle, \langle B \rangle, \omega \rangle$ and $\langle \langle p_k, E[\phi_1 U^a \phi_2] \rangle, \langle B \rangle, \omega_k \rangle$ where $\langle p_k, \omega_k \rangle$ is an abstract successor of $\langle p, \omega \rangle$.

- If $\phi = E[\phi_1 R^a \phi_2]$, then, for every $\omega \in \Gamma^*$, $\langle p, \omega \rangle \models_\lambda \phi$ iff $\langle p, \omega \rangle \models_\lambda \phi_1$ and $\langle p, \omega \rangle \models_\lambda \phi_2$ or $\langle p, \omega \rangle \models_\lambda \phi_1$ and there exists an abstract successor $\langle p_k, \omega_k \rangle$ of $\langle p, \omega \rangle$ s.t. $\langle p_k, \omega_k \rangle \models_\lambda \phi$ (A3). Similar to the case $\phi = E[\phi_1 R^a \phi_2]$ in Section 4.3.2, we get that for every abstract-successor $\langle p_k, \omega_k \rangle$ of $\langle p, \omega \rangle$, if $\langle p_k, \omega_k \rangle \not\models \bot$, then, $\langle \langle p, E[\phi_1 R^a \phi_2] \rangle, \langle B \rangle, \omega \rangle \Rightarrow_{\mathcal{BP}_\varphi} \langle \langle p_k, E[\phi_1 R^a \phi_2] \rangle, \langle B \rangle, \omega_k \rangle$; otherwise $\langle \langle p, E[\phi_1 R^a \phi_2] \rangle, \langle B \rangle, \omega \rangle \Rightarrow_{\mathcal{BP}_\varphi} \langle \langle p_\perp, \langle B \rangle, \gamma_\bot \omega'' \rangle \Rightarrow_{\mathcal{BP}_\varphi} \langle \langle p_\perp, \langle B \rangle, \gamma_\bot \omega'' \rangle$ which is not accepted by $\mathcal{BP}_\varphi$. Therefore, (A3) is ensured by the transition rules in (h17) stating that $\mathcal{BP}_\varphi$ has an accepting run
from \( \langle \langle p, \phi \rangle, B \rangle, \omega \rangle \) iff \( \mathcal{BP}_\varphi \) has an accepting run from both \( \langle \langle p, \phi_1 \rangle, B \rangle, \omega \rangle \) and \( \langle \langle p, \phi_2 \rangle, B \rangle, \omega \rangle \); or \( \mathcal{BP}_\varphi \) has an accepting run from both \( \langle \langle p, \phi_2 \rangle, B \rangle, \omega \rangle \) and \( \langle \langle p_k, E[\phi_1 R^a \phi_2], B \rangle, \omega_k \rangle \) where \( \langle p_k, \omega_k \rangle \) is an abstract successor of \( \langle p, \omega \rangle \).

- The intuition behind the rules corresponding to the cases \( \phi = A[\phi_1 U^a \phi_2] \), \( \phi = A[\phi_1 R^a \phi_2] \) are similar to the previous case.

- If \( \phi = e(e \in \mathcal{V}) \). Given \( p \in P \), \( e \in \mathcal{V} \), \( \omega \in \Gamma^* \), we get that the SABPDS \( \mathcal{BP}_\varphi \) should accept \( \langle \langle p, e \rangle, B \rangle, \omega \rangle \) iff \( \langle \langle p, \omega \rangle, B \rangle \in L(M_e) \). To check whether \( \langle \langle p, \omega \rangle, B \rangle \in L(M_e) \), we let \( \mathcal{BP}_\varphi \) go to state \( s_e \), the initial state corresponding to \( p \) in \( M_e \) by adding rules in \( \langle h22 \rangle \); and then, from this state, we will check whether \( \omega \) is accepted by \( M_e \) under \( B \). This is ensured by the transition rules in \( \langle h24 \rangle \) and \( \langle h25 \rangle \). \( \langle h24 \rangle \) lets \( \mathcal{BP}_\varphi \) mimic a run of \( M_e \) on \( \omega \) under \( B \), which includes three possibilities:

  - if \( \mathcal{BP}_\varphi \) is in a state \( [q, B] \) with \( \gamma \) on the top of the stack where \( \gamma \in \Gamma \), and if \( q \xrightarrow{\gamma} \{q_1, ..., q_n\} \) is a transition rule in \( M_e \), then, \( \mathcal{BP}_\varphi \) will move to states \( [q_1, B], ..., [q_n, B] \) and pop \( \gamma \) from its stack. Note that popping \( \gamma \) allows us to check the rest of the word. This is ensured by the rules corresponding to \( \langle h24 \rangle(a) \). Then function \( equal \) ensures that all these environments are the same.

  - if \( \mathcal{BP}_\varphi \) is in a state \( [q, B] \) with \( \gamma \) on the top of the stack, and if \( q \xrightarrow{\gamma} \{q_1, ..., q_n\} \) is a transition rule in \( M_e \) where \( x \in \mathcal{X} \), then, \( \mathcal{BP}_\varphi \) can mimic a run of \( M_e \) under \( B \) iff \( B(x) = \gamma \). If this condition is guaranteed, \( \mathcal{BP}_\varphi \) will move to states \( [q_1, B], ..., [q_n, B] \) and pop \( \gamma \) from its stack. Again, popping \( \gamma \) allows us to check the rest of the word. This is ensured by the rules corresponding to \( \langle h24 \rangle(b) \). Then function \( join \) ensures that all these environments are the same \( B \) and \( B(x) = \gamma \).

  - Similar to \( \langle h24 \rangle(b) \), \( \langle h24 \rangle(c) \) deals with the cases where \( q \xrightarrow{\gamma} \{q_1, ..., q_n\} \) is a transition rule in \( M_e \) where \( x \in \mathcal{X} \).

In each VA \( M_e \), a configuration is accepted if the run with the word \( \omega \) reaches a final state in \( F_e \); i.e., if \( \mathcal{BP}_\varphi \) reaches a state \( q \in F_e \) with an empty stack, i.e., with a stack containing the bottom stack symbol \( \sharp \). Thus, we should add \( F_e \times B \) as a set of accepting control locations in \( \mathcal{BP}_\varphi \). This is why \( F_e \) is a set of accepting control locations. In addition, since \( \mathcal{BP}_\varphi \) only recognizes infinite paths, \( \langle h25 \rangle \) adds a loop on every configuration \( \langle [q, B], \sharp \rangle \) where \( q \in F_e \).
5.2. SBPCARET Model-Checking for Pushdown Systems

- If $\phi = \neg e (e \in \mathcal{V})$. This case is ensured by the transition rules in (h23), (h24) and (h25). The intuition behind this case is similar to the case $\phi = e$.

The Büchi accepting condition. The elements of the Büchi accepting condition set $F$ of $\mathcal{BP}_{\varphi}$ ensure the liveness requirements of until-formulas on infinite global paths, infinite abstract paths as well as on finite abstract paths.

- With regards to infinite global paths, the fact that the liveness requirement $\phi_2$ in $E[\phi_1 \Upsilon \phi_2]$ is eventually satisfied in $\mathcal{P}$ is ensured by the fact that $[[p, E[\phi_1 \Upsilon \phi_2]], B]$ doesn’t belong to $F$. The intuition behind this case is similar to the case in Section 4.3.2.

- With regards to infinite abstract paths, the fact that the liveness requirement $\phi_2$ in $E[\phi_1 \Upsilon \phi_2]$ is eventually satisfied in $\mathcal{P}$ is ensured by the fact that $[[p, E[\phi_1 \Upsilon \phi_2]], B]$ doesn’t belong to $F$. The intuition behind this case is similar to the intuition of $E[\phi_1 \Upsilon \phi_2]$. The liveness requirements of $A[\phi_1 \Upsilon \phi_2]$ are ensured as for the case of $E[\phi_1 \Upsilon \phi_2]$.

- With regards to finite abstract paths $\langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \ldots \langle p_m, \omega_m \rangle$ where $\langle p_m, \omega_m \rangle \Rightarrow_\mathcal{P} \langle p_{m+1}, \omega_{m+1} \rangle$ corresponds to a return statement, the fact that the liveness requirement $\phi_2$ in $E[\phi_1 \Upsilon \phi_2]$ is eventually satisfied in $\mathcal{P}$ is ensured by the fact that $[\langle p_1, B \rangle]$ doesn’t belong to $F$. The intuition behind this case is similar to the corresponding case in Section 4.3.2.

- With regards to finite abstract paths $\langle p_0, \omega_0 \rangle \langle p_1, \omega_1 \rangle \ldots \langle p_m, \omega_m \rangle$ where $\langle p_m, \omega_m \rangle \Rightarrow_\mathcal{P} \langle p_{m+1}, \omega_{m+1} \rangle$ corresponds to a call statement but this call never reaches its corresponding return-point, the fact that the liveness requirement $\phi_2$ in $E[\phi_1 \Upsilon \phi_2]$ is eventually satisfied in $\mathcal{P}$ is ensured by the fact that $[\langle p, B \rangle] \notin F$ where $p \in P$, $B \in B$. The intuition behind this case is similar to the corresponding case in Section 4.3.2.

Theorem 20. Given a PDS $\mathcal{P} = (P, \Gamma, \Delta)$, a set of atomic propositions $\mathcal{AP}$, a labelling function $\lambda : \mathcal{AP}_P \rightarrow 2^P$ and a SBPCARET formula $\varphi$, we can compute an SABPDS $\mathcal{BP}_\varphi$ such that for every configuration $\langle p, \omega \rangle$, for every $B \in B$, $\langle p, \omega \rangle \models_B^{\lambda} \varphi$ iff $\mathcal{BP}_\varphi$ has an accepting run from the configuration $[[\langle p, \varphi \rangle], B], \omega]$.

Formal proof: Given $c_0 \Rightarrow_{\mathcal{BP}_\varphi} \{c_1, c_2, \ldots, c_n\}$ where for every $0 \leq i \leq n$, $c_i$ is a configuration of the SABPDS $\mathcal{BP}_\varphi$. For presentation reasons, we also write $c_0 \Rightarrow_{\mathcal{BP}_\varphi} c_1 \land c_2 \land \ldots \land c_n$. We prove the following two directions:
Assume that \( \langle p, \omega \rangle \models^B \varphi \), we need to prove that \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle \langle [p, \varphi], B \rangle, \omega \rangle \). In what follows, we show how this is ensured by induction on the structure for the SBPCARET formula \( \varphi \).

