Model checking self modifying code

Xin Ye

▶ To cite this version:

Xin Ye. Model checking self modifying code. Logic in Computer Science [cs.LO]. Université Paris Cité; East China normal university (Shanghai), 2019. English. NNT : 2019UNIP7010 . tel-02972592

HAL Id: tel-02972592
https://theses.hal.science/tel-02972592

Submitted on 20 Oct 2020

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.

Université de Paris

En cotutelle avec

East China Normal University

École Doctorale de Sciences Mathématiques de Paris-Centre (ED 386)

Laboratoire d'informatique de Paris Nord

Model Checking Self Modifying Code

Par Xin Ye

Thèse de doctorat en informatique

Dirigée par Tayssir Touili et Jifeng He

Présentée et soutenue publiquement à Villetaneuse le 30/09/2019

Guillaume Bonfante : Maître de Conférences HDR, LORIA , Université de Lorraine, rapporteur

Jifeng He : Professeur, East China Normal University, co-directeur de thèse

Laure Petrucci : Professeur, LIPN, Université Paris 13, Présidente du jury

Mihaela Sighireanu : Maître de Conférences HDR, IRIF, Université Paris Diderot, examinatrice

Jean-Marc Talbot : Professeur, Université d'Aix-Marseille, rapporteur

Tayssir Touili : Directrice de recherche, CNRS, LIPN, Université Paris 13, directrice de thèse

Valérie Viet Triem Tong: Professeur, CentraleSupelec, examinatrice

Title : Model Checking Self Modifying Code

Abstract :
A Self modifying code is code that modifies its own instructions during execution time. It is nowadays widely used, especially in malware to make the code hard to analyse and to detect by anti-viruses. Thus, the analysis of such self modifying programs is a big challenge. Pushdown Systems (PDSs) is a natural model that is extensively used for the analysis of sequential programs because it allows to accurately model procedure calls and mimic the program's stack. In this thesis, we propose to extend the PushDown System model with self-modifying rules. We call the new model Self-Modifying PushDown System (SM-PDS). A SM-PDS is a PDS that can modify its own set of transitions during execution. First, we show how SM-PDSs can be used to naturally represent self-modifying programs and provide efficient algorithms to compute the backward and forward reachable configurations of SM-PDSs. Then, we consider the LTL model-checking problem of self-modifying code. We reduce this problem to the emptiness problem of Self-modifying Büchi Pushdown Systems (SM-BPDSs). We also consider the CTL model-checking problem of self-modifying code. We reduce this problem to the emptiness problem of Self-modifying Alternating Büchi Pushdown Systems (SM-ABPDSs). We implement our techniques in a tool called SMODIC. We obtained encouraging results. In particular, our tool was able to detect several self-modifying malwares; it could even detect several malwares that well-known anti-viruses such as McAfee, Norman, BitDefender, Kinsoft, Avira, eScan, Kaspersky, Qihoo-360, Avast and Symantec failed to detect.

Keywords : Self-modifying Code, Model-checking, Pushdown System, Malware Detection, LTL, CTL, Reachability Analysis, Binary Code.

Titre : Vérification de Code Auto-modifiant

Résumé : Le code auto-modifiant est un code qui modifie ses propres instructions pendant le temps d'exécution. Il est aujourd'hui largement utilisé, notamment dans les logiciels malveillants pour rendre le code difficile à analyser et être détecté par les anti-virus. Ainsi, l'analyse de tels programmes auto-modifiants est un grand défi. Pushdown System(PDSs) est un modèle naturel qui est largement utilisé pour l'analyse des programmes séquentiels car il permet de modéliser précisément les appels de procédures et de simuler la pile du programme. Dans cette thèse, nous proposons d'étendre le modèle du PDS avec des règles auto-modifiantes. Nous appelons le nouveau modèle Self-Modifying PushDown System (SM- PDS). Un SM-PDS est un PDS qui peut modifier l'ensemble des règles de transitions pendant l'exécution. Tout d'abord, nous montrons comment les SM-PDS peuvent être utilisés pour représenter des programmes auto-modifiants et nous fournissons des algorithmes efficaces pour calculer les configurations accessibles des SM-PDSs. Ensuite, nous résolvons le problème de vérification de propriétés LTL et CTL pour le code auto-modifiant. Nous implémentons nos techniques dans un outil appelé SMODIC. Nous avons obtenu des résultats encourageants. En particulier, notre outil est capable de détecter plusieurs logiciels malveillants auto-modifiants ; il peut même détecter plusieurs logiciels malveillants que les autres logiciels anti-virus bien connus comme McAfee, Norman, BitDefender, Kinsoft, Avira, eScan, Kaspersky, Qihoo-360, Avast et Symantec n'ont pas pu détecter.

Mots clefs : Code auto-modifiant, Vérification, Pushdown System, Détection de Malware, l'analyse de l'accessibilité, code binaire, LTL, CTL.

Résumé Détaillé

Un code auto-modifiant est un code qui modifie ses propres instructions pendant le temps d'exécution. Il est aujourd'hui largement utilisé, principalement pour rendre les programmes difficiles à comprendre. Par exemple, le code auto-modifiant est largement utilisé pour protéger la propriété intellectuelle des logiciels, car il permet d'inverser du code. Il est également abondamment utilisé par les auteurs de malwares afin d'obscurcir leur code malveillant et de le rendre difficile à analyser par les analyseurs statiques et les anti-virus. Il existe plusieurs types d'implémentations pour les codes auto-modifiants. Packing consiste à appliquer des techniques de compression pour réduire la taille du fichier exécutable. Ceci convertit le fichier exécutable en une forme où le contenu exécutable est caché. Ensuite, le code est "déballé" au moment de l'exécution. Un tel code emballé est auto-modifiant. Le chiffrement est une autre technique pour cacher le code. Il utilise une sorte d'opérations inversibles pour cacher le code exécutable à l'aide d'une clé de cryptage. Ensuite, le code est "décrypté" au moment de l'exécution. Les programmes cryptés sont auto-modifiants. Ces deux formes de codes auto-modifiants ont été bien étudiés dans la littérature et pourraient être traitées par plusieurs outils de décryptage.

Dans cette thèse, nous considérons un autre type de code auto-modifiant, causé par des instructions auto-modifiantes, où le code est traité comme des données qui peuvent donc être lues et écrites par des instructions auto-modifiantes. Ces instructions auto-modifiantes sont généralement des instructions de mov, puisque le mov peut accéder à la mémoire, la lire et y écrire. Pour ce faire, nous devons d'abord trouver un modéle adéquat pour de tels programmes. PushDown Systems (PDSs) est connu pour être un modèle naturel pour les programmes séquentiels, car il permet de suivre les contextes des différents appels dans le programme. De plus, les systèmes PushDown permettent d'enregistrer et d'imiter la pile du programme, ce qui est très important pour la détection des malwares. En effet, pour vérifier si un programme est malveillant, les anti-virus commencent par identifier les appels qu'il fait aux fonctions API. Pour échapper à ces contrôles, les auteurs de malwares essaient d'obscurcir les appels qu'ils font au

système d'exploitation en utilisant des pushs et des sauts. Il est donc important de pouvoir suivre la pile pour détecter de tels appels obscurs. C'est pourquoi les systèmes PushDown ont été utilisés pour modéliser des programmes binaires afin de détecter les malwares. Cependant, ces travaux ne tiennent pas compte des malwares qui utilisent du code auto-modifiant, car les systèmes PushDown ne sont pas capables de modéliser des instructions auto-modifiantes.

Pour surmonter cette limitation, nous proposons dans cette thèse d'étendre le modéle du système PushDown avec des règles auto-modifiantes. Nous appelons ce nouveau modèle le système SM-PDS (Self-Modifying PushDown System). En gros, un SM-PDS est un PDS qui peut modifier son propre ensemble de transitions pendant l'exécution. Nous montrons comment les SM-PDS peuvent être utilisés pour représenter naturellement des programmes auto-modifiants. Il s'avère que les SM-PDS sont équivalents aux PDS standards. Nous montrons comment traduire un SM-PDS en PDS standard. Cette traduction est exponentielle. Par conséquent, il n'est pas efficace d'effectuer l'analyse de vérification du modèle sur le PDS équivalent. Nous proposons donc dans cette thèse des algorithmes directs pour les SM-PDS.

Analyse de l'accessibilité du code auto-modifiant

Tout d'abord, nous considérons le problème de l'accessibilité. Nous proposons des algorithmes directs pour calculer les ensembles d'accessibilité avant ($post^*$) et arrière (pre^*) pour les SM-PDS. Ceci permet d'effectuer efficacement l'analyse d'accessibilité pour les programmes auto-modifiants. Nos algorithmes sont basés sur (1) la représentation d'ensembles réguliers (potentiellement infinis) de configurations de SM-PDS en utilisant des automates à états finis, et (2) l'application de procédures de saturation sur les automates à états finis afin de prendre en compte l'effet de l'application des règles du SM-PDS. Ces résultats ont été publiés dans ICECCS 2017.

Vérification de propriétés LTL pour le code auto-modifiant

Au chapitre 3, nous proposons un algorithme direct de vérification de propriétés LTL pour les SM-PDS. Notre algorithme est basé sur la réduction du problème de vérification de propriétés LTL au problème du vide de Büchi Pushdown Systems auto-modifiants(SM-BPDSs). Intuitivement, on obtient ce SM-BPDS en prenant le produit du SM-PDS avec un automate de Büchi acceptant une formule LTL φ. Ensuite, on résout le problème du vide d'un SM-BPDS en calculant ses repeating heads. Ce calcul est basé sur le calcul de configurations labellisées pre^* en appliquant une procédure de saturation sur des automates finis labellisés. Ces résultats sont publiés dans ICECCS 2019.

Vérification de propriétés CTL pour le code auto-modifiant

Au chapitre 4, nous examinons le problème de vérification de propriétés CTL pour les SM-PDS. Ceci permet de détecter les comportements malveillants de type CTL sur du code auto-modifiant. Nous réduisons ce problème au problème de la vérification du vide de Self-modifying Alternating Büchi Pushdown Systems (SM-ABPDSs), et nous proposons un algorithme qui calcule un automate fini qui caractérise l'ensemble des configurations acceptées par les SM-ABPDS.

SMODIC : un outil d'analyse de code auto-modifiant

Nous avons implémenté nos techniques dans un outil d'analyse de code auto-modifiant appelé SMODIC. Nous avons utilisé avec succès SMODIC pour modéliser et vérifier plus de 900 codes binaires auto-modifiants. En particulier, nous avons appliqué SMODIC pour la détection des logiciels malveillants, puisque les logiciels malveillants utilisent généralement des instructions auto-modifiantes, et que les comportements malicieux peuvent être décrits par des formules LTL ou CTL. Dans nos expériences, SMODIC a pu détecter 895 malwares et prouver que 19 programmes bénins étaient bénins. SMODIC a également été capable de détecter plusieurs logiciels malveillants que des antivirus connus tels que Bit-Defender, Kinsoft, Avira, eScan, Kaspersky, Avast et Symantec n'ont pas détectés. SMODIC

peut être trouvé à `https://lipn.univ-paris13.fr/~xin/smodic/index.html`.

Acknowledgements

It has been an unforgettable and extraordinary experience of my Ph.D. years at LIPN, not only from the scientific perspective, but also from personal willpower. Here, I would like to take the opportunity to sincerely acknowledge all the people who helped and supported me in different ways. I would not have done it without them. First of all, I sincerely thank my two supervisors, Prof. Tayssir Touili and Prof. Jifeng He for giving me the opportunity to write this thesis. Prof. Tayssir Touili is also an excellent teacher. Thanks so much for her excellent and patient support and supervision throughout these four-year Ph.D. study. Without her detailed comments, this thesis cannot be completed. I would like to thank Prof. Jifeng He, my supervisor in China. His attitude in research inspired me a lot. Without his support, I cannot finish my thesis. Their attitude and devotion to the science encourages me all the time, and will continue inspiring me and be a role model for me in the rest of our lives.

My special thanks go to Prof. Guillaume Bonfante and Prof. Jean-Marc Talbot who kindly agreed to be my thesis referees. I would like to thank them for the valuable remarks.

I would like to thank Prof. Laure Petrucci and Prof. Valérie Viet Triem Tong who kindly agreed to be the members of my jury. My special thanks to Prof. Mihaela Sighireanu. I would like to thank her not only for accepting to be a member of my defense jury, but also for her help when I arrived in Paris to start my Ph.D study. Without her help, I might have no place to stay for my first year in Paris.

I am very grateful to many staff members of the Laboratoire d'Informatique de Paris Nord (LIPN) for their logistics help. I would like to thank

Ms. Amina Hariti, the secretary of the doctoral school for the help with the paper work.

Many thanks to all my labmates both in LIPN and Shanghai for being there, for sharing good and bad times, for the encouragements.

I also would like to take it as an honor to express my thanks to my friends for their supportive influences in my life all the way. I thank them very much for coming to my life and helping me in any way.

Finally I would like to express my deep sense of gratitude to my ever loving big family. My parents, aunts, uncles and cousins have given me numerous encouragement and support. I would like to thank my dearest mother Yuying and father Jinshou, who give me unconditional support, love, and encouragement in my life. Thank you so much, my dearest families.

Contents

1 Introduction **1**
1.1 Reachability Analysis of Self Modifying Code 3
1.2 LTL Model Checking of Self Modifying Code 3
1.3 CTL Model Checking of Self Modifying Code 4
1.4 SMODIC: A Model Checker for Self Modifying Code 4
1.5 Related Works 4
1.6 Thesis Organization 6

2 Reachability Analysis of Self Modifying Code **7**
2.1 An Example of A Self-modifying Code 7
2.2 Self Modifying Pushdown Systems 9
2.2.1 Definition 9
2.2.2 From SM-PDSs to PDSs 10
2.2.3 From SM-PDSs to Symbolic PDSs 12
2.3 Modeling Self-modifying Code with SM-PDSs 14
2.3.1 Self-modifying Instructions 14
2.3.2 From Self-modifying Code to SM-PDS 15
2.4 Representing Infinite Sets of Configurations of a SM-PDS 15
2.5 Efficient Computation of pre^* images 16
2.5.1 Proof of Theorem 2.5.1 21
2.6 Efficient Computation of $post^*$ Images 25
2.6.1 Proof of Theorem 2.6.1 30
2.7 Experiments 35
2.7.1 Our Algorithms vs. Standard pre^* and $post^*$ Algorithms of PDSs 35
2.7.2 Malware Detection 38

3 LTL Model-Checking of Self-modifying Code 41
3.1 LTL Model-Checking of SM-PDSs 41
3.1.1 The linear-time temporal logic LTL 41
3.1.2 Self Modifying Büchi Pushdown Systems 42
3.1.3 From LTL Model-Checking of SM-PDSs to the emptiness problem of SM-BPDSs 43
3.2 The Emptiness Problem of SM-BPDSs 44
3.2.1 The Head Reachability Graph $\mathcal{G}$ 45
3.2.2 Labelled configurations and labelled $\mathcal{BP}$-automata 51
3.2.3 Computing $pre^*\big((\langle p', \epsilon \rangle, \theta')\big)$ 52
3.2.4 Computing the Head Reachability Graph $\mathcal{G}$ 58
3.3 Experiments 59
3.3.1 Our approach vs. standard LTL for PDSs 59
3.3.2 Malicious Behavior Detection on Self-Modifying Code . . . 61
3.3.2.1 Specifying Malicious Behaviors using LTL. 61
3.3.2.2 Applying our tool for malware detection. 63

4 CTL Model-Checking of Self-modifying Code 71
4.1 CTL Model-Checking of SM-PDSs 71
4.1.1 The Computation Tree Logic CTL 71
4.1.2 Self-modifying Alternating Büchi Pushdown Systems . . . 72
4.1.3 From CTL Model-Checking of SM-PDSs to the emptiness problem of SM-ABPDSs 74
4.2 Computing the language of a SM-ABPDS 89
4.2.1 Characterizing $L(\mathcal{BP})$ 89
4.2.2 Computing $Y_{\mathcal{BP}}$ 91
4.3 Experiments 104
4.3.1 Our algorithm vs. standard CTL on PDSs 104
4.3.2 Malicious Behavior Detection on Self-Modifying Code . . . 105
4.3.2.1 Specifying malicious behaviors using CTL. 105
4.3.2.2 Applying our tool for malware detection. 107

5 SMODIC: A Model Checker for Self Modifying Code 111
5.1 Architecture 112
5.2 Experiments 113

5.2.1 Analysing Self-modifying Binary Code 113
5.2.2 Comparison with Well-known Anti-viruses 113
5.3 Description of SMODIC 113
5.3.1 Reachability Analysis in SMODIC 115
5.3.2 LTL and CTL in SMODIC 116
5.4 Applying SMODIC for Malware Detection 118

6 Conclusion **121**
6.1 Summary 121
6.2 Future Work 122

References **125**

1

Introduction

Self-modifying code is code that modifies its own instructions during execution time. It is nowadays widely used, mainly to make programs hard to understand. For example, self-modifying code is extensively used to protect software intellectual property, since it makes reverse code engineering harder. It is also abundantly used by malware writers in order to obfuscate their malicious code and make it hard to analyse by static analysers and anti-viruses.

There are several kinds of implementations for self-modifying codes. **Packing** [36] consists in applying compression techniques to make the size of the executable file smaller. This converts the executable file to a form where the executable content is hidden. Then, the code is "unpacked" at runtime before execution. Such packed code is self-modifying. **Encryption** is another technique to hide the code. It uses some kind of invertible operations to hide the executable code with an encryption key. Then, the code is "decrypted" at runtime prior to execution. Encrypted programs are self-modifying. These two forms of self-modifying codes have been well studied in the litterature and could be handled by several unpacking tools such as [43, 45].

In this thesis, we consider another kind of self-modifying code, caused by **self-modifying instructions**, where code is treated as data that can thus be read and written by **self-modifying instructions**. These self-modifying instructions are usually **mov** instructions, since **mov** can access memory, and read and write to it. For example, consider the program shown in Figure 1.1. For simplification matters, we suppose that the addresses' length is 1 byte. The binary code is given in the left side, while in the right side, we give its corresponding assembly

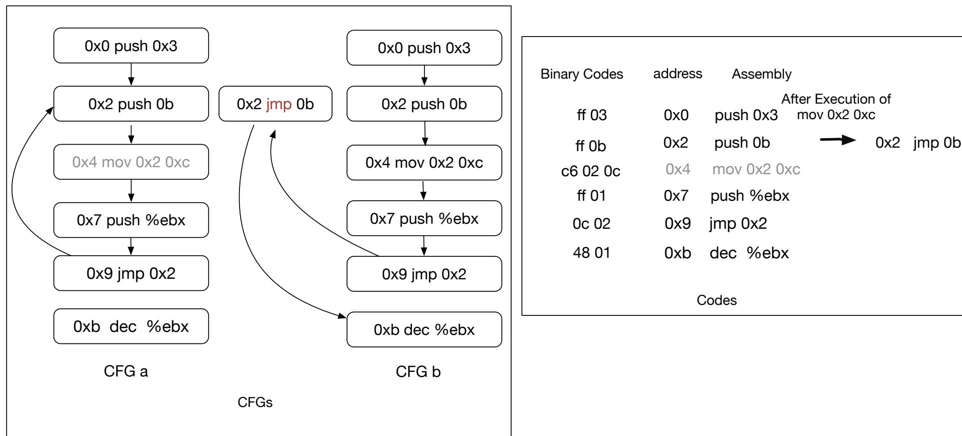

Figure 1.1: A Simple Example of Self-modifying Codes

code obtained by translating syntactically the binary code at each address. For example, `ff` is the binary code of the instruction `push`, thus, the first line is translated to `push 0x3`, the second line to `push 0b`, etc. Let us execute this code. First, we execute `push 0x3`, then `push 0b`, then `mov 0x2 0xc`. This last instruction will replace the first byte at address `0x2` by `0xc`. Thus, at address `0x2`, `ff 0b` is replaced by `0c 0b`. Since `0c` is the binary code of `jmp`, this means the instruction `push 0b` is replaced by `jmp 0xb`. Therefore, this code is self-modifying. If we treat it blindly, without looking at the semantics of the different instructions, we will extract from it the Control Flow Graph `CFG a`, whereas its correct Control Flow Graph is `CFG b`. You can see that the **mov** instruction was able to modify the instructions of the program successfully via its ability to read and write the memory.

In this thesis, we consider the analysis of self-modifying programs where the code is modified by **mov** instructions. To this aim, we first need to find an adequate model for such programs. PushDown Systems (PDSs) is known to be a natural model for sequential programs [42], as it allows to track the contexts of the different calls in the program. Moreover, PushDown Systems allow to record and mimic the program's stack, which is very important for malware detection. Indeed, to check whether a program is malicious, anti-viruses start by identifying the calls it makes to the API functions. To evade these checks, malware writers try to obfuscate the calls they make to the Operating System by using pushes and jumps. Thus, it is important to be able to track the stack to detect such

obfuscated calls. This is why PushDown Systems were used in [13, 14] to model binary programs in order to perform malware detection. However, these works do not consider malwares that use self-modifying code, as PushDown Systems are not able to model self-modifying instructions.

To overcome this limitation, we propose in this thesis to extend the PushDown System model with self-modifying rules. We call the new model Self-Modifying PushDown System (SM-PDS). Roughly speaking, a SM-PDS is a PDS that can modify its own set of transitions during execution. We show how SM-PDSs can be used to naturally represent self-modifying programs. It turns out that SM-PDSs are equivalent to standard PDSs. We show how to translate a SM-PDS to a standard PDS. This translation is exponential. Thus, performing the model-checking analysis on the equivalent PDS is not efficient. We propose then in this thesis *direct* model-checking algorithms for SM-PDSs.

1.1 Reachability Analysis of Self Modifying Code

First, we consider the reachability problem. We propose direct algorithms to compute the forward ($post^*$) and backward (pre^*) reachability sets for SM-PDSs. This allows to efficiently perform reachability analysis for self-modifying programs. Our algorithms are based on (1) representing regular (potentially infinite) sets of configurations of SM-PDSs using finite state automata, and (2) applying saturation procedures on the finite state automata in order to take into account the effect of applying the rules of the SM-PDS. These results were published in [46]. They are described in Chapter 2.

1.2 LTL Model Checking of Self Modifying Code

In Chapter 3, we propose a **direct** LTL model checking algorithm for SM-PDSs. Our algorithm is based on reducing the LTL model checking problem to the emptiness problem of Self Modifying Büchi Pushdown Systems (SM-BPDSs). Intuitively, we obtain this SM-BPDS by taking the product of the SM-PDS with a Büchi automaton accepting an LTL formula φ. Then, we solve the emptiness problem of a SM-BPDS by computing its repeating heads. This computation is

based on computing labelled pre^* configurations by applying a saturation procedure on labelled finite automata. These results are published in [47].

1.3 CTL Model Checking of Self Modifying Code

In Chapter 4, we consider the CTL model-checking problem for SM-PDSs. This allows to detect CTL-like malicious behaviors on self-modifying code. We reduce this problem to the emptiness checking problem of Self-modifying Alternating Büchi Pushdown Systems (SM-ABPDSs), and we propose an algorithm that computes a finite automaton that characterizes the set of configurations accepted by the SM-ABPDS.

1.4 SMODIC: A Model Checker for Self Modifying Code

We implemented our techniques in a tool for self-modifying code analysis called SMODIC. We successfully used SMODIC to model-check more than 900 self-modifying binary codes. In particular, we applied SMODIC for malware detection, since malwares usually use self-modifying instructions, and since malicious behaviors can be described by LTL or CTL formulas. In our experiments, SMODIC was able to detect 895 malwares and to prove that 19 benign programs were benign. SMODIC was also able to detect several malwares that well-known antiviruses such as Bit-Defender, Kinsoft, Avira, eScan, Kaspersky, Avast, and Symantec failed to detect. SMODIC can be found in

`https://lipn.univ-paris13.fr/~xin/smodic/index.html`.

1.5 Related Works

Reachability analysis and LTL/CTL model-checking of pushdown systems was considered e.g. in [6, 20, 39, 42]. Our algorithms are extensions of these works.

Model checking and static analysis approaches have been widely used to analyze binary programs, for instance, in [7, 12, 14, 15, 19, 21, 29, 29, 33]. These works cannot handle self-modifying code.

Cai et al. [18] use a Hoare-logic-style framework to describe self-modifying code by applying local reasoning and separation logic, and treating program code uniformly as regular data structure. However, [18] requires programs to be manually annotated with invariants. In [36], the authors describe a formal semantics for self-modifying codes, and use that semantics to represent self-unpacking code. This work only deals with packing and unpacking behaviours, it cannot capture self-modifying instructions as we do. In [16], Bonfante et al. provide an operational semantics for self-modifying programs and show that they can be constructively rewritten to a non-modifying program. All these specifications [16, 18, 36] are too abstract to be used in practice.

In [1], the authors propose a new representation of self-modifying code named State Enhanced-Control Flow Graph (SE-CFG). SE-CFG extends standard control flow graphs with a new data structure, keeping track of the possible states programs can reach, and with edges that can be conditional on the state of the target memory location. It is not easy to analyse a binary program only using its SE-CFG, especially that this representation does not allow to take into account the stack of the program.

[34] propose abstract interpretation techniques to compute an over-approximation of the set of reachable states of a self-modifying program, where for each control point of the program, an over-approximation of the memory state at this control point is provided. [28] combine static and dynamic analysis techniques to analyse self-modifying programs. Unlike our self-modifying pushdown systems, these techniques [28, 34] cannot handle the program's stack.

Unpacking binary code is considered in [23, 27, 32, 36]. These works do not consider self-modifying **mov** instructions.

There are a lot of tools that can deal with binary code analysis [2, 3, 4, 8, 9, 10, 11, 13, 14, 22, 24, 25, 26, 37, 38, 44]. POMMADE [13, 14] is a malware detector based on LTL and CTL model-checking of PDSs. STAMAD [24, 25, 26] is a malware detector based on PDSs and machine learning. However, all these tools cannot handle self-modifying code. The only tools that we know of and that can deal with self-modifying code are BE-PUM [17] and CoDisasm [5].

BE-PUM (Binary Emulation for PUshdown Model) [17] focuses on generating CFG (Control Flow Graph) of malwares. BE-PUM can construct a pushdown model from x86 binaries in an on-the-fly manner. Concolic testing is applied to

determine the precise destinations of branches for indirect jumps. This tool can deal with self-modifying code caused by modifying the destinations of indirect jumps, including overwriting the return address of a function (in the stack). But it cannot handle self-modifying instructions.

CoDisasm [5] is a tool that focuses on the disassembly of x86 code that includes self-modifying instructions and code overlapping. CoDisasm deals only with disassembling the code. It does not consider model-checking problems of code. Currently, we use Jakstab [22] to disassemble binary code. CoDisasm might help our disassembly process and make it more precise. We plan to use CoDisasm in the future (instead of Jakstab) and see whether it will improve the precision of our extracted CFGs.

1.6 Thesis Organization

In Chapter 2, we give the definition of SM-PDS and show how a SM-PDS can describe self-modifying codes. We also present our direct algorithms for reachability analysis of SM-PDSs. Chapter 3 shows how to reduce the LTL model checking problem of SM-PDSs to the emptiness problem of self-modifying büchi pushdown systems. We tackle the CTL model checking problem on SM-PDSs in Chapter 4. Chapter 5 presents the tool SMODIC that implements our algorithms.

2

Reachability Analysis of Self Modifying Code

A Self modifying code is code that modifies its own instructions during execution time. It is nowadays widely used, especially in malware to make the code hard to analyse and to detect by anti-viruses. Thus, the analysis of such self modifying programs is a big challenge. Pushdown Systems (PDSs) is a natural model that is extensively used for the analysis of sequential programs because it allows to accurately model procedure calls and mimic the program's stack. In this chapter, we propose to extend the PushDown System model with self-modifying rules. We call the new model Self-Modifying PushDown System (SM-PDS). A SM-PDS is a PDS that can modify its own set of transitions during execution. We show how SM-PDSs can be used to naturally represent self-modifying programs and provide efficient algorithms to compute the backward and forward reachable configurations of SM-PDSs. We implemented our techniques in a tool and obtained encouraging results. In particular, we successfully applied our tool for the detection of self-modifying malware.

2.1 An Example of A Self-modifying Code

Fig. 2.1 shows how malware can use self-modifying instructions to evade from static analysis techniques. This figure shows a fragment of the malware Bagle.J equipped with such self-modifying instructions. First let us recall the semantics

of the **mov** instruction. It copies the data item referred to by its second operand (register or memory location) into its first operand. In Fig. 2.1, in the box on the left, we give, respectively, the binary code, the addresses of the different instructions, and the corresponding assembly code, obtained by translating syntactically the binary code at each address. For example, `ff` is the binary code of the instruction `push`. Thus, the first line is translated to `push 0b`. The second instruction `mov 0x2 0xc` will replace the first byte at address `0x2` by `0xc`. Thus, at address `0x2`, `ff 0b` is replaced by `0c 0b`, i.e., the instruction `push 0b` is replaced by `jmp 0b`. If we analyse this code without taking into account the fact that `mov 0x2 0xc` is a self-modifying instruction, then, we will obtain the Control Flow Graph "CFG a", and we will reach the conclusion that the Bagle malicious behaviour implemented at address `0b` by the API functions RegCreateKeyA, RegDeleteValueA, and RegCloseKey is not reachable. However, the actual CFG is "CFG b", where the malicious fragment of the malware Bagle.J that starts at address `0b` is reached and will be executed.

It can be seen from this example that self-modifying codes can make malware detection harder, and that the **mov** instruction is able to modify instructions of the program successfully via its ability to read and write the memory. Thus, it is crucial to be able to analyse this kind of self-modifying code.

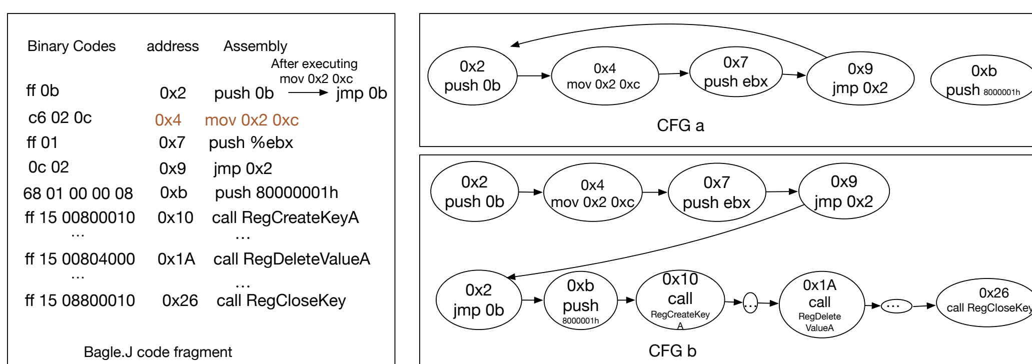

Figure 2.1: An Example of Self-modifying Code

2.2 Self Modifying Pushdown Systems

2.2.1 Definition

We introduce in this section our new model: Self-modifying Pushdown Systems.

Definition 1 *A Self-modifying Pushdown System (SM-PDS) is a tuple* $\mathcal{P} = (P, \Gamma, \Delta, \Delta_c)$*, where* P *is a finite set of control points,* Γ *is a finite set of stack symbols,* $\Delta \subseteq (P \times \Gamma) \times (P \times \Gamma^*)$ *is a finite set of transition rules, and* $\Delta_c \subseteq P \times (\Delta \cup \Delta_c) \times (\Delta \cup \Delta_c) \times P$ *is a finite set of modifying transition rules. If* $((p,\gamma),(p',w)) \in \Delta$*, we also write* $\langle p,\gamma\rangle \hookrightarrow \langle p',w\rangle \in \Delta$*. If* $(p, r_1, r_2, p') \in \Delta_c$*, we also write* $p \xhookrightarrow{(r_1,r_2)} p' \in \Delta_c$*. A Pushdown System (PDS) is a SM-PDS where* $\Delta_c = \emptyset$*.*

Intuitively, a Self-modifying Pushdown System is a Pushdown System that can dynamically modify its set of rules during the execution time: rules Δ are standard PDS transition rules, while rules Δ_c modify the current set of transition rules: $\langle p,\gamma\rangle \hookrightarrow \langle p',w\rangle \in \Delta$ expresses that if the SM-PDS is in control point p and has γ on top of its stack, then it can move to control point p', pop γ and push w onto the stack, while $p \xhookrightarrow{(r_1,r_2)} p' \in \Delta_c$ expresses that when the PDS is in control point p, then it can move to control point p', remove the rule r_1 from its current set of transition rules, and add the rule r_2. Formally, a configuration of a SM-PDS is a tuple $c = (\langle p,w\rangle, \theta)$ where $p \in P$ is the control point, $w \in \Gamma^*$ is the stack content, and $\theta \subseteq \Delta \cup \Delta_c$ is the current set of transition rules of the SM-PDS. θ is called the current **phase** of the SM-PDS. When the SM-PDS is a PDS, i.e., when $\Delta_c = \emptyset$, a configuration is a tuple $c = (\langle p,w\rangle, \Delta)$, since there is no changing rule, so there is only one possible phase. In this case, we can also write $c = \langle p,w\rangle$. Let $\mathcal{C}$ be the set of configurations of a SM-PDS. A SM-PDS defines a transition relation $\Rightarrow_{\mathcal{P}}$ between configurations as follows: Let $c = (\langle p,w\rangle, \theta)$ be a configuration, and let r be a rule in θ, then:

1. if $r \in \Delta_c$ is of the form $r = p \xhookrightarrow{(r_1,r_2)} p'$, such that $r_1 \in \theta$, then $(\langle p,w\rangle, \theta) \Rightarrow_{\mathcal{P}} (\langle p',w\rangle, \theta')$, where $\theta' = (\theta \setminus \{r_1\}) \cup \{r_2\}$. In other words, the transition rule r updates the current set of transition rules θ by removing r_1 from it and adding r_2 to it.

2. if $r \in \Delta$ is of the form $r = \langle p, \gamma \rangle \hookrightarrow \langle p', w' \rangle \in \Delta$, then $(\langle p, \gamma w \rangle, \theta) \Rightarrow_{\mathcal{P}} (\langle p', w'w \rangle, \theta)$. In other words, the transition rule r moves the control point from p to p', pops γ from the stack and pushes w' onto the stack. This transition keeps the current set of transition rules θ unchanged.

Let $\Rightarrow_{\mathcal{P}}^*$ be the transitive, reflexive closure of $\Rightarrow_{\mathcal{P}}$. We define $\stackrel{i}{\Rightarrow}$ as follows: $c \stackrel{i}{\Rightarrow} c'$ iff there exists a sequence of configurations $c_0 \Rightarrow_{\mathcal{P}} c_1 \Rightarrow_{\mathcal{P}} ... \Rightarrow_{\mathcal{P}} c_i$ s.t. $c_0 = c$ and $c_i = c'$ Given a configuration c, the set of immediate predecessors (resp. successors) of c is $pre_{\mathcal{P}}(c) = \{c' \in \mathcal{C} \ : \ c' \Rightarrow_{\mathcal{P}} c\}$ (resp. $post_{\mathcal{P}}(c) = \{c' \in \mathcal{C} \ : \ c \Rightarrow_{\mathcal{P}} c'\}$). These notations can be generalized straightforwardly to sets of configurations. Let $pre_{\mathcal{P}}^*$ (resp. $post_{\mathcal{P}}^*$) denote the reflexive-transitive closure of $pre_{\mathcal{P}}$ (resp. $post_{\mathcal{P}}$). We omit the subscript $\mathcal{P}$ when it is understood from the context.

Example 1 *Let $\mathcal{P} = (P, \Gamma, \Delta, \Delta_c)$ be a SM-PDS where $p = \{p_1, p_2, p_3, p_4\}$, $\Gamma = \{\gamma_1, \gamma_2, \gamma_3\}$, $\Delta = \{r_1 : \langle p_1, \gamma_1 \rangle \hookrightarrow \langle p_2, \gamma_2\gamma_1 \rangle, r_2 : \langle p_2, \gamma_2 \rangle \hookrightarrow \langle p_3, \epsilon \rangle, r_3 : \langle p_4, \gamma_1 \rangle \hookrightarrow \langle p_2, \gamma_2\gamma_3 \rangle\}$, $\Delta_c = \{r' : p_3 \xhookrightarrow{(r_1, r_3)} p_4\}$. Let $c_0 = (\langle p_1, \gamma_1\gamma_1 \rangle, \theta_0)$ where $\theta_0 = \{r_1, r_2, r'\}$. Applying rule r_1, we get $(\langle p_1, \gamma_1\gamma_1 \rangle, \theta_0) \Rightarrow_{\mathcal{P}} (\langle p_2, \gamma_2\gamma_1\gamma_1 \rangle, \theta_0)$. Then, applying rule r_2, we get $(\langle p_2, \gamma_2\gamma_1\gamma_1 \rangle, \theta_0) \Rightarrow_{\mathcal{P}} (\langle p_3, \gamma_1\gamma_1 \rangle, \theta_0)$. Then, applying rule r', we get $(\langle p_3, \gamma_1\gamma_1 \rangle, \theta_0) \Rightarrow_{\mathcal{P}} (\langle p_4, \gamma_1\gamma_1 \rangle, \theta_1)$ where r' is self-modifying, thus, it leads the SM-PDS from phase $\theta_0 = \{r_1, r_2, r'\}$ to phase $\theta_1 = \theta_0 \setminus \{r_1\} \cup \{r_3\} = \{r_2, r_3, r'\}$. Then, applying rule r_3, we get $(\langle p_4, \gamma_1\gamma_1 \rangle, \theta_1) \Rightarrow_{\mathcal{P}} (\langle p_2, \gamma_2\gamma_3\gamma_1 \rangle, \theta_1)$. Then, applying rule r_2 again, we get $(\langle p_2, \gamma_2\gamma_3\gamma_1 \rangle, \theta_1) \Rightarrow_{\mathcal{P}} (\langle p_3, \gamma_3\gamma_1 \rangle, \theta_1)$.*

2.2.2 From SM-PDSs to PDSs

A SM-PDS can be described by a PDS. This is due to the fact that the number of phases is finite, thus, we can encode phases in the control points of the PDS: Let $\mathcal{P} = (P, \Gamma, \Delta, \Delta_c)$ be a SM-PDS, we compute the PDS $\mathcal{P}' = (P', \Gamma, \Delta')$ as follows: $P' = P \times 2^{\Delta \cup \Delta_c}$. Initially, $\Delta' = \emptyset$. For every $\theta \in 2^{\Delta \cup \Delta_c}$, $r \in \theta$:

1. If $r = \langle p, \gamma \rangle \hookrightarrow \langle p', w \rangle \in \Delta \cap \theta$, we add $\langle (p, \theta), \gamma \rangle \hookrightarrow \langle (p', \theta), w \rangle$ to Δ'

2. if $r = p \xhookrightarrow{(r_1, r_2)} p' \in \Delta_c \cap \theta$, then for every $\gamma \in \Gamma$, we add $\langle (p, \theta), \gamma \rangle \hookrightarrow \langle (p', \theta'), \gamma \rangle$ to Δ', where $\theta' = (\theta \setminus \{r_1\}) \cup \{r_2\}$.