**Proof.**

- **Base case:**

  \[ \varphi = b(\alpha_1, ..., \alpha_n): \langle p, \omega \rangle \models^B \varphi \implies b(B(\alpha_1), ..., B(\alpha_n)) \in \lambda(p) \].

  According to the transition rule in (h1), we get \( \langle \langle [p, b(\alpha_1, ..., \alpha_n)], B \rangle, \omega \rangle \Rightarrow_{\mathcal{BP}_\varphi} \langle \langle [p, b(\alpha_1, ..., \alpha_n)], B \rangle, \omega \rangle \). In addition, we get that \( \langle \langle [p, b(\alpha_1, ..., \alpha_n)], B \rangle \in F \) for every \( b(B(\alpha_1), ..., B(\alpha_n)) \in \lambda(p) \). Therefore, \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle \langle [p, b(\alpha_1, ..., \alpha_n)], B \rangle, \omega \rangle \). In other words, \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle \langle [p, \varphi], B \rangle, \omega \rangle \). The property holds for this case.

- **Case \( \varphi = \neg b(\alpha_1, ..., \alpha_n): \langle p, \omega \rangle \models^B \varphi \implies b(B(\alpha_1), ..., B(\alpha_n)) \not\in \lambda(p) \).**

  According to the transition rule in (h2), we get \( \langle \langle [p, \neg b(\alpha_1, ..., \alpha_n)], B \rangle, \omega \rangle \Rightarrow_{\mathcal{BP}_\varphi} \langle \langle [p, \neg b(\alpha_1, ..., \alpha_n)], B \rangle, \omega \rangle \). In addition, we get that \( \langle \langle [p, \neg b(\alpha_1, ..., \alpha_n)], B \rangle \in F \) for every \( b(B(\alpha_1), ..., B(\alpha_n)) \not\in \lambda(p) \). Therefore, \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle \langle [p, \neg b(\alpha_1, ..., \alpha_n)], B \rangle, \omega \rangle \). In other words, \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle \langle [p, \varphi], B \rangle, \omega \rangle \). The property holds for this case.

- **Case \( \varphi = e(e \in \mathcal{V}): \langle p, \omega \rangle \models^B \varphi \implies (\langle p, \omega \rangle, B) \in L(M_e) \implies M_e \) has an accepting run from the corresponding initial state \( s_e \) on the word \( \omega \) under \( B \).**

  Firstly, we show that if \( M_e \) has an accepting run from a state \( q \in Q_e \) over the word \( u \) under the environment \( B \), then \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle \langle [q, B], u \rangle \) by induction on the length of \( u \) (denoted by \( |u| \)).

  * **Basis.** \( |u| = 1 \) (note that \( \sharp \) will never be popped) \( \implies u = \sharp \implies q \in F_e \implies \langle [q, B] \rangle \) is an accepting control location of \( \mathcal{BP}_\varphi \) (by the accepting condition et \( F_e \)). In addition, according to (h25), we get that \( \langle [q, B], \sharp \rangle \Rightarrow_{\mathcal{BP}_\varphi} \langle [q, B], \sharp \rangle \). Therefore, \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle [q, B], \sharp \rangle \). In other words, \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle [q, B], \sharp \rangle \). The property holds for this case.

  * **Step.** \( |u| \geq 2 \), then, there exists \( \gamma \in \Gamma, \upsilon \in \Gamma^* \) s.t. \( u = \gamma \upsilon \) and \( M_e \) has an accepting run from \( q \) over the word \( \gamma \upsilon \) under the environment \( B \). Let \( t \) be the first transition rule used by that accepting run from \( q \) in \( M_e \). There are three possibilities:

    - **Case \( t = q \xrightarrow{\gamma} \{q_1, ..., q_n\} \), where \( \gamma \in \Gamma \).** From (2), we get that for \( 1 \leq i \leq n \), \( M_e \) has an accepting run from \( q_i \) over
the word \( v \) under the environment \( B \). By applying the induction hypothesis, we get that \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle [q, B], v \rangle \) for every \( 1 \leq i \leq n \) (3). Also, from the transition rules in \((h24)(a)\), we get that: 
\[
\langle q, \gamma \rangle \xrightarrow{\text{equal}} \{\langle q_1, \varepsilon \rangle, ..., \langle q_n, \varepsilon \rangle\} \in \Delta' \quad (4).
\]
From (4) and (3), we obtain that \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle [q, B], \gamma v \rangle \). In other words, \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle [q, B], u \rangle \). The property holds for this case.

- Case \( t = q \xrightarrow{z} \{q_1, ..., q_n\} \), where \( x \in X \). From (2) and the definition of an accepting run of a VA, we get that \( B(x) = \gamma \) and for every \( 1 \leq i \leq n \), \( M_e \) has an accepting run from \( q_i \) over the word \( v \) under the environment \( B \). By applying the induction hypothesis, we get that \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle [q_i, B], v \rangle \) for every \( 1 \leq i \leq n \) (5). From the transition rules in \((h24)(b)\), we get that:
\[
\langle q, \gamma \rangle \xrightarrow{\text{join}^{\approx}} \{\langle q_1, \varepsilon \rangle, ..., \langle q_n, \varepsilon \rangle\} \in \Delta' \quad (6).
\]
From (6) and (5), we obtain that \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle [q, B], \gamma v \rangle \) (since \( \text{join}^{\approx}(B, ..., B) = \{B \mid B(x) = \gamma \} \)). In other words, \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle [q, B], u \rangle \). The property holds for this case.

- Case \( t = q \xrightarrow{\neg z} \{q_1, ..., q_n\} \), where \( x \in X \). This case is similar to the case \( t = q \xrightarrow{z} \{q_1, ..., q_n\} \). The property holds for this case.

In conclusion, we get that if \( M_e \) has an accepting run from a state \( q \in Q_e \) over the word \( u \) under the environment \( B \), then, \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle [q, B], u \rangle \) (7).

From (1) and (7), we get that \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle [s_e, B], \omega \rangle \). Therefore, \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle [\langle p, \epsilon \rangle, B], \omega \rangle \) (by the rules in \((h22)\)). The property holds for this case.

- Induction Step:

  - Case \( \varphi = \exists x \varphi_1 \):

    * Since \( \langle p, \omega \rangle \models^{B}_X \varphi \), we obtain that there exists \( d \in \mathcal{D} \) s.t. \( \langle p, \omega \rangle \models^{[B \leftarrow d]}_X \varphi_1 \). By applying the induction hypothesis, we get that \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle [\langle p, \varphi_1 \rangle, [B \leftarrow d]], \omega \rangle \) (8).
Proof. • prove this by induction on the structure of $\langle p, \varphi \rangle$. Assume that $M, B \models \varphi$. – Case $K$ for this case.

* From the transition rule in $(h5)$, we get that $\langle \langle p, \varphi \rangle, \gamma \rangle \overset{\text{meet}_d}{\rightarrow} \langle \langle p, \varphi \rangle, \gamma \rangle$ is a transition rule in $BP_\varphi$. Also, note that $B \in \text{meet}_d(B[x \leftarrow d])$. Therefore, $\langle \langle \langle p, \varphi \rangle, B \rangle, \omega \rangle$ is an immediate predecessor of $\langle \langle p, \varphi \rangle, B[x \leftarrow d], \omega \rangle$ for every $1 \leq i \leq n$.

* From the transition rule in $(h6)$, we get that $\langle \langle p, \varphi \rangle, \gamma \rangle \overset{\text{meet}_d'}{\rightarrow} \langle \langle \langle p, \varphi \rangle, \gamma \rangle, ..., \langle \langle p, \varphi \rangle, \gamma \rangle \rangle$ is a transition rule in $BP_\varphi$. Also, note that $B \in \text{meet}_d'(B[x \leftarrow d])$. Therefore, $\langle \langle p, \varphi \rangle, B \rangle, \omega \rangle$ is an immediate predecessor of $\langle \langle \langle p, \varphi \rangle, B[x \leftarrow d], \omega \rangle \rangle$ for every $1 \leq i \leq n$.

From (8) and (9), we get that $BP_\varphi$ has an accepting run from the configuration $\langle \langle p, \varphi \rangle, B \rangle, \omega \rangle$. The property holds for this case.

– Remaining cases: The proof is similar to the proof of the corresponding cases of Theorem 13 in Section 4.3.2.

$(\Longleftarrow)$ Assume that $BP_\varphi$ has an accepting run from the configuration $\langle \langle p, \varphi \rangle, B \rangle, \omega \rangle$. We need to prove that $\langle \langle p, \varphi \rangle, B \rangle, \omega \rangle$. In what follows, we prove this by induction on the structure of $\varphi$.