It is easy to see that:

Proposition 1 $(\langle p, w\rangle, \theta) \Rightarrow_{\mathcal{P}} (\langle p', w'\rangle, \theta')$ *iff* $\langle (p, \theta), w\rangle \Rightarrow_{\mathcal{P}'} \langle (p', \theta'), w'\rangle$.

Proof:
$\Rightarrow$: We will show that if $(\langle p, w\rangle, \theta) \Rightarrow_{\mathcal{P}} (\langle p', w'\rangle, \theta')$, then we have $\langle (p, \theta), w\rangle \Rightarrow_{\mathcal{P}'} \langle (p', \theta'), w'\rangle$. There are two cases depending on the form of the rule that led to this transition.

- Case $\theta = \theta'$: it means that the transition does not correspond to a self-modifying transition rule. Thus there is a rule $r \in \theta$ of the form $r = \langle p, \gamma\rangle \hookrightarrow \langle p', u'\rangle$ that led to this transition. Let u be such that $w = \gamma u, w' = u'u$. By the construction rule of the PDS $\mathcal{P}'$, we have $\langle (p, \theta), \gamma\rangle \hookrightarrow \langle (p', \theta), u'\rangle \in \Delta'$. Therefore, $\langle (p, \theta), \gamma u\rangle \Rightarrow_{\mathcal{P}'} \langle (p', \theta), u'u\rangle$ holds. This implies that $\langle (p, \theta), w\rangle \Rightarrow_{\mathcal{P}'} \langle (p', \theta), w'\rangle$.

- Case $\theta \neq \theta'$: it means that the transition corresponds to a self-modifying transition rule. Thus there is a rule $r \in \theta$ of the form $p \xhookrightarrow{(r_1, r_2)} p'$ that led to this transition. Let u be such that $w = \gamma u, w' = \gamma u$. By the construction rule of the PDS $\mathcal{P}'$, we have $\langle (p, \theta), \gamma\rangle \hookrightarrow \langle (p', \theta'), \gamma\rangle \in \Delta'$ where $\theta' = (\theta \backslash \{r_1\}) \cup \{r_2\}$. Therefore, $\langle (p, \theta), \gamma u\rangle \Rightarrow_{\mathcal{P}'} \langle (p', \theta'), \gamma u\rangle$ holds. This implies that $\langle (p, \theta), w\rangle \Rightarrow_{\mathcal{P}'} \langle (p', \theta'), w'\rangle$.

$\Leftarrow$: We will show that if $\langle (p, \theta), w\rangle \Rightarrow_{\mathcal{P}'} \langle (p', \theta'), w'\rangle$, then $(\langle p, w\rangle, \theta) \Rightarrow_{\mathcal{P}} (\langle p', w'\rangle, \theta')$. Let $\gamma \in \Gamma, u, u' \in \Gamma^*$ be such that $w = \gamma u, w' = u'u$. There are two cases.

- Case $\theta = \theta'$. Let $r = \langle (p, \theta), \gamma\rangle \hookrightarrow \langle (p', \theta), u'\rangle \in \Delta'$ be the rule that led to the transition. By the construction of PDS $\mathcal{P}'$, there must exist a rule $r \in \theta$ such that $r = \langle p, \gamma\rangle \hookrightarrow \langle p', u'\rangle$. Therefore, $(\langle p, \gamma u\rangle, \theta) \Rightarrow_{\mathcal{P}} (\langle p, u'u\rangle, \theta)$ holds. This implies that $(\langle p, w\rangle, \theta) \Rightarrow_{\mathcal{P}} (\langle p, w'\rangle, \theta')$.

- Case $\theta \neq \theta'$. Let $r = \langle (p, \theta), \gamma\rangle \hookrightarrow \langle (p', \theta'), \gamma\rangle \in \Delta'$ be the rule leading to the transition and $u' = \gamma$. By the construction of PDS $\mathcal{P}'$, there must exist a rule $r \in \theta$ such that $r = p \xhookrightarrow{(r_1, r_2)} p'$ where $\theta' = (\theta \backslash \{r_1\}) \cup \{r_2\}$. Therefore, $(\langle p, \gamma u\rangle, \theta) \Rightarrow_{\mathcal{P}} (\langle p', \gamma u\rangle, \theta')$ holds. This implies that $(\langle p, w\rangle, \theta) \Rightarrow_{\mathcal{P}} (\langle p, w'\rangle, \theta')$.

$\square$

Thus, we get:

Theorem 2.2.1 *Let $\mathcal{P} = (P, \Gamma, \Delta, \Delta_c)$ be a SM-PDS, we can compute an equivalent PDS $\mathcal{P}' = (P', \Gamma, \Delta')$ such that $|\Delta'| = \big(|\Delta| + |\Delta_c| \cdot |\Gamma|\big) \cdot 2^{\mathcal{O}(|\Delta|+|\Delta_c|)}$ and $|P'| = |P| \cdot 2^{\mathcal{O}(|\Delta|+|\Delta_c|)}$.*

2.2.3 From SM-PDSs to Symbolic PDSs

Instead of recording the phases θ of the SM-PDS in the control points of the equivalent PDS, we can have a more compact translation from SM-PDSs to **symbolic** PDSs [42], where each SM-PDS rule is represented by a **single, symbolic** transition, where the different values of the phases are encoded in a symbolic way using relations between phases:

Definition 2 *A symbolic pushdown system is a tuple $\mathcal{P} = (P, \Gamma, \delta)$, where P is a set of control points, Γ is the stack alphabet, and δ is a set of symbolic rules of the form: $\langle p, \gamma \rangle \xhookrightarrow{R} \langle p', w \rangle$, where $R \subseteq 2^{\Delta \cup \Delta_c} \times 2^{\Delta \cup \Delta_c}$ is a relation.*

A symbolic PDS defines a transition relation $\rightsquigarrow_{\mathcal{P}}$ between SM-PDS configurations as follows: Let $c = (\langle p, \gamma w' \rangle, \theta)$ be a configuration and let $\langle p, \gamma \rangle \xhookrightarrow{R} \langle p', w \rangle$ be a rule in δ, then: $(\langle p, \gamma w' \rangle, \theta) \rightsquigarrow_{\mathcal{P}} (\langle p', ww' \rangle, \theta')$ for $(\theta, \theta') \in R$. Let $\rightsquigarrow_{\mathcal{P}}^{*}$ be the transitive, reflexive closure of $\rightsquigarrow_{\mathcal{P}}$. Then, given a SM-PDS $\mathcal{P} = (P, \Gamma, \Delta, \Delta_c)$, we can compute an equivalent symbolic PDS $\mathcal{P}' = (P, \Gamma, \Delta')$ such that: Initially, $\Delta' = \emptyset$;

- For every $\langle p, \gamma \rangle \hookrightarrow \langle p', w \rangle \in \Delta$, add $\langle p, \gamma \rangle \xhookrightarrow{R_{id}} \langle p', w \rangle$ to Δ', where R_{id} is the identity relation.
- For every $r = p \xhookrightarrow{(r_1, r_2)} p' \in \Delta_c$ and every $\gamma \in \Gamma$, add $\langle p, \gamma \rangle \xhookrightarrow{R} \langle p', \gamma \rangle$ to Δ', where $R = \{(\theta_1, \theta_2) \in 2^{\Delta \cup \Delta_c} \times 2^{\Delta \cup \Delta_c} \mid r \in \theta_1$ and $\theta_2 = (\theta_1 \setminus \{r_1\}) \cup \{r_2\}\}$.

It is easy to see that:

Proposition 2 $(\langle p, w \rangle, \theta) \Rightarrow_{\mathcal{P}} (\langle p', w' \rangle, \theta')$ *iff* $(\langle p, w \rangle, \theta) \rightsquigarrow_{\mathcal{P}'} (\langle p', w' \rangle, \theta')$.

Proof:
$\Rightarrow$: we will show that if $(\langle p, w \rangle, \theta) \Rightarrow_{\mathcal{P}} (\langle p', w' \rangle, \theta')$, then $(\langle p, w \rangle, \theta) \rightsquigarrow_{\mathcal{P}'} (\langle p', w' \rangle, \theta')$. There are two cases depending on the form of the rule that led to this transition.

- Case $\theta = \theta'$, it means that the transition does not correspond to a self-modifying transition rule. Thus there is a rule $r \in \theta$ of the form $r = \langle p, \gamma \rangle \hookrightarrow \langle p', u' \rangle$ that led to this transition. Let u be such that $w = \gamma u, w' = u'u$. By construction of the symbolic pushdown system $\mathcal{P}'$, $\langle p, \gamma \rangle \xhookrightarrow{R_{id}} \langle p', u' \rangle \in \Delta'$, therefore, $(\langle p, \gamma u \rangle, \theta) \rightsquigarrow_{\mathcal{P}'} (\langle p', u'u \rangle, \theta)$ holds. This implies that $(\langle p, w \rangle, \theta) \rightsquigarrow_{\mathcal{P}'} (\langle p', w' \rangle, \theta')$.

- Case $\theta \neq \theta'$, it means that the transition corresponds to a self-modifying transition rule. Thus there is a rule $r \in \theta$ of the form $r = p \xhookrightarrow{(r_1,r_2)} p'$ that led to this transition and $\theta' = (\theta \backslash \{r_1\}) \cup \{r_2\}$. Let u be such that $w = \gamma u, w' = \gamma u$. By construction of the symbolic pushdown system $\mathcal{P}'$, $\langle p, \gamma \rangle \xhookrightarrow{R} \langle p', \gamma \rangle \in \Delta'$ and $R = \{(\theta, \theta') \in 2^{\Delta \cup \Delta_c} \times 2^{\Delta \cup \Delta_c} \mid r \in \theta \text{ and } \theta' = (\theta \setminus \{r_1\}) \cup \{r_2\}\}$, therefore, $(\langle p, \gamma u \rangle, \theta) \rightsquigarrow_{\mathcal{P}'} (\langle p', \gamma u \rangle, \theta')$ holds. This implies that $(\langle p, w \rangle, \theta) \rightsquigarrow_{\mathcal{P}'} (\langle p', w' \rangle, \theta')$.

$\Leftarrow$: we will show that if $(\langle p, w \rangle, \theta) \rightsquigarrow_{\mathcal{P}'} (\langle p', w' \rangle, \theta')$, then $(\langle p, w \rangle, \theta) \Rightarrow_{\mathcal{P}} (\langle p', w' \rangle, \theta')$. Let $\gamma \in \Gamma, u, u' \in \Gamma^*$ be such that $w = \gamma u, w' = u'u$. There are two cases.

- Case $\theta = \theta'$. Let $\langle p, \gamma \rangle \xhookrightarrow{R_{id}} \langle p', u' \rangle \in \Delta'$ be the rule applied to this transition. By the construction of the symbolic pushdown system $\mathcal{P}'$, there must exist a rule $r \in \theta$ s.t. $r = \langle p, \gamma \rangle \hookrightarrow \langle p', u' \rangle \in \Delta$. Therefore, $(\langle p, \gamma u \rangle, \theta) \Rightarrow_{\mathcal{P}} (\langle p', u'u \rangle, \theta)$ holds. This implies that $(\langle p, w \rangle, \theta) \Rightarrow_{\mathcal{P}} (\langle p', w' \rangle, \theta')$.

- Case $\theta \neq \theta'$. Let $\langle p, \gamma \rangle \xhookrightarrow{R} \langle p', \gamma \rangle \in \Delta'$ be the rule applied to this transition with $w' = \gamma u$. By the construction of the symbolic pushdown system $\mathcal{P}'$, there must exist a rule $r \in \theta$ of the form $r = p \xhookrightarrow{(r_1,r_2)} p' \in \Delta_c$ s.t. $R = \{(\theta_1, \theta_2) \in 2^{\Delta \cup \Delta_c} \times 2^{\Delta \cup \Delta_c} \mid r \in \theta_1 \text{ and } \theta_2 = (\theta_1 \setminus \{r_1\}) \cup \{r_2\}\}$. Therefore, $\theta' = (\theta \backslash \{r_1\}) \cup \{r_2\}$ and $(\langle p, \gamma u \rangle, \theta) \rightsquigarrow_{\mathcal{P}'} (\langle p', \gamma u \rangle, \theta')$ hold. This implies that $(\langle p, w \rangle, \theta) \rightsquigarrow_{\mathcal{P}'} (\langle p', w' \rangle, \theta')$.

$\square$

Thus, we get:

Theorem 2.2.2 *Let $\mathcal{P} = (P, \Gamma, \Delta, \Delta_c)$ be a SM-PDS, we can compute an equivalent symbolic PDS $\mathcal{P}' = (P', \Gamma, \Delta')$ such that $|P'| = |P|$, $|\Delta'| = |\Delta| + |\Delta_c| \cdot |\Gamma|$, and the size of the relations used in the symbolic transitions is $2^{\mathcal{O}(|\Delta| + |\Delta_c|)}$.*

2.3 Modeling Self-modifying Code with SM-PDSs

2.3.1 Self-modifying Instructions

There are different techniques to implement self-modifying code. We consider in this work code that uses self-modifying instructions. These are instructions that can access the memory locations and write onto them, thus changing the instructions that are in these memory locations. In assembly, the only instructions that can do this are the **mov** instructions. In this case, the self-modifying instructions are of the form `mov` l v, where l is a location of the program that stores executable data and v is a value. This instruction replaces the value at location l (in the binary code) with the value v. This means if at location l there is a binary value v' that is involved in an assembly instruction i_1, and if by replacing v' by v, we obtain a new assembly instruction i_2, then the instruction i_1 is replaced by i_2. E.g., `ff` is the binary code of `push`, `40` is the binary code of `inc`, `0c` is the binary code of `jmp`, `c6` is the binary code of `mov`, etc. Thus, if we have `mov` l `ff`, and if at location l there was initially the value 40 01 (which corresponds to the assembly instruction inc %edx), then 40 is replaced by `ff`, which means the instruction `inc %edx` is replaced by `push 01`. If at location l there was initially the value c6 01 02 (which corresponds to the assembly instruction `mov edx 0x2`), then c6 is replaced by `ff`, which means the instruction `mov edx 0x2` is replaced by `push 02`.

Note that if the instructions i_1 and i_2 do not have the same number of operands, then `mov` l v will, in addition to replacing i_1 by i_2, change several other instructions that follow i_1. Currently, we cannot handle this case, thus we assume that i_1 and i_2 have the same number of operands.

Note also that *mov l v* is self-modifying only if l is a location of the program that stores executable data, otherwise, it is not; e.g., *mov eax v* does not change the instructions of the program, it just writes the value v to the register *eax*. Thus, from now on, by self-modifying instruction, we mean an instruction of the form *mov l v*, where l is a location of the program that stores executable data. Moreover, to ensure that only one instruction is modified, we assume that the corresponding instructions i_1 and i_2 have the same number of operands.

2.3.2 From Self-modifying Code to SM-PDS

We show in what follows how to build a SM-PDS from a binary program. We suppose we are given an oracle $\mathcal{O}$ that extracts from the binary code a corresponding assembly program, together with informations about the values of the registers and the memory locations at each control point of the program. In our implementation, we use Jakstab [22] to get this oracle. We translate the assembly program into a self-modifying pushdown system where the control locations store the control points of the binary program and the stack mimics the program's stack. The non self-modifying instructions of the program define the rules Δ of the SM-PDS (which are standard PDS rules), and can be obtained following the translation of [13] that models non self-modifying instructions of the program by a PDS.

As for the self-modifying instructions of the program, they define the set of changing rules Δ_c. As explained above, these are instructions of the form *mov l v*, where l is a location of the program that stores executable data. This instruction replaces the value at location l (in the binary code) with the value v. Let i_1 be the initial instruction involving the location l, and let i_2 be the new instruction involving the location l, after applying the *mov l v* instruction. As mentioned previously, we assume that i_1 and i_2 have the same number of operands (to ensure that only one instruction is modified). Let r_1 (resp. r_2) be the SM-PDS rule corresponding to the instruction i_1 (resp. i_2). Suppose from control point n to n', we have this *mov l v* instruction, then we add $n \xhookrightarrow{(r_1,r_2)} n'$ to Δ_c. This is the SM-PDS rule corresponding to the instruction *mov l v* at control point n.

2.4 Representing Infinite Sets of Configurations of a SM-PDS

Multi-automata were introduced in [6, 20] to finitely represent regular infinite sets of configurations of a PDS. A configuration $c = (\langle p, w\rangle, \theta)$ of a SM-PDS involves a PDS configuration $\langle p, w\rangle$, together with the current set of transition rules (phase) θ. To finitely represent regular infinite sets of such configurations, we extend multi-automata in order to take into account the phases θ:

Definition 3 *Let $\mathcal{P} = (P, \Gamma, \Delta, \Delta_c)$ be a SM-PDS. A $\mathcal{P}$-automaton is a tuple $\mathcal{A} = (Q, \Gamma, T, P, F)$ where Γ is the automaton alphabet, Q is a finite set of states, $P \times 2^{\Delta \cup \Delta_c} \subseteq Q$ is the set of initial states, $T \subset Q \times \big((\Gamma \cup \{\epsilon\})\big) \times Q$ is the set of transitions and $F \subseteq Q$ is the set of final states.*

If $\big(q, \gamma, q'\big) \in T$, we write $q \xrightarrow{\gamma}_T q'$. We extend this notation in the obvious manner to sequences of symbols: (1) $\forall q \in Q, q \xrightarrow{\epsilon}_T q$, and (2) $\forall q, q' \in Q, \forall \gamma \in \Gamma \cup \{\epsilon\}, \forall w \in \Gamma^*$ for $w = \gamma_0\gamma_1 \cdots \gamma_n, q \xrightarrow{\gamma w}_T q'$ iff $\exists q'' \in Q, q \xrightarrow{\gamma}_T q''$ and $q'' \xrightarrow{w}_T q'$. If $q \xrightarrow{w}_T q'$ holds, we say that $q \xrightarrow{w}_T q'$ is a path of $\mathcal{A}$. A configuration $(\langle p, w \rangle, \theta)$ is accepted by $\mathcal{A}$ iff $\mathcal{A}$ contains a path $(p, \theta) \xrightarrow{\gamma_0}_T q_1 \xrightarrow{\gamma_1}_T q_2 \cdots q_n \xrightarrow{\gamma_n}_T q$ where $q \in F$. Let $L(\mathcal{A})$ be the set of configurations accepted by $\mathcal{A}$. Let $\mathcal{C}$ be a set of configurations of the SM-PDS $\mathcal{P}$. $\mathcal{C}$ is regular if there exists a $\mathcal{P}$-automaton $\mathcal{A}$ such that $\mathcal{C} = L(\mathcal{A})$.

2.5 Efficient Computation of pre^* images

Let $\mathcal{P} = (P, \Gamma, \Delta, \Delta_c)$ be a SM-PDS, and let $\mathcal{A} = (Q, \Gamma, T, P, F)$ be a $\mathcal{P}$-automaton that represents a regular set of configurations $\mathcal{C}$ ($\mathcal{C} = L(\mathcal{A})$). To compute $pre^*(\mathcal{C})$, one can use the translation of Section 2.2.2 to compute an equivalent PDS, and then apply the algorithms of [6, 20]. This procedure is too complex since the size of the obtained PDS is huge. One can also use the translation of Section 2.2.3 to compute an equivalent symbolic PDS, and then use the algorihms of [42]. However, this procedure is not optimal neither since the number of elements of the relations considered in the rules of the symbolic PDSs are huge. We present in this section a **direct** and **more efficient** algorithm that computes $pre^*(\mathcal{C})$ without any need to translate the SM-PDS to an equivalent PDS or symbolic PDS. We assume w.l.o.g. that $\mathcal{A}$ has no transitions leading to an initial state. We also assume that the self-modifying rules $r = p \xhookrightarrow{(r_1, r_2)} p'$ in Δ_c are such that $r \neq r_1$. This is not a restriction since a rule of the form $r = p \xhookrightarrow{(r, r_2)} p'$ can be replaced by these rules that meet this constraint: $r = p \xhookrightarrow{(r_\perp, r_\perp)} p_i$ and $p_i \xhookrightarrow{(r, r_2)} p'$, where $r_\perp$ is a new fake rule that we can add to all phases.

The construction of $\mathcal{A}_{pre^*}$ follows the same idea as for standard pushdown systems (see [6, 20]). It consists in adding iteratively new transitions to the

automaton $\mathcal{A}$ according to **saturation** rules (reflecting the backward application of the transition rules in the system), while the set of states remains unchanged. Therefore, let $\mathcal{A}_{pre^*}$ be the $\mathcal{P}$-automaton (Q, Γ, T', P, F), where T' is computed using the following saturation rules: initially $T' = T$.

α_1: If $r = \langle p, \gamma \rangle \hookrightarrow \langle p_1, w \rangle \in \Delta$, where $w \in \Gamma^*$. For every $\theta \subseteq \Delta \cup \Delta_c$ s.t. $r \in \theta$, if there exists in T' a path $\pi = (p_1, \theta) \xrightarrow{w}_T q$, then add $((p, \theta), \gamma, q)$ to T'.

α_2: if $r = p \xhookrightarrow{(r_1, r_2)} p_1 \in \Delta_c$ for every $\theta \subseteq \Delta \cup \Delta_c$ s.t. $r \in \theta, r_2 \in \theta$ and for every $\gamma \in \Gamma$, if there exists in T' a transition $t = (p_1, \theta) \xrightarrow{\gamma}_T q$, then add $((p, \theta'), \gamma, q)$ to T' where $\theta = (\theta' \setminus \{r_1\}) \cup \{r_2\}$.

The procedure above terminates since there is a finite number of states and phases.
Let us explain intuitively the role of the saturation rule (α_1). Let $r = \langle p, \gamma \rangle \hookrightarrow \langle p', w \rangle \in \Delta$. Consider a path in the automaton of the form $(p', \theta') \xrightarrow{w}_{T'} q \xrightarrow{w'}_{T'} q_F$, where $q_F \in F$. This means, by definition of $\mathcal{P}$-automata, that the configuration $c = (\langle p', ww' \rangle, \theta')$ is accepted by $\mathcal{A}_{pre^*}$. If r is in θ', then the configuration $c' = (\langle p, \gamma w' \rangle, \theta')$ is a predecessor of c. Therefore, it should be added to $\mathcal{A}_{pre^*}$. This configuration is accepted by the run $(p, \theta') \xrightarrow{\gamma}_{T'} q \xrightarrow{w'}_{T'} q_F$ added by rules (α_1).

Rule (α_2) deals with modifying rules: Let $r = p \xhookrightarrow{(r_1, r_2)} p' \in \Delta_c$. Consider a path in the automaton of the form $(p', \theta') \xrightarrow{\gamma}_{T'} q \xrightarrow{w'}_{T'} q_F$, where $q_F \in F$. This means, by definition of $\mathcal{P}$-automata, that the configuration $c = (\langle p', \gamma w' \rangle, \theta')$ is accepted by $\mathcal{A}_{pre^*}$. If r and r_2 are in θ', then the configuration $c' = (\langle p, \gamma w' \rangle, \theta)$ is a predecessor of c, where $\theta' = (\theta \setminus \{r_1\}) \cup \{r_2\}$. Therefore, it should be added to $\mathcal{A}_{pre^*}$. This configuration is accepted by the run $\pi' = (p, \theta) \xrightarrow{\gamma}_{T'} q \xrightarrow{w'}_{T'} q_F$ added by rules (α_2).

Thus, we can show that:

Theorem 2.5.1 *$\mathcal{A}_{pre^*}$ recognizes $pre^*(L(\mathcal{A}))$.*

Before proving this theorem, let us illustrate the construction on 2 examples.

Example 2 *Let us illustrate the procedure by an example. Consider the SM-PDS with control points $P = \{p_0, p_1, p_2, p_3, p_4, p_5\}$ and Δ, Δ_c as shown in the left half*

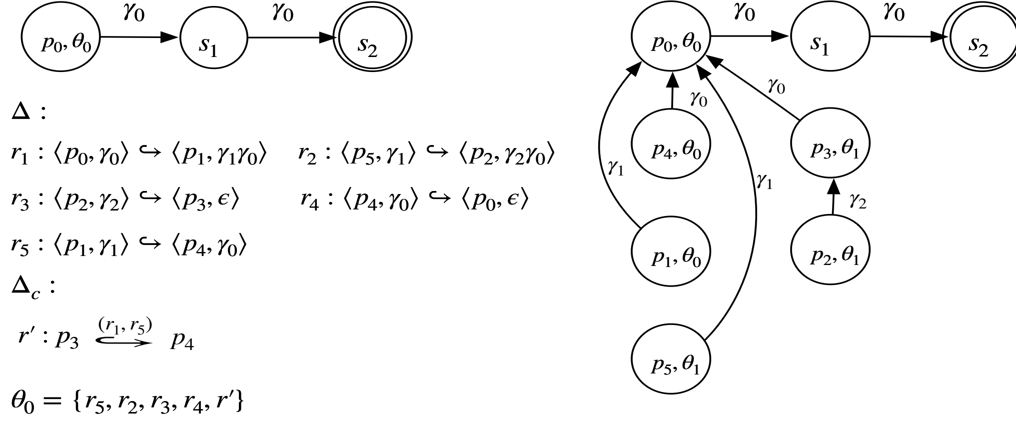

Figure 2.2: The automata $\mathcal{A}$ (left) and $\mathcal{A}_{pre^*}$ (right)

of Fig. 2.2. Let $\mathcal{A}$ be the automaton that accepts the set $C = \{(\langle p_0, \gamma_0\gamma_0 \rangle, \theta_0)\}$, also shown on the left where (p_0, θ_0) is the initial state and s_2 is the final state. The result of the algorithm is shown in the right half of Fig. 2.2. The result is obtained through the following steps:

1. *First, we note that $(p_0, \theta_0) \xrightarrow{\epsilon}_{T'} (p_0, \theta_0)$ holds. Since $\langle p_0, \epsilon \rangle$ occurs on the right hand side of rule r_4 and $r_4 \in \theta_0$, then Rule (α_1) adds the transition $(p_4, \theta_0) \xrightarrow{\gamma_0} (p_0, \theta_0)$ to T'.*

2. *Now that we have $(p_4, \theta_0) \xrightarrow{\gamma_0}_{T'} (p_0, \theta_0)$, since $r_5 \in \theta_0$, Rule (α_1) adds $(p_1, \theta_0) \xrightarrow{\gamma_1} (p_0, \theta_0)$ to T'.*

3. *Since we have $(p_4, \theta_0) \xrightarrow{\gamma_0}_{T'} (p_0, \theta_0)$, the self-modifying transition $r' \in \theta_0$ can be applied. Thus, Rule (α_2) adds $(p_3, \theta_1) \xrightarrow{\gamma_0} (p_0, \theta_0)$ to T' where $\theta_1 = (\theta_0 \setminus \{r_5\}) \cup \{r_1\} = \{r_1, r_2, r_3, r_4, r'\}$.*

4. *Since $(p_3, \theta_1) \xrightarrow{\epsilon} (p_3, \theta_1)$ and $r_3 \in \theta_1$, Rule (α_1) adds $(p_2, \theta_1) \xrightarrow{\gamma_2} (p_3, \theta_1)$ to T'.*

5. *Then, there is a path $(p_2, \theta_1) \xrightarrow{\gamma_2}_{T'} (p_3, \theta_1) \xrightarrow{\gamma_0}_{T'} (p_0, \theta_0)$. Since $\langle p_2, \gamma_2\gamma_0 \rangle$ occurs on the right hand side of r_2 and $r_2 \in \theta_1$, then Rule (α_1) adds the transition $(p_5, \theta_1) \xrightarrow{\gamma_1} (p_0, \theta_0)$ to T'.*

6. *No further additions are possible. Thus, the procedure terminates.*

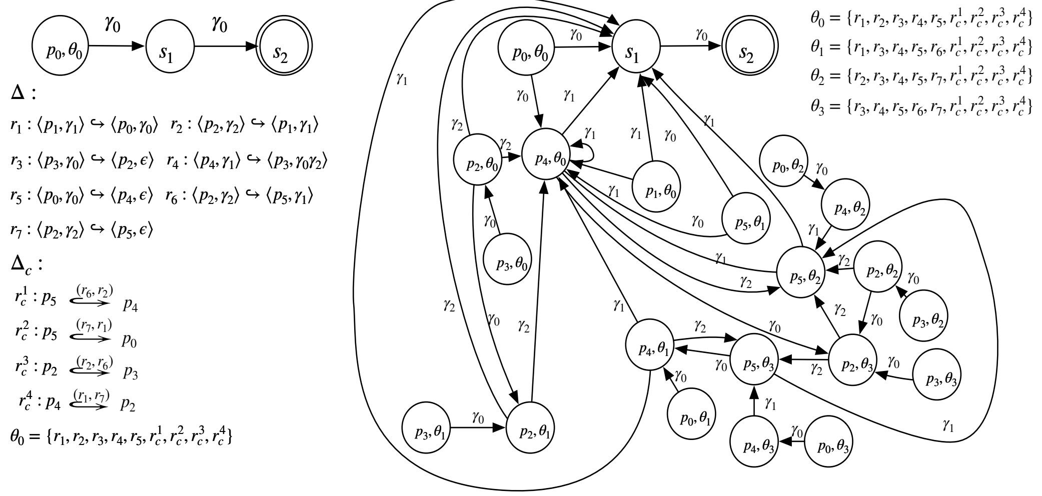

Figure 2.3: The automata $\mathcal{A}$ (left) and $\mathcal{A}_{pre^*}$ (right)

Example 3 *Let us give another example. Consider the SM-PDS with control points* $P = \{p_1, p_2, p_3, p_4, p_5\}$ *and* Δ, Δ_c *as shown in the left half of Fig. 2.3. Let* $\mathcal{A}$ *be the automaton that accepts the set* $C = \{(\langle p_0, \gamma_0\gamma_0 \rangle, \theta_0)\}$ *where* (p_0, θ_0) *is the initial state and* s_2 *is the final state as shown on the left. The result* $\mathcal{A}_{pre^*}$ *of the algorithm is on the right half of Fig. 2.3. The result is obtained through the following steps:*

1. *Since* $(p_0, \theta_0) \xrightarrow{\gamma}_{T'} s_1$ *and* $r_1 \in \theta_0$, *then Rule* (α_1) *adds* $(p_1, \theta_0) \xrightarrow{\gamma_1} s_1$ *to* T'.

2. *Since* $(p_1, \theta_0) \xrightarrow{\gamma_1}_{T'} s_1$ *and* $r_2 \in \theta_0$, *Rule* (α_1) *adds the transition* $(p_2, \theta_0) \xrightarrow{\gamma_2} s_1$ *to* T'.

3. *Since* $(p_2, \theta_0) \xrightarrow{\epsilon}_{T'} (p_2, \theta_0)$ *and* $r_3 \in \theta_0$, *Rule* (α_1) *adds the transition* $(p_3, \theta_0) \xrightarrow{\gamma_0} (p_2, \theta_0)$ *to* T'.

4. *Then, there is a path* $(p_3, \theta_0) \xrightarrow{\gamma_0}_{T'} (p_2, \theta_0) \xrightarrow{\gamma_2}_{T'} s_1$ *and* $r_4 \in \theta_0$, *Rule* α_1 *adds the transition* $(p_4, \theta_0) \xrightarrow{\gamma_1} s_1$ *to* T'.

5. *Because* $(p_4, \theta_0) \xrightarrow{\epsilon}_{T'} (p_4, \theta_0)$ *and* $r_5 \in \theta_0$, *Rule* (α_1) *adds the transition* $(p_0, \theta_0) \xrightarrow{\gamma_0} (p_4, \theta_0)$ *to* T'.

6. *Since* $(p_0,\theta_0) \xrightarrow{\gamma_0}_{T'} (p_4,\theta_0)$ *and* $r_1 \in \theta_0$, *Rule* (α_1) *adds the transition* $(p_1,\theta_0) \xrightarrow{\gamma_1} (p_4,\theta_0)$ *to* T'. *Then, since* $r_2 \in \theta_0$, *Rule* (α_1) *adds the transition* $(p_2,\theta_0) \xrightarrow{\gamma_2} (p_4,\theta_0)$ *to* T'.

7. *Since there is a path* $(p_3,\theta_0) \xrightarrow{\gamma_0}_{T'} (p_2,\theta_0) \xrightarrow{\gamma_2}_{T'} (p_4,\theta_0)$ *and* $r_4 \in \theta_0$, *Rule* (α_1) *adds* $(p_4,\theta_0) \xrightarrow{\gamma_1} (p_4,\theta_0)$ *to* T'.

8. *Since* $(p_4,\theta_0) \xrightarrow{\gamma_1}_{T'} s_1, (p_4,\theta_0) \xrightarrow{\gamma_1}_{T'} (p_4,\theta_0)$ *and* $r_c^1, r_2 \in \theta_0$, *Rule* (α_2) *adds* $(p_5,\theta_1) \xrightarrow{\gamma_1} s_1$ *and* $(p_5,\theta_1) \xrightarrow{\gamma_1} (p_4,\theta_0)$ *to* T' *where* $\theta_1 = (\theta_0 \setminus \{r_2\}) \cup r_6 = \{r_1, r_3, r_4, r_5, r_6, r_c^1, r_c^2, r_c^3, r_c^4\}$. *For the same reason, since* $(p_0,\theta_0) \xrightarrow{\gamma_0}_{T'} (p_4,\theta_0)$, $(p_0,\theta_0) \xrightarrow{\gamma}_{T'} s_1$ *and* $r_1, \in \theta_1$, $r_c^2 \in \theta_0$, *Rule* (α_2) *adds the transitions* $(p_5,\theta_2) \xrightarrow{\gamma_0} (p_4,\theta_0)$ *and* $(p_5,\theta_2) \xrightarrow{\gamma_0} s_1$ *to* T' *where* $\theta_2 = (\theta_0 \setminus \{r_1\}) \cup \{r_7\} = \{r_2, r_3, r_4, r_5, r_7, r_c^1, r_c^2, r_c^3, r_c^4\}$.

9. *Since* $(p_5,\theta_1) \xrightarrow{\gamma_1}_{T'} s_1$, $(p_5,\theta_1) \xrightarrow{\gamma_1}_{T'} (p_4,\theta_0)$ *and* $r_6 \in \theta_1$, *Rule* (α_1) *adds the transitions* $(p_2,\theta_1) \xrightarrow{\gamma_2} s_1$ *and* $(p_2,\theta_1) \xrightarrow{\gamma_2} (p_4,\theta_0)$ *to* T'.

10. *Since* $(p_2,\theta_1) \xrightarrow{\epsilon}_{T'} (p_2,\theta_1)$ *and* $r_3 \in \theta_1$, *Rule* (α_1) *adds* $(p_3,\theta_1) \xrightarrow{\gamma_0} (p_2,\theta_1)$.

11. *Because there are paths* $(p_3,\theta_1) \xrightarrow{\gamma_0}_{T'} (p_2,\theta_1) \xrightarrow{\gamma_2}_{T'} (p_4,\theta_0)$ *and* $(p_3,\theta_1) \xrightarrow{\gamma_0}_{T'} (p_2,\theta_1) \xrightarrow{\gamma_2}_{T'} s_1$, *Rule* (α_1) *adds the transitions* $(p_4,\theta_1) \xrightarrow{\gamma_1} (p_4,\theta_0)$ *and* $(p_4,\theta_1) \xrightarrow{\gamma_1} s_1$ *to* T'.

12. *Since* $(p_4,\theta_0) \xrightarrow{\epsilon}_{T'} (p_4,\theta_0)$ *and* $r_5 \in \theta_1$, *Rule* (α_1) *adds* $(p_0,\theta_1) \xrightarrow{\gamma_0} (p_4,\theta_1)$.

13. *Now we have* $(p_0,\theta_1) \xrightarrow{\gamma_0}_{T'} (p_4,\theta_1)$ *and* $r_c^2, r_1 \in \theta_1$, *Rule* (α_2) *adds the transition* $(p_5,\theta_3) \xrightarrow{\gamma_0} (p_4,\theta_1)$ *to* T' *where* $\theta_3 = \{r_3, r_4, r_5, r_6, r_7, r_c^1, r_c^2, r_c^3, r_c^4\}$. *For the same reason, since* $(p_3,\theta_1) \xrightarrow{\gamma_0} (p_2,\theta_1)$ *and* $r_c^3, r_6 \in \theta_1$, *Rule* α_2 *adds the transition* $(p_2,\theta_0) \xrightarrow{\gamma_0} (p_2,\theta_1)$ *to* T' *because* $\theta_0 = (\theta_1 \setminus \{r_6\}) \cup \{r_2\}$.

14. *Since* $(p_5,\theta_3) \xrightarrow{\epsilon}_{T'} (p_5,\theta_3)$ *and* $r_7 \in \theta_3$, *Rule* (α_1) *adds the transition* $(p_2,\theta_3) \xrightarrow{\gamma_2} (p_5,\theta_3)$ *to* T'.

15. *Because* $(p_2,\theta_3) \xrightarrow{\epsilon}_{T'} (p_2,\theta_3)$ *and* $r_3 \in \theta_3$, *Rule* (α_1) *adds the transition* $(p_3,\theta_3) \xrightarrow{\gamma_0} (p_2,\theta_3)$ *to* T'. *Then, since there is a path* $(p_3,\theta_3) \xrightarrow{\gamma_0}_{T'} (p_2,\theta_3) \xrightarrow{\gamma_2}_{T'} (p_5,\theta_3)$ *and* $r_4 \in \theta_3$, *Rule* (α_1) *adds the transition* $(p_4,\theta_3) \xrightarrow{\gamma_1} (p_5,\theta_3)$ *to* T'. *Then, since* $r_5 \in \theta_3$, *Rule* (α_1) *adds the transition* $(p_0,\theta_3) \xrightarrow{\gamma_0} (p_4,\theta_3)$ *to* T'.