Proof. • Base case:

$\varphi = b(\alpha_1, ..., \alpha_n)$: Since $BP_\varphi$ has an accepting run from $\langle \langle p, b(\alpha_1, ..., \alpha_n) \rangle, B \rangle, \omega \rangle$, we get that $\langle \langle p, b(\alpha_1, ..., \alpha_n) \rangle, B \rangle, \omega \rangle$. From all transition rules of $BP_\varphi$, the unique way to have immediate successors of $\langle \langle p, b(\alpha_1, ..., \alpha_n) \rangle, B \rangle, \omega \rangle$ is from the rules in $(h1)$, which means that $\langle \langle p, b(\alpha_1, ..., \alpha_n) \rangle, B \rangle, \omega \rangle \Rightarrow_{BP_\varphi} \langle \langle p, b(\alpha_1, ..., \alpha_n) \rangle, B \rangle, \omega \rangle$. In addition, this run is accepting implies that $\langle \langle p, b(\alpha_1, ..., \alpha_n) \rangle, B \rangle, \omega \rangle$ must be an accepting control location of $BP_\varphi$. Therefore, we obtain that $b(B(\alpha_1), ..., B(\alpha_n)) \in \lambda(p)$ (by the condition of the accepting set $F$). In other words, $\langle p, \omega \rangle \vdash b(\alpha_1, ..., \alpha_n)$. The property holds for this case.
\[\varphi = -b(\alpha_1, ..., \alpha_n): \text{Since } \mathcal{BP}_\varphi \text{ has an accepting run from } \langle \langle q, B \rangle, \omega \rangle \implies \langle \langle \langle p, -b(\alpha_1, ..., \alpha_n) \rangle, B \rangle, \omega \rangle \text{ must have immediate successors. From all transition rules of } \mathcal{BP}_\varphi, \text{ the unique way to have immediate successors of } \langle \langle p, -b(\alpha_1, ..., \alpha_n) \rangle, B \rangle, \omega \rangle \text{ is from the rules in } (h2), \text{ which means that } \langle \langle p, -b(\alpha_1, ..., \alpha_n) \rangle, B \rangle, \omega \rangle \Rightarrow_{\mathcal{BP}_\varphi} \langle \langle p, -b(\alpha_1, ..., \alpha_n) \rangle, B \rangle, \omega \rangle. \text{ In addition, this run is accepting implies that } [p, -b(\alpha_1, ..., \alpha_n), B] \text{ must be an accepting control location of } \mathcal{BP}_\varphi. \text{ Therefore, we obtain that } b(B(\alpha_1), ..., B(\alpha_n)) \notin \lambda(p) \text{ (by the condition of the accepting set } F_2). \text{ In other words, } [p, \omega] \models^\mathcal{B}_\varphi b(\alpha_1, ..., \alpha_n). \text{ The property holds for this case.}

\[\varphi = e(e \in \mathcal{V}).\]

Firstly, we will prove that for every \( q \in Q_e \), \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle q, B \rangle, u \) implies that \( M_e \) has an accepting run from \( q \) over the word \( u \) under the environment \( B \). We show this by induction on the length of \( u \) (denoted by \( |u| \)).

* Basis. \( |u| = 1 \), then, \( u = \varepsilon \). Since \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle q, B \rangle, \varepsilon \rangle \), we get that \( \langle q, B \rangle, \varepsilon \rangle \) must have immediate successors. From all transition rules of \( \mathcal{BP}_\varphi \), the unique way to have immediate successors of \( \langle q, B \rangle, \varepsilon \rangle \) is from the rules in \( (h25) \), which means that \( \langle q, B \rangle, \varepsilon \rangle \Rightarrow_{\mathcal{BP}_\varphi} \langle q, B \rangle, \varepsilon \rangle \). By the condition in \( (h25) \), we must have \( q \in F \implies M_e \) has an accepting run from the state \( q \) over the word \( u \) under \( B \). The property holds in this case.

* Step. \( |u| \geq 2 \), then, there exists \( \gamma \in \Gamma, v \in \Gamma^* \) s.t. \( u = \gamma v \) and \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle q, B \rangle, u \rangle \) \( (12) \). Let \( t \) be the first transition rule used by that accepting run in \( \mathcal{BP}_\varphi \). There are three possibilities:

  - Case \( t = \langle q, \gamma \rangle \xrightarrow{\text{equal}} \{q_1, \varepsilon, ..., q_n, \varepsilon \} \), then, we get that \( q \xrightarrow{\gamma} \{q_1, ..., q_n\} \in \delta_e \) (by the transition rules in \( (h24)(a) \)) \( (13) \). From \( (12) \), we get that \( \mathcal{BP}_\varphi \) has an accepting run from \( \langle q, B \rangle, \gamma v \rangle \Rightarrow_{\mathcal{BP}_\varphi} \langle q, B \rangle, \gamma v \rangle \) for every \( 1 \leq i \leq m \). By applying the induction hypothesis, we get that \( M_e \) has an accepting run from \( q_i \) over the word \( v \) under \( B \) for every \( 1 \leq i \leq m \) \( (14) \). From \( (13) \) and \( (14) \), we obtain that \( M_e \) has an accepting run from \( q \) over the word \( \gamma v \) under \( B \). In other words, \( M_e \) has an accepting run from \( q \) over the word \( u \) under \( B \). The property holds for this case.
• Case $t = \langle q, \gamma \rangle \xrightarrow{\text{join}^\varphi} \{\langle q_1, \varepsilon \rangle, ..., \langle q_n, \varepsilon \rangle\}$, then, we get that $q \xrightarrow{x} \{q_1, ..., q_n\} \in \delta_e$ and $B(x) = \gamma$ (by the transition rules in (h24)(b)) (15). From (12), we get that $BP_\varphi$ has an accepting run from $\langle [q, B], \gamma v \rangle$ $\implies BP_\varphi$ has an accepting run from $\langle [q, B], v \rangle$ for every $1 \leq i \leq m$. By applying the induction hypothesis, we get that $M_e$ has an accepting run from $q_i$ over the word $v$ under $B$ for every $1 \leq i \leq m$ (16). From (15) and (16), we obtain that $M_e$ has an accepting run from $q$ over the word $\gamma v$ under $B$. In other words, $M_e$ has an accepting run from $q$ over the word $u$ under $B$. The property holds for this case.

• Case $t = \langle q, \gamma \rangle \xrightarrow{\text{join}^\varphi} \{\langle q_1, \varepsilon \rangle, ..., \langle q_n, \varepsilon \rangle\}$, where $x \in \mathcal{X}$. This case is similar to the case $t = \langle q, \gamma \rangle \xrightarrow{\text{join}^\varphi} \{\langle q_1, \varepsilon \rangle, ..., \langle q_n, \varepsilon \rangle\}$. The property holds for this case.

$BP_\varphi$ has an accepting run from $\langle [[p, e], B], \omega \rangle \implies \langle [[p, e], B], \omega \rangle$ must have immediate successors. From all transition rules of $BP_\varphi$, the unique way to have immediate successors of $\langle [[p, e], B], \omega \rangle$ is from the rules in (h22), which means that $\langle [[p, e], B], \omega \rangle$ $\not\Rightarrow BP_\varphi \langle [s_e, B], \omega \rangle$. Therefore, $BP_\varphi$ must have an accepting run from $\langle [s_e, B], \omega \rangle$. From the above result, we obtain that $M_e$ has an accepting run from $s_e$ over the word $\omega$ under the environment $B$ which means that $\langle p, \omega \rangle, B \in L(\varepsilon)$. In other words, $\langle p, \omega \rangle \vDash^\varphi \varepsilon$. The property holds for this case.

• Induction Step:
  
  - Case $\varphi = \exists x \varphi_1$:
    
    * Suppose $D = \{d_1, ..., d_n\}$. Since $BP_\varphi$ has an accepting run from $\langle [[p, \exists x \varphi_1], B], \omega \rangle \implies \langle [[p, \exists x \varphi_1], B], \omega \rangle$ must have immediate successors. From all transition rules of $BP_\varphi$, the unique way to have immediate successors of $\langle [[p, \exists x \varphi_1], B], \omega \rangle$ is from the rules in (h5), which means that for every $1 \leq i \leq n$, $\langle [[p, \varphi_1], B[x \leftarrow d_i]], \omega \rangle$ can be the child of the configuration $\langle [[p, \varphi], B], \omega \rangle$ in the accepting run of $BP_\varphi$ (17).
    
    * From (17), and the fact that $BP_\varphi$ has an accepting run from $\langle [[p, \exists x \varphi_1], B], \omega \rangle$, we get that there exits an $i$, $1 \leq i \leq n$ s.t. $BP_\varphi$ has an accepting run from $\langle [[p, \varphi_1], B[x \leftarrow d_i]], \omega \rangle$. By applying the induction hypothesis, we get that there exists $1 \leq i \leq n$ s.t. $\langle p, \omega \rangle \vDash^{[B \leftarrow d_i]} \varphi_1$. Therefore, $\langle p, \omega \rangle \vDash^\varphi \varphi_1$ (by the
5.3 Conclusion

In this chapter, we present the logic SBPCARET and show how it can precisely and succinctly specify malicious behaviors. We then propose an efficient algorithm for SBPCARET model-checking for PDSs. Our algorithm is based on reducing the model checking problem to the emptiness problem of Symbolic Alternating Büchi Pushdown Systems.
Chapter 5. BCARET Model Checking for Malware Detection
Dynamic Pushdown Networks (DPNs) are a natural model for multithreaded programs with (recursive) procedure calls and thread creation. We consider in this chapter the model-checking problem of DPNs against CARET formulas. We show that this problem can be effectively solved by a reduction to the emptiness problem of Büchi Dynamic Pushdown Systems. We then show that CARET model checking is also decidable for DPNs communicating with locks. Our results can, in particular, be used for the detection of concurrent malware.

Outline. In Section 6.1, we define DPNs and single-indexed CARET formulas for DPNs. Section 6.2 shows how single-indexed CARET for DPNs can be applied to detect concurrent malware. Section 6.3 presents how to model check DPNs against single-indexed CARET formulas. The model checking for DPNs with regular valuations is discussed in Section 6.4. Section 6.5 presents the model-checking problem for DPNs with nested locks against single-indexed CARET formulas. We discuss the related work in Section 6.6 and conclude in Section 6.7.

6.1 Dynamic Pushdown Networks (DPNs)

6.1.1 Definitions

Dynamic Pushdown Networks (DPNs) is a natural model for multithreaded programs [BMT05]. To be able to define CARET formulas over DPNs, we must extend this model to record whether a transition rule corresponds to a call, ret or a simple statement (neither call nor ret).

Definition 18. A Dynamic Pushdown Network (DPN) \( \mathcal{M} \) is a set \( \{P_1, ..., P_n\} \) s.t. for every \( 1 \leq i \leq n \), \( P_i = (P_i, \Gamma_i, \Delta_i) \) is a Labelled Dynamic Pushdown System (DPDS), where \( P_i \) is a finite set of control locations, \( P_i \cap P_j = \emptyset \) for all \( j \neq i \), \( \Gamma_i \) is a finite set of stack alphabet, and \( \Delta_i \) is a finite set of transition rules. Rules of \( \Delta_i \) are of the following form, where \( p, p_1 \in P_i, \gamma, \gamma_1, \gamma_2 \in \Gamma_i, \omega_1 \in \Gamma_i^*, d \in \{\square, p_1 \omega_1 | p_1 \omega_1 \in \bigcup_{1 \leq j \leq n} P_j \times \Gamma_j^*\} \):
• \((r_1)\) \(p^\gamma \overset{\text{call}}{\rightarrow}_i p_1 p_1 p_2 d\)

• \((r_2)\) \(p^\gamma \overset{\text{ret}}{\rightarrow}_i p_1 ε d\)

• \((r_3)\) \(p^\gamma \overset{\text{int}}{\rightarrow}_i p_1 ω_1 d\)

Intuitively, there are two kinds of transition rules depending on the nature of \(d\). A rule with a suffix of the form \(d\) is a nonspawn rule (does not spawn a new process), while a rule with a suffix \(p_ω\) describes a spawn rule (a new process is spawned). A nonspawn step describes pushdown operations of one single process in the network. Roughly speaking, a call statement is described by a rule in the form \(p^\gamma \overset{\text{call}}{\rightarrow}_i p_1 p_2 d\). This rule usually models a statement of the form \(\gamma \overset{\text{call}}{\rightarrow}_i \gamma_2\) where \(\gamma\) is the control point of the program where the function call is made, \(\gamma_1\) is the entry point of the called procedure \(\gamma\), and \(\gamma_2\) is the return point of the call; \(p_1\) and \(p_2\) can be used to encode various information, such as the return values of functions, shared data between procedures, etc. A return statement is modeled by a rule \((r_2)\), while a rule \((r_3)\) is used to model a simple statement (neither a call nor a return).

A spawn step allows in addition the creation of a new process. For instance, a \(p_\omega\) describes a spawn rule (a new process is spawned). For every \(d\) of \(\{\text{call, ret, int}\}\) describes that a process \(P_i\) at control location \(p_1\) and having \(\gamma\) on top of the stack can (1) change the control location to \(p_1\) and modify the stack by replacing \(\gamma\) with \(\omega_1\) and also (2) create a new instance of a process \(P_j\) \((1 ≤ j ≤ n)\) starting at \(p_\omega\). Note that in this case, \(p_1\) is call, \(\omega_1\) is \(\gamma_1\), and \(p_1\) is ret, then \(\omega_1\) is ε.

A DPDS \(P_i\) can be seen as a Pushdown System (PDS) if there are no spawn rules in \(\Delta_i\). Generally speaking, a DPN consists of a set of PDSs \(\{P_1, ..., P_n\}\) running in parallel where each PDS can dynamically spawn new instances of PDSs in the set \(\{P_1, ..., P_n\}\) during the run. An initial local configuration of a newly created instance \(p_\omega\) is called a Dynamically Created Local Initial Configuration (DCLIC). For every \(i \in \{1, 2, ..., n\}\), let \(D_i = \{p_\omega \in \bigcup_{1 ≤ j ≤ n} P_j \times Γ_j^* \mid P_i \overset{\gamma}{\rightarrow}_j p_1 p_1 p_2 p_3 p_\omega d\} \subseteq \text{DCLICs}\) that can be created by the DPDS \(P_i\).

A local configuration of an instance of a DPDS \(P_i\) is a tuple \(p_\omega\) where \(p_1 \in P_i\) is the control location, \(\omega \in Γ_i^*\) is the stack content. A global configuration of \(\mathcal{M}\) is a multiset over \(\bigcup_{1 ≤ i ≤ n} P_i \times Γ_i^*\), in which \(p_\omega \in P_i \times Γ_i^*\) is a local configuration of an instance of \(P_i\) which is running in parallel in the network \(\mathcal{M}\).

A DPDS \(P_i\) defines a transition relation \(\Rightarrow_i\) as follows: if \(p_\gamma \overset{\gamma}{\rightarrow}_i p_1 p_2 d\) then \(p_\gamma \Rightarrow_i p_1 p_2 p_3 D\) for every \(\omega \in Γ_i^*\) where \(D = \emptyset\) if \(d = \square\), \(D = \{p_\omega\}\) if \(d = p_\omega\). Let \(\Rightarrow_i^*\) be the transitive and reflexive closure of \(\Rightarrow_i\), then, for every \(p_\omega \in P_i \times Γ_i^*\):

• \(p_\omega \Rightarrow_i^* p_\omega \emptyset\)
6.2. Applications

- if \( p_{\omega} \Rightarrow_{i} p_{1\omega_1} \triangleright D_1 \) and \( p_{1\omega_1} \Rightarrow_{i} p_{2\omega_2} \triangleright D_2 \), then, \( p_{\omega} \Rightarrow_{i} p_{2\omega_2} \triangleright D_1 \cup D_2 \).

A local run of an instance of a DPDS \( P_i \) starting at a local configuration \( c_0 \) is a sequence \( c_0 c_1 \ldots \) s.t. for every \( x \geq 0 \), \( c_x \in P_i \times \Gamma_i^{*} \) is a local configuration of \( P_i \), \( c_x \Rightarrow_{i} c_{x+1} \triangleright D \) for some \( D \). A global run \( \rho \) of \( M \) from a global configuration \( G = \{ p_0\omega_0, \ldots, p_k\omega_k \} \) is a set of local runs (possibly infinite) where each local run describes the execution of one instance of a certain DPDS \( P_i \). Initially, \( \rho \) consists of \( k \) local runs of \( k \) instances starting from \( \{ p_0\omega_0, \ldots, p_k\omega_k \} \), when a new instance is created, a new local run of this instance is added to \( \rho \). For example, when a DCLIC \( c \) is created by a certain local run of \( \rho \), a new local run that starts at \( c \) is added to \( \rho \). Note that from a global configuration, we can obtain a set of global runs because from a local configuration, we can have different local runs.

6.1.2 Single-indexed CARET for DPNs

Given a DPN \( M = \{ P_1, \ldots, P_n \} \), a single-indexed CARET formula \( f \) is a formula in the form \( \bigwedge_{i=1}^{n} f_i \) s.t. for every \( 1 \leq i \leq n \), \( f_i \) is a CARET formula in which the satisfiability of its atomic propositions depends only on the DPDS \( P_i \).

Given a set of atomic propositions \( AP \), let \( \lambda : \bigcup_{i=1}^{n} P_i \rightarrow 2^{AP} \) be a labeling function that associates each control location with a set of atomic propositions.

Let \( \pi = p_0\omega_0p_1\omega_1 \ldots \) be a local run of the DPDS \( P_i \). We associate to each local configuration \( p_x\omega_x \) of \( \pi \) a tag \( t_x \) in \( \{ call, int, ret \} \) as follows, where \( D = \emptyset \) or \( D = \{ p_s\omega_s \} \):

- If \( p_x\omega_x \Rightarrow_{i} p_{x+1}\omega_{x+1} \triangleright D \) corresponds to a transition rule \( p_{\gamma} \Rightarrow_{i} p_{i\omega_1} \triangleright d \), then \( t_x = t \).

Then, we say that \( \pi \) satisfies \( f_i \) iff the \( \omega \)-word \( (\lambda(p_0), t_0)(\lambda(p_1), t_1) \ldots \) satisfies \( f_i \). A local configuration \( c \) of \( P_i \) satisfies \( f_i \) (denoted \( c \models f_i \)) iff there exists a local run \( \pi \) starting from \( c \) such that \( \pi \) satisfies \( f_i \). If \( D \) is the set of DCLICs created during the run \( \pi \), then, we write \( c \models_{D} f_i \). A DPN \( M \) satisfies a single-indexed CARET formula \( f \) iff there exist a global run \( \rho \) s.t. for every \( 1 \leq i \leq n \), each local run of \( P_i \) in \( \rho \) satisfies the formula \( f_i \).

6.2 Applications

Several malwares are multithreaded programs that involve recursive procedures and dynamic thread creation. Therefore, DPNs can be used to model such
programs. We show in what follows how single-indexed CARET for DPNs can describe malicious behaviors of concurrent malwares.

More precisely, we show how this logic can specify email worms. To this aim, let us consider a typical email worm: the worm Bagle. Bagle is a multithreaded email worm. In the main thread, one of the first things the worm does is to register itself into the registry listing to be started at the boot time. Then, it does some different actions to hide itself from users. After this, the malware creates one thread (named \texttt{Thread2}) that listens on the port 6777 to receive different commands and also allow the attacker to upload a new file and execute it. This grants the attacker the ability to update new versions for his malware. In addition, the attacker can send a crafted byte sequence to this port to force the malware to kill itself and delete it from the system. Thus, the attacker can remove his malware remotely. In the next step, the malware creates one more thread (named \texttt{Thread3}) which contacts a list of websites every 10 minutes to announce the infection of the current machine. The malware sends the port it is listening to as well as the IP of the infected machine to these sites. At some point in the program, the malware continues to spawn a thread named \texttt{Thread4} to search on local drives to look for valid email addresses. In this thread, for each email address found, the malware attaches itself and sends itself to this email address.

Thus, you can see that Bagle is a mutithreaded malware with dynamic thread creation, i.e., the main process can create threads to fulfill various tasks. To model Bagle, DPNs is a good candidate since DPNs allow dynamic thread creation. Let $\mathcal{M} = \{P_1, P_2, P_3, P_4\}$ be a model of Bagle where $P_1$ is a PDS that represents the main process of the malware; $P_2, P_3, P_4$ are PDSs that model the code segments corresponding to Thread1, Thread2, Thread3 respectively. Note that $P_2, P_3, P_4$ are designed to execute specific tasks, while $P_1$ is a main process able to dynamically create an arbitrary number of instances of $P_2, P_3, P_4$ to fulfill tasks in need.

We show now how the malicious behavior of the different threads can be described by a CARET formula. Let us start with the main process. The typical behaviour of this process is to add its own executable name to the registry listing so that it can be started at the boot time. As already explained in Section 2.5.1, to do this, the malware needs to invoke the API function \texttt{GetModuleFileNameA} with 0 and $x$ as parameters. \texttt{GetModuleFileNameA} will put the file name of its current executable on the memory address pointed by x. After that, the malware calls the API function \texttt{RegSetValueExA} with the same $x$ as parameter. \texttt{RegSetValueExA} will use the file name stored at $x$ to add itself into the registry key listing. This malicious behaviour can be specified by CARET as follows:
6.2. Applications

\[ \psi_1 = \bigvee_{x \in K} F^a \left( \text{call(GetModuleNameA)} \land x \Gamma^* \land F^a \left( \text{call(RegSetValueExA)} \land x \Gamma^* \right) \right) \]

where the \( \bigvee \) is taken over all possible memory addresses \( x \) over domain \( K \).