16. *Since* $(p_3, \theta_3) \xrightarrow{\gamma_0}_{T'} (p_2, \theta_3)$ *and* $r_c^3 \in \theta_3$*, Rule* (α_2) *adds the transition* $(p_2, \theta_2) \xrightarrow{\gamma_0}_{T'} (p_2, \theta_3)$ *to* T' *where* $(\theta_3 \setminus \{r_6\}) \cup \{r_2\} = \theta_2$*. Meanwhile, since* $(p_2, \theta_3) \xrightarrow{\gamma_2} (p_5, \theta_3)$ *and* $r_c^4, r_7 \in \theta_3$*, Rule* (α_2) *adds the transition* $(p_4, \theta_1) \xrightarrow{\gamma_2} (p_5, \theta_3)$ *to* T' *where* $(\theta_3 \setminus \{r_7\}) \cup \{r_1\} = \theta_1$*.*

17. *Because* $r_7 \in \theta_2$ *and* $(p_5, \theta_2) \xrightarrow{\epsilon}_{T'} (p_5, \theta_2)$*, Rule* (α_1) *adds the transition* $(p_2, \theta_2) \xrightarrow{\gamma_2} (p_5, \theta_2)$ *to* T'*.*

18. *Since* $(p_2, \theta_2) \xrightarrow{\epsilon}_{T'} (p_2, \theta_2)$ *and* $r_3 \in \theta_2$*, Rule* (α_1) *adds* $(p_3, \theta_2) \xrightarrow{\gamma_0}_{T'} (p_2, \theta_2)$ *to* T'*. Then, there is a path* $(p_3, \theta_2) \xrightarrow{\gamma_0\gamma_2}{}^*_{T'} (p_5, \theta_2)$*, since* $r_4 \in \theta_2$*, Rule* (α_1) *adds the transition* $(p_4, \theta_2) \xrightarrow{\gamma_1}_{T'} (p_5, \theta_2)$ *to* T'*. Then, since* $(p_5, \theta_2) \xrightarrow{\epsilon}_{T'} (p_5, \theta_2)$ *and* $r_5 \in \theta_2$*, Rule* (α_1) *adds the transition* $(p_0, \theta_2) \xrightarrow{\gamma_0} (p_4, \theta_2)$ *to* T'*.*

19. *Now we have* $(p_2, \theta_2) \xrightarrow{\gamma_2}_{T'} (p_5, \theta_2)$ *and* $(p_2, \theta_2) \xrightarrow{\gamma_0}_{T'} (p_2, \theta_3)$*, since* $r_c^4, r_7 \in \theta_2$*, Rule* α_2 *adds the transitions* $(p_4, \theta_0) \xrightarrow{\gamma_2} (p_5, \theta_2)$ *and* $(p_4, \theta_0) \xrightarrow{\gamma_0} (p_2, \theta_3)$ *to* T' *where* $(\theta_2 \setminus \{r_7\} \cup) \{r_1\} = \theta_0$*.*

20. *Since* $(p_4, \theta_2) \xrightarrow{\gamma_1}_{T'} (p_5, \theta_2)$ *and* $r_2, r_c^1 \in \theta_2$*, Rule* (α_2) *adds the transition* $(p_5, \theta_3) \xrightarrow{\gamma_1} (p_5, \theta_2)$ *to* T' *where* $(\theta_2 \setminus \{r_2\}) \cup \{r_6\} = \theta_3$*.*

21. *Since* $(p_5, \theta_3) \xrightarrow{\gamma_1}_{T'} (p_5, \theta_2)$ *and* $r_6 \in \theta_3$*, Rule* (α_1) *adds the transition* $(p_2, \theta_3) \xrightarrow{\gamma_2} (p_5, \theta_2)$ *to* T'*.*

22. *No further additions are possible, so the procedure terminates.*

2.5.1 Proof of Theorem 2.5.1

Let us now prove Theorem 2.5.1. To prove this theorem, we first introduce the following lemma.

Lemma 1 *For every configuration* $(\langle p, w \rangle, \theta_0) \in L(\mathcal{A})$*, if* $(\langle p', w' \rangle, \theta) \Rightarrow^*_{\mathcal{P}} (\langle p, w \rangle, \theta_0)$*, then* $(p', \theta) \xrightarrow{w'}_{T'} q$ *for some final state* q *of* $\mathcal{A}_{pre^*}$*.*

Proof: Assume $(\langle p', w' \rangle, \theta) \overset{i}{\Rightarrow}_{\mathcal{P}} (\langle p, w \rangle, \theta_0)$. We proceed by induction on i.

Basis. $i = 0$. Then $\theta = \theta_0, p' = p$ and $w = w'$. Since $(\langle p, w \rangle, \theta_0) \in L(\mathcal{A})$, we have $(p, \theta_0) \xrightarrow{w}_{T'} q$ always holds for some final state q i.e. $(p', \theta) \xrightarrow{w'}_{T'} q$ holds.

Step. $i > 0$. Then there exists a configuration $(\langle p'', u \rangle, \theta'')$ such that

$$(\langle p', w' \rangle, \theta) \Rightarrow_{\mathcal{P}} (\langle p'', u \rangle, \theta'') \overset{i-1}{\Rightarrow}_{\mathcal{P}} (\langle p, w \rangle, \theta_0)$$

We apply the induction hypothesis to $(\langle p'', u\rangle, \theta'') \overset{i-1}{\Rightarrow} (\langle p, w\rangle, \theta_0)$, and obtain $(p'', \theta'') \xrightarrow{u}_{T'} q$ for $q \in F$.

Let $w_1, u_1 \in \Gamma^*, \gamma' \in \Gamma$ be such that $w' = \gamma' w_1$, $u = u_1 w_1$. Let q' be a state of $\mathcal{A}_{pre^*}$ s.t.

$$(p'', \theta'') \xrightarrow{u_1}_{T'} q' \xrightarrow{w_1}_{T'} q \tag{1}$$

There are two cases depending on which rule is applied to get $(\langle p', w'\rangle, \theta) \Rightarrow (\langle p'', u\rangle, \theta'')$.

1. Case $(\langle p', w'\rangle, \theta) \Rightarrow (\langle p'', u\rangle, \theta'')$ is obtained by a rule of the form: $\langle p', \gamma'\rangle \hookrightarrow \langle p'', u_1\rangle \in \Delta$. In this case, $\theta'' = \theta$. By the saturation rule α_1, we have

$$(p', \theta'') \xrightarrow{\gamma'}_{T'} q' \tag{2}$$

   Putting (1) and (2) together, we can obtain that

$$\pi = (p', \theta'') \xrightarrow{\gamma'}_{T'} q' \xrightarrow{w_1}_{T'} q \tag{3}$$

   Thus, $(p', \theta'') \xrightarrow{\gamma' w_1}_{T'} q$ i.e. $(p', \theta) \xrightarrow{w'} q$ for some final state $q \in F$.

2. Case $(\langle p', w'\rangle, \theta) \Rightarrow (\langle p'', u\rangle, \theta'')$ is obtained by a rule of the form $p' \xhookrightarrow{(r_1, r_2)} p'' \in \Delta_c$. I.e $\theta'' \neq \theta$. In this case, $u_1 = \gamma'$. By the saturation rule α_2, we obtain that

$$(p', \theta) \xrightarrow{\gamma'}_{T'} q' \text{ where } \theta'' = \theta \backslash \{r_1\} \cup \{r_2\}. \tag{4}$$

   Putting (1) and (4) together, we have the following path

$$(p', \theta) \xrightarrow{\gamma'}_{T'} q' \xrightarrow{w_1}_{T'} q. \text{ I.e. } (p', \theta) \xrightarrow{w'}_{T'} q \text{ for } q \in F \tag{5}$$

□

Lemma 2 *If a path $\pi = (p, \theta) \xrightarrow{w}_{T'} q$ for $\theta \subseteq \Delta \cup \Delta_c$ is in $\mathcal{A}_{pre^*}$, then*

(I) $(\langle p, w\rangle, \theta) \Rightarrow^* (\langle p', w'\rangle, \theta_0)$ holds for a configuration $(\langle p', w'\rangle, \theta_0)$ s.t. $(p', \theta_0) \xrightarrow{w'}_{T} q$ in the initial $\mathcal{P}$-automaton $\mathcal{A}$;

(II) Moreover, if q is an initial state i.e. in the form (p, θ), then $w' = \epsilon$.

Proof: Let $\mathcal{A}_{pre^*} = (Q, \Gamma, T, P, F)$ be the $\mathcal{P}$-automaton computed by the saturation procedure. In this proof, we use $\underset{i}{\longrightarrow}_{T'}$ to denote the transition relation of $\mathcal{A}_{pre^*}$ obtained after adding i-transitions using the saturation procedure. In particular, since initially $\mathcal{A}_{pre^*} = \mathcal{A}$, $\mathcal{A}_{pre^*}$ contains the path $(p', \theta_0) \xrightarrow{w'}_T q$ where $(\langle p', w' \rangle, \theta_0) \in L(\mathcal{A})$, then we write $(p', \theta_0) \underset{0}{\xrightarrow{w'}}_T q$.

Let i be an index such that $\pi = (p, \theta) \underset{i}{\xrightarrow{w}}_{T'} q$ holds. We shall prove (I) by induction on i. Statement (II) then follows immediately from the fact that initial states have no incoming transitions in $\mathcal{A}$.

Basis. $i = 0$. Since $(\langle p, w \rangle, \theta) \Rightarrow^* (\langle p, w \rangle, \theta)$ always holds, take then $p = p'$, $w = w'$ and $\theta_0 = \theta$.

Step. $i > 0$. Let $t = ((p_1, \theta_1), \gamma, q')$ be the i-th transition added to $\mathcal{A}_{pre^*}$ and j be the number of times that t is used in the path $(p, \theta) \underset{i}{\xrightarrow{w}}_{T'} q$. The proof is by induction on j. If $j = 0$, then we have $(p, \theta) \underset{i-1}{\xrightarrow{w}}_{T'} q$ in the automaton, and we apply the induction hypothesis (induction on i) then we obtain $(\langle p, w \rangle, \theta) \Rightarrow^* (\langle p', w' \rangle, \theta_0)$ for a configuration $(\langle p', w' \rangle, \theta_0)$ s.t. $(p', \theta_0) \xrightarrow{w'}_T q$ in the initial $\mathcal{P}$-automaton $\mathcal{A}$. So assume that $j > 0$. Then, there exist u and v such that $w = u\gamma v$ and

$$(p, \theta) \underset{i-1}{\xrightarrow{u}}_{T'} (p_1, \theta_1) \underset{i}{\xrightarrow{\gamma}}_{T'} q' \underset{i}{\xrightarrow{v}}_{T'} q \quad (1)$$

The application of the induction hypothesis (induction on i) to $(p, \theta) \underset{i-1}{\xrightarrow{u}}_{T'} (p_1, \theta_1)$ (notice that (p_1, θ_1) is an initial state) gives that

$$(\langle p, u \rangle, \theta) \Rightarrow^* (\langle p_1, \epsilon \rangle, \theta_1) \quad (2)$$

There are 2 cases depending on whether transition t was added by saturation rule α_1 or α_2.

1. Case t was added by rule α_1: There exist $p_2 \in P$ and $w_2 \in \Gamma^*$ such that

$$r = \langle p_1, \gamma \rangle \hookrightarrow \langle p_2, w_2 \rangle \in \Delta \cap \theta_1 \quad (3)$$

and $\mathcal{A}_{pre^*}$ contains the following path:

$$\pi' = (p_2, \theta_1) \underset{i-1}{\xrightarrow{w_2}}_{T'} q' \underset{i}{\xrightarrow{v}}_{T'} q \quad (4)$$

Applying the transition rule r gets that

$$(\langle p_1, \gamma v\rangle, \theta_1) \Rightarrow (\langle p_2, w_2 v\rangle, \theta_1) \tag{5}$$

By induction on j (since transition t is used $j-1$ times in π'), we get from (4) that

$$(\langle p_2, w_2 v\rangle, \theta_1) \Rightarrow^* (\langle p', w'\rangle, \theta_0) \text{ s.t. } (p', \theta_0) \xrightarrow{w'}_{T'} q \text{ in the initial } \mathcal{P}\text{-automaton } \mathcal{A} \tag{6}$$

Putting (2) ,(5) and (6) together, we can obtain that

$$(\langle p, w\rangle, \theta) = (\langle p, u\gamma v\rangle, \theta) \Rightarrow^* (\langle p_1, \gamma v\rangle, \theta_1) \Rightarrow (\langle p_2, w_2 v\rangle, \theta_1) \Rightarrow^* (\langle p', w'\rangle, \theta_0)$$

such that $(p', \theta_0) \xrightarrow{w'}_T q$ in the initial $\mathcal{P}$-automaton $\mathcal{A}$

2. Case t was added by rule α_2 : there exist $p_2 \in P$ and $\theta'' \subseteq \Delta \cup \Delta_c$ such that

$$p_1 \xhookrightarrow{(r_1, r_2)} p_2 \in \Delta_c \cap \theta'', \theta'' = (\theta_1 \backslash \{r_1\}) \cup \{r_2\} \tag{7}$$

and the following path in the current automaton (self-modifying rule won't change the stack) with $r \in \theta''$:

$$(p_2, \theta'') \xrightarrow[i-1]{\gamma}_{T'} q' \xrightarrow[i]{v}_{T'} q \tag{8}$$

Applying the transition rule, we can get from (7) that

$$(\langle p_1, \gamma v\rangle, \theta_1) \Rightarrow (\langle p_2, \gamma v\rangle, \theta'') \tag{9}$$

We can apply the induction hypothesis (on j) to (8), and obtain

$$(\langle p_2, \gamma v\rangle, \theta'') \Rightarrow^* (\langle p', w'\rangle, \theta_0) \text{ s.t. } (p', \theta_0) \xrightarrow{w'}_T q \text{ in the initial } \mathcal{P}\text{-automaton } \mathcal{A} \tag{10}$$

From (2),(9) and (10), we get

$$(\langle p, w\rangle, \theta) = (\langle p, u\gamma v\rangle, \theta) \Rightarrow^* (\langle p_1, \gamma v\rangle, \theta_1) \Rightarrow (\langle p_2, \gamma v\rangle, \theta'') \Rightarrow^* (\langle p', w'\rangle, \theta_0)$$

such that $(p', \theta_0) \xrightarrow{w'}_T q$ in the initial $\mathcal{P}$-automaton $\mathcal{A}$.

□

Then, we can prove Theorem 2.5.1:

Proof: Let $(\langle p, w\rangle, \theta)$ be a configuration of $pre^*(L(\mathcal{A}))$. Then $(\langle p, w\rangle, \theta) \Rightarrow^* (\langle p', w'\rangle, \theta_0)$ for a configuration $(\langle p', w'\rangle, \theta_0)$ s.t. $(p', \theta_0)\xrightarrow{w'}_{T'} q$ is a path in $\mathcal{A}$ for $q \in F$. By lemma 1, we can obtain that there exists a path $(p, \theta) \xrightarrow{w}_{T'} q$ for some final state q of $\mathcal{A}_{pre^*}$. So $(\langle p, w\rangle, \theta)$ is recognized by $\mathcal{A}_{pre^*}$.

Conversely, let $(\langle p, w\rangle, \theta)$ be a configuration accepted by $\mathcal{A}_{pre^*}$ i.e. there exists a path $(p, \theta) \xrightarrow{w}_{T'} q$ in $\mathcal{A}_{pre^*}$ for some final state $q \in F$. By Lemma 2, there exists a configuration $(\langle p', w'\rangle, \theta_0)$ s.t. there exist a path $(p', \theta_0) \xrightarrow{w'}_{T} q$ in the initial automaton $\mathcal{A}$ and $(\langle p, w\rangle, \theta) \Rightarrow^* (\langle p', w'\rangle, \theta_0)$. Because q is a final state, we have $(\langle p', w'\rangle, \theta_0) \in L(\mathcal{A})$ i.e. $(\langle p, w\rangle, \theta) \in pre^*(L(\mathcal{A}))$.

□

2.6 Efficient Computation of $post^*$ Images

Let $\mathcal{P} = (P, \Gamma, \Delta, \Delta_c)$ be a SM-PDS, and let $\mathcal{A} = (Q, \Gamma, T, P, F)$ be a $\mathcal{P}$-automaton that represents a regular set of configurations $\mathcal{C}$ ($\mathcal{C} = L(\mathcal{A})$). Similarly, it is not optimal to compute $post^*(\mathcal{C})$ using the translations of Sections 2.2.2 and 2.2.3 to compute equivalent PDSs or symbolic PDSs, and then apply the algorithms of [20, 42]. We present in this section a **direct** and **efficient** algorithm that computes $post^*(\mathcal{C})$. We assume w.l.o.g. that $\mathcal{A}$ has no transitions leading to an initial state. Moreover, we assume that the rules of Δ are of the form $\langle p, \gamma\rangle \hookrightarrow \langle p', w\rangle$, where $|w| \leq 2$. This is not a restriction, indeed, a rule of the form $\langle p, \gamma\rangle \hookrightarrow \langle p', \gamma_1 \cdots \gamma_n\rangle$, $n > 2$ can be replaced by the following rules:

- $\langle p, \gamma\rangle \hookrightarrow \langle p_1, a_1\gamma_n\rangle$
- $\langle p_1, a_1\rangle \hookrightarrow \langle p_2, a_2\gamma_{n-1}\rangle$
- $\langle p_2, a_2\rangle \hookrightarrow \langle p_3, a_3\gamma_{n-2}\rangle$
- $\cdots$,
- $\langle p_{n-2}, a_{n-2}\rangle \hookrightarrow \langle p', \gamma_1\gamma_2\rangle$

As previously, the construction of $\mathcal{A}_{post^*}$ consists in adding iteratively new transitions to the automaton $\mathcal{A}$ according to saturation rules (reflecting the forward application of the transition rules in the system). We define $\mathcal{A}_{post^*}$ to be the $\mathcal{P}$-automaton (Q', Γ, T', P, F), where T' is computed using the following saturation rules and Q' is the smallest set s.t. $Q \subseteq Q'$ and for every $r = \langle p, \gamma \rangle \hookrightarrow \langle p', \gamma_1\gamma_2 \rangle \in \Delta, q^{\theta}_{p'\gamma_1} \in Q'$ where $q^{\theta}_{p'\gamma_1}$ is the new state labelled with p', γ_1 and θ: initially $T' = T$;

β_1: If $r = \langle p, \gamma \rangle \hookrightarrow \langle p', \epsilon \rangle \in \Delta$ and there exists in T' a path $\pi = (p, \theta) \xrightarrow{\gamma}_{T'} q$ with $r \in \theta$, then add $((p', \theta), \epsilon, q)$ to T'.

β_2: If $r = \langle p, \gamma \rangle \hookrightarrow \langle p', \gamma' \rangle \in \Delta$ and there exists in T' a path $\pi = (p, \theta) \xrightarrow{\gamma}_{T'} q$ with $r \in \theta$, then add $((p', \theta), \gamma', q)$ to T'.

β_3: If $r = \langle p, \gamma \rangle \hookrightarrow \langle p', \gamma_1\gamma_2 \rangle \in \Delta$ and there exists in T' a path $\pi = (p, \theta) \xrightarrow{\gamma}_{T'} q$ with $r \in \theta$. Add $t' = ((p', \theta), \gamma_1, q^{\theta}_{p'\gamma_1})$ and $t'' = (q^{\theta}_{p'\gamma_1}, \gamma_2, q)$ to T'.

β_4: if $r = p \xhookrightarrow{(r_1, r_2)} p' \in \Delta_c$ and there exists in T' a path $\pi = (p, \theta) \xrightarrow{\gamma}_{T'} q$, where $\gamma \in \Gamma$ with $r \in \theta$, and $r_1 \in \theta$, then add $t' = ((p', \theta'), \gamma, q)$ where $\theta' = (\theta \setminus \{r_1\}) \cup \{r_2\}\}$.

The procedure above terminates since there is a finite number of states and phases.

Let us explain intuitively the role of the saturation rules above. Consider a path in the automaton of the form $(p, \theta) \xrightarrow{\gamma}_{T'} q \xrightarrow{w'}_{T'} q_F$, where $q_F \in F$. This means, by definition of $\mathcal{P}$-automata, that the configuration $c = (\langle p, \gamma w' \rangle, \theta)$ is accepted by $\mathcal{A}_{post^*}$.

Let $r = \langle p, \gamma \rangle \hookrightarrow \langle p', \epsilon \rangle \in \Delta$. If r is in θ, then the configuration $c' = (\langle p', w' \rangle, \theta)$ is a successor of c. Therefore, it should be added to $\mathcal{A}_{post^*}$. This configuration is accepted by the run $(p', \theta) \xrightarrow{\epsilon}_{T'} q \xrightarrow{w'}_{T'} q_F$ added by rules (β_1).

If θ contains the rule $r = \langle p, \gamma \rangle \hookrightarrow \langle p', \gamma' \rangle \in \Delta$, then the configuration $c' = (\langle p', \gamma' w' \rangle, \theta)$ is a successor of c. Therefore, it should be added to $\mathcal{A}_{post^*}$. This configuration is accepted by the run $(p', \theta) \xrightarrow{\gamma'}_{T'} q \xrightarrow{w'}_{T'} q_F$ added by rules (β_2).

If $r = \langle p, \gamma \rangle \hookrightarrow \langle p', \gamma_1\gamma_2 \rangle \in \Delta$ is in θ, then the configuration $c' = (\langle p', \gamma_1\gamma_2 w' \rangle, \theta)$ is a successor of c. Therefore, it should be added to $\mathcal{A}_{post^*}$. This configuration is accepted by the run $(p', \theta) \xrightarrow{\gamma_1}_{T'} q^{\theta}_{p'\gamma_1} \xrightarrow{\gamma_2}_{T'} q \xrightarrow{w'}_{T'} q_F$ added by rules (β_3).

Rule (β_4) deals with modifying rules: Let $r = p \xhookrightarrow{(r_1,r_2)} p' \in \Delta_c$. If r and r_1 are in θ, then the configuration $c' = (\langle p', \gamma w' \rangle, \theta')$ is a successor of c, where $\theta' = (\theta \setminus \{r_1\}) \cup \{r_2\}$. Therefore, it should be added to $\mathcal{A}_{post^*}$. This configuration is accepted by the run $(p', \theta') \xrightarrow{\gamma}_{T'} q \xrightarrow{w'}_{T'} q_F$ added by rules (β_4).

Thus, we can show that:

Theorem 2.6.1 *$\mathcal{A}_{post^*}$ recognizes the set $post^*(L(\mathcal{A}))$.*

Before proving this theorem, let us illustrate the construction on 2 examples.

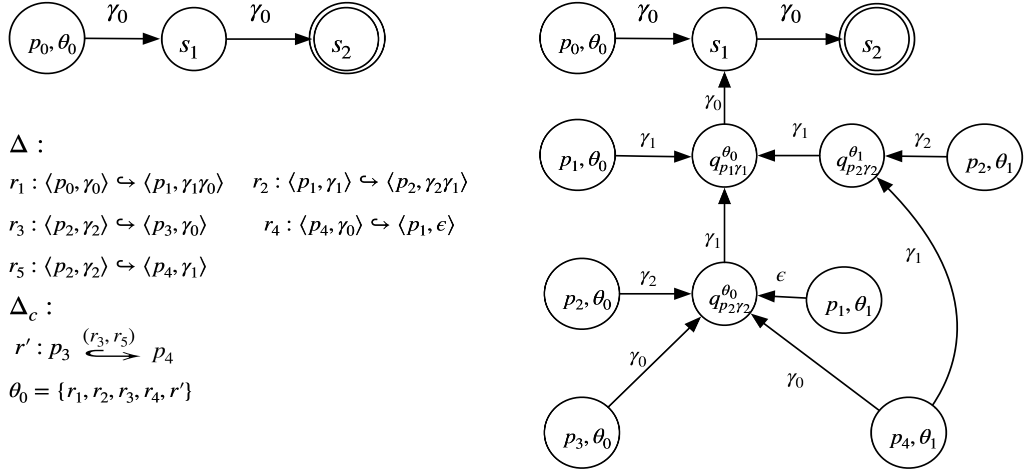

Figure 2.4: The automata $\mathcal{A}$ (left) and $\mathcal{A}_{post^*}$ (right)

Example 4 *Let us illustrate this procedure by an example. Consider the SM-PDS shown in the left half of Fig. 2.4 and the automaton $\mathcal{A}$ from Fig. 2.4 that accepts the set $C = \{(\langle p_0, \gamma_0\gamma_0 \rangle, \theta_0)\}$ where (p_0, θ_0) is the initial state and s_2 is the final state. Then the result $\mathcal{A}_{post^*}$ of the algorithm is shown in the right half of Fig. 2.4. The result is derived through the following steps:*

1. *First, since $(p_0, \theta_0) \xrightarrow{\gamma_0}_{T'} s_1$ and $r_1 \in \theta_0$, Rule (β_3) generates a new state $q^{\theta_0}_{p_1\gamma_1}$ and adds the two transitions: $(p_1, \theta_0) \xrightarrow{\gamma_1} q^{\theta_0}_{p_1\gamma_1}$ and $q^{\theta_0}_{p_1\gamma_1} \xrightarrow{\gamma_0} s_1$ to T'.*

2. *Since $(p_1, \theta_0) \xrightarrow{\gamma_1}_{T'} q^{\theta_0}_{p_1\gamma_1}$ and $r_2 \in \theta_0$, Rule (β_3) generates a new state $q^{\theta_0}_{p_2\gamma_2}$ and adds two transitions : $(p_2, \theta_0) \xrightarrow{\gamma_2} q^{\theta_0}_{p_2\gamma_2}$ and $q^{\theta_0}_{p_2\gamma_2} \xrightarrow{\gamma_1} q^{\theta_0}_{p_1\gamma_1}$ to T'.*

3. *Because* $(p_2, \theta_0) \xrightarrow{\gamma_2}_{T'} q^{\theta_0}_{p_2\gamma_2}$ *and* $r_3 \in \theta_0$*, Rule* (β_1) *adds the transition* $(p_3, \theta_0) \xrightarrow{\gamma_0} q^{\theta_0}_{p_2\gamma_2}$ *to* T'*.*

4. *Since* $(p_3, \theta_0) \xrightarrow{\gamma_0}_{T'} q^{\theta_0}_{p_2\gamma_2}$ *and* $r' \in \theta_0$*, Rule* (β_4) *adds the transition* $(p_4, \theta_1) \xrightarrow{\gamma_0} q^{\theta_0}_{p_2\gamma_2}$ *to* T' *where* $\theta_1 = (\theta_0 \setminus \{r_3\}) \cup \{r_5\} = \{r_1, r_2, r_4, r_5, r'\}$*.*

5. *Since* $(p_4, \theta_1) \xrightarrow{\gamma_0}_{T'} q^{\theta_0}_{p_2\gamma_2}$ *and* $r_4 \in \theta_1$*, Rule* (β_1) *adds the transition* $(p_1, \theta_1) \xrightarrow{\epsilon} q^{\theta_0}_{p_2\gamma_2}$ *to* T'*.*

6. *Then, since there is a path* $(p_1, \theta_1) \xrightarrow{\gamma_1}{}^{*}_{T'} q^{\theta_0}_{p_1\gamma_1}$ *and* $r_2 \in \theta_1$*, Rule* (β_3) *generates new state* $q^{\theta_1}_{p_2,\gamma_2}$ *and adds two transitions* $(p_2, \theta_1) \xrightarrow{\gamma_2} q^{\theta_1}_{p_2,\gamma_2}$ *and* $q^{\theta_1}_{p_2,\gamma_2} \xrightarrow{\gamma_1} q^{\theta_0}_{p_1\gamma_1}$ *to* T'*.*

7. *Since* $(p_2, \theta_1) \xrightarrow{\gamma_2}_{T'} q^{\theta_1}_{p_2,\gamma_2}$ *and* $r_5 \in \theta_1$*, Rule* (β_2) *adds the transition* $(p_4, \theta_1) \xrightarrow{\gamma_1} q^{\theta_1}_{p_2,\gamma_2}$ *to* T'*.*

8. *No unprocessed matches remain. The procedure terminates.*

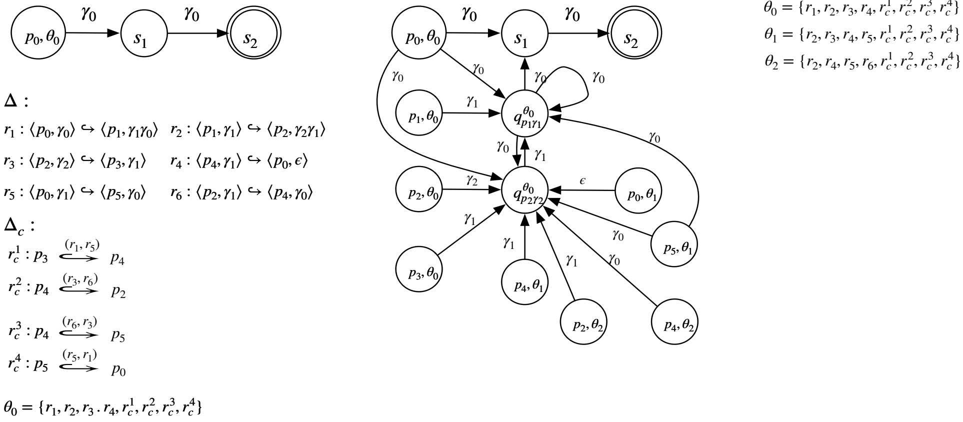

Figure 2.5: The automata $\mathcal{A}$ (left) and $\mathcal{A}_{post^*}$ (right)

Example 5 *Let us illustrate this procedure by another example. Consider the SM-PDS shown in the left half of Fig. 2.5 where* (p_0, θ_0) *is the initial state and* s_2 *is the final state. The result* $\mathcal{A}_{post^*}$ *of the algorithm is shown in the right half of Fig. 2.5 obtained as follows:*

1. *First, since* $(p_0, \theta_0) \xrightarrow{\gamma_0}_{T'} s_1$ *and* $r_1 \in \theta_0$, *Rule* (β_3) *generates a new state* $q^{\theta_0}_{p_1\gamma_1}$ *and adds two transitions:* $(p_1, \theta_0) \xrightarrow{\gamma_1} q^{\theta_0}_{p_1\gamma_1}$ *and* $q^{\theta_0}_{p_1\gamma_1} \xrightarrow{\gamma_0} s_1$ *to* T'.

2. *Since* $(p_1, \theta_0) \xrightarrow{\gamma_1}_{T'} q^{\theta_0}_{p_1\gamma_1}$ *and* $r_2 \in \theta_0$, *Rule* (β_3) *generates a new state* $q^{\theta_0}_{p_2\gamma_2}$ *and adds two transitions:* $(p_2, \theta_0) \xrightarrow{\gamma_2} q^{\theta_0}_{p_2\gamma_2}$ *and* $q^{\theta_0}_{p_2\gamma_2} \xrightarrow{\gamma_1} q^{\theta_0}_{p_1\gamma_1}$ *to* T'.

3. *Because* $(p_2, \theta_0) \xrightarrow{\gamma_2}_{T'} q^{\theta_0}_{p_2\gamma_2}$ *and* $r_3 \in \theta_0$, *Rule* (β_2) *adds* $(p_3, \theta_0) \xrightarrow{\gamma_1} q^{\theta_0}_{p_2\gamma_2}$ *to* T'.

4. *Since* $(p_3, \theta_0) \xrightarrow{\gamma_1}_{T'} q^{\theta_0}_{p_2\gamma_2}$ *and* $r^1_c, r_1 \in \theta_0$, *Rule* (β_4) *adds the transition* $(p_4, \theta_1) \xrightarrow{\gamma_1} q^{\theta_0}_{p_2\gamma_2}$ *to* T' *where* $\theta_1 = (\theta_0 \setminus \{r_1\}) \cup \{r_5\}$.

5. *Since* $(p_4, \theta_1) \xrightarrow{\gamma_1}_{T'} q^{\theta_0}_{p_2\gamma_2}$ *and* $r_4 \in \theta_1$, *Rule* (β_1) *adds the transition* $(p_0, \theta_1) \xrightarrow{\epsilon} q^{\theta_0}_{p_2\gamma_2}$ *to* T'. *Then there is a path* $(p_0, \theta_1) \xrightarrow{\gamma_1}{}^{*}_{T'} q^{\theta_0}_{p_1\gamma_1}$, *since* $r_5 \in \theta_1$, *Rule* (β_2) *adds the transition* $(p_5, \theta_1) \xrightarrow{\gamma_0} q^{\theta_0}_{p_1\gamma_1}$ *to* T'.

6. *Since* $(p_5, \theta_1) \xrightarrow{\gamma_0}_{T'} q^{\theta_0}_{p_1\gamma_1}$ *and* $r^4_c, r_5 \in \theta_1$, *Rule* (β_4) *adds the transition* $(p_0, \theta_0) \xrightarrow{\gamma_0} q^{\theta_0}_{p_1\gamma_1}$ *to* T' *where* $(\theta_1 \setminus \{r_5\}) \cup \{r_1\} = \theta_0$.

7. *Since* $(p_0, \theta_0) \xrightarrow{\gamma_0}_{T'} q^{\theta_0}_{p_1\gamma_1}$ *and* $r_1 \in \theta_0$, *Rule* (β_3) *adds the transitions* $(p_1, \theta_0) \xrightarrow{\gamma_1} q^{\theta_0}_{p_1\gamma_1}$ *and* $q^{\theta_0}_{p_1\gamma_1} \xrightarrow{\gamma_0} q^{\theta_0}_{p_1\gamma_1}$ *to* T'.

8. *Because* $(p_4, \theta_1) \xrightarrow{\gamma_1}_{T'} q^{\theta_0}_{p_2\gamma_2}$ *and* $r^2_c \in \theta_1$, *Rule* (β_4) *adds the transition* $(p_2, \theta_2) \xrightarrow{\gamma_1} q^{\theta_0}_{p_2\gamma_2}$ *to* T'.

9. *Since* $(p_2, \theta_2) \xrightarrow{\gamma_1}_{T'} q^{\theta_0}_{p_2\gamma_2}$ *and* $r_6 \in \theta_2$, *Rule* (β_2) *adds the transition* $(p_4, \theta_2) \xrightarrow{\gamma_0} q^{\theta_0}_{p_2\gamma_2}$ *to* T'.

10. *Since* $p_4, \theta_2 \xrightarrow{\gamma_0}_{T'} q^{\theta_0}_{p_2\gamma_2}$ *holds and* $r_6, r^3_c \in \theta_2$, *Rule* (β_4) *adds the transition* $(p_5, \theta_1) \xrightarrow{\gamma_0} q^{\theta_0}_{p_2\gamma_2}$ *to* T'.

11. *Then, since* $(p_5, \theta_1) \xrightarrow{\gamma_0}_{T'} q^{\theta_0}_{p_2\gamma_2}$ *and* $r^4_c \in \theta_1$, *Rule* (β_4) *adds the transition* $(p_0, \theta_0) \xrightarrow{\gamma_0} q^{\theta_0}_{p_2\gamma_2}$ *to* T'.

12. *Since* $r_1 \in \theta_0$ *and* $(p_0, \theta_0) \xrightarrow{\gamma_0}_{T'} q^{\theta_0}_{p_2\gamma_2}$, *Rule* (β_3) *adds two transitions:* $(p_1, \theta_0) \xrightarrow{\gamma_1} q^{\theta_0}_{p_1\gamma_1}$ *and* $q^{\theta_0}_{p_1\gamma_1} \xrightarrow{\gamma_0} q^{\theta_0}_{p_2\gamma_2}$ *to* T'.

13. *No more rules can be applied. Thus, the procedure terminates.*

2.6.1 Proof of Theorem 2.6.1

Let us now prove Theorem 2.6.1. To prove this theorem, we first show the following lemma:

Lemma 3 *For every configuration* $(\langle p, w\rangle, \theta_0) \in L(\mathcal{A})$*, if* $(\langle p, w\rangle, \theta_0) \Rightarrow^* (\langle p', w'\rangle, \theta)$ *then we have a path* $\pi = (p', \theta) \xrightarrow{w'}_{T'} q$ *for some final state* q *of* $\mathcal{A}_{post^*}$*.*

Proof:

Let i be the index s.t. $(\langle p, w\rangle, \theta_0) \overset{i}{\Rightarrow} (\langle p', w'\rangle, \theta)$ holds. We proceed by induction on i.

Basis. $i = 0$. Then $p' = p$, $w = w'$ and $\theta_0 = \theta$. Since $(\langle p, w\rangle, \theta_0) \in L(\mathcal{A})$, we have $(p, \theta_0) \xrightarrow{w}_{T'} q$ for some final state q that implies $\pi = (p', \theta) \xrightarrow{w'}_{T'} q$ is a path of $\mathcal{A}_{post^*}$.

Step. $i > 0$. Then there exists a configuration $(\langle p'', u\rangle, \theta'')$ with

$$(\langle p, w\rangle, \theta_0) \overset{i-1}{\Rightarrow} (\langle p'', u\rangle, \theta'') \Rightarrow (\langle p', w'\rangle, \theta)$$

By applying the induction hypothesis (induction on i), we can get that

$$(p'', \theta'') \xrightarrow{u}_{T'} q \text{ for some } q \in F \tag{1}$$

Then, let $\gamma \in \Gamma$, $u_1, w_1 \in \Gamma^*$ be such that $u = \gamma u_1$, $w' = w_1 u_1$. Let q_1 be a state of $\mathcal{A}_{post^*}$ s.t. we have the following path in $\mathcal{A}_{post^*}$:

$$(p'', \theta'') \xrightarrow{\gamma}_{T'} q_1 \xrightarrow{u_1}_{T'} q \tag{2}$$

There are two cases depending on whether $(\langle p'', u\rangle, \theta'') \Rightarrow (\langle p', w'\rangle, \theta)$ is corresponding to a self-modifying transition (i.e. $(\theta'' = \theta)$) or not.

1. Case: $\theta'' = \theta$. Then there exists a transition rule $r : \langle p'', \gamma\rangle \hookrightarrow \langle p', w_1\rangle \in \Delta$ s.t. $r \in \theta$. There are three possible cases depending on the length of w_1 :

   - Case $|w_1| = 0$ i.e. $w_1 = \epsilon$, by applying the saturation rule β_1, we can get

     $$(p', \theta) \xrightarrow{\epsilon}_{T'} q_1 \tag{3}$$

     Putting (2) and (3) together, we can have $(p', \theta) \xrightarrow{\epsilon}_{T'} q_1 \xrightarrow{u_1}_{T'} q$ i.e. $(p', \theta) \xrightarrow{w'}_{T'} q$ for some final state q of $\mathcal{A}_{post^*}$.

- Case $|w_1| = 1$, then let $\gamma' \in \Gamma$ s.t. $w_1 = \gamma'$. By applying the saturation rule α_2, we can get
$$(p', \theta) \xrightarrow{\gamma'}_{T'} q_1 \tag{4}$$
Putting (2) and (4) together, we can have $(p', \theta) \xrightarrow{\gamma'}_{T'} q_1 \xrightarrow{u_1}_{T'} q$ i.e. $(p', \theta) \xrightarrow{w'}_{T'} q$ for some final state q of $\mathcal{A}_{post^*}$.
- Case $|w_1| = 2$, let $\gamma'_0, \gamma'_1 \in \Gamma$ be such that $w_1 = \gamma'_0\gamma'_1$. By applying the saturation rule α_3, we can get
$$(p', \theta) \xrightarrow{\gamma'_0}_{T'} q^{\theta}_{p'\gamma'_0} \xrightarrow{\gamma'_1}_{T'} q_1 \tag{5}$$
Putting (2) and (5) together, then we have a path $(p', \theta) \xrightarrow{\gamma'_0}_{T'} q^{\theta}_{p'\gamma'_0} \xrightarrow{\gamma'_1}_{T'} q_1 \xrightarrow{u_1}_{T'} q$ i.e. $(p', \theta) \xrightarrow{w'}_{T'} q$ for some final state q of $\mathcal{A}_{post^*}$.

2. Case $\theta'' \neq \theta$. Then there exists a self-modifying transition rule s.t. $r : p'' \xhookrightarrow{(r_1,r_2)} p' \in \Delta_c \cap \theta''$ and $\gamma = w_1$ and $\theta = (\theta'' \backslash \{r_1\}) \cup \{r_2\}$.

   By applying rule β_4 to (2), we have the following path in the automaton:
$$(p', \theta) \xrightarrow{\gamma}_{T'} q_1 \xrightarrow{u_1}_{T'} q \tag{5}$$
   i.e. $(p', \theta) \xrightarrow{w'}_{T'} q$ for some final state q of $\mathcal{A}_{post^*}$.

□

Lemma 4 *If a path $\pi = (p, \theta) \xrightarrow{w}_{T'} q$ is in $\mathcal{A}_{post^*}$, then the following holds:*

(I) if q is a state of $\mathcal{A}$, then $(\langle p', w'\rangle, \theta_0) \Rightarrow^* (\langle p, w\rangle, \theta)$ for a configuration $(\langle p', w'\rangle, \theta_0)$ such that $(p', \theta_0) \xrightarrow{w'}_{T} q$ is a path in the initial $\mathcal{P}$-automaton $\mathcal{A}$;

(II) if q is a new state of the form $q = q^{\theta_1}_{p_1\gamma_1}$, then $(\langle p_1, \gamma_1\rangle, \theta_1) \Rightarrow^* (\langle p, w\rangle, \theta)$.

Proof: Let $\mathcal{A}_{post^*} = (Q', \Gamma, T', P, F)$ be the $\mathcal{P}$-automaton computed by the saturation procedure. In this proof, we use $\underset{i}{\longrightarrow}_{T'}$ to denote the transition relation $\rightarrow_{T'}$ of $\mathcal{A}_{post^*}$ obtained after adding i transitions using the saturation procedure.

Let i be an index such that $(p, \theta) \overset{w}{\underset{i}{\longrightarrow}}_{T'} q$ holds. We prove both parts of the lemma by induction on i.

Basis. $i = 0$. Only (I) applies. Thus, $p' = p$, $\theta_0 = \theta$ and $w = w'$. $(\langle p', w'\rangle, \theta) \Rightarrow^* (\langle p', w'\rangle, \theta)$ always holds.

Step. $i > 1$. Let t be the i-th transition added to the automaton. Let j be the number of times that t is used in $(p, \theta) \xrightarrow[i]{w}_{T'} q$. $\mathcal{A}$ has no transitions leading to initial states, and the algorithm does not add any such transitions; therefore, if t starts in an initial state, t can only be used at the start of the path.

The proof is by induction on j. If $j = 0$, then we have $(p, \theta) \xrightarrow[i-1]{w}_{T'} q$. We apply the induction hypothesis (induction on i) then we obtain that there exists a configuration $(\langle p', w'\rangle, \theta_0)$ s.t. $(\langle p', w'\rangle, \theta_0) \Rightarrow^* (\langle p, w\rangle, \theta)$ and $(p', \theta_0) \xrightarrow{w'}_T q$ is a path of initial $\mathcal{P}$-automaton $\mathcal{A}$. So assume that $j > 0$. We distinguish three possible cases:

1. If t was added by the rule β_1, β_2 or β_3, then $t = ((p_1, \theta_1), v, q_1)$, where $v = \epsilon$ or $v = \gamma_1$. Then, necessarily, $j = 1$ and there exists the following path in the current automaton:

$$(p, \theta) = (p_1, \theta_1) \xrightarrow[i]{v}_{T'} q_1 \xrightarrow[i-1]{w_1}_{T'} q \tag{1}$$

There are 2 cases depending on whether transition t was added by rule β_4 or not.

- Case t was added by rule β_4: there exists a self-modifying transition rule such that $r = p_2 \xhookrightarrow{(r_1,r_2)} p_1 \in \Delta_c$, and there exists the following path in the current automaton:

$$(p_2, \theta_2) \xrightarrow[i-1]{v}_{T'} q_1 \xrightarrow[i-1]{w_1}_{T'} q, \theta_1 = \theta_2 \backslash \{r_1\} \cup \{r_2\} \tag{2}$$

By induction on (i), we get from (2) that there exists a configuration $(\langle p', w'\rangle, \theta_0)$ s.t. $(p', \theta_0) \xrightarrow{w'}_T q$ is a path in the initial $\mathcal{P}$-automaton $\mathcal{A}$:

$$(\langle p', w'\rangle, \theta_0) \Rightarrow^* (\langle p_2, vw_1\rangle, \theta_2) \tag{3}$$

By applying the rule $p_2 \xhookrightarrow{(r_1,r_2)} p_1$, we get that

$$(\langle p_2, vw_1\rangle, \theta_2) \Rightarrow (\langle p_1, vw_1\rangle, \theta_1) \tag{4}$$

Thus, putting (3) and (4) together, we get that there exists a configuration $(\langle p', w'\rangle, \theta_0)$ s.t. $(p', \theta_0) \xrightarrow{w'}_T q$ is a path in the initial $\mathcal{P}$-automaton $\mathcal{A}$ and:

$$(\langle p', w'\rangle, \theta_0) \Rightarrow^* (\langle p_2, vw_1\rangle, \theta_2) \Rightarrow (\langle p_1, w\rangle, \theta_1) = (\langle p, w\rangle, \theta) \quad (5)$$

- Case t is added by β_1 or β_2: then there exists $p_2 \in P$, $\gamma_2 \in \Gamma$ such that

$$r = \langle p_2, \gamma_2\rangle \hookrightarrow \langle p_1, v\rangle \in \Delta \quad (6)$$

and $\mathcal{A}_{post^*}$ contains the following path:

$$(p_2, \theta_1) \xrightarrow[i-1]{\gamma_2}_{T'} q_1 \xrightarrow[i-1]{w_1}_{T'} q \quad (7)$$

By induction on (i), We can get from (7) that there exists a configuration $(\langle p', w'\rangle, \theta_0)$ s.t. $(p', \theta_0) \xrightarrow{w'}_T q$ is a path in the initial $\mathcal{P}$-automaton $\mathcal{A}$ and:

$$(\langle p', w'\rangle, \theta_0) \Rightarrow^* (\langle p_2, \gamma_2 w_1\rangle, \theta_1) \quad (8)$$

Thus, putting (6) and (8) together, we have that there exists a configuration $(\langle p', w'\rangle, \theta_0)$ s.t. $(p', \theta_0) \xrightarrow{w'}_T q$ is a path in the initial $\mathcal{P}$-automaton $\mathcal{A}$ and:

$$(\langle p', w'\rangle, \theta_0) \Rightarrow^* (\langle p_2, \gamma_2 w_1\rangle, \theta_1) \Rightarrow (\langle p_1, w\rangle, \theta_1) = (\langle p, w\rangle, \theta) \quad (9)$$

2. If t is the first transition added by rule β_3 i.e. t is in the form of $((p_1, \theta''), \gamma_1, q^{\theta_1}_{p_1\gamma_1})$. If this transition is new, then there are no transitions outgoing from $q^{\theta_1}_{p_1\gamma_1}$. So the only path using t is $(p_1, \theta'') \xrightarrow[i]{\gamma_1}_{T'} q^{\theta_1}_{p_1\gamma_1}$. For this path, we only need to prove part (II), and $(\langle p_1, \gamma_1\rangle, \theta_1) \Rightarrow^* (\langle p_1, \gamma_1\rangle, \theta_1)$ holds trivially.

3. Let $t = (q^{\theta_1}_{p_1\gamma_1}, \gamma'', q')$ be the second transition added by saturation rule β_3. Then there exist u, $v \in \Gamma^*$ s.t. $w = u\gamma''v$ and the current automaton contains the following path:

$$(p, \theta) \xrightarrow[i-1]{u}_{T'} q^{\theta_1}_{p_1\gamma_1} \xrightarrow[i]{\gamma''}_{T'} q' \xrightarrow[i]{v}_{T'} q \quad (10)$$

Because t was added via the saturation rule, then there exist $p_2 \in P$, $\gamma_2 \in \Gamma$ and a rule of the form

$$\langle p_2, \gamma_2\rangle \hookrightarrow \langle p_1, \gamma_1\gamma''\rangle \in \Delta \cap \theta_1 \quad (11)$$

and $\mathcal{A}_{post^*}$ contains the following path:

$$(p_2, \theta_1) \xrightarrow[i-1]{\gamma_2}_{T'} q' \xrightarrow[i]{v}_{T'} q \tag{12}$$

We apply the induction hypothesis on i and obtain that

$$(\langle p_1, \gamma_1 \rangle, \theta_1) \Rightarrow^* (\langle p, u \rangle, \theta) \tag{13}$$

We apply the induction hypothesis on i to obtain that there exists a configuration $(\langle p', w' \rangle, \theta_0)$ s.t. $(p', \theta_0) \xrightarrow{w'}_T q$ is a path in the initial $\mathcal{P}$-automaton $\mathcal{A}$ and:

$$(\langle p', w' \rangle, \theta_0) \Rightarrow^* (\langle p_2, \gamma_2 v \rangle, \theta_1) \tag{14}$$

Thus, putting (11) (13) and (14) together, we have that there exists a configuration $(\langle p', w' \rangle, \theta_0)$ s.t. $(p', \theta_0) \xrightarrow{w'}_T q$ is a path in the initial $\mathcal{P}$-automaton $\mathcal{A}$ and:

$$(\langle p', w' \rangle, \theta_0) \Rightarrow^* (\langle p_2, \gamma_2 v \rangle, \theta_1) \Rightarrow (\langle p_1, \gamma_1 \gamma'' v \rangle, \theta_1) \Rightarrow^* (\langle p, u\gamma'' v \rangle, \theta) = (\langle p, w \rangle, \theta) \tag{15}$$

□

Then we continue to prove Theorem 2.6.1:

Proof: Let $(\langle p', w' \rangle, \theta)$ be a configuration of $post^*(L(\mathcal{A}))$. Then there exists a configuration $(\langle p, w \rangle, \theta_0)$ such that there exists a path $(p, \theta_0) \xrightarrow{w}_T q$ in the initial automaton $\mathcal{A}$ and $(\langle p, w \rangle, \theta_0) \Rightarrow^* (\langle p', w' \rangle, \theta)$. By Lemma 3, we can have $(p', \theta) \xrightarrow{w'}_{T'} q$ for q is a final state of $\mathcal{A}_{post^*}$. So $(\langle p', w' \rangle, \theta)$ is recognized by $\mathcal{A}_{post^*}$.

Conversely, let $(\langle p', w' \rangle, \theta)$ be a configuration recognized by $\mathcal{A}_{post^*}$. Then there exists a path $(p', \theta) \xrightarrow{w'}_{T'} q$ in $\mathcal{A}_{post^*}$ for some final state q. By Lemma 4, since q is a final state, we have $(\langle p, w \rangle, \theta_0) \Rightarrow^*_{\mathcal{P}} (\langle p', w' \rangle, \theta)$ s.t. there exists a configuration $(\langle p, w \rangle, \theta_0)$ s.t. $(p, \theta_0) \xrightarrow{w}_{T'} q$ is a path in the initial automaton $\mathcal{A}$ i.e. $(\langle p, w \rangle, \theta_0) \in L(\mathcal{A})$. Therefore, $(\langle p', w' \rangle, \theta) \in post^*(L(\mathcal{A}))$

□

2.7 Experiments

2.7.1 Our Algorithms vs. Standard *pre** and *post** Algorithms of PDSs

We implemented our algorithms in a tool. To compare the performance of our algorithms against the approach that consists in translating the SM-PDS into an equivalent PDS or symbolic PDS and then apply the standard *post** and *pre** algorithms for PDSs and symbolic PDSs [20, 42], we first applied our tool on randomly generated SM-PDSs of various sizes. The results of the comparision using the *pre** (resp. *post**) algorithms are reported in Table 2.1 (resp. Table 2.2).

In Table 2.1, **Column** $|\Delta| + |\Delta_c|$ is the number of transitions of the SM-PDS (changing and non changing rules). **Column** SM-PDS gives the cost it takes to apply our direct algorithm to compute the *pre** for the given SM-PDS. **Column** PDS shows the cost it takes to get the equivalent PDS from the SM-PDS. **Column** Symbolic PDS reports the cost it takes to get the equivalent Symbolic PDS from the SM-PDS. **Column** Result1 reports the cost it takes to get the *pre** analysis of Moped [42] for the PDS we got. **Column** Total1 is the total cost it takes to translate the SM-PDS into a PDS and then apply the standard *pre** algorithm of Moped (Total1=PDS+Result1). **Column** Result2 reports the cost it takes to get the *pre** analysis of Moped for the symbolic PDS we got. **Column** Total2 is the total cost it takes to translate the SM-PDS into a symbolic PDS and then apply the standard *pre** algorithm of Moped (Total2=Symbolic PDS+Result2). "error" in the table means failure of Moped, because the size of the relations involved in the symbolic transitions is huge. Hence, we mark $-$ for the total execution time. You can see that our direct algorithm (**Column** SM-PDS) is much more efficient.

Table 2.2 shows the performance of our *post** algorithm. The meaning of the columns are exactly the same as for the *pre** case, but using the *post** algorithms instead. You can see from this table that applying our direct *post** algorithm on the SM-PDS is much better than translating the SM-PDS to an equivalent PDS or symbolic PDS, and then applying the standard *post** algorithms of Moped. Going through PDSs or symbolic PDSs is less efficient and leads to memory out in several cases.

$\lvert\Delta\rvert + \lvert\Delta_c\rvert$	SM-PDS	PDS	Result1	Total1	Symbolic PDS	Result2	Total2
$10 + 3$	**0.08s & 2MB**	0.15s & 3MB	0.00s	0.15s	0.10s & 2MB	0.00s	0.10s
$13 + 3$	**0.10s & 2MB**	0.15s & 3MB	0.00s	0.15s	0.10s & 2MB	0.00s	0.10s
$13 + 3$	**0.12s & 2MB**	0.15s & 3MB	0.00s	0.15s	0.10s & 2MB	0.00s	0.10s
$43 + 7$	**0.24s &3MB**	3.44s &4MB	0.02s	3.46s	4.80s &5MB	0.01s	4.81s
$110 + 10$	**0.38s &7MB**	5.15s &6MB	0.01s	5.16s	2.71s &8MB	0.00s	2.71s
$120 + 10$	**0.42s &11MB**	5.20s &15MB	0.01s	5.21s	2.79s &10MB	0.01s	2.80s
$255 + 8$	**0.65s & 15MB**	295.41s & 86MB	0.05s	295.46s	21.41s & 76MB	0.02s	21.43s
$1009 + 10$	**1.49s &97MB**	11504.2s & 117MB	2.46s	11506.66s	14.10s &471MB	1.74s	15.84s
$1899 + 7$	**2.98s & 210MB**	6538s & 171MB	4.09s	6542.09s	124.10s & 558MB	2.71s	173.71s
$2059 + 8$	**3.82s &423MB**	19525.1s &113MB	4.19s	19529.29s	20.70s &713MB	error	-
$2099 + 8$	**4.05s & 32MB**	19031s & 192MB	4.19s	19035.19s	124.12s & 757MB	error	-
$2099 + 9$	**7.08s & 252MB**	29742s & 198MB	4.28s	29746.28s	128.12s & 760MB	error	-
$3060 + 9$	**11.36s & 282MB**	29993.05s & 241MB	18.72s	30011.77s	261.07s & 610MB	error	-
$3160 + 9$	**11.99s & 285MB**	29252.05s & 257MB	26.15s	29278.2s	162.55s & 611MB	error	-
$4058 + 7$	**18.06s & 332MB**	81408.51s &307MB	92.68s	81501.19s	802.07s &1013MB	error	-
$4058 + 8$	**19.42s & 397MB**	82812.51s &399MB	91.91s	82904.42s	899.07s & 1020MB	error	-
$4158 + 8$	**21.68s &491MB**	83112.51s &401MB	97.68s	83210.19	899.19s &1021MB	error	-
$5050 + 8$	**23.26s &499MB**	93912.51s &298MB	118.12	94030.63s	205.12s &375MB	error	-

Table 2.1: Our direct pre^* algorithm vs. standard pre^* algorithms of PDSs

$\|\Delta\| + \|\Delta_c\|$	SM-PDS	PDS	Result1	Total1	Symbolic PDS	Result2	Total2
$10 + 3$	**0.12s & 2MB**	0.15s & 3MB	0.00s	0.15s	0.10s &2MB	0.00s	0.10s
$13 + 3$	**0.12s & 2MB**	0.15s &3MB	0.00s	0.15s	0.10s &2MB	0.00s	0.10s
$43 + 7$	**0.28s & 2MB**	3.44s & 4MB	0.04s	3.48s	4.80s & 5MB	0.02s	4.82s
$110 + 10$	**0.36s & 8MB**	5.15s &6MB	0.01s	5.16s	2.71s &8MB	0.00s	2.71s
$120 + 10$	**0.39s & 13MB**	5.20s &15MB	0.01s	5.21s	2.79s & 10MB	0.01s	2.80s
$255 + 8$	**0.44s & 15MB**	295.41s & 86MB	0.05s	295.46s	21.41s & 76MB	0.02s	21.43s
$1009 + 10$	**1.48s & 97MB**	11504.2s & 117MB	2.56s	11506.76s	14.10s & 471MB	1.75s	15.85s
$1899 + 7$	**3.47s & 212MB**	6538s & 171MB	3.89s	6541.89s	124.10s & 558MB	2.71s	126.81s
$2059 + 8$	**4.03s &323MB**	19525.1s & 113MB	3.99s	19528.99s	20.70s &713MB	error	-
$2099 + 8$	**4.15s &332MB**	19031s &192MB	3.99s	19034.99s	124.12s & 757MB	error	-
$2099 + 9$	**4.95s & 352MB**	29742s & 198MB	4.18s	29746.18s	128.12s & 760MB	error	-
$3060 + 9$	**5.71s & 388MB**	29993.05s & 241MB	18.12s	30011.17s	261.07s &610MB	error	-
$3160 + 9$	**5.79s & 415MB**	29252.05s & 257MB	26.10s	29278.15s	162.55s & 611MB	error	-
$4058 + 7$	**7.56s & 364MB**	81408.51s & 307MB	91.68s	81500.19s	802.07s & 1013MB	error	-
$4058 + 8$	**9.76s & 387MB**	82812.51s & 399MB	91.71s	82904.22s	899.07s & 1020MB	error	-
$4158 + 8$	**11.85s & 487MB**	83112.51s & 401MB	97.28s	83209.79s	899.19s & 1021MB	error	-
$5050 + 8$	**13.04s & 498MB**	93912.51s &498MB	112.53s	94025.04s	205.12s & 375MB	error	-

Table 2.2: Our direct *post** algorithm vs. standard *post** algorithms of PDSs

2.7.2 Malware Detection

Self-modifying code is widely used as an obfuscation technique for malware writers. Thus, we applied our tool for malware detection.

Example	SM-PDS	PDS
Email-Worm.Win32.Klez.b	Y	N
Backdoor.Win32.Allaple.b	Y	N
Email-Worm.Win32.Avron.a	Y	N
Email-Worm.Win32.Anar.a	Y	N
Email-Worm.Win32.Anar.b	Y	N
Email-Worm.Win32.Bagle.a	Y	N
Email-Worm.Win32.Bagle.am	Y	N
Email-Worm.Win32.Bagle.ao	Y	N
Email-Worm.Win32.Bagle.ap	Y	N
Email-Worm.Win32.Ardurk.d	Y	N
Email-Worm.Win32.Atak.k	Y	N
Email-Worm.Win32.Atak.g	Y	N
Email-Worm.Win32.Hanged	Y	N

Table 2.3: Malware Detection

We consider self-modifying versions of 13 well known malwares. In these versions, the malicious behaviors are unreachable if one does not take into account that the self-modifying piece of code will change the malware code: if the code does not change, the part that contains the malicious behavior cannot be reached; after executing the self-modifying code, the control point will jump to the part containing the malicious behavior.

We model such malwares in two ways: (1) first, we take into account the self-modifying piece of code and use SM-PDSs to represent these programs as discussed in Section 2.3.2, (2) second, we don't take into account that this part of the code is self-modifying and we treat it as all the other instructions of the program. In this case, we model these programs by a standard PDS following the translation of [13].

The results are reported in Table 2.3, **Column** Example reports the name of the worm. **Column** SM-PDS shows the result obtained by applying our method

to check the reachability of the entry point of the malicious block. **Column** PDS gives the result if we apply the traditional PDS translation of programs (without taking into account the semantics of self modifying code) method to check the reachability of the entry point of the malicious block. Y stands for yes (the program is malicious) and N stands for no (the program is benign). As it can be seen, our techniques that go through SM-PDS to model self modifying code is able to conclude that the entry point of the malicious block is reachable, whereas the standard PDS translation from programs fails to reach this conclusion.

3

LTL Model-Checking of Self-modifying Code

In this chapter, we consider the LTL model-checking problem of SM-PDSs. We reduce this problem to the emptiness problem of Self-modifying Büchi PushDown Systems (SM-BPDSs).

3.1 LTL Model-Checking of SM-PDSs

3.1.1 The linear-time temporal logic LTL

Let At be a finite set of atomic propositions. LTL formulas are defined as follows (where $A \in At$):

$$\varphi := \ A \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid X\varphi \mid \varphi_1 U \varphi_2$$

Formulae are interpreted on infinite words over 2^{At}. Let $\omega = \omega^0\omega^1...$ be an infinite word over 2^{At}. We write ω_i for the suffix of ω starting at ω^i. We denote $\omega \models \varphi$ to express that ω satisfies a formula φ:

$\omega \models A \iff A \in \omega^0$
$\omega \models \neg\varphi \iff \omega \not\models \varphi$
$\omega \models \varphi_1 \vee \varphi_2 \iff \omega \models \varphi_1$ or $\omega \models \varphi_2$
$\omega \models X\varphi \iff \omega_1 \models \varphi$
$\omega \models \varphi_1 U \varphi_2 \iff \exists i \geq 0, \omega_i \models \varphi_2$ and $\forall 0 \leq j < i, \omega_j \models \varphi_1$

The temporal operators G (globally) and F (eventually) are defined as follows: $F\varphi = (A \vee \neg A)U\varphi$ and $G\varphi = \neg F \neg\varphi$. Let $W(\varphi)$ be the set of infinite words that

satisfy an LTL formula φ. It is well known that $W(\varphi)$ can be accepted by Büchi automata:

Definition 4 *A Büchi automaton $\mathcal{B}$ is a quintuple $(Q, \Gamma, \eta, q_0, F)$ where Q is a finite set of states, Γ is a finite input alphabet, $\eta \subseteq (Q \times \Gamma \times Q)$ is a set of transitions, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is the set of accepting states. A run of $\mathcal{B}$ on a word $\gamma_0\gamma_1... \in \Gamma^\omega$ is a sequence of states $q_0q_1q_2...$ s.t. $\forall i \geq 0, (q_i, \gamma_i, q_{i+1}) \in \eta$. An infinite word ω is accepted by $\mathcal{B}$ if $\mathcal{B}$ has a run on ω that starts at q_0 and visits accepting states from F infinitely often.*

Theorem 3.1.1 *[30] Given an LTL formula φ, one can effectively construct a Büchi automaton $\mathcal{B}_\varphi$ which accepts $W(\varphi)$.*

3.1.2 Self Modifying Büchi Pushdown Systems

Definition 5 *A Self Modifying Büchi Pushdown Systems (SM-BPDS) is a tuple $\mathcal{BP} = (P, \Gamma, \Delta, \Delta_c, G)$ where P is a set of control locations, $G \subseteq P$ is a set of accepting control locations, $\Delta \subseteq (P \times \Gamma) \times (P \times \Gamma^*)$ is a finite set of transition rules, and $\Delta_c \subseteq P \times 2^{\Delta \cup \Delta_c} \times 2^{\Delta \cup \Delta_c} \times P$ is a finite set of modifying transition rules in the form $p \xhookrightarrow{(\sigma,\sigma')} p'$ where $\sigma, \sigma' \subseteq \Delta \cup \Delta_c$.*

Let $\Rightarrow_{\mathcal{BP}}$ be the transition relation between configurations as follows: Let $\theta \subseteq \Delta \cup \Delta_c, \gamma \in \Gamma, w \in \Gamma^*$, and $p \in P$, then

1. *If $r : \langle p, \gamma \rangle \hookrightarrow \langle p', w' \rangle \in \Delta$ and $r \in \theta$, then $(\langle p, \gamma w \rangle, \theta) \Rightarrow_{\mathcal{BP}} (\langle p', w'w \rangle, \theta)$.*

2. *If $r : p \xhookrightarrow{(\sigma,\sigma')} p' \in \Delta_c$, $\sigma \cap \theta \neq \emptyset$ and $r \in \theta$, then $(\langle p, \gamma w \rangle, \theta) \Rightarrow_{\mathcal{BP}} (\langle p', \gamma w \rangle, \theta')$ where $\theta' = \theta \backslash \sigma \cup \sigma'$.*

A run π of $\mathcal{BP}$ is a sequence of configurations $\pi = c_0c_1...$ s.t. $c_i \Rightarrow_{\mathcal{BP}} c_{i+1}$ for every $i \geq 0$. π is accepting iff it infinitely often visits configurations having control locations in G.

Let c and c' be two configurations of the SM-BPDS $\mathcal{BP}$. The relation $\Rightarrow^r_{\mathcal{BP}}$ is defined as follows: $c \Rightarrow^r_{\mathcal{BP}} c'$ iff there exists a configuration $(\langle g, u \rangle, \theta)$, $g \in G$ s.t. $c \Rightarrow^*_{\mathcal{BP}} (\langle g, u \rangle, \theta) \Rightarrow^+_{\mathcal{BP}} c'$. We remove the subscript $\mathcal{BP}$ when it is clear from the context. We define $\stackrel{i}{\Rightarrow}$ as follows: $c \stackrel{i}{\Rightarrow} c'$ iff there exists a sequence of configurations $c_0 \Rightarrow_{\mathcal{BP}} c_1 \Rightarrow_{\mathcal{BP}} ... \Rightarrow_{\mathcal{BP}} c_i$ s.t. $c_0 = c$ and $c_i = c'$.

A head of SM-BPDS is a tuple $(\langle p, \gamma\rangle, \theta)$ *where* $p \in P$, $\gamma \in \Gamma$ *and* $\theta \subseteq \Delta \cup \Delta_c$. *A head* $((p, \gamma), \theta)$ *is repeating if there exists* $v \in \Gamma^*$ *such that* $(\langle p, \gamma\rangle, \theta) \Rightarrow^r_{\mathcal{BP}} (\langle p, \gamma v\rangle, \theta)$. *The set of repeating heads of SM-BPDS is called* $Rep_{\mathcal{BP}}$.

We assume w.l.o.g. that for every rule in Δ_c of the form $r : p \xhookrightarrow{(\sigma,\sigma')} p'$, $r \notin \sigma$.

3.1.3 From LTL Model-Checking of SM-PDSs to the emptiness problem of SM-BPDSs

Let $\mathcal{P} = (P, \Gamma, \Delta, \Delta_c)$ be a self modifying pushdown system. Let At be a set of atomic propositions. Let $\nu : P \to 2^{At}$ be a labelling function. Let $\pi = (\langle p_0, w_0\rangle, \theta_0)(\langle p_1, w_1\rangle, \theta_1)...$ be an execution of the SM-PDS $\mathcal{P}$. Let φ be an LTL formula over the set of atomic propositions At. We say that

$$\pi \models_\nu \varphi \text{ iff } \nu(p_0)\nu(p_1)\cdots \models \varphi$$

Let $(\langle p, w\rangle, \theta)$ be a configuration of $\mathcal{P}$. We say that $(\langle p, w\rangle, \theta) \models_\nu \varphi$ iff $\mathcal{P}$ has a path π starting at $(\langle p, w\rangle, \theta)$ such that $\pi \models_\nu \varphi$.

Our goal in this chapter is to perform LTL model-checking for self-modifying pushdown systems. Since SM-PDSs can be translated to standard (symbolic) pushdown systems, one way to solve this LTL model-checking problem is to compute the (symbolic) pushdown system that is equivalent to the SM-PDS, and then apply the standard LTL model-checking algorithms on standard PDSs [42]. However, this approach is not efficient (as will be witnessed later in the experiments). Thus, we need a *direct* approach that performs LTL model-checking on the SM-PDS, without translating it to an equivalent PDS. Let $\mathcal{B}_\varphi = (Q, 2^{At}, \eta, q_0, F)$ be a Büchi automaton that accepts $W(\varphi)$. We compute the SM-BPDS $\mathcal{BP}_\varphi = (P \times Q, \Gamma, \Delta', \Delta'_c, G)$ by performing a kind of product between the SM-PDS $\mathcal{P}$ and the Büchi automaton $\mathcal{B}_\varphi$ as follows:

1. if $r = \langle p, \gamma\rangle \hookrightarrow \langle p', w\rangle \in \Delta$ and $(q, \nu(p), q') \in \eta$, then $\langle (p, q), \gamma\rangle \hookrightarrow \langle (p', q'), w\rangle \in \Delta'$. Let $prod(r)$ be the set of rules of Δ' obtained from the rule r, i.e., rules of Δ' of the form $\langle (p, q), \gamma\rangle \hookrightarrow \langle (p', q'), w\rangle$.

2. if a rule $r = p \xhookrightarrow{(r_1,r_2)} p' \in \Delta_c$ and $(q, \nu(p), q') \in \eta$, then $(p,q) \xhookrightarrow{(\sigma,\sigma')} (p',q') \in \Delta'_c$ where $\sigma = prod(r_1), \sigma' = prod(r_2)$. Let $prod(r)$ be the set of rules of Δ' obtained from the rule r, i.e., rules of Δ'_c of the form $(p,q) \xhookrightarrow{(\sigma,\sigma')} (p',q')$.

3. $G = P \times F$.

We can show that:

Theorem 3.1.2 *Let $(\langle p, w\rangle, \theta)$ be a configuration of the SM-PDS $\mathcal{P}$. $(\langle p, w\rangle, \theta) \models_\nu \varphi$ iff $\mathcal{BP}_\varphi$ has an accepting run from $(\langle (p, q_0), w\rangle, prod(\theta))$ where $prod(\theta)$ is the set of rules of $\Delta \cup \Delta_c$ obtained from the rules of θ as described above.*

Thus, LTL model-checking for SM-PDSs can be reduced to checking whether a SM-BPDS has an accepting run. The rest of the chapter is devoted to this problem.

3.2 The Emptiness Problem of SM-BPDSs

From now on, we fix a SM-BPDS $\mathcal{BP} = (P, \Gamma, \Delta, \Delta_c, G)$. We can show that $\mathcal{BP}$ has an accepting run starting from a configuration c if and only if from c, it can reach a configuration with a repeating head:

Proposition 3 *A SM-BPDS $\mathcal{BP}$ has an accepting run starting from a configuration c if and only if there exists a repeating head $((p,\gamma),\theta)$ such that $c \Rightarrow^*_{\mathcal{BP}} (\langle p, \gamma w\rangle, \theta)$ for some $w \in \Gamma^*$.*

Proof: “ $\Rightarrow$ ”: Let $\sigma = c_0 c_1 ...$ be an accepting run starting at configuration c where $c_0 = c$ and $c_i = (\langle p_i, w_i\rangle, \theta_i)$. We construct an increasing sequence of indices $i_0, i_1 ...$ with a property that once any of the configurations c_{i_k} is reached, the rest of the run never changes the bottom $|w_{i_k}|-1$ elements of the stack anymore. This property can be written as follows:

$$|w_{i_0}| = \min\{|w_j| \mid j \geq 0\}$$

$$|w_{i_k}| = \min\{|w_j| \mid j > i_{k-1}\}, k \geq 1$$

Because $\mathcal{BP}$ has only finitely many different heads, there must be a head $(\langle p, \gamma \rangle, \theta)$ which occurs infinitely often as a head in the sequence $c_{i_0} c_{i_1} \dots$. Moreover, as some $g \in G$ becomes a control location infinitely often, we can find a subsequence of indices $i_{j_0}, i_{j_1}, \dots$ with the following property: for every $k \geq 1$, there exist $v, w \in \Gamma^*$

$$c_{i_{j_k}} = (\langle p, \gamma w \rangle, \theta) \Rightarrow^r (\langle p, \gamma v w \rangle, \theta) = c_{i_{j_{k+1}}}$$

Because w is never looked at or changed in this path, we can have $(\langle p, \gamma \rangle, \theta) \Rightarrow^r (\langle p, \gamma v \rangle, \theta)$. This proves this direction of the proposition.

" $\Leftarrow$ ": Because $(\langle p, \gamma \rangle, \theta)$ is a repeating head, we can construct the following run for some $u, v, w \in \Gamma^*$, $\theta' \subseteq (\Delta \cup \Delta_c)$ and $g \in G$:

$$c \Rightarrow^* (\langle p, \gamma w \rangle, \theta) \Rightarrow^* (\langle g, uw \rangle, \theta') \Rightarrow^+ (\langle p, \gamma v w \rangle, \theta) \Rightarrow^* (\langle g, uvw \rangle, \theta') \Rightarrow^+ (\langle p, \gamma v v w \rangle, \theta) \Rightarrow^* \dots$$

Since g occurs infinitely often, the run is accepting. □

Thus, since there exists an efficient algorithm to compute the *pre** of SM-PDSs [46], the emptiness problem of a SM-BPDS can be reduced to computing its repeating heads.

3.2.1 The Head Reachability Graph $\mathcal{G}$

Our goal is to compute the set of repeating heads $Rep_{\mathcal{BP}}$, i.e., the set of heads $(\langle p, \gamma \rangle, \theta)$ such that there exists $v \in \Gamma^*$, $(\langle p, \gamma \rangle, \theta) \Rightarrow^r (\langle p, \gamma v \rangle, \theta)$. I.e., $(\langle p, \gamma \rangle, \theta) \Rightarrow^* (\langle p, \gamma v \rangle, \theta)$ s.t. this path goes through an accepting location in G. To this aim, we will compute a finite graph $\mathcal{G}$ whose nodes are the heads of $\mathcal{BP}$ of the form $((p, \gamma), \theta)$, where $p \in P$, $\gamma \in \Gamma$ and $\theta \subseteq \Delta \cup \Delta_c$; and whose edges encode the reachability relation between these heads. More precisely, given two heads $((p, \gamma), \theta)$ and $((p', \gamma'), \theta')$, $((p, \gamma), \theta) \xrightarrow{b} ((p', \gamma'), \theta')$ is an edge of the graph $\mathcal{G}$ means that the configuration $(\langle p, \gamma \rangle, \theta)$ can reach a configuration having $(\langle p', \gamma' \rangle, \theta')$ as head, i.e., it means that there exists $v \in \Gamma^*$ s.t. $(\langle p, \gamma \rangle, \theta) \Rightarrow^* (\langle p', \gamma' v \rangle, \theta')$. Moreover, we need to keep the information whether this path visits an accepting location in G or not. This information is recorded in the label of the edge b: $b = 1$ means that the path visits an accepting location in G, i.e. that $(\langle p, \gamma \rangle, \theta) \Rightarrow^r (\langle p', \gamma' v \rangle, \theta')$. Otherwise, $b = 0$. Therefore, if the graph $\mathcal{G}$ contains a loop from a head $((p, \gamma), \theta)$ to itself such that this loop goes through an edge labelled by 1, then $((p, \gamma), \theta)$

is a repeating head. Thus, computing $Rep_{\mathcal{BP}}$ can be reduced to computing the graph $\mathcal{G}$ and finding 1-labelled loops in this graph.