Note that parameters are passed via the stack in binary programs. For succinctness, we use regular variable expression \( x \Gamma^* \) (resp. \( 0x \Gamma^* \)) to describe the requirement that \( x \) (resp. \( 0x \)) is on top of the stack. Then, this formula states that there is a call to the API \( \text{GetModuleNameA} \) with 0 and \( x \) on the top of the stack (i.e., with 0 and \( x \) as parameters), followed by a call to the API \( \text{RegSetValueExA} \) with \( x \) on the top of the stack. Using the operator \( F^a \) guarantees that \( \text{GetModuleNameA} \) and \( \text{RegSetValueExA} \) are invoked in the same function. This allows to avoid the case where \( \text{RegSetValueExA} \) is invoked before the function \( \text{GetModuleNameA} \) terminates.

The primary behavior of \( \text{Thread2} \) is to set up the malware to listen to a certain port to get updated information (new attack targets, ...). As already explained in Section 2.5.1., to achieve this task, it needs to call the API \( \text{socket} \) to create a socket, followed by a call to the API \( \text{bind} \) to associate a local address with the socket and a call to \( \text{listen} \) to put the socket in the listening state. The call to the API \( \text{socket} \) returns a descriptor referencing the new socket which is used as input of the calls to the APIs \( \text{bind} \) and \( \text{listen} \). Thus, when \( \text{bind} \) and \( \text{listen} \) are invoked, the socket descriptor must be on top of the program’s stack (since parameters are passed via the stack in binary programs). Using CARET, this malicious behaviour can be specified as follows:

\[ \psi_2 = \bigvee_{x \in K} F^a \left( \text{call(socket)} \land X^a(eax = x) \land F^a \left( \text{call(bind)} \land x \Gamma^* \land F^a \left( \text{call(listen)} \land x \Gamma^* \right) \right) \right) \]

where the \( \bigvee \) is taken over all possible memory addresses \( x \) over the domain \( K \) which stores the socket descriptors in the program.

Remember that the return value of an API function is put in \( eax \) when the function terminates. Thus, the return value of an API function is the value of \( eax \) at its return-point. Then, the subformula \( \text{call(socket)} \land X^a(eax = x) \) states that there is a call to the API function \( \text{socket} \) whose return value is \( x \) (since the return-point of a call is its abstract successor). When \( \text{bind} \) is invoked, one required parameter is the socket descriptor and this descriptor must be put on top of the stack (since parameters are passed via the stack in binary programs). The regular expression \( x \Gamma^* \) describes the requirement that \( x \) is on top of the stack. Then, the subformulas \( \text{call(bind)} \land x \Gamma^* \) and \( \text{call(listen)} \land x \Gamma^* \) state that there are calls to \( \text{bind} \) and \( \text{listen} \) whose socket descriptor is \( x \). Thus, \( \psi_2 \) expresses that there is a call to the API \( \text{socket} \) with a return value \( x \), followed by a call to the function \( \text{bind} \) and a call to the function \( \text{listen} \) with \( x \) on top of the stack. Note that in this case, \( x \) is the memory address storing the descriptor.

The main malicious behavior of \( \text{Thread3} \) is to contact several specific websites
to inform them about the infection or the IP number of the victim computer, ...
To do this, the malware needs to check if the Internet connection is up or not.
If yes, the malware sets up a connection to specific urls. For the first task,
the malware needs to invoke the API function `InternetGetConnectedState`
to obtain the Internet connection state. The second task is obtained by an
invocation to `InternetOpen` to open the connection, followed by a call to
`InternetOpenUrl` to contact specific urls. This malicious behaviour can be
specified by the following formula:
$$\psi_3 = F^g(\text{call(InternetGetConnectedState)} \land X^a(eax = 1) \land F^a(\text{call(InternetOpen)} \land F^a(\text{call(InternetOpenUrl)}))$$

The sub-formula `call(InternetGetConnectedState) \land X^a(eax = 1)` expresses
that a call to the API `InternetGetConnectedState` returns the value 1 (stored
in `eax`) which implies that the Internet connection is up. The sub-formula
`F^a(\text{call(InternetOpen)} \land F^a(\text{call(InternetOpenUrl)})` expresses that the mal-
ware first opens a connection (via API `InternetOpen`), then contacts specific
websites (by calling API `InternetOpenUrl`) to inform them about the infection.

The typical behavior of `Thread4` is to hunt for emails on local drives by
searching files matching certain conditions. As presented before, to do this,
the malware first calls the API function `FindFirstFileA` to obtain the first
matching file. `FindFirstFileA` will return a search handle `h`. To obtain all
matching files, the malware must continuously call the function `FindNextFileA`
with `h` as parameter. Similarly, this behaviour cannot be specified by LTL or
CTL since it requires that the return value of the API `FindFirstFileA` must
be used as the input of the function `FindNextFileA`. Using CARET, the above
behavior can be expressed by the following formula:
$$\psi_4 = \bigvee_{d \in K} F^g \left( \text{call(FindFirstFileA)} \land X^a(eax = d) \land F^a(\text{call(FindNextFileA) \land d} \Gamma^*) \right)$$
where the $\bigvee$ is taken over all possible memory addresses `d` over domain `K`
which contain the values of search handles `h` in the program.

Similarly, the return value of `FindFirstFileA` is the value of `eax` at its
corresponding return-point. Then, the subformula $F^a(\text{call(FindFirstFileA)} \land
X^a(eax = d))$ states that there is a call to the API `FindFirstFileA` and
the return value of this function is `d`. When `FindNextFileA` is invoked, it
requires a search handle as parameter. The requirement that `d` is on top of the
program stack is expressed by the regular expression $d \Gamma^*$. Thus, the subformula
`call(FindNextFileA) \land d \Gamma^*` expresses that `FindNextFileA` is called with `d` as
parameter ($d$ stores the information of the search handle). $\psi_4$ expresses then
that there is a call to the API `FindFirstFileA` with the return value `d` (the search handle), followed by a call to the function `FindNextFileA` with `d` on
the top of the stack.

Thus, the malicious behavior of the concurrent worm Bagle can be described
6.3. Single-indexed CARET model-checking for DPNs

In this section, we consider the CARET model-checking problem of DPNs. Let \( \lambda : \bigcup_{i=1}^{n} P_i \rightarrow 2^{AP} \) be a labeling function that associates each control location with a set of atomic propositions. Let \( M = \{P_1, ..., P_n\} \) be a DPN, \( f = \bigwedge_{i=1}^{n} f_i \) be a single-indexed CARET formula.

6.3.1 Büchi DPNs (BDPNs)

**Definition 19.** A Büchi DPDS (BDPDS) is a tuple \( B\mathcal{P}_i = (P_i, \Gamma_i, \Delta_i, F_i) \) s.t. \( P_i = (P_i, \Gamma_i, \Delta_i) \) is a DPDS, \( F_i \subseteq P_i \) is the set of accepting control locations. A run of a BDPDS is accepted iff it visits infinitely often some control locations in \( F_i \).

**Definition 20.** A Generalized Büchi DPDS (GBDPDS) is a tuple \( B\mathcal{P}_i = (P_i, \Gamma_i, \Delta_i, F_i) \), where \( P_i = (P_i, \Gamma_i, \Delta_i) \) is a DPDS and \( F_i = \{F_i^1, ..., F_i^k\} \) is a set of sets of accepting control locations. A run of a GBDPDS is accepted iff it visits infinitely often some control locations in \( F_i^j \) for every \( 1 \leq j \leq k \).

Given a BDPDS or a GBDPDS \( B\mathcal{P}_i \), let \( c \in P_i \times \Gamma_i^* \) be a local configuration of \( B\mathcal{P}_i \). Then, let \( L(B\mathcal{P}_i) \) be the set of all pairs \((c, D) \in P_i \times \Gamma_i^* \times 2^{D_i} \) s.t. \( B\mathcal{P}_i \) has an accepting run from \( c \) and \( D \) is the set of DCLICs generated during that run. We get the following properties:

**Proposition 6.** Given a GBDPDS \( B\mathcal{P}_i \), we can effectively compute a BDPDS \( B\mathcal{P}'_i \) s.t. \( L(B\mathcal{P}_i) = L(B\mathcal{P}'_i) \).

This result comes from the fact that we can translate a GBDPDS to a corresponding BDPDS by applying the similar approach as the translation from a Generalized Büchi automaton to a corresponding Büchi automaton [EMCP99].

**Definition 21.** A Büchi Dynamic Pushdown Network (BDPN) is a set \( \{B\mathcal{P}_1, ..., B\mathcal{P}_n\} \) s.t. for every \( 1 \leq i \leq n \), \( B\mathcal{P}_i = (P_i, \Gamma_i, \Delta_i, F_i) \) is a BDPDS. A (global) run \( \rho \) of a BDPN is accepted iff all local runs in \( \rho \) are accepting (local) runs.

**Definition 22.** A Generalized Büchi Dynamic Pushdown Network (GBDPN) is a set \( \{B\mathcal{P}_1, ..., B\mathcal{P}_n\} \) s.t. for every \( 1 \leq i \leq n \), \( B\mathcal{P}_i = (P_i, \Gamma_i, \Delta_i, F_i) \) is a GBDPDS. A (global) run \( \rho \) of a GBDPN is accepted iff all local runs in \( \rho \) are accepting (local) runs.
Given a BDPN or a GBDPN $BM = \{BP_1, ..., BP_n\}$, let $L(BM)$ be the set of all global configurations $G$ s.t. $BM$ has an accepting run from $G$. We get the following properties:

**Proposition 7.** Given a GBDPN $BM$, we can effectively compute a BDPN $BM'$ s.t. $L(BM) = L(BM')$.

This result is obtained due to the fact that we can translate each GBDPDS in $BM$ to a corresponding BDPDS in $BM'$.