More precisely, we define the head reachability graph $\mathcal{G}$ as follows:

Definition 6 *The head reachability graph $\mathcal{G}$ is a tuple $(P \times \Gamma \times 2^{\Delta \cup \Delta_c}, \{0,1\}, \delta)$ such that $((p,\gamma),\theta) \xrightarrow{b} ((p',\gamma'),\theta')$ is an edge of δ iff:*

1. *there exists a transition $r_c : p \xhookrightarrow{(\sigma,\sigma')} p' \in \theta \cap \Delta_c$, $\gamma = \gamma'$, $\theta' = \theta \setminus \sigma \cup \sigma'$, and $b = 1$ iff $p \in G$;*

2. *there exists a transition $\langle p,\gamma \rangle \hookrightarrow \langle p',\gamma' \rangle \in \theta \cap \Delta, \theta = \theta'$ and $b = 1$ iff $p \in G$;*

3. *there exists a transition $\langle p,\gamma \rangle \hookrightarrow \langle p'',\gamma_1\gamma' \rangle \in \theta \cap \Delta$, for $\gamma_1 \in \Gamma$, $p'' \in P$, s.t. $(\langle p'',\gamma_1 \rangle,\theta) \Rightarrow^*_{\mathcal{BP}} (\langle p',\epsilon \rangle,\theta')$, and $b = 1$ iff $p \in G$ or $(\langle p'',\gamma_1 \rangle,\theta) \Rightarrow^r_{\mathcal{BP}} (\langle p',\epsilon \rangle,\theta')$*

Let $\mathcal{G}$ be the head reachability graph. We define $\underset{i}{\rightarrow}$ as follows: let $((p,\gamma),\theta)$ and $((p',\gamma'),\theta')$ be two heads of $\mathcal{BP}$. We write $((p,\gamma),\theta) \underset{i}{\rightarrow} ((p',\gamma'),\theta')$ iff $\exists$ booleans $b_1, b_2 ... b_i \in \{0,1\}$, $\exists$ heads $((p_j,\gamma_j),\theta_j), 0 \leq j \leq i$ s.t. $\mathcal{G}$ contains the following path $((p_0,\gamma_0),\theta_0) \xrightarrow{b_1} ((p_1,\gamma_1),\theta_1) \xrightarrow{b_2} ... \xrightarrow{b_i} ((p_i,\gamma_i),\theta_i)$ where $((p_0,\gamma_0),\theta_0) = ((p,\gamma),\theta)$ and $((p_i,\gamma_i),\theta_i) = ((p',\gamma'),\theta')$.

Let $\rightarrow^*$ be the reflexive transitive closure of the graph relation $\xrightarrow{b}$, and let $\rightarrow^r$ be defined as follows: Given two heads $((p,\gamma),\theta)$ and $((p',\gamma'),\theta')$, $((p,\gamma),\theta) \rightarrow^r ((p',\gamma'),\theta')$ iff there is in $\mathcal{G}$ a path between $((p,\gamma),\theta)$ and $((p',\gamma'),\theta')$ that goes through a 1-labelled edge, i.e., iff there exist heads $((p_1,\gamma_1),\theta_1)$ and $((p_2,\gamma_2),\theta_2)$ s.t. $((p,\gamma),\theta) \rightarrow^* ((p_1,\gamma_1),\theta_1) \xrightarrow{1} ((p_2,\gamma_2),\theta_2) \rightarrow^* ((p',\gamma'),\theta')$.

We can show that:

Theorem 3.2.1 *Let $\mathcal{BP} = (P,\Gamma,\Delta,\Delta_c,G)$ be a self-modifying Büchi pushdown system, and let $\mathcal{G}$ be its corresponding head reachability graph. A head $((p,\gamma),\theta)$ of $\mathcal{BP}$ is repeating iff $\mathcal{G}$ has a loop on the node $((p,\gamma),\theta)$ that goes through a 1-labeled edge.*

To prove this theorem, we first need to prove the following lemma:

Lemma 5 *The relations $\rightarrow^*$ and $\rightarrow^r$ have the following properties: For any heads $((p,\gamma),\theta_1)$ and $((p',\gamma'),\theta_2)$:*

(a) $((p,\gamma),\theta_1) \rightarrow^* ((p',\gamma'),\theta_2)$ *iff* $(\langle p,\gamma\rangle,\theta_1) \Rightarrow^* (\langle p',\gamma' v\rangle,\theta_2)$ *for some* $v \in \Gamma^*$.

(b) $((p,\gamma),\theta_1) \rightarrow^r ((p',\gamma'),\theta_2)$ *iff* $(\langle p,\gamma\rangle,\theta_1) \Rightarrow^r (\langle p',\gamma' v\rangle,\theta_2)$ *for some* $v \in \Gamma^*$.

Proof: "$\Rightarrow$": Assume $((p,\gamma),\theta_1) \underset{i}{\rightarrow} ((p',\gamma'),\theta_2)$. We proceed by induction on i.

(a) **Basis.** $i = 0$. In this case, $((p,\gamma),\theta_1) = ((p',\gamma'),\theta_2)$, then we can get $(\langle p,\gamma\rangle,\theta_1) \Rightarrow^* (\langle p,\gamma\rangle,\theta_1) = (\langle p',\gamma'\rangle,\theta_2)$

Step. $i > 0$. Then there exist $p_1 \in P, \gamma'' \in \Gamma^*$ and $\theta' \subseteq \Delta \cup \Delta_c$ such that $((p,\gamma),\theta_1) \underset{1}{\rightarrow} ((p_1,\gamma''),\theta') \underset{i-1}{\longrightarrow} ((p',\gamma'),\theta_2)$. From the induction hypothesis, there exists $u \in \Gamma^*$ such that $(\langle p_1,\gamma''\rangle,\theta') \Rightarrow^* (\langle p',\gamma' u\rangle,\theta_2)$

Since $((p,\gamma),\theta_1) \rightarrow ((p_1,\gamma''),\theta')$, we have $(\langle p,\gamma\rangle,\theta_1) \Rightarrow^* (\langle p_1,\gamma'' w\rangle,\theta')$ for $w \in \Gamma^*$, hence $(\langle p,\gamma\rangle,\theta_1) \Rightarrow^* (\langle p',\gamma' uw\rangle,\theta_2)$.

The property holds.

(b) $((p,\gamma),\theta_1) \rightarrow^r ((p,\gamma),\theta_1)$ cannot hold for the case $i = 0$.

Basis. $i = 1$. In this case, $((p,\gamma),\theta_1) \rightarrow^r ((p',\gamma'),\theta_2)$, then we can get $p \in G$ and $(\langle p,\gamma\rangle,\theta_1) \Rightarrow^r (\langle p',\gamma'\rangle,\theta_2)$. The property holds.

Step. $i > 0$. As done in the proof of part (a) of this lemma, there exists $p_1, \gamma'' \in \Gamma, \theta'' \subseteq \Delta \cup \Delta_c$ s.t. $((p,\gamma),\theta_1) \underset{1}{\rightarrow} ((p_1,\gamma''),\theta') \underset{i-1}{\longrightarrow} ((p',\gamma'),\theta_2)$. Then if $((p,\gamma),\theta_1) \rightarrow^r ((p',\gamma'),\theta_2)$, either $((p_1,\gamma''),\theta') \rightarrow^r ((p',\gamma'),\theta_2)$ or $((p,\gamma),\theta_1) \xrightarrow{1} ((p_1,\gamma''),\theta')$ holds. In the first case i.e. $((p_1,\gamma''),\theta') \rightarrow^r ((p',\gamma'),\theta_2)$, by the induction hypothesis, we can have $(\langle p_1,\gamma''\rangle,\theta') \Rightarrow^r (\langle p',\gamma' u\rangle,\theta_2)$, hence, $(\langle p,\gamma\rangle,\theta_1) \Rightarrow^r (\langle p',\gamma' u\rangle,\theta_2)$ holds

The second case depends on the rule applied to get $((p,\gamma),\theta_1) \xrightarrow{1} ((p_1,\gamma''),\theta')$ according to Definition 6.

- If this edge corresponds to a transition $r_c : p \xhookrightarrow{(\sigma,\sigma')} p_1 \in \theta_1$, then $\gamma = \gamma'', \theta' = \theta_1 \backslash \sigma \cup \sigma'$ and $p \in G$. Since we can obtain $(\langle p,\gamma\rangle,\theta_1) \Rightarrow_{\mathcal{BP}} (\langle p_1,\gamma\rangle,\theta') \Rightarrow^* (\langle p',\gamma' uw\rangle,\theta_2)$ from part (a) and $p \in G$, then $(\langle p,\gamma\rangle,\theta_1) \Rightarrow^r (\langle p_1,\gamma\rangle,\theta') \Rightarrow^* (\langle p',\gamma' uw\rangle,\theta_2)$. This implies that $(\langle p,\gamma\rangle,\theta_1) \Rightarrow^r (\langle p',\gamma' v\rangle,\theta_2)$ for some $v \in \Gamma^*$.

- If this edge corresponds to a transition $r : \langle p, \gamma\rangle \hookrightarrow \langle p_1, \gamma''\rangle \in \theta_1 \cap \Delta$, then $\theta' = \theta_1$ and $p \in G$. Since we can obtain $(\langle p, \gamma\rangle, \theta_1) \Rightarrow_{\mathcal{BP}} (\langle p_1, \gamma''\rangle, \theta_1) \Rightarrow^* (\langle p', \gamma' uw\rangle, \theta_2)$ from part (a) and $p \in G$, then

$$(\langle p, \gamma\rangle, \theta_1) \Rightarrow^r (\langle p_1, \gamma''\rangle, \theta_1) \Rightarrow^* (\langle p', \gamma' uw\rangle, \theta_2).$$

  This implies that $(\langle p, \gamma\rangle, \theta_1) \Rightarrow^r (\langle p', \gamma' v\rangle, \theta_2)$ for some $v \in \Gamma^*$.

- If this edge corresponds to a transition $r : \langle p, \gamma\rangle \hookrightarrow \langle p'', \gamma_1\gamma''\rangle \in \theta_1$, then either $p \in G$ or $(\langle p'', \gamma_1\rangle, \theta_1) \Rightarrow^r (\langle p_1, \epsilon\rangle, \theta')$ holds. If $p \in G$, then we have $(\langle p, \gamma\rangle, \theta_1) \Rightarrow^r (\langle p'', \gamma_1\gamma''\rangle, \theta_1)$. Otherwise, $(\langle p'', v_1\gamma'' w\rangle, \theta_1) \Rightarrow^r (\langle p_1, \gamma'' w\rangle, \theta')$. Since we can obtain $(\langle p_1, \gamma''\rangle, \theta') \Rightarrow^* (\langle p', \gamma' u\rangle, \theta_2)$ from part (a). Therefore, $(\langle p, \gamma\rangle, \theta_1) \Rightarrow^r (\langle p_1, \gamma''\rangle, \theta') \Rightarrow^* (\langle p', \gamma' u\rangle, \theta_2)$. This implies that $(\langle p, \gamma\rangle, \theta_1) \Rightarrow^r (\langle p', \gamma' v\rangle, \theta_2)$ for some $v \in \Gamma^*$.

'$\Leftarrow$": Assume $(\langle p, \gamma\rangle, \theta_1) \overset{i}{\Rightarrow} (\langle p', \gamma' v\rangle, \theta_2)$. We proceed by induction on i.

(a) **Basis.** $i = 0$. In this case, $v = \epsilon$ and $(\langle p, \gamma\rangle, \theta_1) = (\langle p', \gamma'\rangle, \theta_2)$, then $((p, \gamma), \theta_1) \rightarrow^* ((p', \gamma'), \theta_2)$ holds.

Step. $i > 0$. Then there exist $p_1 \in P, u \in \Gamma^*$ and $\theta' \subseteq \Delta \cup \Delta_c$ such that $(\langle p, \gamma\rangle, \theta_1) \overset{1}{\Rightarrow} (\langle p_1, u\rangle, \theta') \overset{i-1}{\Rightarrow} (\langle p', \gamma' v\rangle, \theta_2)$. There are 2 cases:

1. Case $\theta' = \theta_1$: There must exist a rule $r : \langle p, \gamma\rangle \hookrightarrow \langle p_1, u\rangle \in \Delta$ such that $r \in \theta'$ and $|u| \geq 1$. Let l denote the minimal length of the stack on the path from $(\langle p_1, u\rangle, \theta_1)$ to $(\langle p', \gamma' v\rangle, \theta_2)$. Then u can be written as $u''\gamma_1 u'$ where $|u'| = l-1$ (that means u' will remain on the stack for the path). Furthermore, there exists p''' such that $(\langle p_1, u''\rangle, \theta_1) \Rightarrow^* (\langle p''', \epsilon\rangle, \theta'')$ for some $\theta'' \subseteq (\Delta_c \cup \Delta)$. We have $(\langle p, \gamma\rangle, \theta_1) \overset{k}{\Rightarrow} (\langle p''', \gamma_1 u'\rangle, \theta'')$ for $k < i$. By the induction on i, we have $((p, \gamma), \theta_1) \rightarrow^* ((p''', \gamma_1), \theta'')$. Because u' has to remain on the stack for the rest of the path, v is of the form $v'u'$ for some $v' \in \Gamma^*$. That means $(\langle p''', \gamma_1\rangle, \theta'') \overset{j}{\Rightarrow} (\langle p', \gamma' v'\rangle, \theta_2)$ for $j < i$. By the induction hypothesis, $((p''', \gamma_1), \theta'') \rightarrow^* ((p', \gamma'), \theta_2)$ holds. Moreover, we have $((p, \gamma), \theta_1) \rightarrow^* ((p''', \gamma_1), \theta'')$, hence $((p, \gamma), \theta_1) \rightarrow^* ((p', \gamma'), \theta_2)$.

2. Case $\theta' \neq \theta_1$: There must be a rule $r_c : p \xrightarrow{(\sigma,\sigma')} p_1 \in \Delta_c$ such that $r_c \in \theta_1$ and $\sigma \cap \theta_1 \neq \emptyset$, then $\theta' = \theta_1 \setminus \sigma \cup \sigma'$. After the execution of r_c, the content of the stack will remain the same, thus, $u = \gamma$. Then $(\langle p, \gamma \rangle, \theta_1) \overset{1}{\Rightarrow} (\langle p_1, \gamma \rangle, \theta') \overset{i-1}{\Rightarrow} (\langle p', \gamma' v \rangle, \theta_2)$. By the induction hypothesis to $(\langle p_1, \gamma \rangle, \theta') \overset{i-1}{\Rightarrow} (\langle p', \gamma' v \rangle, \theta_2)$, we can obtain that $((p_1, \gamma), \theta') \rightarrow^* ((p', \gamma'), \theta_2)$. Since $(\langle p, \gamma \rangle, \theta_1) \overset{1}{\Rightarrow} (\langle p_1, \gamma \rangle, \theta')$, then we can have a path $((p, \gamma), \theta_1) \rightarrow ((p_1, \gamma), \theta') \rightarrow^* ((p', \gamma'), \theta_2)$ that implies $((p, \gamma), \theta_1) \rightarrow^* ((p', \gamma'), \theta_2)$. The property holds.

(b) $(\langle p, \gamma \rangle, \theta_1) \Rightarrow^r (\langle p, \gamma' v \rangle, \theta_1)$ is impossible in 0 steps.

Basis. $i = 1$. $(\langle p, \gamma \rangle, \theta_1) \Rightarrow^r (\langle p, \gamma \rangle, \theta_1)$, then $p \in G$. Thus, $((p, \gamma), \theta_1) \rightarrow^r ((p, \gamma), \theta_1)$ holds.

Step. $i > 1$. $(\langle p, \gamma \rangle, \theta_1) \Rightarrow^r (\langle p', \gamma' v \rangle, \theta_2)$ holds, then there exist $p_1 \in P, u \in \Gamma^*$ and $\theta' \subseteq \Delta \cup \Delta_c$ such that $(\langle p, \gamma \rangle, \theta_1) \overset{1}{\Rightarrow} (\langle p_1, u \rangle, \theta') \overset{i-1}{\Rightarrow} (\langle p', \gamma' v \rangle, \theta_2)$. Thus, either $(\langle p, \gamma \rangle, \theta_1) \Rightarrow^r (\langle p_1, u \rangle, \theta')$ or $(\langle p_1, u \rangle, \theta') \Rightarrow^r (\langle p', \gamma' v \rangle, \theta_2)$ holds.

The first case implies $p \in G$. There are 2 cases:

1. Case $\theta' = \theta_1$: then as in the previous proof of part (a), we can have a path $((p, \gamma), \theta_1) \rightarrow^* ((p''', \gamma_1), \theta'') \rightarrow^* ((p', \gamma'), \theta_2)$. Since $p \in G$, we get by Definition 6 $((p, \gamma), \theta_1) \rightarrow^* ((p''', \gamma_1), \theta'') \rightarrow^* ((p', \gamma'), \theta_2)$. Thus, we have that $((p, \gamma), \theta_1) \rightarrow^r ((p', \gamma'), \theta_2)$. The property holds.
2. Case $\theta' \neq \theta_1$: then as in the previous proof of part (a), we can have a path $((p, \gamma), \theta_1) \rightarrow ((p_1, \gamma), \theta') \rightarrow^* ((p', \gamma'), \theta_2)$. Since $p \in G$, we get $((p, \gamma), \theta_1) \overset{1}{\rightarrow} ((p_1, \gamma), \theta') \rightarrow^* ((p', \gamma'), \theta_2)$. Thus, we have that $((p, \gamma), \theta_1) \rightarrow^r ((p', \gamma'), \theta_2)$. The property holds.

In the second case, $(\langle p_1, u \rangle, \theta') \Rightarrow^r (\langle p', \gamma' v \rangle, \theta_2)$ holds. As previously, there are 2 cases:

1. Case $\theta' = \theta_1$: then as in case (a) we have $(\langle p_1, u \rangle, \theta_1) \Rightarrow^* (\langle p''', \gamma_1 u' \rangle, \theta'')$ and $(\langle p''', \gamma_1 \rangle, \theta'') \Rightarrow^* (\langle p', \gamma' v' \rangle, \theta_2)$. If $(\langle p_1, u \rangle, \theta_1) \Rightarrow^r (\langle p', \gamma' v \rangle, \theta_2)$, then either

$(\langle p_1, u\rangle, \theta_1) \Rightarrow^r (\langle p''', \gamma_1 u'\rangle, \theta'')$ or $(\langle p''', \gamma_1\rangle, \theta'') \Rightarrow^r (\langle p', \gamma' v'\rangle, \theta_2)$.

- If $(\langle p_1, u\rangle, \theta_1) \Rightarrow^r (\langle p''', \gamma_1 u'\rangle, \theta'')$, let $u'' \in \Gamma^*$ s.t. $u = u''\gamma_1 u'$ and $(\langle p_1, u''\rangle, \theta_1) \Rightarrow^r (\langle p''', \epsilon\rangle, \theta'')$, then, we have $((p, \gamma), \theta_1) \rightarrow^r ((p''', \gamma_1), \theta'')$. We have $(\langle p, \gamma\rangle, \theta_1) \overset{k}{\Rightarrow} (\langle p''', \gamma_1 u'\rangle, \theta'')$ for $k < i$. By the induction on i, we have $((p, \gamma), \theta_1) \rightarrow^* ((p''', \gamma_1), \theta'')$. Because u' has to remain on the stack for the rest of the path, v is of the form $v'u'$ for some $v' \in \Gamma^*$.
  That means $(\langle p''', \gamma_1\rangle, \theta'') \overset{j}{\Rightarrow} (\langle p', \gamma' v'\rangle, \theta_2)$ for $j < i$. By the induction hypothesis, $((p''', \gamma_1), \theta'') \rightarrow^* ((p', \gamma'), \theta_2)$ holds. Moreover, we have $((p, \gamma), \theta_1) \rightarrow^* ((p''', \gamma_1), \theta'')$, hence $((p, \gamma), \theta_1) \rightarrow^* ((p', \gamma'), \theta_2)$. So we can have a path $((p, \gamma), \theta_1) \rightarrow^* ((p''', \gamma_1), \theta'') \rightarrow^* ((p', \gamma'), \theta_2)$, thus we have that $((p, \gamma), \theta_1) \rightarrow^r ((p', \gamma'), \theta_2)$;
- If $(\langle p''', \gamma_1\rangle, \theta'') \Rightarrow^r (\langle p', \gamma' v'\rangle, \theta_2)$, then by the induction hypothesis we have $((p''', \gamma_1), \theta'') \rightarrow^r ((p', \gamma'), \theta_2)$. Thus, we can have a path $((p, \gamma), \theta_1) \rightarrow^* ((p''', \gamma_1), \theta'') \rightarrow^* ((p', \gamma'), \theta_2)$, then we have that $((p, \gamma), \theta_1) \rightarrow^r ((p', \gamma'), \theta_2)$;

2. Case $\theta' \neq \theta_1$: then $(\langle p_1, \gamma\rangle, \theta') \Rightarrow^r (\langle p', \gamma' v\rangle, \theta_2)$. By the induction hypothesis we have $((p_1, \gamma), \theta') \rightarrow^r ((p', \gamma'), \theta_2)$. Since $(\langle p, \gamma\rangle, \theta_1) \overset{1}{\Rightarrow} (\langle p_1, \gamma\rangle, \theta') \overset{i-1}{\Rightarrow} (\langle p', \gamma' v\rangle, \theta_2)$.
   By the induction hypothesis to $(\langle p_1, \gamma\rangle, \theta') \overset{i-1}{\Rightarrow} (\langle p', \gamma' v\rangle, \theta_2)$, we can obtain that $((p_1, \gamma), \theta') \rightarrow^* ((p', \gamma'), \theta_2)$. Since $(\langle p, \gamma\rangle, \theta_1) \overset{1}{\Rightarrow} (\langle p_1, \gamma\rangle, \theta')$, then we can have a path $((p, \gamma), \theta_1) \rightarrow ((p_1, \gamma), \theta') \rightarrow^* ((p', \gamma'), \theta_2)$. Thus, we have that $((p, \gamma), \theta_1) \rightarrow^r ((p', \gamma'), \theta_2)$;

Thus, the property holds.

□

Proof of Theorem 3.2.1

We can now prove Theorem 3.2.1.

Proof: Let $((p, \gamma), \theta)$ be a repeating head, then there exists some $v \in \Gamma^*, \theta \subseteq \Delta_c \cup \Delta$ such that $(\langle p, \gamma\rangle, \theta) \Rightarrow^r (\langle p, \gamma v\rangle, \theta)$. By Lemma 5, this is the case if and only if $((p, \gamma), \theta) \rightarrow^r ((p, \gamma), \theta)$. From the definition of $\rightarrow^r$, that means that there exist heads $((p_1, \gamma_1), \theta')$ and $((p_2, \gamma_2), \theta'')$ such that $((p, \gamma), \theta) \rightarrow^* ((p_1, \gamma_1), \theta') \overset{1}{\rightarrow} ((p_2, \gamma_2), \theta'') \rightarrow^* ((p, \gamma), \theta)$. Then $((p, \gamma), \theta), ((p_1, \gamma_1), \theta')$ and $((p_2, \gamma_2), \theta'')$ are all

in the same loop with a 1-labelled edge. Conversely, whenever $((p,\gamma),\theta)$ is in a component with such an edge, $((p,\gamma),\theta) \rightarrow^r ((p,\gamma),\theta)$ holds, then Lemma 5 implies that $(\langle p,\gamma\rangle,\theta) \Rightarrow^r (\langle p,\gamma v\rangle,\theta)$ which means that $((p,\gamma),\theta)$ is a repeating head.

□

3.2.2 Labelled configurations and labelled $\mathcal{BP}$-automata

To compute $\mathcal{G}$, we need to be able to compute predecessors of configurations of the form $(\langle p',\epsilon\rangle,\theta')$, and to determine whether these predecessors were backward-reachable using some control points in G (item 3 in Definition 6). To solve this question, we will label configurations $(\langle p'',w\rangle,\theta)$ s.t. $(\langle p'',w\rangle,\theta) \Rightarrow^* (\langle p',\epsilon\rangle,\theta')$ by 1 if this path went through an accepting location in G, i.e., if $(\langle p'',w\rangle,\theta) \Rightarrow^r (\langle p',\epsilon\rangle,\theta')$, and by 0 if not. To this aim, we define a labelled configuration as a tuple $[(\langle p,w\rangle,\theta),b]$, s.t. $(\langle p,w\rangle,\theta)$ is a configuration and $b \in \{0,1\}$.

Multi-automata were introduced in [6, 20] to finitely represent regular infinite sets of configurations of a PDS. Since a labelled configuration $c = [(\langle p,w\rangle,\theta),b]$ of a SM-PDS involves a PDS configuration $\langle p,w\rangle$, together with the current set of transition rules (phase) θ, and a boolean b, in order to take into account the phases θ, and these new 0/1-labels in configurations, we extend multi-automata to labelled $\mathcal{BP}$-automata as follows:

Definition 7 *Let $\mathcal{BP} = (P,\Gamma,\Delta,\Delta_c,G)$ be a SM-BPDS. A labelled $\mathcal{BP}$-automaton is a tuple $\mathcal{A} = (Q,\Gamma,T,I,F)$ where Γ is the automaton alphabet, Q is a finite set of states, $I \subseteq P \times 2^{\Delta\cup\Delta_c} \subseteq Q$ is the set of initial states, $T \subset Q \times \big((\Gamma \cup \{\epsilon\}) \times \{0,1\}\big) \times Q$ is the set of transitions, $F \subseteq Q$ is the set of final states.*

If $\big(q,[\gamma,b],q'\big) \in T$, we write $q \xrightarrow{[\gamma,b]}_T q'$. We extend this notation in the obvious way to sequences of symbols: (1) $\forall q \in Q, q \xrightarrow{[\epsilon,0]}_T q$, and (2) $\forall q,q' \in Q, \forall b \in \{0,1\}, \forall w \in \Gamma^*$ for $w = \gamma_0...\gamma_{n+1}, q \xrightarrow{[w,b]}_T q'$ iff $\exists q_0,...,q_n \in Q, b_0,...,b_{n+1} \in \{0,1\}, b = b_0 \vee b_1 \vee ... \vee b_{n+1}$ and $q \xrightarrow{[\gamma_0,b_0]}_T q_0 \xrightarrow{[\gamma_1,b_1]}_T q_1 \cdots q_n \xrightarrow{[\gamma_{n+1},b_{n+1}]}_T q'$. If $q \xrightarrow{[w,b]}_T q'$ holds, we say that
$q \xrightarrow{[w,b]}_T q'$ and $q \xrightarrow{[\gamma_0,b_0]}_T q_0 \xrightarrow{[\gamma_1,b_1]}_T q_1 \cdots q_n \xrightarrow{[\gamma_{n+1},b_{n+1}]}_T q'$ is a path of $\mathcal{A}$.

A labelled configuration $[(\langle p, w\rangle, \theta), b]$ is accepted by the automaton $\mathcal{A}$ iff there exists a path $(p,\theta)\xrightarrow{[\gamma_0,b_0]}_T q_1 \xrightarrow{[\gamma_1,b_1]}_T q_2 \cdots q_n \xrightarrow{[\gamma_n,b_n]}_T q_{n+1}$ in $\mathcal{A}$ such that $w = \gamma_0\gamma_1\cdots\gamma_n$, $b = b_0 \vee b_1 \vee ... \vee b_n$, $(p,\theta) \in I$, and $q_{n+1} \in F$. Let $L(\mathcal{A})$ be the set of labelled configurations accepted by $\mathcal{A}$.

3.2.3 Computing $pre^*\big((\langle p', \epsilon\rangle, \theta')\big)$

Given a configuration of the form $(\langle p', \epsilon\rangle, \theta')$, our goal is to compute a labelled $\mathcal{BP}$-automaton $\mathcal{A}_{pre^*}\big((\langle p', \epsilon\rangle, \theta')\big)$ that accepts labelled configurations of the form $[c, b]$ where c is a configuration and $b \in \{0, 1\}$ such that $c \Rightarrow^* (\langle p', \epsilon\rangle, \theta')$ (i.e., $c \in pre^*\big((\langle p', \epsilon\rangle, \theta')\big)$) and $b = 1$ iff this path went through final control points, i.e., $c \Rightarrow^r (\langle p', \epsilon\rangle, \theta')$. Otherwise, $b = 0$.

Let $p \in P$, we define $B(p) = 1$ if $p \in G$ and $B(p) = 0$ otherwise. $\mathcal{A}_{pre^*}\big((\langle p', \epsilon\rangle, \theta')\big) = (Q, \Gamma, T, I, F)$ is computed as follows: Initially, $Q = I = F = \{(p', \theta')\}$ and $T = \emptyset$. We add to T transitions as follows:

α_1: If $r = \langle p, \gamma\rangle \hookrightarrow \langle p_1, w\rangle \in \Delta$. If there exists in T a path $(p_1, \theta) \xrightarrow{[w,b]}_T q$ (in case $|w| = 0$, we have $w = \epsilon$) with $r \in \theta$. Then, add (p, θ) to I, and $\big((p, \theta), [\gamma, B(p) \vee b], q\big)$ to T.

α_2: if $r = p \xhookrightarrow{(\sigma,\sigma')} p_1 \in \Delta_c$ and there exists in T a transition $(p_1, \theta)\xrightarrow{[\gamma,b]}_T q$ with $r \in \theta$, where $\gamma \in \Gamma$. Then add (p, θ') to I, and $\big((p, \theta'), [\gamma, B(p) \vee b], q\big)$ to T, for θ' such that $\theta = \theta' \setminus \sigma \cup \sigma'$.

The procedure above terminates since there is a finite number of states and phases. Note that by construction, $F = \{(p', \theta')\}$, and, since initially $Q = \{(p', \theta')\}$, states of $\mathcal{A}_{pre^*}\big((\langle p', \epsilon\rangle, \theta')\big)$ are all of the form (p, θ) for $p \in P$ and $\theta \subseteq \Delta \cup \Delta_c$.

Let us explain the intuition behind rule (α_1). Let $r = \langle p, \gamma\rangle \hookrightarrow \langle p_1, w\rangle \in \Delta$. Let $c = (\langle p_1, ww'\rangle, \theta)$ and $c' = (\langle p, \gamma w'\rangle, \theta)$. Then, if $c \Rightarrow^* (\langle p', \epsilon\rangle, \theta')$, then necessarily, $c' \Rightarrow^* (\langle p', \epsilon\rangle, \theta')$. Moreover, $c' \Rightarrow^r (\langle p', \epsilon\rangle, \theta')$ iff either $c \Rightarrow^r (\langle p', \epsilon\rangle, \theta')$ or $p \in G$ (i.e. $B(p) = 1$). Thus, we would like that if the automaton $\mathcal{A}_{pre^*}\big((\langle p', \epsilon\rangle, \theta')\big)$ accepts the labelled configuration $[c, b]$ (where $b = 1$ means $c \Rightarrow^r (\langle p', \epsilon\rangle, \theta')$), then it should also accept the labelled configuration $[c', b \vee B(p)]$ ($b \vee B(p) = 1$ means $c' \Rightarrow^r (\langle p', \epsilon\rangle, \theta')$). Thus, if the automaton $\mathcal{A}_{pre^*}\big((\langle p', \epsilon\rangle, \theta')\big)$ contains a

path of the form $\pi = (p_1, \theta) \xrightarrow{[w,b_1]}_T q \xrightarrow{[w',b_2]}_T q_f$ where $q_f \in F$ that accepts the labelled configuration $[c, b]$, then the automaton should also accept the labelled configuration $[c', b \vee B(p)]$. This configuration is accepted by the run $(p, \theta) \xrightarrow{[\gamma, B(p)\vee b_1]}_T q \xrightarrow{[w',b_2]}_T q_f$ added by rule (α_1).

Rule (α_2) deals with modifying rules: Let $r = p \xhookrightarrow{(\sigma,\sigma')} p_1 \in \Delta_c$. Let $c = (\langle p_1, \gamma w'\rangle, \theta)$ and $c' = (\langle p, \gamma w'\rangle, \theta'')$ s.t. $\theta = \theta''\backslash\sigma \cup \sigma'$. Then, if $c \Rightarrow^* (\langle p', \epsilon\rangle, \theta')$, then necessarily, $c' \Rightarrow^* (\langle p', \epsilon\rangle, \theta')$. Moreover, $c' \Rightarrow^r (\langle p', \epsilon\rangle, \theta')$ iff either $c \Rightarrow^r (\langle p', \epsilon\rangle, \theta')$ or $p \in G$ (i.e. $B(p) = 1$). Thus, we need to impose that if the automaton $\mathcal{A}_{pre^*}\big((\langle p', \epsilon\rangle, \theta')\big)$ contains a path of the form $(p_1, \theta) \xrightarrow{[\gamma,b_1]}_T q \xrightarrow{[w',b_2]}_T q_f$ (where $q_f \in F$) that accepts the labelled configuration $[c, b], b = b_1 \vee b_2$ ($b = 1$ means $c \Rightarrow^r (\langle p', \epsilon\rangle, \theta')$), then necessarily, the automaton $\mathcal{A}_{pre^*}\big((\langle p', \epsilon\rangle, \theta')\big)$ should also accept the labelled configuration $[c', b \vee B(p)]$. This configuration is accepted by the run $(p, \theta'') \xrightarrow{[\gamma, B(p)\vee b_1]}_T q \xrightarrow{[w',b_2]}_T q_f$ added by rule (α_2).

Before proving that our construction is correct, we introduce the following definition:

Definition 8 *Let $\mathcal{A}_{pre^*}\big((\langle p', \epsilon\rangle, \theta')\big) = (Q, \Gamma, T, P, F)$ be the labelled $\mathcal{P}$-automaton computed by the saturation procedure above. In this section, we use $\underset{i}{\rightarrow}_T$ to denote the transition relation of $\mathcal{A}_{pre^*}\big((\langle p', \epsilon\rangle, \theta')\big)$ obtained after adding i transitions using the saturation procedure above. Let us notice that due to the fact that initially $Q = \{(p', \theta')\}$ and due to rules (α_1) and (α_2) that at step i add only transitions of the form $(p, \theta) \xrightarrow{\gamma}_T q$ for a state q that is already in the automaton at step $i-1$, then, states of $\mathcal{A}_{pre^*}\big((\langle p', \epsilon\rangle, \theta')\big)$ are all of the form (p, θ) for $p \in P$ and $\theta \subseteq \Delta \cup \Delta_c$.*

We can show that:

Lemma 6 *Let $p, p'' \in P$ and $\theta, \theta'' \subseteq \Delta \cup \Delta_c$. Let $w \in \Gamma^*$ and $b \in \{0, 1\}$. If a path $(p, \theta) \xrightarrow{[w,b]}_T (p'', \theta'')$ is in $\mathcal{A}_{pre^*}\big((\langle p', \epsilon\rangle, \theta')\big)$, then $(\langle p, w\rangle, \theta) \Rightarrow^* (\langle p'', \epsilon\rangle, \theta'')$. Moreover, if $b = 1$, then $(\langle p, w\rangle, \theta) \Rightarrow^r (\langle p'', \epsilon\rangle, \theta'')$.*

Proof: Initially, the automaton contains no transitions. Let i be an index such that $(p, \theta) \underset{i}{\xrightarrow{[w,b]}}_T (p'', \theta'')$ holds. We proceed by induction on i.

Basis. $i = 0$, then $(p'', \theta'') \underset{0}{\xrightarrow{[\epsilon,0]}}_T (p'', \theta'')$. This means $p'' = p'$, $\theta'' = \theta'$. Since initially $Q = \{(p', \theta')\}$, then $(\langle p'', \epsilon\rangle, \theta'') \Rightarrow^* (\langle p'', \epsilon\rangle, \theta'')$ always holds.

Step. $i > 0$. Let $t = \big((p_1,\theta_1),[\gamma,b_1],(p_0,\theta_0)\big)$ be the i-th transition added to $\mathcal{A}_{pre^*}$ and j be the number of times that t is used in the path $(p,\theta)\xrightarrow[i]{[w,b]}_T(p'',\theta'')$. The proof is by induction on j. If $j = 0$, then we have $(p,\theta)\xrightarrow[i-1]{[w,b]}_T(p'',\theta'')$ in the automaton, and we apply the induction hypothesis (induction on i) then we obtain $(\langle p,w\rangle,\theta) \Rightarrow^* (\langle p'',\epsilon\rangle,\theta'')$. So assume that $j > 0$. Then, there exist $u,v \in \Gamma^*$, $b',b'' \in \{0,1\}$ such that $w = u\gamma v$, $b = b' \vee b_1 \vee b''$ and

$$(p,\theta)\xrightarrow[i-1]{[u,b']}_T(p_1,\theta_1)\xrightarrow[i]{[\gamma,b_1]}_T(p_0,\theta_0)\xrightarrow[i]{[v,b'']}_T(p'',\theta'') \tag{1}$$

The application of the induction hypothesis (induction on i) to $(p,\theta) \xrightarrow[i-1]{[u,b']}_T (p_1,\theta_1)$ gives that

$$(\langle p,u\rangle,\theta) \Rightarrow^* (\langle p_1,\epsilon\rangle,\theta_1),\ \text{moreover, if } b' = 1, (\langle p,u\rangle,\theta) \Rightarrow^r (\langle p_1,\epsilon\rangle,\theta_1) \tag{2}$$

There are 2 cases depending on whether transition t was added by saturation rule α_1 or α_2.

1. Case t was added by rule α_1: There exist $p_2 \in P$ and $w_2 \in \Gamma^*$ such that

   $$r = \langle p_1,\gamma\rangle \hookrightarrow \langle p_2,w_2\rangle \in \Delta \cap \theta_1 \tag{3}$$

   and $\mathcal{A}_{pre^*}$ contains the following path:

   $$\pi' = (p_2,\theta_1) \xrightarrow[i-1]{[w_2,b_2]}_T (p_0,\theta_0) \xrightarrow[i]{[v,b'']}_T (p'',\theta''),\quad b_1 = b_2 \vee B(p_1) \tag{4}$$

   Applying the transition rule r, we get that

   $$(\langle p_1,\gamma v\rangle,\theta_1) \Rightarrow (\langle p_2,w_2 v\rangle,\theta_1) \tag{5}$$

   By induction on j (since transition t is used $j-1$ times in π'), we get from (4) that

   $$\begin{aligned} (\langle p_2,w_2v\rangle,\theta_1) &\Rightarrow^* (\langle p'',\epsilon\rangle,\theta'') \\ \text{moreover, if } b_2 \vee b'' = 1, (\langle p_2,w_2v\rangle,\theta_1) &\Rightarrow^r (\langle p'',\epsilon\rangle,\theta'') \end{aligned} \tag{6}$$

   Putting (2), (5) and (6) together, we can obtain that

$$(\langle p, w\rangle, \theta) = (\langle p, u\gamma v\rangle, \theta) \Rightarrow^* (\langle p_1, \gamma v\rangle, \theta_1) \Rightarrow (\langle p_2, w_2 v\rangle, \theta_1) \Rightarrow^* (\langle p'', \epsilon\rangle, \theta'')$$

Furthermore, if $b = b' \vee b_1 \vee b'' = 1$, then $b' = 1$ or $b_1 \vee b'' = 1$.

For the first case, $b' = 1$, then we can have $(\langle p, u\rangle, \theta) \Rightarrow^r (\langle p_1, \epsilon\rangle, \theta_1)$ from (2). Thus, we can obtain that $(\langle p, u\gamma v\rangle, \theta) \Rightarrow^r (\langle p_1, \gamma v\rangle, \theta_1) \Rightarrow^* (\langle p'', \epsilon\rangle, \theta'')$ i.e. $(\langle p, w\rangle, \theta) \Rightarrow^r (\langle p'', \epsilon\rangle, \theta'')$.

The second case $b_1 \vee b'' = 1$ i.e. $B(p_1) \vee b_2 \vee b'' = 1$ implies that $B(p_1) = 1$ (that means $p_1 \in G$ and $(\langle p_1, \gamma v\rangle, \theta_1) \Rightarrow^r (\langle p'', \epsilon\rangle, \theta'')$) or $b_2 \vee b'' = 1$ (that implies $(\langle p_2, w_2 v\rangle, \theta_1) \Rightarrow^r (\langle p'', \epsilon\rangle, \theta'')$ from (6)). Therefore, $(\langle p, w\rangle, \theta_1) \Rightarrow^r (\langle p'', \epsilon\rangle, \theta'')$.

2. Case t was added by rule α_2 : there exist $p_2 \in P$ and $\theta_2 \subseteq \Delta \cup \Delta_c$ such that

$$r = p_1 \xhookrightarrow{(\sigma,\sigma')} p_2 \in \Delta_c \cap \theta_2, \theta_2 = (\theta_1 \backslash \sigma) \cup \sigma' \tag{7}$$

and the following path in the current automaton (self-modifying rule won't change the stack) with $r \in \theta_2$:

$$(p_2, \theta_2) \xrightarrow[i-1]{[\gamma, b_1']}_T (p_0, \theta_0) \xrightarrow[i]{[v, b'']}_T (p'', \theta''), \quad b_1 = B(p_1) \vee b_1' \tag{8}$$

Applying the transition rule, we can get from (7) that

$$(\langle p_1, \gamma v\rangle, \theta_1) \Rightarrow (\langle p_2, \gamma v\rangle, \theta_2) \tag{9}$$

We can apply the induction hypothesis (on j) to (8), and obtain

$$\begin{aligned} (\langle p_2, \gamma v\rangle, \theta_2) \Rightarrow^* (\langle p'', \epsilon\rangle, \theta'',) \\ \text{moreover, if } b_1' \vee b'' = 1, (\langle p_2, \gamma v\rangle, \theta_2) \Rightarrow^r (\langle p'', \epsilon\rangle, \theta'') \end{aligned} \tag{10}$$

From (2),(9) and (10), we get

$$(\langle p, w\rangle, \theta) = (\langle p, u\gamma v\rangle, \theta) \Rightarrow^* (\langle p_1, \gamma v\rangle, \theta_1) \Rightarrow (\langle p_2, \gamma v\rangle, \theta_2) \Rightarrow^* (\langle p'', \epsilon\rangle, \theta'')$$

Furthermore, if $b = b' \vee b_1 \vee b'' = 1$, then $b' = 1$ or $b_1 \vee b'' = 1$.

For the first case, $b' = 1$, then we can have $(\langle p, u\rangle, \theta) \Rightarrow^r (\langle p_1, \epsilon\rangle, \theta_1)$ from (2). Thus, we can obtain that $(\langle p, u\gamma v\rangle, \theta) \Rightarrow^r (\langle p_1, \gamma v\rangle, \theta_1) \Rightarrow^* (\langle p'', \epsilon\rangle, \theta'')$ i.e. $(\langle p, w\rangle, \theta) \Rightarrow^r (\langle p'', \epsilon\rangle, \theta'')$. The second case $b_1 \vee b'' = 1$ i.e. $B(p_1) \vee b'_1 \vee b'' = 1$ implies that $B(p_1) = 1$ (that means $p_1 \in G$ and $(\langle p_1, \gamma v\rangle, \theta_1) \Rightarrow^r (\langle p', \epsilon\rangle, \theta')$) or $b'_1 \vee b'' = 1$ (that implies $(\langle p_2, \gamma v\rangle, \theta_2) \Rightarrow^r (\langle p'', \epsilon\rangle, \theta'')$ from (10)) i.e. $(\langle p, w\rangle, \theta_1) \Rightarrow^r (\langle p', \epsilon\rangle, \theta')$. Therefore, we can get that if $b = 1$, then $(\langle p, w\rangle, \theta_1) \Rightarrow^r (\langle p'', \epsilon\rangle, \theta'')$.

□

Lemma 7 *If there is a labelled configuration $[(\langle p, w\rangle, \theta), b]$ such that $(\langle p, w\rangle, \theta) \Rightarrow^* (\langle p', \epsilon\rangle, \theta')$, then there is a path $(p, \theta) \xrightarrow{[w,b]}_T (p', \theta')$ in $\mathcal{A}_{pre^*}\big((\langle p', \epsilon\rangle, \theta')\big)$. Moreover, if $(\langle p, w\rangle, \theta) \Rightarrow^r (\langle p', \epsilon\rangle, \theta')$, then $b = 1$.*

Proof: Assume $(\langle p, w\rangle, \theta) \stackrel{i}{\Rightarrow} (\langle p', \epsilon\rangle, \theta')$. We proceed by induction on i.

Basis. $i = 0$. Then $\theta = \theta', p' = p$ and $w = \epsilon$. Initially, we have that $Q = \{(p', \theta')\}$, therefore, by the definition of $\rightarrow_T$, we have $(p', \theta') \xrightarrow{\epsilon}_T (p', \theta')$. We cannot have $(\langle p', \epsilon\rangle, \theta') \Rightarrow^r (\langle p', \epsilon\rangle, \theta')$ in 0-step.

Step. $i > 0$. Then there exists a configuration $(\langle p'', u\rangle, \theta'')$ such that

$$(\langle p, w\rangle, \theta) \Rightarrow (\langle p'', u\rangle, \theta'') \stackrel{i-1}{\Rightarrow} (\langle p', \epsilon\rangle, \theta')$$

We apply the induction hypothesis to $(\langle p'', u\rangle, \theta'') \stackrel{i-1}{\Rightarrow} (\langle p', \epsilon\rangle, \theta')$, and obtain that there exists in $\mathcal{A}_{pre^*}\big((\langle p', \epsilon\rangle, \theta')\big)$ a path $(p'', \theta'') \xrightarrow{[u,b'']}_T (p', \theta')$. If $(\langle p'', u\rangle, \theta'') \Rightarrow^r (\langle p', \epsilon\rangle, \theta')$, $b'' = 1$.

Let (p_0, θ_0) be a state of $\mathcal{A}_{pre^*}$. Let $w_1, u_1 \in \Gamma^*, \gamma \in \Gamma, b''_0, b''_1 \in \{0, 1\}$ be such that $w = \gamma w_1$, $u = u_1 w_1$, $b'' = b''_0 \vee b''_1$ and

$$(p'', \theta'') \xrightarrow{[u_1,b''_0]}_T (p_0, \theta_0) \xrightarrow{[w_1,b''_1]}_T (p', \theta') \tag{6}$$

There are two cases depending on which rule is applied to get $(\langle p, w\rangle, \theta) \Rightarrow (\langle p'', u\rangle, \theta'')$.

1. Case $(\langle p, w\rangle, \theta) \Rightarrow (\langle p'', u\rangle, \theta'')$ is obtained by a rule of the form: $\langle p, \gamma\rangle \hookrightarrow \langle p'', u_1\rangle \in \Delta$. In this case, $\theta'' = \theta$. By the saturation rule α_1, we have

$$(p, \theta'') \xrightarrow{[\gamma,b_0]}_T (p_0, \theta_0),\ b_0 = B(p) \vee b''_0 \tag{7}$$

Putting (1) and (2) together, we can obtain that

$$\pi = (p, \theta'') \xrightarrow{[\gamma, b_0]}_T (p_0, \theta_0) \xrightarrow{[w_1, b_1'']}_T (p', \theta') \tag{8}$$

Thus, $(p, \theta'') \xrightarrow{[\gamma w_1, b_0 \vee b_1'']}_T (p', \theta')$ i.e. $(p, \theta) \xrightarrow{[w,b]}_T (p', \theta')$ where $b = b_0 \vee b_1''$.

2. Case $(\langle p, w\rangle, \theta) \Rightarrow (\langle p'', u\rangle, \theta'')$ is obtained by a rule of the form $p \xhookrightarrow{(\sigma,\sigma')} p'' \in \Delta_c$ i.e $\theta'' \neq \theta$. In this case, $u_1 = \gamma$. By the saturation rule β_2, we obtain that

$$(p, \theta) \xrightarrow{[\gamma, b_0]}_T (p_0, \theta_0) \text{ where } \theta'' = \theta \backslash \{r_1\} \cup \{r_2\}, b_0 = B(p) \vee b_0''. \tag{9}$$

Putting (1) and (4) together, we have the following path

$$(p, \theta) \xrightarrow{[\gamma, b_0]}_T (p_0, \theta_0) \xrightarrow{[w_1, b_1'']}_T (p', \theta') \text{ i.e. } (p, \theta) \xrightarrow{[w,b]}_T (p', \theta') \text{ where } b = b_0 \vee b_1'' \tag{10}$$

Furthermore, if $(\langle p, w\rangle, \theta) \Rightarrow^r (\langle p', \epsilon\rangle, \theta')$, then $(\langle p, w\rangle, \theta) \Rightarrow^r (\langle p'', u\rangle, \theta'')$ or $(\langle p'', u\rangle, \theta'') \Rightarrow^r (\langle p', \epsilon\rangle, \theta')$.

For the first case, $(\langle p, w\rangle, \theta) \Rightarrow^r (\langle p'', u\rangle, \theta'')$, then $p \in G$ i.e. $B(p) = 1$. For the second case, $(\langle p'', u\rangle, \theta'') \Rightarrow^r (\langle p', \epsilon\rangle, \theta')$, we can get $b'' = 1$ (from induction hypothesis). Thus, $b = b_0 \vee b_1'' = B(p) \vee b_0'' \vee b_1'' = B(p) \vee b'' = 1$. Therefore, if $(\langle p, w\rangle, \theta) \Rightarrow^r (\langle p', \epsilon\rangle, \theta')$, then we can obtain $b = 1$.

□

From these two lemmas, we get:

Theorem 3.2.2 *Let $[c, b]$ be a labelled configuration.*
Then $[c, b]$ is in $L(\mathcal{A}_{pre^*}\big((\langle p', \epsilon\rangle, \theta')\big)$ iff $c \in pre^*\big((\langle p', \epsilon\rangle, \theta')\big)$. Moreover, $c \Rightarrow^r (\langle p', \epsilon\rangle, \theta')$ iff $b = 1$.

Proof: Let $[(\langle p, w\rangle, \theta), b]$ be a configuration of $pre^*\big((\langle p', \epsilon\rangle, \theta')\big))$.
Then $(\langle p, w\rangle, \theta) \Rightarrow^* (\langle p', \epsilon\rangle, \theta')$. By Lemma 6, we can obtain that there exists a path $(p, \theta) \xrightarrow{[w,b]}_T (p', \theta')$ in $\mathcal{A}_{pre^*}\big((\langle p', \epsilon\rangle, \theta')\big)$.
So $[(\langle p, w\rangle, \theta), b]$ is in $L(\mathcal{A}_{pre^*}\big((\langle p', \epsilon\rangle, \theta')\big))$. Moreover, if $(\langle p, w\rangle, \theta) \Rightarrow^r (\langle p', \epsilon\rangle, \theta')$, then $b = 1$.

Conversely, let $[(\langle p,w\rangle,\theta),b]$ be a configuration accepted by $\mathcal{A}_{pre^*}\big((\langle p',\epsilon\rangle,\theta')\big)$ i.e. there exists a path $(p,\theta) \xrightarrow{[w,b]}_T (p',\theta')$ in $\mathcal{A}_{pre^*}\big((\langle p',\epsilon\rangle,\theta')\big)$. By Lemma 7, $(\langle p,w\rangle,\theta) \Rightarrow^* (\langle p',\epsilon\rangle,\theta')$ i.e. $(\langle p,w\rangle,\theta) \in pre^*(L(A))$. Moreover, if $b=1$, $(\langle p,w\rangle,\theta) \Rightarrow^r (\langle p',\epsilon\rangle,\theta')$.

□

3.2.4 Computing the Head Reachability Graph $\mathcal{G}$

Based on the definition of the Head Reachability Graph $\mathcal{G}$, and on Theorem 3.2.2, we can compute $\mathcal{G}$ as follows. Initially, $\mathcal{G}$ has no edges.

α_1': if $r_c : p \xhookrightarrow{(\sigma,\sigma')} p' \in \Delta_c$, then for every phase θ such that $r_c \in \theta$ and every $\gamma \in \Gamma$, we add the edge $((p,\gamma),\theta) \xrightarrow{B(p)} ((p',\gamma),\theta_0)$ to the graph $\mathcal{G}$, where $\theta_0 = \theta \setminus \sigma \cup \sigma'$.

α_2': if $r : \langle p,\gamma\rangle \hookrightarrow \langle p_0,\gamma_0\rangle \in \Delta$, then for every phase θ such that $r \in \theta$, we add the edge $((p,\gamma),\theta) \xrightarrow{B(p)} ((p_0,\gamma_0),\theta)$ to the graph $\mathcal{G}$.

α_3': if $r : \langle p,\gamma\rangle \hookrightarrow \langle p_0,\gamma_0\gamma'\rangle \in \Delta$, then for every phase θ such that $r \in \theta$, we add to the graph $\mathcal{G}$ the edge $((p,\gamma),\theta) \xrightarrow{B(p)} ((p_0,\gamma_0),\theta)$. Moreover, for every control point $p' \in P$ and phase θ' such that $\mathcal{A}_{pre^*}\big((\langle p',\epsilon\rangle,\theta')\big)$ contains a transition of the form $t = (p_0,\theta) \xrightarrow{[\gamma_0,b]}_T (p',\theta')$, we add to the graph $\mathcal{G}$ the edge $((p,\gamma),\theta) \xrightarrow{b\vee B(p)} ((p',\gamma'),\theta')$.

Items α_1' and α_2' are obvious. They respectively correspond to item 1 and item 2 of Definition 6 (since $B(p)=1$ iff $p \in G$). Item α_3' is based on Lemma 5 and on item 3 of Definition 6. Indeed, it follows from Lemma 5 that $\mathcal{A}_{pre^*}\big((\langle p',\epsilon\rangle,\theta')\big)$ contains a transition of the form $(p_0,\theta) \xrightarrow{[\gamma_0,b]}_T (p',\theta')$ implies that $(\langle p_0,\gamma_0\rangle,\theta) \Rightarrow^* (\langle p',\epsilon\rangle,\theta')$, and if $b=1$, then $(\langle p_0,\gamma_0\rangle,\theta) \Rightarrow^r (\langle p',\epsilon\rangle,\theta')$. Thus, in this case, the edge $((p,\gamma),\theta) \xrightarrow{b\vee B(p)} ((p',\gamma'),\theta')$ is added to $\mathcal{G}$ (item 3 of Definition 6) since $\langle p,\gamma\rangle \hookrightarrow \langle p_0,\gamma_0\gamma'\rangle \in \Delta$.

3.3 Experiments

3.3.1 Our approach vs. standard LTL for PDSs

We implemented our approach in a tool and we compared its performance against the approaches that consist in translating the SM-PDS to an equivalent standard (or symbolic) PDS, and then applying the standard LTL model checking algorithms implemented in the PDS model-checker tool Moped [42]. All our experiments were run on Ubuntu 16.04 with a 2.7 GHz CPU, 2GB of memory. To perform the comparison, we randomly generate several SM-PDSs and LTL formulas of different sizes. The results (CPU Execution time) are shown in Table 3.1 and 3.2. **Column** *Size* is the size of SM-PDS (S_1 for non-modifying transitions Δ and S_2 for modifying transitions Δ_c). **Column** *LTL* gives the size of the transitions of the Büchi automaton generated from the LTL formula (using the tool LTL2BA[31]). **Column** *SM-PDS* gives the cost of our direct algorithm presented in this thesis. **Column** *PDS* shows the cost it takes to get the equivalent PDS from the SM-PDS. **Column** *Result* reports the cost it takes to run the LTL PDS model-checker Moped [42] for the PDS we got. **Column** *Total* is the total cost it takes to translate the SM-PDS into a PDS and then apply the standard LTL model checking algorithm of Moped (Total=PDS+Result). **Column** *Symbolic PDS* reports the cost it takes to get the equivalent Symbolic PDS from the SM-PDS. **Column** $Result_1$ is the cost to run the Symbolic PDS LTL model-checker Moped. **Column** $Total_1$ is the total cost it takes to translate the SM-PDS into a symbolic PDS and then apply the standard LTL model checking algorithm of Moped. You can see that our direct algorithm (**Column** *SM-PDS*) is much more efficient than translating the SM-PDS to an equivalent (symbolic) PDS, and then run the standard LTL model-checker Moped. **Translating the SM-PDS to a standard PDS may take more than 20 days, whereas our direct algorithm takes only a few seconds.** Moreover, since the obtained standard (symbolic) PDS is huge, Moped failed to handle several cases (the time limit that we set for Moped is 20 minutes), whereas our tool was able to deal with all the cases in only a few seconds.

Size	LTL	SM-PDS	PDS	Result	Total	Symbolic PDS	$Result_1$	$Total_1$
$S_1 : 5, S_2 : 2$	$\|\delta\|$:15	**0.07s**	0.09s	0.01s	0 .10s	0.08s	0.00s	0.08s
$S_1 : 5, S_2 : 3$	$\|\delta\|$:8	**0.06s**	0.08s	0.01s	0.09s	0.09s	0.00s	0.09s
$S_1 : 11, S_2 : 4$	$\|\delta\|$:8	**0.16s**	0.13s	0.05s	0.18s	0.10s	0.00s	0.10s
$S_1 : 5, S_2 : 3$	$\|\delta\|$:10	**0.06s**	0.15s	0.01s	0.16s	0.09s	0.00s	0.09s
$S_1 : 110, S_2 : 4$	$\|\delta\|$:8	**0.34s**	186.10s	0.79s	186.99s	0.35s	0.00s	0.35s
$S_1 : 255, S_2 : 8$	$\|\delta\|$:8	**0.39s**	281.02s	0.94s	281.96s	4.82s	0.05s	4.87s
$S_1 : 255, S_2 : 8$	$\|\delta\|$:10	**0.42s**	281.02s	0.97s	281.99s	4.82s	0.06s	4.88s
$S_1 : 110, S_2 : 4$	$\|\delta\|$:15	**0.28s**	186.10s	1.05s	187.15s	0.35s	0.06s	0.41s
$S_1 : 255, S_2 : 8$	$\|\delta\|$:15	**0.46s**	281.02s	1.92s	282.94s	4.82s	0.08s	4.90s
$S_1 : 110, S_2 : 4$	$\|\delta\|$:20	**0.37s**	186.10s	1.05s	187.15s	0.35s	0.06s	0.41s
$S_1 : 255, S_2 : 8$	$\|\delta\|$:20	**0.55s**	281.02s	1.97s	282.99s	4.82s	0.17s	4.99s
$S_1 : 255, S_2 : 8$	$\|\delta\|$:25	**0.59s**	281.02s	1.23s	282.99s	4.82s	0.24s	5.36s
$S_1 : 2059, S_2 : 7$	$\|\delta\|$:8	**0.86s**	19525.01s	20.71s	19545.72s	20.70s	error	-
$S_1 : 2059, S_2 : 9$	$\|\delta\|$:8	**1.49s**	19784.7s	79.12s	19863.32	128.12s	error	-
$S_1 : 2059, S_2 : 11$	$\|\delta\|$:8	**3.73s**	30011.67s	168.15s	30179.82s	261.07s	error	-
$S_1 : 2059, S_2 : 11$	$\|\delta\|$:28	**6.88s**	30011.67s	169.55s	30180.22s	261.07s	error	-
$S_1 : 3050, S_2 : 10$	$\|\delta\|$:8	**5.21s**	39101.57s	killed	-	438.27s	error	-
$S_1 : 3090, S_2 : 10$	$\|\delta\|$:8	**5.86s**	40083.07s	killed	-	438.69s	error	-
$S_1 : 3050, S_2 : 10$	$\|\delta\|$:20	**7.24s**	39101.57s	killed	-	438.27s	error	-
$S_1 : 3090, S_2 : 10$	$\|\delta\|$:30	**8.38s**	40083.07s	killed	-	438.69s	error	-
$S_1 : 3090, S_2 : 10$	$\|\delta\|$:25	**8.89s**	40083.07s	killed	-	438.69s	error	-
$S_1 : 4050, S_2 : 10$	$\|\delta\|$:8	**9.21s**	81408.91s	killed	-	699.19s	error	-
$S_1 : 4050, S_2 : 10$	$\|\delta\|$:28	**11.64s**	81408.91s	killed	-	699.19s	error	-
$S_1 : 4058, S_2 : 11$	$\|\delta\|$:8	**9.83s**	93843.37s	killed	-	802.07s	error	-
$S_1 : 4058, S_2 : 11$	$\|\delta\|$:25	**13.59s**	93843.37s	killed	-	802.07s	error	-
$S_1 : 5050, S_2 : 11$	$\|\delta\|$:8	**10.34s**	173943.37s	killed	-	921.16s	error	-
$S_1 : 5090, S_2 : 11$	$\|\delta\|$:8	**10.52s**	179993.54s	killed	-	929.32s	error	-
$S_1 : 5090, S_2 : 11$	$\|\delta\|$:10	**12.89s**	179993.54s	killed	-	929.32s	error	-
$S_1 : 6090, S_2 : 11$	$\|\delta\|$:8	**13.49s**	190293.64s	killed	-	1002.73s	error	-
$S_1 : 6090, S_2 : 11$	$\|\delta\|$:10	**15.81s**	190293.64s	killed	-	1002.73s	error	-
$S_1 : 6090, S_2 : 11$	$\|\delta\|$:40	**32.39s**	190293.64s	killed	-	1002.73s	error	-
$S_1 : 7090, S_2 : 11$	$\|\delta\|$:25	**39.86s**	198932.32s	killed	-	1092.28s	error	-
$S_1 : 7090, S_2 : 11$	$\|\delta\|$:30	**43.24s**	198932.32s	killed	-	1092.28s	error	-
$S_1 : 9090, S_2 : 11$	$\|\delta\|$:8	**29.98s**	199987.98s	killed	-	1128.19s	error	-
$S_1 : 9090, S_2 : 11$	$\|\delta\|$:20	**45.29s**	199987.98s	killed	-	1128.19s	error	-

Table 3.1: Our approach vs. standard LTL for PDSs (Part 1)

Size	LTL	SM-PDS	PDS	Result	Total	Symbolic PDS	$Result_1$	$Total_1$
$S_1 : 10050, S_2 : 12$	$\|\delta\|$:8	**48.53s**	2134587.14s	killed	-	1469.28s	error	-
$S_1 : 10050, S_2 : 12$	$\|\delta\|$:25	**59.69s**	2134587.14s	killed	-	1469.28s	error	-
$S_1 : 10050, S_2 : 12$	$\|\delta\|$:30	**61.42s**	2134587.14s	kille d	-	1469.28s	error	-
$S_1 : 10150, S_2 : 12$	$\|\delta\|$:35	**64.17s**	2134633.28s	killed	-	1469.28s	error	-
$S_1 : 10150, S_2 : 14$	$\|\delta\|$:8	**58.34s**	2181975.64s	killed	-	2849.96s	error	-
$S_1 : 10150, S_2 : 14$	$\|\delta\|$:40	**82.72s**	2181975.64s	killed	-	2849.96s	error	-
$S_1 : 10150, S_2 : 12$	$\|\delta\|$:40	**76.61s**	2134633.28s	killed	-	1469.28s	error	-
$S_1 : 10150, S_2 : 16$	$\|\delta\|$:45	**89.83s**	2211008.82s	killed	-	3665.59s	error	-
$S_1 : 10150, S_2 : 12$	$\|\delta\|$:60	**97.56s**	2134633.28s	killed	-	1469.28s	error	-
$S_1 : 10150, S_2 : 12$	$\|\delta\|$:65	**105.89s**	2134633.28s	killed	-	1469.28s	error	-
$S_1 : 10150, S_2 : 16$	$\|\delta\|$:65	**134.45s**	2211008.82s	killed	-	3665.59s	error	-
$S_1 : 10180, S_2 : 16$	$\|\delta\|$:65	**175.29s**	2134643.52s	killed	-	3689.83s	error	-
$S_1 : 10180, S_2 : 16$	$\|\delta\|$:78	**214.36s**	2134643.52s	killed	-	3689.83s	error	-

Table 3.2: Our approach vs. standard LTL for PDSs (Part 2)

3.3.2 Malicious Behavior Detection on Self-Modifying Code

3.3.2.1 Specifying Malicious Behaviors using LTL.

As described in [14], several malicious behaviors can be described by LTL formulas. We give in what follows four examples of such malicious behaviors and show how they can be described by LTL formulas:

Registry Key Injecting: In order to get started at boot time, many malwares add themselves into the registry key listing. This behavior is typically implemented by first calling the API function GetModuleFileNameA to retrieve the path of the malware's executable file. Then, the API function RegSetValueExA is called to add the file path into the registry key listing. This malicious behavior can be described in LTL as follows:

$\phi_{rk} = \mathbf{F}\big(call\ GetModuleFileNameA \wedge \mathbf{F}(\ call\ RegSetValueExA)\big)$

This formula expresses that if a call to the API function GetModuleFileNameA is followed by a call to the API function RegSetValueExA, then probably a malware is trying to add itself into the registry key listing.

Data-Stealing: Stealing data from the host is a popular malicious behavior

that intend to steal any valuable information including passwords, software codes, bank information, etc. To do this, the malware needs to scan the disk to find the interesting file that he wants to steal. After finding the file, the malware needs to locate it. To this aim, the malware first calls the API function GetModuleHandleA to get a base address to search for a location of the file. Then the malware starts looking for the interesting file by calling the API function FindFirstFileA. Then the API functions CreateFileMappingA and MapViewOfFile are called to access the file. Finally, the specific file can be copied by calling the API function CopyFileA. Thus, this data-stealing malicious behavior can be described by the following LTL formula as follows:
$\phi_{ds} = \mathbf{F}(call\ GetModuleHandleA\ \wedge \mathbf{F}(call\ FindFirstFileA$
$\wedge\, \mathbf{F}\ (call\ CreateFileMappingA\ \wedge \mathbf{F}\ (call\ MapViewofFile \wedge \mathbf{F}\ call\ CopyFileA))))$

Spy-Worm: A spy worm is a malware that can record data and send it using the Socket API functions. For example, Keylogger is a spy worm that can record the keyboard states by calling the API functions GetAsyKeyState and GetKeyState and send that to the specific server by calling the socket function sendto. Another spy worm can also spy on the I/O device rather than the keyboard. For this, it can use the API function GetRawInputData to obtain input from the specified device, and then send this input by calling the socket functions send or sendto. Thus, this malicious behavior can be described by the following LTL formula:

$\phi_{sw} = \mathbf{F}\big((call\ GetAsyncKeyState \vee call\ GetRawInputData) \wedge \mathbf{F}(call\ sendto \vee call\ send)\big)$

Appending virus: An appending virus is a virus that inserts a copy of its code at the end of the target file. To achieve this, since the real OFFSET of the virus' variables depends on the size of the infected file, the virus has to first compute its real absolute address in the memory. To perform this, the virus has to call the sequence of instructions: l_1: call f; l_2: ....; f: pop eax;. The instruction call f will push the return address l_2 onto the stack. Then, the pop instruction in f will put the value of this address into the register eax. Thus, the virus can get its real absolute address from the register eax. This malicious behavior can be described by the following LTL formula:

$\phi_{av} = \bigvee \mathbf{F}\Big(call \wedge \mathbf{X}(\text{top-of-stack} = a) \wedge \mathbf{G}\neg\big(ret \wedge (\text{top-of-stack} = a)\big)\Big)$

where the $\bigvee$ is taken over all possible return addresses a, and top-of-stack$=a$ is a predicate that indicates that the top of the stack is a. The subformula

$call \wedge \mathbf{X}(\text{top-of-stack} = a)$ means that there exists a procedure call having a as return address. Indeed, when a procedure call is made, the program pushes its corresponding return address a to the stack. Thus, at the next step, a will be on the top of the stack. Therefore, the formula above expresses that there exists a procedure call having a as return address, such that there is no *ret* instruction which will return to a.

Note that this formula uses predicates that indicate that the top of the stack is a. Our techniques work for this case as well: it suffices to encode the top of the stack in the control points of the SM-PDS. Our implementation works for this case as well and can handle appending viruses.

3.3.2.2 Applying our tool for malware detection.

We applied our tool to detect several malwares. We use the unpack tool unpacker [45] to handle packers like UPX, and we use Jakstab [22] as disassembler. We consider 160 malwares from the malware library VirusShare [49], 184 malwares from the malware library MalShare [35], 288 email-worms from VX heaven [48] and 260 new malwares generated by NGVCK, one of the best malware generators. We also choose 19 benign samples from Windows XP system. We consider self-modifying versions of these programs. In these versions, the malicious behaviors are unreachable if the semantics of the self-modifying instructions are not taken into account, i.e., if the self-modifying instructions are considered as "standard" instructions that do not modify the code, then the malicious behaviors cannot be reached. To check this, we model such programs in two ways:

1. First, we take into account the self-modifying instructions and model these programs using SM-PDSs as described in Section 2.3.2. Then, we check whether these SM-PDSs satisfy at least one of the malicious LTL formulas presented above. If yes, the program is declared as malicious, if not, it is declared as benign. Our tool was able to detect all the 892 self-modifying malwares as malicious, and to determine that benign programs are benign. We report in Tables 3.5, 3.6, 3.7 and 3.8 the results we obtained. **Column** *Size* is the number of control locations, **Column** *Result* gives the result of our algorithm: **Yes** means malicious and **No** means benign; and **Column** *cost* gives the cost to apply our LTL model-checker to check one of the LTL properties described above.

2. Second, we abstract away the self-modifying instructions and proceed as if these instructions were not self-modifying. In this case, we translate the binary codes to standard pushdown systems as described in [13]. By using PDSs as models, none of the malwares that we consider was detected as malicious, whereas, as reported in Tables 3.5, 3.6, 3.7 and 3.8 , using self-modifying PDSs as models, and applying our LTL model-checking algorithm allowed to detect all the 892 malwares that we considered.

Note that checking the formulas ϕ_{rk}, ϕ_{ds}, and ϕ_{sw} could be done using multiple *pre** queries on SM-PDSs using the *pre** algorithm of Section 2.5. However, this would be less efficient than performing our direct LTL model-checking algorithm, as shown in Tables 3.3, 3.4, where **Column** *Size* gives the number of control locations, **Column** *LTL* gives the time of applying our LTL model-checking algorithm; and **Column** *Multiple pre** gives the cost of applying multiple *pre** on SM-PDSs to check the properties ϕ_{rk}, ϕ_{ds}, and ϕ_{sw}. It can be seen that applying our *direct* LTL model checking algortihm is more efficient. Furthermore, the appending virus formula ϕ_{av} cannot be solved using multiple *pre** queries. Our direct LTL model-checking algorithm is needed in this case. Note that some of the malwares we considered in our experiments are appending viruses. Thus, our algorithm and our implementation are crucial to be able to detect these malwares.

Example	Size	LTL	Multiple pre^*	Example	Size	LTL	Multiple pre^*
Tanatos.b	12315	16.261s	46.635s	Netsky.c	45	0.002s	0.092s
Win32.Happy	23	0.042s	0.075s	Mydoom.c	155	0.014s	0.206s
Netsky.a	45	0.047s	0.085s	MyDoom-N	16980	30.231s	98.418s
Mydoom.y	26902	12.462s	102.559s	Mydoom.j	22355	11.262s	111.617s
klez-N	6281	3.252s	78.419s	Mydoom.v	5965	3.971s	83.988s
klez.c	30	0.039s	0.088s	Netsky.b	45	0.057s	0.183s
Repah.b	221	2.428s	8.852s	Gibe.b	5358	4.229s	17.239s
Magistr.b	4670	3.699s	93.818s	Ardurk.d	1913	0.482s	3.212s
Netsky.d	45	0.083s	0.123s	klez.f	27	0.054s	4.518s
Kelino.l	495	0.326s	5.468s	Kipis.t	20378	23.345s	48.689s
klez.d	31	0.085s	0.291s	Plage.b	395	0.291s	3.138s
Kelino.g	470	0.672s	3.446s	Urbe.a	123	0.376s	2.981s
klez.e	27	0.094s	0.482s	Magistr.b	4670	3.987s	53.235s
Magistr.a.poly	36989	49.863s	159.195s	Spam.Tedroo.AB	487	0.924s	4.894s
Adon.1703	37	0.358s	0.884s	Adon.1559	37	0.255s	4.088s
Akez	273	0.136s	1.863s	Alcaul.d	845	0.165s	0.392s
Alaul.c	355	0.109s	5.757s	fsAutoB.F026	245	1.698s	4.503s
Haharin.A	210	1.462s	4.318s	Haharin.dr	235	1.558s	4.312s
LdPinch.BX.DLL	2010	6.965s	8.128s	LdPinch.Win32.5558	2015	6.907s	8.981s
LdPinch.fmye	1845	6.194s	9.232s	Win32/Toga!rfn	590	2.023s	3.978s
LdPinch-15	580	1.008s	3.957s	LdPinch.e	578	1.185s	3.392s
Tanatos.b	12315	16.261s	46.635s	Netsky.c	45	0.002s	0.092s
Win32.Happy	23	0.042s	0.075s	Mydoom.c	155	0.014s	0.206s
Netsky.a	45	0.047s	0.085s	MyDoom-N	16980	30.231s	98.418s
Mydoom.y	26902	12.462s	102.559s	klez-N	6281	3.252s	78.419s
Mydoom.j	22355	11.262s	111.617s	Mydoom.v	5965	3.971s	83.988s
klez.c	30	0.039s	0.088s	Netsky.b	45	0.057s	0.183s
Repah.b	221	2.428s	8.852s	Magistr.b	4670	3.699s	93.818s
Gibe.b	5358	4.229s	17.239s	Ardurk.d	1913	0.482s	3.212s
Netsky.d	45	0.083s	0.123s	klez.f	27	0.054s	4.518s
Kelino.l	495	0.326s	5.468s	Kipis.t	20378	23.345s	48.689s

Table 3.3: Multiple pre^* v.s. our direct LTL model-checking algorithm (part 1)

Example	Size	LTL	Multiple pre^*	Example	Size	LTL	Multiple pre^*
Kelino.g	470	0.672s	3.446s	Plage.b	395	0.291s	3.138s
Urbe.a	123	0.376s	2.981s	Magistr.a.poly	36989	49.863s	159.195s
klez.e	27	0.094s	0.482s	Magistr.b	4670	3.987s	53.235s
Mydoom-EG[Trj]	230	0.242s	6.172s	Email.W32!c	220	0.249s	5.946s
W32.Mydoom.L	235	0.288s	6.452s	Mydoom.5	228	0.307s	8.163s
Mydoom.cjdz5239	225	0.392s	9.968s	Mydoom.DN.worm	220	0.299s	8.928s
Mydoom.R	230	0.322s	9.086s	Win32.Mydoom	235	0.296s	7.985s
Mydoom.o@MM!zip	235	0.403s	10.323s	MyDoom.54464	5935	5.939s	94.026s
Mydoom.M@mm	5965	5.633s	108.129s	MyDoom.N	5970	6.152s	86.468s
Sramota.avf	240	0.383s	2.691s	Mydoom	238	0.278	2.749s
Win32.Mydoom.288	248	0.410s	2.983s	Win32.Chur.A	51895	98.161s	298.047s
Win32.Runouce	51678	92.692s	248.146s	Win32.CNHacker	51095	94.952s	245.452s
Win32.Skybag	4180	6.891s	13.739s	Skybag.A	4310	6.205s	15.452s
Netsky.ah@MM	4480	6.991s	16.018s	Spam.Tedroo.AB	487	0.924s	4.894s
Adon.1703	37	0.358s	0.884s	Adon.1559	37	0.255s	4.088s
Akez	273	0.136s	1.863s	Alcaul.d	845	0.165s	0.392s
Alaul.c	355	0.109s	5.757s	Haharin.dr	235	1.558s	4.312s
Haharin.A	210	1.462s	4.318s	fsAutoB.F026	245	1.698s	4.503s
LdPinch.BX.DLL	2010	6.965s	8.128s	LdPinch.fmye	1845	6.194s	9.232s
LdPinch..5558	2015	6.907s	8.981s	LdPinch.e	578	1.185s	3.392s
LdPinch-15	580	1.008s	3.957s	Win32/Toga!rfn	590	2.023s	3.978s
LdPinch.by	970	4.092s	11.327s	Generic.2026199	433	2.402s	9.614s
LdPinch.arr	1250	1.848s	9.986s	Troj.LdPinch.er	205	2.529s	6.154s
LdPnch-Fam	195	1.440s	4.097s	LdPinch.Gen.3	210	1.482s	4.973s
Androm	95	0.028s	0.192s	Ardurk.d	1913	3.679s	5.588s
Generic.12861	30183	72.264s	224.809s	Tanatos.O	9284	21.481s	79.773s
Jorik	837	4.159s	11.733s	Bugbear-B	9278	17.737s	52.549s

Table 3.4: Multiple pre^* v.s. our direct LTL model-checking algorithm (part 2)