Given a BDPN $BM = \{BP_1, ..., BP_n\}$ where $BP_i = (P_i, \Gamma, \Delta, F)$. Let $I(c)$ be the index $i$ of the local configuration $c \in P_i \times \Gamma^*$. Let $D = \bigcup_{i=1}^n D_i$. Then, we get the following theorem:

**Theorem 21.** [ST13b, ST16] The membership problem of a BDPN is decidable in time $O(\sum_{i=1}^n |\Delta_i| \cdot |\Gamma_i| \cdot |P_i|^3 \cdot 2^{|D_i|} + \Sigma_{c \in D} (|c| \cdot |P_{I(c)}| \cdot 2^{|P_{I(c)}|} + |D| \cdot 2^{|D|})$.

Thus, from Proposition 7 and Theorem 21, we get that the membership problem of a GBDPN is decidable.

**Theorem 22.** The membership problem of GBDPNs is decidable.

### 6.3.2 From CARET model checking of DPNs to the membership problem in BDPNs

Given a local run $\pi$, let $\vartheta(\pi)$ be the index of the DPDS corresponding to $\pi$. Let $G$ be an initial global configuration of the DPN $M$, then we say that $G$ satisfies $f$ iff $M$ has a global run $\rho$ starting from $G$ s.t. every local run $\pi$ in $\rho$ satisfies $f_{\vartheta(\pi)}$. Determining whether $G$ satisfies $f$ is a non-trivial problem since the number of global runs can be unbounded and the number of local runs of each global run can also be unbounded. Note that it is not sufficient to check whether every pushdown process $P_i$ satisfies the corresponding CARET formula $f_i$. Indeed, we need to ensure that all instances of $P_i$ created during a global run satisfy the formula $f_i$. Also, it is not correct to check whether all possible instances of $P_i$ satisfy the formula $f_i$. Indeed, an instance of $P_i$ should not be checked if it is not created during a global run. To solve these problems, we reduce the CARET model-checking problem for DPNs to the membership problem for GBDPNs. To do this, we compute a GBDPN $BM = \{BP_1, ..., BP_n\}$ where $BP_i$ ($i \in \{1..n\}$) is a GBDPDS s.t. (1) the problem of checking whether each instance of $P_i$ satisfies a CARET formula $f_i$ can be reduced to the membership problem of $BP_i$; (2) if $P_i$ creates a new instance of $P_j$ starting from $p_s \omega_s$, which requires that $p_s \omega_s \models f_j$, $BP_i$ must also create an instance of $BP_j$ starting from a certain configuration (computed
from $p_s\omega_s$) from which $\mathcal{BP}_j$ has an accepting run. In what follows, we present how to compute such GBDPDSs.

Let $Label = \{exit, unexit\}$ (we explain later the need to these labels). Given a DPDS $\mathcal{P}_i$ ($i \in \{1..n\}$), a corresponding CARET formula $f_i$, we define $Initial_i$ as the set of atoms $A$ ($A \in Atoms(f_i)$) such that $f_i \in A$ and $NextCallerFormulas(A) = \emptyset$. Our goal is that for every $\mathcal{P}_i$ ($i \in \{1..n\}$), we compute a GBDPDS $\mathcal{BP}_i$ s.t. for every $p\omega \in \mathcal{P}_i \times \Gamma_i^*$, $p\omega$ satisfies $f_i$ iff there exists an atom $A$ where $A \in Initial_i$ s.t. $\mathcal{BP}_i$ has an accepting run from $L_{p,A,unexit}M_s\omega_s$.

**GBDPDSs Computation.**

Let us fix a DPDS $\mathcal{P}_i = (P, \Gamma, \Delta)$ in the DPN $M$, a CARET formula $f_i$ in $f = \bigwedge_{i=1}^n f_i$ corresponding to the DPDS $\mathcal{P}_i$. In this section, we show how to compute such a GBDPDS $\mathcal{BP}_i$ corresponding to $\mathcal{P}_i$. Given a local configuration $p\omega$, let $\delta(p\omega)$ be the index of the DPDS corresponding to $p\omega$. We define $\mathcal{BP}_i = (P', \Gamma', \Delta', F)$ as follows:

- $P' = \{ (p, A, l) | p \in P, l \in Label, A \in Atoms(f_i) \text{ and } A \cap AP = \lambda(p) \}$ is the finite set of control locations of $\mathcal{BP}_i$
- $\Gamma' = \Gamma \cup (\Gamma \times Atoms(f_i) \times Label)$ is the finite set of stack symbols of $\mathcal{BP}_i$.

The transition relation $\Delta'$ of $\mathcal{BP}_i$ is the smallest set of transition rules satisfying the following:

- $(\alpha_1)$ for every $p\gamma \xrightarrow{\text{call}} q\gamma'\gamma'' \triangleright d \in \Delta$: $(p, A, l) \gamma \rightarrow_i (q, A', l') \gamma'\gamma'' (A, l) \triangleright d_0 \in \Delta'$ for every $A, A' \in Atoms(f_i); l, l' \in Label$ such that:
  - $(\beta_0)$ $A \cap \{\text{call, ret, int}\} = \{\text{call}\}$
  - $(\beta_1)$ $A \cap AP = \lambda(p)$
  - $(\beta_2)$ $A' \cap AP = \lambda(q)$
  - $(\beta_3)$ $\text{GlNext}(A, A')$
  - $(\beta_4)$ $\text{CallerNext}(A', A)$
  - $(\beta_5)$ $l' = \text{unexit}$ implies $(l = \text{unexit}$ and $\text{NextAbsForms}(A) = \emptyset)$
  - $(\beta_6)$ $d_0 = \square$ if $d = \square$; $d_0 = (p_s, A_0, \text{unexit})\omega_s$ where $A_0 \in Initial_{\delta(p_s\omega_s)}$ if $d = p_s\omega_s$

- $(\alpha_2)$ for every $p\gamma \xrightarrow{\text{ret}} q \triangleright d \in \Delta$:
Chapter 6. CARET analysis of multithreaded programs

- \((\alpha_{2.1})\) \(\{p, A, \text{exit}\} \gamma \xrightarrow{\tau_i} \{q, A', l'\} \varepsilon \triangleright d_0 \in \Delta'\) for every \(A, A' \in \text{Atoms}(f_i)\); \(l, l' \in \text{Label}\) such that:
  - \((\beta_0)\) \(A \cap \{\text{call, ret, int}\} = \{\text{int}\}\)
  - \((\beta_1)\) \(A \cap AP = \lambda(p)\)
  - \((\beta_2)\) \(A' \cap AP = \lambda(q)\)
  - \((\beta_3)\) \(\text{GlNext}(A, A')\)
  - \((\beta_4)\) \(\text{NexAbsForms}(A) = \emptyset\)
  - \((\beta_5)\) \(d_0 = \square\) if \(d = \square\); \(d_0 = \{p_s, A_0, \text{unexit}\} \omega_s\) where \(A_0 \in \text{Initial}_{\delta(p_s, \omega_s)}\) if \(d = p_s \omega_s\)

- \((\alpha_{2.2})\) \(\{q, A', l'\}, \{\gamma_0, A_0, l_0\} \xrightarrow{\tau_i} \{q, A', l'\} \gamma_0 \in \Delta'\) for every \(\gamma_0 \in \Gamma, A_0, A' \in \text{Atoms}(f_i)\); \(l', l_0 \in \text{Label}\) such that:
  - \((\beta_0)\) \(\text{AbsNext}(A_0, A')\)
  - \((\beta_1)\) \(\text{NexCallerForms}(A') = \text{NexCallerForms}(A_0)\)
  - \((\beta_2)\) \(A' \cap AP = \lambda(q)\)
  - \((\beta_3)\) \(l_0 = l'\)

- \((\alpha_3)\) for every \(p \gamma \xrightarrow{\tau_i} q \omega \triangleright d \in \Delta: \{p, A, l\} \gamma \xrightarrow{\tau_i} \{q, A', l\} \omega \triangleright d_0 \in \Delta'\) for every \(A, A' \in \text{Atoms}(f_i), l \in \text{Label}\) such that:
  - \((\beta_0)\) \(A \cap \{\text{call, ret, int}\} = \{\text{int}\}\)
  - \((\beta_1)\) \(A \cap AP = \lambda(p)\)
  - \((\beta_2)\) \(A' \cap AP = \lambda(q)\)
  - \((\beta_3)\) \(\text{GlNext}(A, A')\)
  - \((\beta_4)\) \(\text{AbsNext}(A, A')\)
  - \((\beta_5)\) \(\text{NexCallerForms}(A) = \text{NexCallerForms}(A')\)
  - \((\beta_6)\) \(d_0 = \square\) if \(d = \square\); \(d_0 = \{p_s, A_0, \text{unexit}\} \omega_s\) where \(A_0 \in \text{Initial}_{\delta(p_s, \omega_s)}\) if \(d = p_s \omega_s\)

Let \(cl_{US}(f_i) = \{\phi_1 U^g \chi_1, \ldots, \phi_k U^g \chi_k\}\) and \(cl_{US}(f_i) = \{\xi_1 U^a \tau_1, \ldots, \xi_k U^a \tau_k\}\) be the set of \(U^g\)-formulas and \(U^a\)-formulas of \(CI(f_i)\) respectively. The generalized Büchi accepting condition \(F\) of \(\mathcal{BP}_1\) is defined as: \(F = \{F_1\} \cup F_2 \cup F_3\) where

- \(F_1 = P \times \text{Atoms}(f_i) \times \{\text{unexit}\}\)

- \(F_2 = \{F_1^g, \ldots, F_k^g\}\) where \(F_x^g = \{P \times F_{\phi_x U^g \chi_x} \times \text{Label}\}\) where \(F_{\phi_x U^g \chi_x} = \{A \in \text{Atoms}(f_i) \mid \phi_x U^g \chi_x \in A\}\) for every \(1 \leq x \leq k\).
6.3. Single-indexed CARET model-checking for DPNs

- \( F_x = \{ F_{i,x}^0, ..., F_{i,x}^{k'} \} \) where \( F_{i,x}^o = \{ p \times F_{i,x} \times \{ \text{unexit} \} \} \) where \( F_{i,x} = \{ A \in \text{Atoms}(f_i) \mid \text{if } \xi_x \in A \text{ then } \tau_x \in A \} \) for every \( 1 \leq x \leq k' \).

A DPDS \( \mathcal{P}_i = (P, \Gamma, \Delta) \) can be seen as a Pushdown System (PDS) if there are no spawn rules in \( \Delta \). Similarly, a GBDPDS \( \mathcal{B} \mathcal{P}_i = (P', \Gamma', \Delta', F) \) can be seen as a Generalized Büchi Pushdown System (GBPDS) if there are no spawn rules in \( \Delta' \). Let \( \mathcal{B} \mathcal{P}_i = (P', \Gamma', \Delta', F) \) be a GBDPDS obtained by the above computation. Then, it is easy to see that if we don’t take into account spawn actions and conditions related to spawn actions, the way we obtain a GBDPDS \( \mathcal{B} \mathcal{P}_i \) from a DPDS \( \mathcal{P}_i \) is the same as the way we obtain a GBPDS \( \mathcal{B} \mathcal{P}_\psi \) from a PDS \( \mathcal{P} \) as presented in Section 2.3. Therefore, from Theorem 2 and the fact that \( \text{Initial}_i = \{ A \mid A \in \text{Atoms}(f_i), f_i \in A \text{ and } \text{NextCallerFormulas}(A) = \emptyset \} \), we obtain the following lemma:

**Lemma 9.** Given a DPDS \( \mathcal{P}_i = (P, \Gamma, \Delta) \), and a CARET formula \( f_i \), we can construct a GBDPDS \( \mathcal{B} \mathcal{P}_i = (P', \Gamma', \Delta', F) \) such that for every configuration \( p\omega \in P_i \times \Gamma' \), \( p\omega \models f_i \) iff there exists an atom \( A \in \text{Initial}_i \) s.t. \( \mathcal{B} \mathcal{P}_i \) has an accepting run from \( \langle p, A, \text{unexit} \rangle \omega \).

**Spawning new instances.** Lemma 9 guarantees that the problem of checking whether an instance of \( \mathcal{P}_i \) starting from \( p\omega \) satisfies \( f_i \) can be reduced to the problem of checking if \( \mathcal{B} \mathcal{P}_i \) has an accepting run from \( \langle p, A, \text{unexit} \rangle \omega \) where \( A \in \text{Initial}_i \). Now, we need to ensure the satisfiability on instances created dynamically. Suppose that \( \mathcal{P}_i \) spawns a new instance of \( \mathcal{P}_j \) starting from \( p_i\omega_i \), this means that we need to guarantee that \( p_i\omega_i \models f_j \). Note that by applying Lemma 9 for the DPDS \( \mathcal{P}_j \), we get that \( p_i\omega_i \models f_j \) iff there exists an atom \( A \in \text{Initial}_j \) s.t. \( \mathcal{B} \mathcal{P}_j \) has an accepting run from \( \langle p_i, A, \text{unexit} \rangle \omega_i \). Then, the requirement \( p_i\omega_i \models f_j \) is ensured by the conditions \( (\beta_0) \) in \( (\alpha_1) \), \( (\beta_0) \) in \( (\alpha_2) \) and \( (\beta_0) \) in \( (\alpha_3) \) stating that for every \( p\gamma \xrightarrow{t} q\omega \triangleright d \in \Delta \) (\( t \in \{ \text{call, ret, int} \}) \), we have \( \langle p, A, \emptyset \rangle \gamma \xrightarrow{t} \langle q, A', t' \rangle \omega \triangleright d_0 \in \Delta' \) such that if \( d = p_i\omega_i \), then, \( d_0 = \langle p_i, A_0, \text{unexit} \rangle \omega_i \) where \( A_0 \in \text{Initial}_j \) (since \( \delta(p_i\omega_i) = j \) in this case).

Given a global configuration \( G = \{ p_0\omega_0, ..., p_k\omega_k \} \), for every \( 0 \leq x \leq k \), let \( IC_x = \{ \langle p_x, A_x^o, \text{unexit} \rangle \omega_x \mid A_x^o \in \text{Initial}_i(p_x) \} \) be the set of initial configurations of the Büchi pushdown process \( \mathcal{B} \mathcal{P}_{\xi}(p_x\omega_x) \) such that \( p_x\omega_x \models f_{\xi}(p_x\omega_x) \) iff there exist a configuration \( \langle p_x, A_x^o, \text{unexit} \rangle \omega_x \in IC_x \) from which \( \mathcal{B} \mathcal{P}_{\xi}(p_x\omega_x) \) has an accepting run. Let \( GC = IC_0 \times ... \times IC_k \). We can show that:

**Theorem 23.** Given a DPN \( \mathcal{M} = \{ \mathcal{P}_1, ..., \mathcal{P}_n \} \), a single-indexed CARET formula \( f = \bigwedge_{i=1}^n f_i \), we can compute a GBDPN \( \mathcal{B} \mathcal{M} = \{ \mathcal{B} \mathcal{P}_1, ..., \mathcal{B} \mathcal{P}_n \} \) such that the global configuration \( G \) of \( \mathcal{M} \) satisfies \( f \) iff \( \exists G' \in GC \) s.t. \( G' \in \mathcal{L}(\mathcal{B} \mathcal{M}) \).
6.4 Single-indexed CARET model-checking for DPNs with regular valuations

In this section, we consider the single-indexed CARET model-checking problem for DPNs with regular valuations, in which the set of configurations where an atomic proposition is satisfied is a regular language.

Definition 23. Let $\mathcal{M} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ be a DPN. For every $i \in \{1..n\}$, a set of configurations of a pushdown process $\mathcal{P}_i = (P_i, \Delta_i, \Gamma_i)$ is regular if it can be written as the union of sets of the form $E_p$, where $p \in P_i$ and $E_p = \{(p, w) | w \in L_p\}$, where $L_p$ is a regular set over $\Gamma_i^*$. 

Definition 24. Let $\mathcal{M} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ be a DPN. Let $\mathcal{AP}$ be a finite set of atomic propositions. Let $\nu : \mathcal{AP} \rightarrow 2^{\bigcup_{i=1}^{n} P_i \times \Gamma_i^*}$ be a valuation. $\nu$ is called regular if for every $e \in \mathcal{AP}$, $\nu(e)$ is a regular set of configurations.

Let $\nu : \mathcal{AP} \rightarrow 2^{\bigcup_{i=1}^{n} P_i \times \Gamma_i^*}$ be a regular valuation. We define $\lambda_{\nu} : P \times \Gamma^* \rightarrow 2^{\mathcal{AP}}$ such that $\lambda_{\nu}(p \omega) = \{e \in \mathcal{AP} \mid p \omega \in \nu(e)\}$. Let $\pi = p_0 \omega_0 p_1 \omega_1 \ldots$ be a local path of $\mathcal{P}_i$. We associate each configuration $p_x \omega_x$ of $\pi$ with a tag $t_x$ in \{call, int, ret\} as presented in Section 6.1.2. Let $f_i$ be a CARET formula over $\mathcal{AP}$. The satisfiability relation w.r.t. the regular valuation $\nu$ is defined as follows:

$$\pi \models_{\nu} f_i \iff (\lambda_{\nu}(p_0 \omega_0), t_0)(\lambda_{\nu}(p_1 \omega_1), t_1) \cdots \models f_i$$

Theorem 24. [ST13b] Single-indexed LTL model-checking with regular valuations for DPNs can be reduced to standard LTL model checking for DPNs.

Given a DPN $\mathcal{M} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ and a regular valuation $\nu : \mathcal{AP} \rightarrow 2^{\bigcup_{i=1}^{n} P_i \times \Gamma_i^*}$, this result is based on translating every DPDS $\mathcal{P}_i$ ($i \in \{1..n\}$) into a DPDS $\mathcal{P}_i' = (P_i, \Gamma_i', \Delta_i')$ where the regular valuation requirements are encoded in $\Gamma_i'$. The same reduction is still true for single-indexed CARET with regular valuations. For details about this reduction, we refer readers to [ST13b]. Therefore, given a single-indexed CARET formula with regular valuations and a DPN $\mathcal{M} = \{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$, we apply the reduction of [ST13b] to obtain a new DPN $\mathcal{M}' = \{\mathcal{P}_1', \ldots, \mathcal{P}_n'\}$ in which for every $i \in \{1..n\}$, $\mathcal{P}_i' = (P_i, \Gamma_i', \Delta_i')$ s.t. the satisfiability of a CARET formula over $\mathcal{P}_i'$ w.r.t. the regular valuation $\nu$ can be reduced to the satisfiability of a CARET formula over $\mathcal{P}_i'$ with simple valuations. Therefore, we can show that:

Theorem 25. Single-indexed CARET model-checking with regular valuations for DPNs can be reduced to standard single-indexed CARET model checking for DPNs.
6.5 DPNs Communicating via Locks

Dynamic Pushdown Network with Locks (L-DPNs) is a natural formalism for multithreaded programs communicating via locks [LMW09, ST16]:

**Definition 25.** A Dynamic Pushdown Network with Locks (L-DPN) $\mathcal{M}$ is a set $\{\mathbb{L}, \text{Act}, \mathcal{P}_1, ..., \mathcal{P}_n\}$ where $\mathbb{L}$ is a set of locks, $\text{Act} = \{\text{acq}(l), \text{rel}(l), \tau \mid l \in \mathbb{L}\}$ is a set of actions on locks s.t. $\text{acq}(l)$ (resp. $\text{rel}(l)$) for $l \in \mathbb{L}$ represents an acquisition (resp. release) of the lock $l$ and the action $\tau$ describes internal actions (neither acquire nor release locks); for every $1 \leq i \leq n$, $\mathcal{P}_i = (P, \Gamma_i, \Delta_i)$ is a Labelled Dynamic Pushdown System with Locks (L-DPDS), where $P$ is a finite set of control locations and $P_i \cap P_j = \emptyset$ for all $j \neq i$, $\Gamma_i$ is a finite set of stack alphabets, and $\Delta_i$ is a finite set of transitions rules. Rules of $\Delta_i$ are of the following form, where $a \in \text{Act}$, $p, p_1 \in P_i, \gamma \in \Gamma_i, \omega_1 \in \Gamma_i^*$, $d \in \{\Box, p\omega_1 \mid p\omega_1 \in \bigcup_{1 \leq j \leq n} P_j \times \Gamma_j^*\}$:

- $(r_1) p \gamma \xrightarrow{(a, \text{call})} p_1 \gamma_1 \gamma_2 \triangleright d$
- $(r_2) p \gamma \xrightarrow{(a, \text{ret})} p_1 \varepsilon \triangleright d$
- $(r_3) p \gamma \xrightarrow{(a, \text{int})} p_1 \omega \triangleright d$

Intuitively, a L-DPN is a DPN where processes communicate via locks. The transition rules of L-DPNs are similar to DPNs where each rule is associated with an element in the set $\{\text{call}, \text{ret}, \text{int}\}$ to denote whether the rule corresponds to a call, ret or a simple statement (neither call nor ret). The difference is that each transition rule of L-DPNs is assigned to one additional action $a \in \text{Act}$. Depending on the nature of the associated action $a$, each transition step of L-DPDSs include one additional operation on a given lock $l$. $\text{acq}(l)$ (resp. $\text{rel}(l)$) represents an acquisition (resp. release) of the lock $l$ and the action $\tau$ describe internal actions (neither acquire nor release locks).