Example	Size	Result	*cost*	Example	Size	Result	*cost*	Example	Size	Result	*cost*
Tanatos.b	12315	Yes	16.261s	Netsky.c	45	Yes	0.002s	Win32.Happy	23	Yes	0.042s
Netsky.a	45	Yes	0.047s	Mydoom.c	155	Yes	0.014s	MyDoom-N	16980	Yes	30.231s
Mydoom.y	26902	Yes	12.462s	Mydoom.j	22355	Yes	11.262s	klez-N	6281	Yes	3.252s
klez.c	30	Yes	0.039s	Mydoom.v	5965	Yes	3.971s	Netsky.b	45	Yes	0.057s
Repah.b	221	Yes	2.428s	Gibe.b	5358	Yes	4.229s	Magistr.b	4670	Yes	3.699s
Netsky.d	45	Yes	0.083s	Ardurk.d	1913	Yes	0.482s	klez.f	27	Yes	0.054s
Kelino.l	495	Yes	0.326s	Kipis.t	20378	Yes	25.345s	klez.d	31	Yes	0.085s
Kelino.g	470	Yes	0.672s	Plage.b	395	Yes	0.291s	Urbe.a	123	Yes	0.376s
klez.e	27	Yes	0.094s	Magistr.b	4670	Yes	3.987s	Magistr.a.poly	36989	Yes	49.863s
Mydoom.M@mm	5965	Yes	5.633s	MyDoom.54464	5935	Yes	5.939s	MyDoom.N!worm	5970	Yes	6.152s
Win32.Runouce	51678	Yes	92.692s	Win32.Chur.A	51895	Yes	98.161s	Win32.CNHacker.C	51095	Yes	94.952s
Win32.Mydoom!O	215	Yes	0.481s	Mydoom.o@MM!zip	257	Yes	0.298s	W.Mydoom.kZ2L	228	Yes	0.729s
Mydoom-EG [Trj]	230	Yes	0.242s	Email.Worm.W32!c	220	Yes	0.249s	W32.Mydoom.L	235	Yes	0.288s
Worm.Mydoom-5	228	Yes	0.307s	Mydoom.CJDZ-5239	225	Yes	0.392s	Mydoom.DN.worm	220	Yes	0.299s
Win32.Mydoom.R	230	Yes	0.322s	Win32.Mydoom.dlnpqi	235	Yes	0.296s	Mydoom.o@MM!zip	235	Yes	0.403s
Sramota.avf	240	Yes	0.383s	BehavesLike.Mydoom	238	Yes	0.278s	Win32.Mydoom.288	248	Yes	0.410s
Mydoom.ACQ	19210	Yes	39.662s	Mydoom.ba	19423	Yes	38.269s	Mydoom.ftde	19495	Yes	39.583s
Worm.Anarxy	210	Yes	1.913s	Malware!15bf	220	Yes	2.017s	Anar.A.2	140	Yes	1.993s
Win32.Anar.a	215	Yes	1.631s	nar.24576	240	Yes	2.738s	Worm-email.Anar.S	155	Yes	2.093s
HLLW.NewApt	4230	Yes	6.954s	Win32.Worm.km	4405	Yes	7.396s	Newapt.Efbh	4550	Yes	7.254s
NewApt!generic	4815	Yes	9.002s	NewApt.A@mm	4485	Yes	8.159s	Newapt.Win32.1	4155	Yes	7.885s
W32.W.Newapt.A!	5015	Yes	8.925s	Worm.Mail.NewApt.a	51550	Yes	9.083s	malicious.154966	5155	Yes	9.291s
Win32.Yanz	2250	Yes	4.357s	Yanzi.QTQX-0894	2120	Yes	4.109s	Win32.Yanz.a	2410	Yes	4.465s
Win32.Skybag	4180	Yes	6.891s	Skybag.A	4310	Yes	6.205s	Netsky.ah@MM	4480	Yes	6.991s
Skybag.b	4955	Yes	6.892s	Worm.Skybag-1	4820	Yes	7.119s	Win32.Agent.R	4490	Yes	7.898s
Skybag [Wrm]	4985	Yes	7.482s	Skybag.Dvgb	4830	Yes	7.564s	Netsky.CI.worm	4550	Yes	7.180s

Table 3.5: Experimental Results (part 1)

Example	Size	Result	*cost*	Example	Size	Result	*cost*	Example	Size	Result	*cost*
Agent.xpro	533	Yes	0.352s	Vilsel.lhb	15036	Yes	4.972s	Generic.2026199	433	Yes	3.489s
Vilsel.lhb	15036	Yes	26.962s	Generic.DF	5358	Yes	7.821s	LdPinch.aoq	7695	Yes	6.290s
Jorik	837	Yes	4.159s	Bugbear-B	9278	Yes	17.737s	Tanatos.O	9284	Yes	21.481s
Gen.2	1510	Yes	5.632s	Gibe.b	5358	Yes	9.615s	Generic26.AXCN	837	Yes	3.792s
Androm	95	Yes	0.028s	Ardurk.d	1913	Yes	3.679s	Generic.12861	30183	Yes	72.264s
LdPinch.by	970	Yes	4.092s	Generic.2026199	433	Yes	2.402s	LdPinch.arr	1250	Yes	1.848s
Generic.12861	30183	Yes	88.294s	Generic.18017273	267	Yes	0.192s	LdPinch.mg	5957	Yes	9.297s
Script.489524	522	Yes	1.458s	Generic.DF	5358	Yes	8.291s	Zafi	433	Yes	1.028s
GenericKD4047614	3495	Yes	4.646s	Win32.Agent.es	3500	Yes	6.083s	W32.HfsAutoB.	3398	Yes	5.092s
Trojan.Sivis-1	5351	Yes	7.029s	Win32.Siggen.28	5440	Yes	6.998s	Trojan/Cosmu.isk.	5345	Yes	6.273s
Trojan.17482-4	381	Yes	1.495s	Delphi.Gen	375	Yes	1.948s	Trojan.b5ac.	370	Yes	2.089s
Delfobfus	798	Yes	3.909s	Troj.Undef	790	Yes	4.068s	Trojan-Ransom.	805	Yes	5.119s
LDPinch.400	1783	Yes	4.893s	PSW.LdPinch.plt	1808	Yes	5.088s	PSW.Pinch.1	1905	Yes	5.757s
LdPinch.BX.DLL	2010	Yes	6.965s	LdPinch.fmye	1845	Yes	6.194s	LdPinch.5558	2015	Yes	6.907s
TrojanSpy.Lydra.a	3450	Yes	8.289s	Trojan.StartPage	2985	Yes	5.982s	PSWTroj.LdPinch.au	2985	Yes	6.198s
LdPinch-21	3180	Yes	6.917s	LdPinch-R	3025	Yes	7.005s	LdPinch.Gen.2	2990	Yes	6.992s
Graftor.46303	3230	Yes	5.898s	LdPinch-AIH [Trj]	3010	Yes	6.095s	Win32.Heur.k	2970	Yes	5.950s
LdPinch-15	580	Yes	1.008s	LdPinch.e	578	Yes	1.185s	Win32/Toga!rfn	590	Yes	2.023s
PSW.LdPinch.mj	595	Yes	1.078s	Gaobot.DIH.worm	590	Yes	1.482s	LDPinch.DF!tr.pws	588	Yes	1.736s
TrojanSpy.Zbot	610	Yes	1.610s	LDPinch.10639	605	Yes	1.185s	SillyProxy.AM	590	Yes	1.882s
LdPinch.mj!c	590	Yes	4.5345s	LdPinch.gen!Eldorado	605	Yes	3.955s	Generic!BT	615	Yes	2.085s
LdPnch-Fam	195	Yes	1.440s	Troj.LdPinch.er	205	Yes	2.529s	LdPinch.Gen.3	210	Yes	1.482s
Malware.wsc	150	Yes	2.843s	malicious.7aa9fd	185	Yes	2.189s	WS.LDPinch.400	195	Yes	1.898s

Table 3.6: Experimental Results (part2)

Example	Size	Result	*cost*	Example	Size	Result	*cost*	Example	Size	Result	*cost*
calculation.exe	9952	No	18.352s	cisvc.exe	4105	No	3.631s	simple.exe	52	No	0.001s
shutdown.exe	2529	No	0.397s	loop.exe	529	No	9.249s	cmd.exe	1324	No	13.466s
notepad.exe	10529	No	24.583s	java.exe	800	No	15.852s	java.exe	21324	No	42.373s
sort.exe	8529	No	29.789s	bibDesk.exe	32800	No	50.279s	interface.exe	1005	No	8.462s
ipv4.exe	968	No	4.186s	TextWrangler.exe	14675	No	45.221s	sogou.exe	45219	No	55.259s
game.exe	34325	No	82.424s	cycle.tex	9014	No	42.555s	calender.exe	892	No	35.039s
SdBot.zk	3430	Yes	23.242s	Virus.Gen	661	Yes	9.437s	AutoRun.PR	240	Yes	4.181s
Adon.1703	37	Yes	0.358s	Adon.1559	37	Yes	0.255s	Spam.Tedroo.AB	487	Yes	0.924s
Akez	273	Yes	0.136s	Alcaul.d	845	Yes	0.165s	Alaul.c	355	Yes	0.109s
Virus.klk	5235	Yes	15.863s	Virus.Win32.Agent	5340	Yes	15.968s	Hoax.Gen	5455	Yes	13.569s
eHeur.Virus02	420	Yes	4.985s	Akez.11255	440	Yes	3.985s	Akez.Win32.1	455	Yes	4.008s
Weird.10240.C	430	Yes	3.929s	PEAKEZ.A	450	Yes	2.998s	Virus.Weird.c	473	Yes	3.302s
W95/Kuang	435	Yes	2.985s	Radar01.Gen	465	Yes	4.005s	Akez.Win32.5	490	Yes	3.958s
Haharin.A	210	Yes	1.462s	fsAutoB.F026	245	Yes	1.698s	Haharin.dr	235	Yes	1.558s
NGVCK1	329	Yes	0.933s	NGVCK2	455	Yes	1.109s	NGVCK3	2300	Yes	1.388s
NGVCK4	550	Yes	1.149s	NGVCK5	1555	Yes	1.825s	NGVCK6	1698	Yes	1.689s
NGVCK7	6902	Yes	14.524s	NGVCK8	2355	Yes	4.254s	NGVCK9	281	Yes	13.301s
NGVCK10	2980	Yes	9.262s	NGVCK11	5965	Yes	11.456s	NGVCK12	4529	Yes	10.094s
NGVCK13	2210	Yes	8.902s	NGVCK14	5358	Yes	10.294s	NGVCK15	970	Yes	1.912s
NGVCK16	658	Yes	0.935s	NGVCK17	913	Yes	1.392s	NGVCK18	90	Yes	0.094s
NGVCK19	1295	Yes	6.958s	NGVCK20	4378	Yes	15.449s	NGVCK21	31	Yes	0.097s
NGVCK22	370	Yes	0.898s	NGVCK23	3955	Yes	9.498s	NGVCK24	6924	Yes	11.983s
NGVCK25	8127	Yes	15.018s	NGVCK26	4970	Yes	9.982s	NGVCK27	7989	Yes	13.197s
NGVCK28	227	Yes	0.098s	NGVCK29	960	Yes	0.692s	NGVCK30	89	Yes	0.088s
NGVCK31	550	Yes	0.875s	NGVCK32	60	Yes	0.059s	NGVCK33	65	Yes	0.069s
NGVCK34	5990	Yes	9.848s	NGVCK35	4590	Yes	10.178s	NGVCK36	825	Yes	2.934s

Table 3.7: Experimental Results (part 3)

Example	Size	Result	*cost*	Example	Size	Result	*cost*	Example	Size	Result	*cost*
NGVCK37	80	Yes	0.998s	NGVCK38	150	Yes	1.093s	NGVCK39	395	Yes	1.048s
NGVCK40	40	Yes	0.921s	NGVCK41	950	Yes	0.704s	NGVCK42	8290	Yes	15.085s
NGVCK43	6220	Yes	2.930s	NGVCK44	5215	Yes	11.006s	NGVCK45	9290	Yes	14.595s
NGVCK46	320	Yes	0.928s	NGVCK47	834	Yes	2.958s	NGVCK48	9810	Yes	14.696s
NGVCK49	12320	Yes	25.395s	NGVCK50	8810	Yes	19.969s	NGVCK51	39810	Yes	68.283s
NGVCK52	520	Yes	0.289s	NGVCK53	15	Yes	0.089s	NGVCK54	8883	Yes	11.393s
NGVCK55	12520	Yes	38.768s	NGVCK56	6218	Yes	15.489s	NGVCK57	32562	Yes	83.482s
NGVCK58	9520	Yes	23.658s	NGVCK59	818	Yes	2.592s	NGVCK60	12962	Yes	38.025s
NGVCK61	10020	Yes	24.976s	NGVCK62	8818	Yes	19.299s	NGVCK63	2068	Yes	3.662s
NGVCK64	273	Yes	1.987s	NGVCK65	5855	Yes	8.995s	NGVCK66	68	Yes	1.002s
NGVCK69	4150	Yes	8.052s	NGVCK70	9860	Yes	24.199s	NGVCK71	3240	Yes	7.951s
NGVCK72	31	Yes	0.591s	NGVCK73	549	Yes	1.052s	NGVCK74	9078	Yes	29.078s
NGVCK75	90	Yes	1.002s	NGVCK76	5890	Yes	10.128s	NGVCK77	1958	Yes	9.559s
NGVCK78	33468	Yes	75.098s	NGVCK79	4735	Yes	10.980s	NGVCK80	45273	Yes	82.396s
NGVCK66	777	Yes	0.198s	NGVCK67	895	Yes	0.223s	NGVCK81	6939	Yes	2.726s
NGVCK82	2931	Yes	0.463s	NGVCK83	8759	Yes	10.316s	NGVCK84	34563	Yes	53.244s
NGVCK85	19024	Yes	29.220s	NGVCK86	1026	Yes	0.572s	NGVCK87	7929	Yes	5.671s
NGVCK88	6126	Yes	8.682s	NGVCK89	580	Yes	2.036s	NGVCK90	27843	Yes	17.353s
NGVCK91	20	Yes	0.001s	NGVCK92	59	Yes	0.903s	NGVCK93	98	Yes	0.021s
NGVCK94	150	Yes	0.146s	NGVCK95	1679	Yes	0.294s	NGVCK96	6299	Yes	5.196s
NGVCK97	4496	Yes	5.272s	NGVCK98	428	Yes	0.329s	NGVCK99	158	Yes	1.153s
NGVCK100	895	Yes	0.961s	NGVCK101	745	Yes	1.117s	NGVCK102	704	Yes	0.269s
NGVCK103	86	Yes	0.282s	NGVCK104	145	Yes	0.998s	NGVCK105	24124	Yes	68.816s

Table 3.8: Experimental Results (part 4)

4

CTL Model-Checking of Self-modifying Code

In this chapter, we reduce the CTL model-checking problem of self-modifying code to the emptiness problem of Self-Modifying Alternating Büchi Pushdown Systems (SM-ABPDSs).

4.1 CTL Model-Checking of SM-PDSs

4.1.1 The Computation Tree Logic CTL

Let At be a finite set of atomic propositions. CTL formulas over At are defined as follows (where $A \in At$):

$$\varphi ::= A \mid \neg A \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid AX\varphi \mid EX\varphi \mid A[\varphi U \varphi] \mid E[\varphi U \varphi] \mid A[\varphi \widetilde{U} \varphi] \mid E[\varphi \widetilde{U} \varphi].$$

Given a CTL formula φ, the closure $cl(\varphi)$ is the set of all the subformulae of φ, including φ. Let $\mathcal{P} = (P, \Gamma, \Delta, \Delta_c)$ be a SM-PDS, $\nu : P \rightarrow 2^{At}$ be a labelling function mapping to each control location $p \in P$ a set of atomic propositions. The satisfiability relation of a CTL formula φ at a configuration $(\langle p_0, w_0 \rangle, \theta_0)$ (denoted by $(\langle p_0, w_0 \rangle, \theta_0) \models_\nu \varphi$) is defined as follows:

- $(\langle p_0, w_0 \rangle, \theta_0) \models_\nu A$ iff $A \in \nu(p_0)$,
- $(\langle p_0, w_0 \rangle, \theta_0) \models_\nu \neg A$ iff $A \notin \nu(p_0)$,
- $(\langle p_0, w_0 \rangle, \theta_0) \models_\nu \varphi_1 \vee \varphi_2$ iff $(\langle p_0, w_0 \rangle, \theta_0) \models_\nu \varphi_1$ or $(\langle p_0, w_0 \rangle, \theta_0) \models_\nu \varphi_2$,
- $(\langle p_0, w_0 \rangle, \theta_0) \models_\nu \varphi_1 \wedge \varphi_2$ iff $(\langle p_0, w_0 \rangle, \theta_0) \models_\nu \varphi_1$ and $(\langle p_0, w_0 \rangle, \theta_0) \models_\nu \varphi_2$,

• $(\langle p_0, w_0\rangle, \theta_0) \models_\nu AX\varphi$ iff $(\langle p_1, w_1\rangle, \theta_1) \models_\nu \varphi$ for every successor $(\langle p_1, w_1\rangle, \theta_1)$ of $(\langle p_0, w_0\rangle, \theta_0)$,
• $(\langle p_0, w_0\rangle, \theta_0) \models_\nu EX\varphi$ iff $(\langle p_1, w_1\rangle, \theta_1) \models_\nu \varphi$ for some successor $(\langle p_1, w_1\rangle, \theta_1)$ of $(\langle p_0, w_0\rangle, \theta_0)$,
• $(\langle p_0, w_0\rangle, \theta_0) \models_\nu A[\varphi_1 U \varphi_2]$ iff for every path $(\langle p_0, w_0\rangle, \theta_0)(\langle p_1, w_1\rangle, \theta_1)\cdots$ of $\mathcal{P}$ starting from $(\langle p_0, w_0\rangle, \theta_0)$, $\exists i \geq 0$ s.t. $(\langle p_i, w_i\rangle, \theta_i) \models_\nu \varphi_2$ and $\forall 0 \leq j < i, (\langle p_j, w_j\rangle, \theta_j) \models_\nu \varphi_1$.
• $(\langle p_0, w_0\rangle, \theta_0) \models_\nu E[\varphi_1 U \varphi_2]$ iff there exists a path $(\langle p_0, w_0\rangle, \theta_0)(\langle p_1, w_1\rangle, \theta_1)\cdots$ of $\mathcal{P}$ starting from $(\langle p_0, w_0\rangle, \theta_0)$, $\exists i \geq 0$ s.t. $(\langle p_i, w_i\rangle, \theta_i) \models_\nu \varphi_2$, $\forall 0 \leq j < i, (\langle p_j, w_j\rangle, \theta_j) \models_\nu \varphi_1$.
• $(\langle p_0, w_0\rangle, \theta_0) \models_\nu A[\varphi_1 \widetilde{U} \varphi_2]$ iff for every path $(\langle p_0, w_0\rangle, \theta_0)(\langle p_1, w_1\rangle, \theta_1)\cdots$ of $\mathcal{P}$ starting from $(\langle p_0, w_0\rangle, \theta_0)$, $\forall i \geq 0$, if $(\langle p_i, w_i\rangle, \theta_i) \not\models_\nu \varphi_2$, then $\exists 0 \leq j < i, (\langle p_j, w_j\rangle, \theta_j) \models_\nu \varphi_1$.
• $(\langle p_0, w_0\rangle, \theta_0) \models_\nu E[\varphi_1 \widetilde{U} \varphi_2]$ iff $\exists$ a path $(\langle p_0, w_0\rangle, \theta_0)(\langle p_1, w_1\rangle, \theta_1)\cdots$ of $\mathcal{P}$ starting with $(\langle p_0, w_0\rangle, \theta_0)$, s.t. $\forall i \geq 0$, if $(\langle p_i, w_i\rangle, \theta_i) \not\models_\nu \varphi_2$, then $\exists 0 \leq j < i, (\langle p_j, w_j\rangle, \theta_j) \models_\nu \varphi_1$.

Standard CTL operators can be expressed by the above operators: $\mathbf{EF}\psi = \mathbf{E}[true\mathbf{U}\psi]$, $\mathbf{AF}\psi = \mathbf{A}[true\mathbf{U}\psi]$, $\mathbf{EG}\psi = \mathbf{E}[false\widetilde{U}\psi]$, $\mathbf{AG}\psi = \mathbf{A}[false\widetilde{U}\psi]$.

4.1.2 Self-modifying Alternating Büchi Pushdown Systems

Definition 9 *A Self Modifying Alternating Büchi Pushdown System (SM-ABPDS) is a tuple $\mathcal{BP} = (P, \Gamma, \Delta, \Delta_c, F)$, where P is a finite set of control points, Γ is a finite set of stack symbols, F is the set of final states, $\Delta \subseteq (P \times \Gamma) \times 2^{2^{\Delta \cup \Delta_c} \cup \{-\}} \times 2^{P \times \Gamma^*}$ is a finite set of transition rules in the form $\langle p, \gamma\rangle \xrightarrow{[\sigma_1, \cdots, \sigma_n]} \{\langle p_1, w_1\rangle, \cdots, \langle p_n, w_n\rangle\}$ where $[\sigma_1, \cdots, \sigma_n]$ is an ordered set and $\forall 1 \leq i \leq n, \sigma_i$ is either a set of rules $\sigma_i \subseteq \Delta \cup \Delta_c$ or $\sigma_i = -$, and $\Delta_c \subseteq P \times 2^{\Delta \cup \Delta_c} \times 2^{\Delta \cup \Delta_c} \times P$ is a finite set of modifying transition rules in the form $p \xrightarrow{(\sigma, \sigma')} p'$ where $\sigma, \sigma' \subseteq \Delta \cup \Delta_c$. A configuration of a SM-ABPDS is a tuple of the form $(\langle p, w\rangle, \theta)$ where $p \in P$, $w \in \Gamma^*$ and $\theta \subseteq \Delta \cup \Delta_c$ is the current phase.*

$\mathcal{BP}$ defines the transition relation $\Rightarrow_{\mathcal{BP}} \subseteq (P \times \Gamma^* \times 2^{\Delta \cup \Delta_c}) \times 2^{(P \times \Gamma^* \times 2^{\Delta \cup \Delta_c})}$ between configurations as follows: Let $\theta \subseteq \Delta \cup \Delta_c$, $\gamma \in \Gamma, w \in \Gamma^*$, and $p \in P$, then:

1. If $r : \langle p, \gamma\rangle \xrightarrow{[\sigma_1, \cdots, \sigma_m]} \{\langle p_1, w_1\rangle, \cdots, \langle p_m, w_m\rangle\}$ is a rule in $\Delta \cap \theta$, if either for every $1 \leq i \leq m, \sigma_i = -$ or $\exists 1 \leq i \leq m, \sigma_i \cap \theta \neq \emptyset$, then $(\langle p, \gamma w\rangle, \theta) \Rightarrow_{\mathcal{BP}} \{(\langle p_i, w_i w\rangle, \theta) | \sigma_i = -, 1 \leq i \leq m\} \cup \{(\langle p_i, w_i w\rangle, \theta) | \sigma_i \cap \theta \neq \emptyset, 1 \leq i \leq m\}$.

2. If $r : p \xhookrightarrow{(\sigma,\sigma')} p'$ is a rule in $\Delta_c \cap \theta$, then $(\langle p, w\rangle, \theta) \Rightarrow_{\mathcal{BP}} \{(\langle p', w\rangle, \theta')\}, \theta' = \theta \backslash \sigma \cup \sigma'$.

Intuitively, $[\sigma_1, \cdots, \sigma_m]$ in the transition $r : \langle p, \gamma\rangle \xhookrightarrow{[\sigma_1,\cdots,\sigma_m]} \{\langle p_1, w_1\rangle, \cdots, \langle p_m, w_m\rangle\}$ ensures that for a given configuration $(\langle p, \gamma w\rangle, \theta)$, for every $1 \leq i \leq n$, $(\langle p_i, w_i w\rangle, \theta)$ is in the set of immediate successor iff

- either for every $1 \leq j \leq n, \sigma_j = -$;
- or $\sigma_i = -$ and $\exists j \neq i, 1 \leq j \leq n$ s.t. $\sigma_j \cap \theta \neq \emptyset$
- or $\sigma_i \cap \theta \neq \emptyset$

Note that $-$ means that there is no constraint on whether θ contains a rule in σ_i or not.

For every $c \in P \times \Gamma^* \times 2^{\Delta\cup\Delta_c}$ and $C \subseteq P \times \Gamma^* \times 2^{\Delta\cup\Delta_c}$, if $c \Rightarrow_{\mathcal{BP}} C$ then c is an immediate predecessor of C and C is an immediate successor of c. Let $\Rightarrow^*_{\mathcal{BP}} \subseteq (P \times \Gamma^* \times 2^{\Delta\cup\Delta_c}) \times 2^{(P\times\Gamma^*\times 2^{\Delta\cup\Delta_c})}$ be the reflexive transitive closure of $\Rightarrow_{\mathcal{BP}}$ defined as follows: (1) $\forall c \in P \times \Gamma^* \times 2^{\Delta\cup\Delta_c}$, $c \Rightarrow^*_{\mathcal{BP}} \{c\}$, (2) if $c \Rightarrow_{\mathcal{BP}} C$, then $c \Rightarrow^*_{\mathcal{BP}} C$, and (3) if $c \Rightarrow_{\mathcal{BP}} \{c_1, ...c_n\}$ and $c_i \Rightarrow^*_{\mathcal{BP}} C_i$ for every $1 \leq i \leq n$, then $c \Rightarrow^*_{\mathcal{BP}} \bigcup_{i=1}^n C_i$. Given a set of configurations C, we define the sets $pre_{\mathcal{BP}}(C), pre^*_{\mathcal{BP}}(C)$ and $pre^+_{\mathcal{BP}}(C)$ as follows: $pre_{\mathcal{BP}}(C) = \{c \in P \times \Gamma^* \times 2^{\Delta\cup\Delta_c} \mid \exists C' \subseteq C$ s.t. C' is an immediate successor of $c\}$, $pre^*_{\mathcal{BP}}(C) = \{c \in P \times \Gamma^* \times 2^{\Delta\cup\Delta_c}, \exists C' \subseteq C$ s.t. $c \Rightarrow^*_{\mathcal{BP}} C'\}$ and $pre^+_{\mathcal{BP}}(C) = pre_{\mathcal{BP}} \circ pre^*_{\mathcal{BP}}(C)$. We omit the subscript $\mathcal{BP}$ when it is clear from the context.

A run ρ of $\mathcal{BP}$ starting from an initial configuration c_0 is a tree whose root is labelled by c_0 and whose other nodes are labelled by configurations of $P \times \Gamma^* \times 2^{\Delta\times\Delta_c}$. A node of ρ labelled by configuration c has n children labelled by $c_1, \ldots, c_n$, respectively, iff $c \Rightarrow_{\mathcal{BP}} \{c_1, ...c_n\}$. A path $c_0c_1\cdots$ of a run ρ is an infinite sequence of configurations s.t. c_0 is the root of ρ and c_{i+1} is one child of c_i. A path is accepting iff it visits some configurations with control locations in F infinitely often. A run is accepting iff all its paths are accepting. A configuration c is accepted by $\mathcal{BP}$ iff it is the root of a run accepted by $\mathcal{BP}$. The language of $\mathcal{BP}$, $L(\mathcal{BP})$, is the set of configurations accepted by $\mathcal{BP}$.

We assume w.l.o.g. that for every rule in Δ_c of the form $r : p \xhookrightarrow{(\sigma,\sigma')} p'$, $r \notin \sigma$.

Representing potentially infinite sets of configurations of SM-ABPDSs.

Alternating Multi-Automata (AMA) were introduced in [6] to finitely represent regular sets of configurations of an alternating PDS. In order to adapt AMA to represent regular sets of SM-ABPDS, we extend this notion taking phases into account as follows:

Definition 10 *Let $\mathcal{BP} = (P, \Gamma, \Delta, \Delta_c, F)$ be a SM-ABPDS. An Extended Alternating Multi-Automaton (EAMA) is a tuple $\mathcal{A} = (Q, \Gamma, T, I, Q_F)$ where $I \subseteq P \times 2^{\Delta \cup \Delta_c} \subseteq Q$ is the set of initial states, $T \subseteq Q \times (\Gamma \cup \{\epsilon\}) \times 2^Q$ is the set of transitions, $Q_F \subseteq Q$ is a finite set of final states.*

Let $\rightarrow_T$ be the transition relation defined as follows: (1) $\forall q \in Q, q \xrightarrow{\epsilon}_T \{q\}$ where ϵ is the empty word; (2) if $(q, \gamma, \{q_1, \cdots, q_n\}) \in T$, $q \xrightarrow{\gamma}_T \{q_1, \cdots, q_n\}$; and (3) if $q \xrightarrow{\gamma}_T \{q_1, \cdots, q_n\}$ and $q_i \xrightarrow{w}_T Q_i$ for every $1 \leq i \leq n$, then $q \xrightarrow{\gamma w}_T \bigcup_{i=1}^{n} Q_i$.

A configuration $(\langle p, w \rangle, \theta)$ is accepted by the EAMA $\mathcal{A}$ iff $(p, \theta) \in I$ and $\exists\ Q' \subseteq Q_F$ such that $(p, \theta) \xrightarrow{w}_T Q'$. Let $L(\mathcal{A})$ be the set of configurations accepted by $\mathcal{A}$. Let $\mathcal{C}$ be a set of configurations of the SM-ABPDS $\mathcal{BP}$. $\mathcal{C}$ is regular if there exists an EAMA $\mathcal{A}$ such that $\mathcal{C} = L(\mathcal{A})$.

4.1.3 From CTL Model-Checking of SM-PDSs to the emptiness problem of SM-ABPDSs

Let $\mathcal{P} = (P, \Gamma, \Delta, \Delta_c)$ be a Self Modifying Pushdown System with an initial configuration $c_0 = (\langle p_0, w_0 \rangle, \theta_0)$. We suppose w.l.o.g. that $\mathcal{P}$ has a bottom stack symbol $\sharp \in \Gamma$ that is never popped from the stack i.e. there is no transition rule of the from $\langle p, \sharp \rangle \hookrightarrow \langle p', w \rangle \in \Delta$. Given a set of atomic propositions At, let $\nu : P \rightarrow 2^{At}$ be a labeling function that associates each control location to a set of atomic propositions. Let φ be a CTL formula over At. Our goal is to check whether $c_0 \models_\nu \varphi$. This can be done by translating the SM-PDS into an equivalent PDS as described in Chapter 2, and then applying the standard CTL model-checking algorithm for PDSs [40]. However, as will be shown in the experiments section (Section 4.3), this approach is not efficient. Thus, we need a **direct** algorithm that operates directly on the SM-PDS without translating it into a PDS. We provide in this section a **direct** algorithm that performs CTL model-checking on SM-PDSs. To this aim, we will compute a kind of product of the SM-PDS with φ: we construct a Self Modifying Alternating Büchi Pushdown

System $\mathcal{BP}_\varphi$ s.t. $\mathcal{BP}_\varphi$ accepts a configuration c iff $c \models_\nu \varphi$. Thus, determining whether $c_0 \models_\nu \varphi$ can be reduced to checking whether $c_0 \in L(\mathcal{BP}_\varphi)$.

Let $\mathcal{BP}_\varphi = (P', \Gamma, \Delta', \Delta'_c, F)$ be the SM-ABPDS defined as follows: $P' = P \times cl(\varphi) \cup P^{cl(\varphi)}$, where $P^{cl(\varphi)}$ is the set of control locations in the form p^ψ where $p \in P$ and $\psi \in cl(\varphi)$, $F = \{[p,a] | \; a \in cl(\varphi) \cap At$ and $a \in \nu(p)\} \cup \{[p, \neg a] | \neg a \in cl(\varphi), a \in At$ and $a \notin \nu(p)\} \cup P \times cl_{\widetilde{U}}(\varphi)$ where $cl_{\widetilde{U}}(\varphi)$ is the set of formulae of $cl(\varphi)$ in the form $E[\psi_1 \widetilde{U} \psi_2]$ or $A[\psi_1 \widetilde{U} \psi_2]$. In what follows, to compute Δ' and Δ'_c, every rule $r \in \Delta \cup \Delta_c$ leads to a set of rules $\{r'_1, \cdots, r'_n\}$ of $\Delta' \cup \Delta'_c$, we call this set of rules $prod(r)$. Moreover, let $prod_E(r) \subseteq prod(r)$ be the set of rules generated from r using subformulas of the form $EX\psi_1$, $E[\psi_1 U \psi_2]$ or $E[\psi_1 \widetilde{U} \psi_2]$(see below for more details about $prod(r)$ and $prod_E(r)$).

The transition relations Δ' and Δ'_c (resp. the sets $prod(r)$ and $prod_E(r)$, for every $r \in \Delta \cup \Delta_c$) are the smallest sets of transitions (resp. of sets of rules) defined as follows: Initially, $\Delta' = \Delta'_c = \emptyset$, $prod_E(r) = \emptyset$ and $prod(r) = \emptyset$, $\forall$ $r \in \Delta \cup \Delta_c$. $\forall p \in P$, $\forall \psi \in cl(\varphi)$ and $\forall \gamma \in \Gamma$, we have:

1. if $\psi = a, a \in At$ and $a \in \nu(p)$; $\langle [p,a], \gamma \rangle \xhookrightarrow{[-]} \langle [p,a], \gamma \rangle \in \Delta'$
2. if $\psi = \neg a, a \in At$ and $a \notin \nu(p)$; $\langle [p,\psi], \gamma \rangle \xhookrightarrow{[-]} \langle [p,\psi], \gamma \rangle \in \Delta'$
3. if $\psi = \psi_1 \vee \psi_2$; $\langle [p,\psi], \gamma \rangle \xhookrightarrow{[-]} \langle [p,\psi_1], \gamma \rangle \in \Delta'$ and $\langle [p,\psi], \gamma \rangle \xhookrightarrow{[-]} \langle [p,\psi_2], \gamma \rangle \in \Delta'$
4. if $\psi = \psi_1 \wedge \psi_2$; $\langle [p,\psi], \gamma \rangle \xhookrightarrow{[-,-]} \{\langle [p,\psi_1], \gamma \rangle, \langle [p,\psi_2], \gamma \rangle\} \in \Delta'$
5. if $\psi = EX\psi_1$, then:
   (a) if $p \in P_N$, for every $R = \langle p, \gamma \rangle \hookrightarrow \langle p', w \rangle \in \Delta$, $R' = \langle [p,\psi], \gamma \rangle \xhookrightarrow{[-]} \langle [p',\psi_1], w \rangle \in \Delta'$, $R' \in prod_E(R)$ and $R' \in prod(R)$ ($R' \in prod(R)$ means that R' is generated from R and $R' \in prod_E(R)$ means that R' is generated from R using a formula of the form $EX\psi_1$, $E[\psi_1 U \psi_2]$ or $E[\psi_1 \widetilde{U} \psi_2]$.)
   (b) if $p \in P_C$, for every $R = p \xhookrightarrow{(r_1,r_2)} p' \in \Delta_c$, $R' = [p,\psi] \xhookrightarrow{(\sigma,\sigma')} [p',\psi_1] \in \Delta'_c$ where $\sigma = prod(r_1), \sigma' = prod(r_2)$, $R' \in prod_E(R)$ and $R' \in prod(R)$
6. if $\psi = AX\psi_1$, then:
   (a) if $p \in P_N$, let $\{R_1 = \langle p, \gamma \rangle \hookrightarrow \langle p_1, w_1 \rangle, \cdots, R_n = \langle p, \gamma \rangle \hookrightarrow \langle p_n, w_n \rangle\}$ be the set of all the rules of Δ that have $\langle p, \gamma \rangle$ in the left-hand-side. Then, $R' = \langle [p,\psi], \gamma \rangle \xhookrightarrow{[\sigma_1,\cdots,\sigma_n]} \{\langle [p_1,\psi_1], w_1 \rangle, \cdots, \langle [p_n,\psi_1], w_n \rangle\} \in \Delta'$, where for every $1 \leq i \leq n$, $\sigma_i = prod_E(R_i)$ and $R' \in prod(R_i)$.

(b) if $p \in P_C$, let $\{R_1 = p \xhookrightarrow{(r_1,r_1')} p_1, \cdots, R_n = p \xhookrightarrow{(r_n,r_n')} p_n\}$ be the set of all the rules of Δ_c that have p in the left-hand-side. Then, for every $\gamma \in \Gamma$, $R'_\perp = \langle [p,\psi],\gamma\rangle \xhookrightarrow{[\sigma_1,\cdots,\sigma_n]} \{\langle p_1^\psi,\gamma\rangle,\cdots,\langle p_n^\psi,\gamma\rangle\} \in \Delta'$ and for every $1 \leq i \leq n$, $R'_i : p_i^\psi \xhookrightarrow{(\sigma,\sigma')} [p_i,\psi_1] \in \Delta'_c$, where for every $1 \leq i \leq n, \sigma_i = prod_E(R_i)$, $\sigma = prod(r_i), \sigma' = prod(r'_i)$, and for every $1 \leq i \leq n$, $R'_\perp$,$R' \in prod(R_i)$.

7. if $\psi = E[\psi_1 U \psi_2]$, then $\langle [p,\psi],\gamma\rangle \xhookrightarrow{[-]} \langle [p,\psi_2],\gamma\rangle \in \Delta'$ and:

   (a) if $p \in P_N$, for every $R = \langle p,\gamma\rangle \hookrightarrow \langle p',w\rangle \in \Delta$,
   $R' = \langle [p,\psi],\gamma\rangle \xhookrightarrow{[-,-]} \{\langle [p,\psi_1],\gamma\rangle, \langle [p',\psi],w\rangle\} \in \Delta'$, $R' \in prod_E(R)$ and $R' \in prod(R)$.

   (b) if $p \in P_C$, for every $R = p \xhookrightarrow{(r_1,r_1')} p' \in \Delta_c$, then for every $\gamma \in \Gamma$, $R'_\perp = \langle [p,\psi],\gamma\rangle \xhookrightarrow{[-,-]} \{\langle [p,\psi_1],\gamma\rangle, \langle p^\psi,\gamma\rangle\} \in \Delta'$ and $p^\psi \xhookrightarrow{(\sigma,\sigma')} [p',\psi] \in \Delta'_c$ where $\sigma = prod(r_1), \sigma' = prod(r'_1)$, $R'_\perp, R' \in prod_E(R)$ and $R'_\perp, R' \in prod(R)$.

8. if $\psi = A[\psi_1 U \psi_2]$, then $\langle [p,\psi],\gamma\rangle \xhookrightarrow{[-]} \langle [p,\psi_2],\gamma\rangle \in \Delta'$, and:

   (a) if $p \in P_N$, let $\{R_1 = \langle p,\gamma\rangle \hookrightarrow \langle p_1,w_1\rangle, \cdots, R_n = \langle p,\gamma\rangle \hookrightarrow \langle p_n,w_n\rangle\}$ be the set of all the rules of Δ that have $\langle p,\gamma\rangle$ in the left-hand-side. Then, $R' = \langle [p,\psi],\gamma\rangle \xhookrightarrow{[-,\sigma_1,\cdots,\sigma_n]} \{\langle [p,\psi_1],\gamma\rangle, \langle [p_1,\psi],w_1\rangle, \cdots, \langle [p_n,\psi],w_n\rangle\} \in \Delta'$ where for every $1 \leq i \leq n$, $\sigma_i = prod_E(R_i)$, and $R' \in prod(R_i)$.