A local configuration of an instance of a L-DPDS $\mathcal{P}_i$ is a tuple $(p\omega, L)$ where $p \in P_i$ is the control location, $\omega \in \Gamma_i^*$ is the stack content and $L \subseteq \mathbb{L}$ is a set of locks owned by the instance. A global configuration of $\mathcal{M}$ is a multiset over $\bigcup_{1 \leq i \leq n} P_i \times \Gamma_i^* \times 2^\mathbb{L}$, in which $(p\omega, L) \in P_i \times \Gamma_i^* \times 2^\mathbb{L}$ represents the local configuration of an instance of a pushdown process $\mathcal{P}_i$ which is running in the network.

A L-DPDS $\mathcal{P}_i$ defines a transition relation $\Rightarrow_i$ as follows where $t \in \{\text{call}, \text{ret}, \text{int}\}$:

- if $p \gamma \xrightarrow{(\tau, t)} p_1 \omega_1 \triangleright d$ then $(p \gamma \omega, L) \Rightarrow_i (p_1 \omega_1 \omega, L) \triangleright D_0$ where $D_0 = \emptyset$ if $d = \Box$, $D_0 = \{(p_s \omega_s, \emptyset)\}$ if $d = p_s \omega_s$ for every $\omega \in \Gamma_i^*$, $L \subseteq \mathbb{L}$
Theorem 26. \[ST16]\Single-indexed LTL model-checking for L-DPNs can be reduced to single-indexed LTL model checking for DPNs.

Given a L-DPN \( M = \{ P_1, ..., P_n \} \), this result is based on translating every \( P_i \) into a DPDS \( P'_i = (P'_i, \Gamma'_i, \Delta'_i) \) s.t. \( P'_i \) is a kind of product between the DPDS \( P_i \) and the acquisition structure, where an acquisition structure (encoded in control locations of \( P'_i \)) stores information about how locks are used such as the number of held locks, the order of acquisition and release of locks. We can compute a DPN \( M' = \{ P'_1, ..., P'_n \} \) s.t. the global runs of \( M' \) mimic the global runs of \( M \) and the acquisition structures reflect the lock usages. Thus, the global runs of \( M' \) correspond to global runs of \( M \) in which the locks are accessed in a nested manner. The same reduction is still true for single-indexed CARET formulas. For details of this reduction, we refer readers to [ST16]. Therefore, given a single-indexed CARET formula and a L-DPN \( M = \{ P_1, ..., P_n \} \), we apply the reduction of [ST16] to obtain a DPN \( M' = \{ P'_1, ..., P'_n \} \) s.t. the satisfiability of a CARET formula over \( P_i \)
can be reduced to the satisfiability of a CARET formula over $\mathcal{P}_1'$. Therefore, we can show that:

**Theorem 27.** Single-indexed CARET model-checking for $L$-DPNs can be reduced to single-indexed CARET model checking for DPNs.

### 6.6 Related Works

[BET03, CCK+06, ABT08, AT09] considered Pushdown networks with communications between processes. However, these works consider only networks with a fixed number of threads. The model-checking problem for pushdown networks where synchronization between threads is ensured by a set of nested locks is considered in [KIG05, KG06, KG07] for single-indexed LTL/CTL and double-indexed LTL. These works do not handle dynamic thread creation.

Multi-pushdown systems were considered in [LN12, BD13] to represent multithreaded programs. [BD12] introduced Multi-CaRet, an extension of CARET dedicated for multi-pushdown systems, and considered Multi-CaRet model-checking for multi-pushdown systems with some bounds. These systems have only a finite number of stacks, and thus, they cannot handle dynamic thread creation.

Pushdown Networks with dynamic thread creation (DPNs) were introduced in [BMT05]. The reachability problems of DPNs and its extensions are considered in [BMT05, GLM+11, LMW09, Lug11, Wen10]. [ST13b] considers the model-checking problem of DPNs against single-indexed LTL and CTL, while [ST16] investigates the single-indexed LTL model checking problem for DPNs with locks.

### 6.7 Conclusion

In this chapter, we present an algorithm for single-indexed CARET model-checking for DPNs. We reduce this problem to the membership problem for Büchi Dynamic Pushdown Networks. In addition, we show that single-indexed CARET model checking for $L$-DPNs with nested lock access can be reduced to single-indexed CARET model checking for DPNs.
Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we presented several temporal logics that take into account matching of calls and returns and we proposed model-checking algorithms for Pushdown Systems against these logics. We showed how these logics can be applied to specify malicious behaviors and applied our model-checking algorithms to malware detection.

In Chapter 2, we proposed an algorithm to model-check PDSs against CARET formulas. We reduced this problem to the emptiness problem of Büchi Pushdown Systems. This latter problem is already solved in [BEM97, ES01]. We reduced the malware detection problem to the CARET model-checking problem for PDSs. We showed that by using CARET formulas, we can describe behaviors that LTL, CTL and their extensions cannot describe.

In Chapter 3, we defined the new logic SPCARET and showed how it can be used to succinctly and precisely describe different malicious behaviors. We identified the sublogic SPCARET$^{\nabla c}$, which is a subclass of SPCARET that does not use the caller operator. We proposed an algorithm to model-check PDSs against SPCARET$^{\nabla c}$ formulas. We reduced malware detection to the SPCARET$^{\nabla c}$ model-checking problem for PDSs. Our algorithms are based on reducing the model checking problem to the emptiness problem of Symbolic Büchi Pushdown Systems. This makes our algorithms more efficient.

In Chapter 4, we defined the new logic BCARET which allows to describe branching-time properties that require matchings of calls and returns and showed how it can be used to describe branching-time malicious behaviors. We proposed an algorithm to model-check PDSs against BCARET formulas. Our algorithms are based on reducing the model checking problem to the emptiness problem of Alternating Büchi Pushdown Systems.

In Chapter 5, we defined the new logic SBPCARET and showed how it can be used to succinctly and precisely describe branching-time malicious behaviors. We proposed an algorithm to model-check PDSs for SBPCARET formulas. Our algorithm is based on reducing the model checking problem to the emptiness problem of Symbolic Alternating Büchi Pushdown Systems.

In Chapter 6, we proposed to use Dynamic Pushdown Networks to model concurrent binary programs and single-indexed CARET formulas to describe
malicious behaviors. We reduced the concurrent malware detection problem to the single-indexed CARET model-checking problem for DPNs. We showed that the latter problem is decidable. We reduced the problem of checking whether DPNs satisfy single-indexed CARET formulas to the membership problem for Büchi Dynamic Pushdown Networks (BDPNs). In addition, we showed that single-indexed CARET model checking is decidable for Dynamic Pushdown Networks communicating via nested locks.

### 7.2 Future Work

The results presented in this thesis can be extended in several directions as follows:

**Performance Improvement of the tools.** In the algorithms proposed in Chapters 2 and 3, we must consider all possible atoms of CARET formulas and all possible sub-formula subsets of SPCARET formulas when computing the Büchi Pushdown Systems. The number of atoms and sub-formula subsets is huge. To improve the efficiency of our tools, we plan to propose algorithms to compute only needed atoms/subsets in an effective manner. The idea is to determine "reachable" atoms/subsets from a CARET/SPCARET formula.

**Extensions of the Tools.** Currently, the input of our tools described in Chapter 2 and 3 is pushdown systems and binary codes. Our plan is to extend our tools to be able to take boolean, C/C++ and Java programs as input. Indeed, sequential boolean programs can naturally be translated to equivalent symbolic PDSs. To obtain equivalent boolean programs for C/C++ programs, we can for example use Satabs [CKSY05]. For Java programs, we can use JimpleToPDSolver [HO10] to retrieve its corresponding PDSs.

**Model-Checking Tools for BCARET, SBPCARET and for DPNs.** In this thesis, we defined the new logics BCARET, SBPCARET and proposed model-checking algorithms for these logics against PDSs. In addition, we considered single-indexed CARET model-checking for DPNs. These algorithms were not implemented. Our plan is to implement these algorithms in tools and apply them for malware detection.

**Model-Checking DPNs for double-indexed properties.** In Chapter 6, we considered single-indexed CARET model-checking for DPNs and DPNs with nested locks access. The model-checking problem for DPNs against double-indexed properties is undecidable in general. Therefore, our plan is to determine fragments of double-indexed CARET that are decidable for DPNs and DPNs with nested lock access.
7.2. Future Work

**Android Malware Detection.** In this thesis, we only considered Windows malwares. Since the number of android malware is increasing significantly in recent years, we plan to extend our tools for Android malware detection.

**BCARET for DPNs.** In this thesis, we considered CARET model-checking for DPNs. We believe that BCARET model-checking for DPNs is also decidable. We plan to investigate this problem in the future.
Bibliography


[CKSY05] Edmund M. Clarke, Daniel Kroening, Natasha Sharygina, and Karen Yorav. SATABS: sat-based predicate abstraction for ANSI-