   (b) if $p \in P_C$, let $\{R_1 = p \xhookrightarrow{(r_1,r_1')} p_1, \cdots, R_n = p \xhookrightarrow{(r_n,r_n')} p_n\}$ be the set of all the rules of Δ_c that have p in the left-hand-side. Then, $\forall 1 \leq i \leq n$, for every $\gamma \in \Gamma$, $R_\perp : \langle [p,\psi],\gamma\rangle \xhookrightarrow{[-,\sigma_1,\cdots,\sigma_n]} \{\langle [p,\psi_1],\gamma\rangle, \langle p_1^\psi,\gamma\rangle, \cdots, \langle p_n^\psi,\gamma\rangle\} \in \Delta'$ and $R'_i : p_i^\psi \xhookrightarrow{(\sigma,\sigma')} [p_i,\psi] \in \Delta'_c$ where for every $1 \leq i \leq n, \sigma_i = prod_E(R_i)$, $\sigma = prod(r_i)$, $\sigma' = prod(r'_i)$ and $R_\perp, R'_i \in prod(R_i)$.

9. if $\psi = E[\psi_1 \widetilde{U} \psi_2]$, then $\langle [p,\psi],\gamma\rangle \xhookrightarrow{[-,-]} \{\langle [p,\psi_2],\gamma\rangle, \langle [p,\psi_1],\gamma\rangle\} \in \Delta'$ and:

   (a) if $p \in P_N$, then for every $R = \langle p,\gamma\rangle \hookrightarrow \langle p',w\rangle \in \Delta$,
   $R' = \langle [p,\psi],\gamma\rangle \xhookrightarrow{[-,-]} \{\langle [p,\psi_2],\gamma\rangle, \langle [p',\psi],w\rangle\} \in \Delta'$, $R' \in prod_E(R)$ and $R' \in prod(R)$.

   (b) if $p \in P_C$, then for every $R = p \xhookrightarrow{(r_1,r_1')} p' \in \Delta_c$, for every $\gamma \in \Gamma$, $R_\perp = \langle [p,\psi],\gamma\rangle \xhookrightarrow{[-,-]} \{\langle [p,\psi_2],\gamma\rangle, \langle p^\psi,\gamma\rangle\} \in \Delta'$ and $R' : p^\psi \xhookrightarrow{(\sigma,\sigma')} [p',\psi] \in \Delta'_c$ where $\sigma = prod(r_1), \sigma' = prod(r'_1)$, $R_\perp, R' \in prod_E(R)$ and $R_\perp, R' \in prod(R)$.

10. if $\psi = A[\psi_1 \widetilde{U} \psi_2]$, then $\langle [p,\psi], \gamma\rangle \xhookrightarrow{[-,-]} \{\langle [p,\psi_2],\gamma\rangle, \langle [p,\psi_1],\gamma\rangle\} \in \Delta'$, and:

    (a) if $p \in P_N$, let $\{R_1 = \langle p,\gamma\rangle \hookrightarrow \langle p_1, w_1\rangle, \cdots, R_n = \langle p,\gamma\rangle \hookrightarrow \langle p_n, w_n\rangle\}$ be the set of all the rules of Δ that have $\langle p,\gamma\rangle$ in the left-hand-side. Then for every $1 \leq i \leq n$, $\sigma_i = prod_E(R_i)$, $R' = \langle [p,\psi],\gamma\rangle \xhookrightarrow{[-,\sigma_1,\cdots,\sigma_n]} \{\langle [p,\psi_2],\gamma\rangle, \langle [p_1,\psi], w_1\rangle, \cdots, \langle [p_n,\psi], w_n\rangle\} \in \Delta'$ and $R' \in prod(R_i)$.

    (b) if $p \in P_C$, let $\{R_1 = p \xhookrightarrow{(r_1,r'_1)} p_1, \cdots, R_n = p \xhookrightarrow{(r_n,r'_n)} p_n\}$ be the set of all the rules of Δ_c that have p in the left-hand-side. Then, for every $\gamma \in \Gamma, R_\perp = \langle [p,\psi],\gamma\rangle \xhookrightarrow{[-,\sigma_1,\cdots,\sigma_n]} \{\langle [p,\psi_2],\gamma\rangle, \langle p_1^\psi,\gamma\rangle, \cdots, \langle p_n^\psi,\gamma\rangle\} \in \Delta', \forall 1 \leq i \leq n, \sigma_i = prod_E(R_i)$ and for every $1 \leq i \leq n, R'_i : p_i^\psi \xhookrightarrow{(\sigma,\sigma')} [p_i,\psi] \in \Delta'_c$ where $\sigma = prod(r_i), \sigma' = prod(r'_i)$ and $R_\perp, R'_i \in prod(R_i)$.

Let $prod(\Delta) = \{r' \in \Delta' \mid \exists r \in \Delta, r' \in prod(r)\}$ be the set of rules of Δ' that are generated from Δ. Let $\delta = \Delta' \setminus prod(\Delta)$ be the set of rules of Δ' that are not generated from any rule of Δ nor Δ_c (e.g., the rules computed by items 1, 2, 3 and 4 are in δ). These rules δ are independent of Δ and Δ_c. They are introduced by the structure of φ. Thus, they need to be present in all the phases of $\mathcal{BP}_\varphi$. Let then $\theta \subseteq \Delta \cup \Delta_c$ be a phase of $\mathcal{P}$. Its corresponding phase in $\mathcal{BP}_\varphi$ is $\beta(\theta) = prod(\theta) \cup \delta$, where $prod(\theta) = \{r' \in \Delta' \cup \Delta'_c \mid \exists r \in \theta, r' \in prod(r)\}$.

Let us explain the above construction intuitively. The above automaton $\mathcal{BP}_\varphi$ can be seen as a kind of product of the SM-PDS $\mathcal{P}$ with the formula φ. For $\psi \in cl(\varphi)$, $(\langle p,w\rangle,\theta) \models_\nu \psi$ iff $\mathcal{BP}_\varphi$ accepts a configuration $\big(\langle [p,\psi], w\rangle, \beta(\theta)\big)$. We give in what follows the intuition behind all the items above:

If $\psi = a \in At$, then $\forall w \in \Gamma^*, \theta \subseteq \Delta \cup \Delta_c$, $(\langle p,w\rangle,\theta) \models_\nu \psi$ iff $a \in \nu(p)$. Hence, the automaton $\mathcal{BP}_\varphi$ should accept a run starting from $(\langle [p,a], w\rangle, \beta(\theta))$ iff $a \in \nu(p)$. $[p,a] \in F$ iff $a \in \nu(p)$. Thus, the loop added in $(\langle [p,a], w\rangle, \beta(\theta))$ by Item 1 makes sure that $\mathcal{BP}_\varphi$ accepts this run.

If $\psi = \neg a$, then $\forall w \in \Gamma^*, \theta \subseteq \Delta \cup \Delta_c$, $(\langle p,w\rangle,\theta) \models_\nu \psi$ iff $a \notin \nu(p)$. Hence, the automaton $\mathcal{BP}_\varphi$ should accept a run starting from $(\langle [p,\neg a], w\rangle, \beta(\theta))$ iff $a \notin \nu(p)$. $[p,\neg a] \in F$ iff $a \notin \nu(p)$. Thus, the loop in $(\langle [p,\neg a], w\rangle, \beta(\theta))$ added by Item 2 ensures that $\mathcal{BP}_\varphi$ accepts this run.

If $\psi = \psi_1 \vee \psi_2$, then $\forall w \in \Gamma^*, \theta \subseteq \Delta \cup \Delta_c$, $(\langle p,w\rangle,\theta) \models_\nu \psi$ iff ($(\langle p,w\rangle,\theta) \models_\nu \psi_1$ or $(\langle p,w\rangle,\theta) \models_\nu \psi_2$). Thus, $\mathcal{BP}_\varphi$ accepts a run starting from $(\langle [p,\psi_1 \vee \psi_2], w\rangle, \beta(\theta))$ iff $\mathcal{BP}_\varphi$ has an accepting run starting from $(\langle [p,\psi_1], w\rangle, \beta(\theta))$ or $(\langle [p,\psi_2], w\rangle, \beta(\theta))$. This is ensured by Item 3. Item 4 is similar to Item 3, it

handles the case $\psi = \psi_1 \wedge \psi_2$ where $(\langle p, w\rangle, \theta)$ satisfies ψ iff it satisfies both ψ_1 and ψ_2.

If $\psi = EX\psi_1$, then $\forall w \in \Gamma^*, \theta \subseteq \Delta \cup \Delta_c$, $(\langle p, w\rangle, \theta) \models_\nu \psi$ iff an immediate successor $(\langle p', w'\rangle, \theta')$ of $(\langle p, w\rangle, \theta)$ satisfies ψ_1. Thus, $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \psi], w\rangle, \beta(\theta))$ iff it can accept a run from $(\langle [p', \psi_1], w'\rangle, \beta(\theta'))$. There are two cases depending on whether $p \in P_N$ or $p \in P_c$, because the form of the rules of the SM-PDS depends on whether $p \in P_N$ or $p \in P_c$: if $p \in P_N$, then necessarily, the rules that can be applied from p are of the form $\langle p, \gamma\rangle \hookrightarrow \langle p', w\rangle \in \Delta$, whereas if $p \in P_c$, then necessarily, the rules that can be applied from p are of the form $r : p \xhookrightarrow{(r_1,r_2)} p' \in \Delta_c$. Thus, if $p \in P_N$, then $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \psi], \gamma u\rangle, \beta(\theta))$ iff there exists a rule $\langle p, \gamma\rangle \hookrightarrow \langle p', w\rangle \in \Delta$ such that $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p', \psi_1], wu\rangle, \beta(\theta))$. This is ensured by Item 5(a). If $p \in P_c$, then $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \psi], \gamma u\rangle, \beta(\theta))$ iff there exists a rule $r : p \xhookrightarrow{(r_1,r_2)} p' \in \Delta_c \cap \theta$ such that $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p', \psi_1], \gamma u\rangle, \beta(\theta'))$, where $\theta' = (\theta \backslash \{r_1\}) \cup \{r_2\}$. This is ensured by Item 5(b).

If $\psi = AX\psi_1$, then $\forall w \in \Gamma^*, \theta \subseteq \Delta \cup \Delta_c$, $(\langle p, w\rangle, \theta) \models_\nu \psi$ iff every immediate successor $(\langle p', w'\rangle, \theta')$ of $(\langle p, w\rangle, \theta)$ satisfies ψ_1. Thus, $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \psi], w\rangle, \beta(\theta))$ iff it can accept a run from all its immediate successors $(\langle [p', \psi_1], w'\rangle, \beta(\theta'))$. As previously, there are two cases depending on whether $p \in P_N$ or $p \in P_c$: if $p \in P_N$, let $\gamma \in \Gamma$ and $u \in \Gamma^*$ such that $w = \gamma u$. Let then $\{\langle p, \gamma\rangle \hookrightarrow \langle p_1, w_1\rangle, \cdots, \langle p, \gamma\rangle \hookrightarrow \langle p_m, w_m\rangle\}$ be the set of all the rules of $\Delta \cap \theta$ that have $\langle p, \gamma\rangle$ in the left-hand-side. Then, $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \psi], \gamma u\rangle, \beta(\theta))$ iff $\mathcal{BP}_\varphi$ has an accepting run from every $(\langle [p_i, \psi_1], w_i u\rangle, \beta(\theta))$, $1 \leq i \leq m$. This is ensured by Item 6(a). Note that Item 6(a) considers all the rules $R_i : \langle p, \gamma\rangle \hookrightarrow \langle p_i, w_i\rangle$ that are in Δ (even those that are not in θ), then the constraints $[\sigma_1, \cdots, \sigma_n]$ of the rule R' of Item 6(a) ensures that only the R_i's that are in θ are applied. Note also that in R', $\sigma_i = prod_E(R_i)$ ensures that $\sigma_i \cap \beta(\theta) \neq \emptyset$ iff $R_i \cap \theta \neq \emptyset$. Here taking $\sigma_i = prod(R_i)$ is not correct because $R' \in prod(R_i)$ and so in this case, $\sigma_i \cap \beta(\theta)$ would always be nonempty. On the other hand, if $p \in P_c$, let $\{p \xhookrightarrow{(r_1,r_1')} p_1, \cdots, p \xhookrightarrow{(r_m,r_m')} p_m\}$ be the set of all the rules of $\Delta_c \cap \theta$ that have p in the left-hand-side. Then $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \psi], \gamma u\rangle, \beta(\theta))$ iff $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p_i, \psi_1], \gamma u\rangle, \beta(\theta_i))$, for every $1 \leq i \leq m$, where $\theta_i = (\theta \backslash \{r_i\}) \cup \{r_i{}'\}$. This is ensured by Item 6(b). As previously, Item 6(b) considers all the rules $R_i : p \xhookrightarrow{(r_i,r_i')} p_i$ that

are in Δ_c (even those that are not in θ), then the constraints $[\sigma_1, \cdots, \sigma_n]$ of the rule $R'_\perp = \langle [p,\psi], \gamma\rangle \xrightarrow{[\sigma_1,\cdots,\sigma_n]} \{\langle p_1^\psi, \gamma\rangle, \cdots, \langle p_n^\psi, \gamma\rangle\}$ of Item 6(b) ensures that only the R_i's that are in θ are applied. Then $R'_i : p_i^\psi \xrightarrow{(\sigma,\sigma')} [p_i, \psi_1]$ ensures $\mathcal{BP}_\varphi$ has an accepting run from $(\langle p_i^\psi, \gamma u\rangle, \beta(\theta))$ iff $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p_i, \psi_1], \gamma u\rangle, \beta(\theta_i))$ where $\theta_i = (\theta \backslash \{r_i\}) \cup \{r_i'\}$ for $1 \leq i \leq n$. Note that $\sigma' = \mathit{prod}(r_i)$ and $\sigma' = \mathit{prod}(r'_i)$, then $\beta(\theta_i) = \beta(\theta) \setminus \sigma \cup \sigma'$. Thus, $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p,\psi], \gamma u\rangle, \beta(\theta))$ iff $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p_i,\psi_1], \gamma u\rangle, \beta(\theta_i))$ for every $1 \leq i \leq n$.

If $\psi = E[\psi_1 U \psi_2]$, then $\forall w \in \Gamma^*, \theta \subseteq \Delta \cup \Delta_c$, $(\langle p, w\rangle, \theta) \models_\nu \psi$ iff either it satisfies ψ_2 or it satisfies ψ_1 and there exists an immediate successor satisfying ψ. Thus, $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p,\psi], w\rangle, \beta(\theta))$ iff:

1. $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p,\psi_2], w\rangle, \beta(\theta))$. This is handled by the rules $\langle [p,\psi], \gamma\rangle \xrightarrow{[-,-]} \{\langle [p,\psi_2], \gamma\rangle, \langle [p,\psi_1], \gamma\rangle\}$ introduced by Item 7.

2. or $\mathcal{BP}_\varphi$ has an accepting run from both $(\langle [p,\psi_1], w\rangle, \beta(\theta))$ and $(\langle [p',\psi], w'\rangle, \beta(\theta'))$ where $(\langle p', w'\rangle, \theta')$ is an immediate successor of $(\langle p, w\rangle, \theta))$. There are two cases depending on whether $p \in P_N$ or $p \in P_C$: the case $p \in P_N$ is handled by Item 7(a). Its intuition is similar to the intuition behind the previous items. Let then $p \in P_C$. Then there exists a rule $r : p \xrightarrow{(r_1, r_1')} p' \in \theta \cap \Delta_c$ such that $\mathcal{BP}_\varphi$ has an accepting run from both $(\langle [p,\psi_1], w\rangle, \beta(\theta))$ and $(\langle [p',\psi], w\rangle, \beta(\theta'))$, where $\theta' = \theta \backslash \{r_1) \cup \{r_1'\}$. This is ensured by the rule $R_\perp = \langle [p,\psi], \gamma\rangle \xrightarrow{[-,-]} \{\langle [p,\psi_2], \gamma\rangle, \langle p^\psi, \gamma\rangle\} \in \Delta'$ and $R' : p^\psi \xrightarrow{(\sigma,\sigma')} [p', \psi] \in \Delta'_c$ added by Item 7(b).

The case $\psi = A[\psi_1 U \psi_2]$ is handled in a similar way using Items 8. If $\psi = E[\psi_1 \widetilde{U} \psi_2]$, then $\forall w \in \Gamma^*, \theta \subseteq \Delta \cup \Delta_c$, $(\langle p, w\rangle, \theta) \models_\nu \psi$ iff it satisfies ψ_2 and either it satisfies also ψ_1, or it has a successor $(\langle p', w'\rangle, \theta')$ that satisfies ψ. Then, $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p,\psi], w\rangle, \theta)$ iff $\mathcal{BP}_\varphi$ has an accepting run from both $((\langle [p,\psi_2], w\rangle, \beta(\theta))$ and $(\langle [p,\psi_1], w\rangle, \beta(\theta)))$, or it has an accepting run from both $((\langle [p,\psi_2], w\rangle, \beta(\theta))$ and $(\langle [p',\psi], w'\rangle, \beta(\theta')))$. This case is handled by Items 9. To ensure that the runs on which ψ_2 always holds are accepted, we add $[p,\psi]$ in F. The case where $\psi = A[\psi_1 \widetilde{U} \psi_2]$ is handled similarly by Items 10.

We can show that:

Theorem 4.1.1 *Let $(\langle p, w\rangle, \theta)$ be a configuration of the SM-PDS $\mathcal{P}$. $(\langle p, w\rangle, \theta) \models_\nu \varphi$ iff $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \varphi], w\rangle, \beta(\theta))$.*

Proof: $(\Rightarrow)$: Suppose $(\langle p, w\rangle, \theta) \models_\nu \psi$, we show that $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \psi], w\rangle, \beta(\theta))$ by induction on the structure of ψ.

Case $\psi = a$: Since $(\langle p, w\rangle, \theta) \models_\nu \psi$, then $a \in \nu(p)$. By the definition of $\mathcal{BP}_\varphi$, $[p, a] \in F$ and $\forall \gamma \in \Gamma$, $r : \langle [p, a], \gamma\rangle \xhookrightarrow{[-]} \langle [p, a], \gamma\rangle \in \Delta'$ and $r \in \delta$ (since it is not from Δ nor Δ_c). Then, $r \in \beta(\theta)$ and there is a loop in $(\langle [p, \psi], w\rangle, \beta(\theta))$. Hence, $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \psi], w\rangle, \beta(\theta))$.

Case $\psi = \neg a$: Since $(\langle p, w\rangle, \theta) \models_\nu \psi$, then $a \notin \nu(p)$. By the definition of $\mathcal{BP}_\varphi$, $[p, \neg a] \in F$ and $\forall \gamma \in \Gamma$, $r : \langle [p, \neg a], \gamma\rangle \xhookrightarrow{[-]} \langle [p, \neg a], \gamma\rangle \in \Delta'$ and $r \in \delta$ (since it is not from Δ nor Δ_c). Then, $r \in \beta(\theta)$ and there is a loop in $(\langle [p, \psi], w\rangle, \beta(\theta))$. Hence, $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \psi], w\rangle, \beta(\theta))$.

Case $\psi = \psi_1 \vee \psi_2$: Since $(\langle p, w\rangle, \theta) \models_\nu \psi$, then $(\langle p, w\rangle, \theta) \models_\nu \psi_1$ or $(\langle p, w\rangle, \theta) \models_\nu \psi_2$. By applying the induction hypothesis, $\mathcal{BP}_\varphi$ has an accepting run from the configuration $(\langle [p, \psi_1], w\rangle, \beta(\theta))$ or $(\langle [p, \psi_2], w\rangle, \beta(\theta))$. Since $r_1 : \langle [p, \psi], \gamma\rangle \xhookrightarrow{[-]} \langle [p, \psi_1], \gamma\rangle \in \Delta'$ is s.t. $r_1 \in \delta$ and $r_2 : \langle [p, \psi], \gamma\rangle \xhookrightarrow{[-]} \langle [p, \psi_2], \gamma\rangle \in \Delta'$ is s.t. $r_2 \in \delta$. Then $r_1, r_2 \in \beta(\theta)$ and we can get that $(\langle [p, \psi], w\rangle, \beta(\theta)) \Rightarrow_{\mathcal{BP}} (\langle [p, \psi_1], w\rangle, \beta(\theta))$ and $(\langle [p, \psi], w\rangle, \beta(\theta)) \Rightarrow_{\mathcal{BP}} (\langle [p, \psi_2], w\rangle, \beta(\theta))$. So $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \psi], w\rangle, \beta(\theta))$.

Case $\psi = \psi_1 \wedge \psi_2$: it is similar to case $\psi = \psi_1 \vee \psi_2$.

Case $\psi = EX\psi_1$: Since $(\langle p, w\rangle, \theta) \models_\nu \psi$, then there exists an immediate successor $(\langle p_1, w_1\rangle, \theta_1)$ of $(\langle p, w\rangle, \theta)$ s.t. $(\langle p_1, w_1\rangle, \theta_1) \models_\nu \psi_1$. By applying the induction hypothesis, $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p_1, \psi_1], w_1\rangle, \beta(\theta_1))$. There are two cases depending on whether $p \in P_N$ or not.

- Case $p \in P_N$, then $\theta_1 = \theta$. Since $(\langle p_1, w_1\rangle, \theta)$ is an immediate successor of $(\langle p, w\rangle, \theta)$, there exist $w'' \in \Gamma^*$ s.t. $w = \gamma w'', w_1 = w'w''$ and $R = \langle p, \gamma\rangle \hookrightarrow \langle p_1, w'\rangle \in \Delta \cap \theta$. By the construction Item 5, we can obtain $R' = \langle [p, \psi], \gamma\rangle \xhookrightarrow{[-]} \langle [p_1, \psi_1], w'\rangle \in \Delta'$ and $R' \in prod(R)$. Since $prod(\theta) = \{r' \in \Delta' \cup \Delta'_c \mid \exists r \in \theta, r' \in prod(r)\}$ and $R \in \theta$, then $R' \in prod(\theta)$. Thus, $R' \in \beta(\theta)$ and $(\langle [p, \psi], w\rangle, \beta(\theta)) \Rightarrow_{\mathcal{BP}} (\langle [p_1, \psi_1], w_1\rangle, \beta(\theta))$. Hence, $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \psi], w\rangle, \beta(\theta))$.

- Case $p \in P_C$, then $\theta_1 = \theta \backslash \{r_1\} \cup \{r_2\}$ for a transition rule $R = p \xhookrightarrow{(r_1,r_2)} p_1 \in \Delta_c \cap \theta$. There exist $\gamma \in \Gamma$ and $w'' \in \Gamma^*$ s.t. $w_1 = w = \gamma w''$. Thus, we can obtain $R' = [p,\psi] \xhookrightarrow{(\sigma,\sigma')} [p',\psi_1] \in \Delta'_c$, $\sigma = prod(r_1), \sigma' = prod(r_2)$, $R' \in prod_E(R)$ and $R' \in prod(R)$. Since $prod(\theta) = \{r' \in \Delta' \cup \Delta'_c \mid \exists r \in \theta, r' \in prod(r)\}$ and $R, r_1 \in \theta$, then $R' \in \beta(\theta)$ and $\sigma \cap prod(\theta) \neq \emptyset$. Based on rule R, $\theta_1 = \theta \backslash \{r_1\} \cup \{r_2\}$, then $\beta(\theta_1) = prod(\theta_1) \cup \delta = (prod(\theta \backslash \{r_1\} \cup \{r_2\}) \cup \delta$. Since $\sigma = prod(r_1)$ and $\sigma' = prod(r_2)$, then we can obtain $\beta(\theta_1) = \beta(\theta) \backslash \sigma \cup \sigma'$. Thus, we can have that $(\langle [p,\psi], w\rangle, \beta(\theta)) \Rightarrow_{\mathcal{BP}} (\langle [p_1,\psi_1], w\rangle, \beta(\theta_1))$. Hence, $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p,\psi], w\rangle, \beta(\theta))$.

Case $\psi = AX\psi_1$: there are 2 cases depending on whether $p \in P_N$ or not.

- Case $p \in P_N$. Let then $\gamma \in \Gamma$ and $u \in \Gamma^*$ be such that $w = \gamma u$. Let $S = \{R_1 = \langle p,\gamma\rangle \hookrightarrow \langle p_1, u_1\rangle, \cdots, R_n = \langle p,\gamma\rangle \hookrightarrow \langle p_n, u_n\rangle\}$ be the set of all the rules of Δ that have $\langle p,\gamma\rangle$ on the left hand-side. Then, by Item 6(a), we obtain that $R' = \langle [p,\psi],\gamma\rangle \xhookrightarrow{[\sigma_1,\cdots,\sigma_n]} \{\langle [p_1,\psi_1], w_1\rangle, \cdots, \langle [p_n,\psi_1], w_n\rangle\} \in \Delta'$, for every $1 \leq i \leq n$, $\sigma_i = prod_E(R_i)$. Let $\{R_{i_1} = \langle p,\gamma\rangle \hookrightarrow \langle p_{i_1}, u_{i_1}\rangle, \cdots, R_{i_k} = \langle p,\gamma\rangle \hookrightarrow \langle p_{i_k}, u_{i_k}\rangle\}$ be the set of rules of $S \cap \theta$. Let $w_{i_j} = u_{i_j} u$, $1 \leq j \leq k$, then $\{(\langle p_{i_1}, w_{i_1}\rangle, \theta), \cdots, (\langle p_{i_k}, w_{i_k}\rangle, \theta)\}$ is an immediate successor of $(\langle p, w\rangle, \theta)$. Since $(\langle p, w\rangle, \theta) \models_\nu \psi$, then $(\langle p_{i_j}, w_{i_j}\rangle, \theta) \models_\nu \psi_1$ for every $j, 1 \leq j \leq k$. By applying the induction hypothesis, $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p_{i_j},\psi_1], w_{i_j}\rangle, \beta(\theta))$. Since $prod(\theta) = \{r' \in \Delta' \cup \Delta'_c \mid \exists r \in \theta, r' \in prod(r)\}$ and $R_i \in \theta$, then $R' \in \beta(\theta)$. Since $\sigma_i = prod_E(R_i)$, $\sigma_l \cap \beta(\theta) \neq \emptyset$ if $l \in \{i_1, \cdots, i_k\}$ and $\sigma_l \cap \beta(\theta) = \emptyset$ if $l \notin \{i_1, \cdots, i_k\}$, then using R', we get that $(\langle [p,\psi], w\rangle, \beta(\theta)) \Rightarrow_{\mathcal{BP}} \{(\langle [p_{i_1},\psi_1], w_{i_1}\rangle, \beta(\theta)), \cdots, (\langle [p_{i_k},\psi_1], w_{i_k}\rangle, \beta(\theta))\}$. Since $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p_{i_j},\psi_1], w_{i_j}\rangle, \beta(\theta))$, then, $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p,\psi], w\rangle, \beta(\theta))$.

- Case $p \in P_C$. Let then $\gamma \in \Gamma$ and $u \in \Gamma^*$ be such that $w = \gamma u$. Let $S = \{R_1 = p \xhookrightarrow{(r_1,r'_1)} p_1, \cdots, R_n = p \xhookrightarrow{(r_n,r'_n)} p_n\}$ be the set of all the rules of Δ_c that have p on the left hand-side. Then, by Item 6(b), we obtain that $R'_\perp = \langle [p,\psi],\gamma\rangle \xhookrightarrow{[\sigma_1,\cdots,\sigma_n]} \{\langle p_1^\psi, \gamma\rangle, \cdots, \langle p_n^\psi, \gamma\rangle\} \in \Delta'$, for every $1 \leq i \leq n$, $\sigma_i = prod_E(R_i)$ and for every $1 \leq i \leq n$, $R'_i : p_i^\psi \xhookrightarrow{(\sigma,\sigma')} [p_i,\psi_1] \in \Delta'_c$ where for every $1 \leq i \leq n, \sigma_i = prod_E(R_i)$, $\sigma = prod(r_i), \sigma' = prod(r'_i)$ and $R'_i \in$

$prod(R_i)$. Let $\{R_{i_1} = p \xhookrightarrow{(r_{i_1}, r'_{i_1})} p_{i_1}, \cdots, R_{i_k} = p \xhookrightarrow{(r_{i_k}, r'_{i_k})} p_{i_k}\rangle\}$ be the set of rules of $S \cap \theta$. Then $\{(\langle p_{i_1}, \gamma u\rangle, \theta_{i_1}), \cdots, (\langle p_{i_k}, \gamma u\rangle, \theta_{i_k})\}$ is an immediate successor of $(\langle p, w\rangle, \theta)$ where $\theta_{i_j} = (\theta \setminus \{r_{i_j}\}) \cup \{r'_{i_j}\}$. Since $(\langle p, w\rangle, \theta) \models_\nu \psi$, then $(\langle p_{i_j}, \gamma u\rangle, \theta_{i_j}) \models_\nu \psi_1$ for every $j, 1 \leq j \leq k$. By applying the induction hypothesis, $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p_{i_j}, \psi_1], \gamma u\rangle, \beta(\theta_{i_j}))$. Since $prod(\theta) = \{r' \in \Delta' \cup \Delta'_c \mid \exists r \in \theta, r' \in prod(r)\}$ and $R_i \in \theta$, then $R'_i, R'_\perp \in \beta(\theta)$. Since $\sigma_l = prod_E(R_l)$, $\sigma_l \cap \beta(\theta) \neq \emptyset$ if $l \in \{i_1, \cdots, i_k\}$ and $\sigma_l \cap \beta(\theta) = \emptyset$ if $l \notin \{i_1, \cdots, i_k\}$, then using $R'_\perp$, we get that $(\langle [p, \psi], w\rangle, \beta(\theta)) \Rightarrow_{\mathcal{BP}} \{(\langle p^\psi_{i_1}, w\rangle, \beta(\theta)), \cdots, (\langle p^\psi_{i_k}, w\rangle, \beta(\theta))\}$. For every $j, 1 \leq j \leq k$, using R'_{i_j}, we get that $(\langle p^\psi_{i_1}, w\rangle, \beta(\theta)) \Rightarrow_{\mathcal{BP}} (\langle [p_{i_j}, \psi_1], \gamma u\rangle, \beta(\theta) \setminus \sigma \cup \sigma')$. Since $\beta(\theta_{i_j}) = prod(\theta_{i_j}) \cup \delta = (prod(\theta \setminus \{r_{i_j}\} \cup \{r'_{i_j}\}) \cup \delta$, then $\beta(\theta_{i_j}) = \beta(\theta) \setminus \sigma \cup \sigma'$ for every $\sigma = prod(r_{i_j})$ and $\sigma' = prod(r'_{i_j})$. Thus, we can obtain that

$$(\langle [p, \psi], w\rangle, \beta(\theta)) \Rightarrow^*_{\mathcal{BP}} \{(\langle [p_{i_1}, \psi_{i_1}], w\rangle, \beta(\theta_{i_1})), \cdots, (\langle [p_{i_k}, \psi_1], w\rangle, \beta(\theta_{i_k}))\}.$$

Since $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p_{i_j}, \psi_1], w_{i_j}\rangle, \beta(\theta))$, then, $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \psi], w\rangle, \beta(\theta))$.

Case $\psi = E[\psi_1 U \psi_2]$: Since $(\langle p, w\rangle, \theta) \models_\nu \psi$, then there exists a path

$$\rho : (\langle p_0, w_0\rangle, \theta_0), (\langle p_1, w_1\rangle, \theta_1), (\langle p_2, w_2\rangle, \theta_2) \cdots$$

from $(\langle p, w\rangle, \theta)$ (i.e. $(\langle p_0, w_0\rangle, \theta_0) = (\langle p, w\rangle, \theta)$) such that there exists $i \geq 0$, $(\langle p_i, w_i\rangle, \theta_i) \models_\nu \psi_2$ and for every $0 \leq j < i, (\langle p_j, w_j\rangle, \theta_j) \models_\nu \psi_1$. Thus, by applying the induction hypothesis, we obtain that $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p_i, \psi_2], w_i\rangle, \beta(\theta_i))$ and for every $0 \leq j < i$, $\mathcal{BP}_\varphi$ has an accepting run from the configuration $(\langle [p_j, \psi_1], w_j\rangle, \beta(\theta_j))$. We first show that $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p_i, \psi], w_i\rangle, \beta(\theta_i))$.

By the construction Item 7, $r' = \langle [p_i, \psi], \gamma\rangle \xhookrightarrow{[-]} \langle [p_i, \psi_2], \gamma\rangle \in \Delta'$ s.t. $r' \in \delta$ (since it is not constructed from Δ nor Δ_c). Then, $r' \in \beta(\theta_i)$. Thus, we have that $(\langle [p_i, \psi], w_i\rangle, \beta(\theta_i)) \Rightarrow_{\mathcal{BP}} (\langle [p_i, \psi_2], w_i\rangle, \beta(\theta_i))$. Hence, $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p_i, \psi], w_i\rangle, \beta(\theta_i))$. If $i = 0$, then $(\langle [p, \psi], w\rangle, \beta(\theta)) = (\langle [p_i, \psi], w_i\rangle, \beta(\theta_i))$ and $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \psi], w\rangle, \beta(\theta))$. Otherwise if $i > 0$, we show that $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p_j, \psi], w_j\rangle, \beta(\theta_j))$, $1 \leq j < i$, by induction on $l = i - j$. (Note that $(\langle [p, \psi], w\rangle, \beta(\theta)) = (\langle [p_0, \psi], w_0\rangle, \beta(\theta_0))$).

- **Basis.** $l = 1$. Then $(\langle p_i, w_i\rangle, \theta_i)$ is an immediate successor of $(\langle p_j, w_j\rangle, \theta_j)$.

  If $p_j \in P_N$, then there exists $R = \langle p_j, \gamma\rangle \hookrightarrow \langle p_i, w'\rangle \in \Delta \cap \theta_j$. By the construction Item 7(a), $R' : \langle [p_j, \psi], \gamma\rangle \xhookrightarrow{[-,-]} \{\langle [p_j, \psi_1], \gamma\rangle, \langle [p_i, \psi], w'\rangle\} \in \Delta'$ and $R' \in prod(R)$. Since $prod(\theta) = \{r' \in \Delta' \cup \Delta'_c \mid \exists r \in \theta, r' \in prod(r)\}$ and $R \in \theta$, then $R' \in prod(\theta)$. Thus, $R' \in \beta(\theta)$ and we can have $(\langle [p_j, \psi], w_j\rangle, \beta(\theta_j)) \Rightarrow_{\mathcal{BP}} \{(\langle [p_j, \psi_1], w_j\rangle, \beta(\theta_j)), (\langle [p_i, \psi], w_i\rangle, \beta(\theta_j))\}$. Hence, $\mathcal{BP}_\varphi$ has an accepting run from the configuration $(\langle [p_j, \psi], w_j\rangle, \beta(\theta_j))$.

  Otherwise if $p_j \in P_C$, for every $R = p_j \xhookrightarrow{(r_1,r_2)} p_i \in \Delta_c \cap \theta_j$, for every $\gamma \in \Gamma$, $R'_\perp = \langle [p_j, \psi], \gamma\rangle \xhookrightarrow{[-,-]} \{\langle [p_j, \psi_1], \gamma\rangle, \langle {p_j}^\psi, \gamma\rangle\} \in \Delta'$ and $R' : {p_j}^\psi \xhookrightarrow{(\sigma,\sigma')} [p_i, \psi] \in \Delta'_c$ where $\sigma = prod(r_1), \sigma' = prod(r_2)$ s.t. $R'_\perp, R' \in prod(R)$. Then, we can obtain that:

  $$(\langle [p_j, \psi], w_j\rangle, \beta(\theta_j)) \Rightarrow_{\mathcal{BP}} \{(\langle [p_j, \psi_1], w_j\rangle, \beta(\theta_j)), (\langle {p_j}^\psi, w_j\rangle, \beta(\theta_j))\}.$$

  Since $R, r_1 \in \theta_j$ and $\theta_i = \theta_j \backslash \{r_1\} \cup \{r_2\}$, then $\beta(\theta_i) = prod(\theta_i) \cup \delta = prod(\theta_j \backslash \{r_1\} \cup \{r_2\}) \cup \delta$. Thus, $\beta(\theta_i) = \beta(\theta_j) \backslash \sigma \cup \sigma'$ and $(\langle {p_j}^\psi, w_j\rangle, \beta(\theta_j)) \Rightarrow_{\mathcal{BP}} \{(\langle [p_i, \psi], w)\rangle, \beta(\theta_i)\}$. Hence, $\mathcal{BP}_\varphi$ has an accepting run from the configuration $(\langle [p_j, \psi], w_j\rangle, \beta(\theta_j))$.

- **Step.** $l > 1$. Then there exists $(\langle p_{j+1}, w_{j+1}\rangle, \theta_{j+1})$ s.t. $(\langle p_j, w_j\rangle, \theta_j) \Rightarrow_{\mathcal{P}} (\langle p_{j+1}, w_{j+1}\rangle, \theta_{j+1}) \Rightarrow^*_{\mathcal{P}} (\langle p_i, w_i\rangle, \theta_i)$. By applying the induction hypothesis (induction on l), we can obtain that $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p_{j+1}, \psi], w_{j+1}\rangle, \beta(\theta_{j+1}))$. Since $(\langle p_j, w_j\rangle, \theta_j) \models_\nu \psi_1$, by applying the induction hypothesis (induction on the structure of ψ), $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p_j, \psi_1], w_j\rangle, \beta(\theta_j))$. There are two cases depending on whether $p_j \in P_N$ or not.

  - Case $p_j \in P_N$, then there exists $R = \langle p_j, \gamma\rangle \hookrightarrow \langle p_{j+1}, w'\rangle \in \Delta \cap \theta_j$. By the construction, $R' = \langle [p_j, \psi], \gamma\rangle \xhookrightarrow{[-,-]} \{\langle [p_j, \psi_1], \gamma\rangle, \langle [p_{j+1}, \psi], w'\rangle\} \in \Delta'$ s.t. $R' \in prod(R)$. Since $prod(\theta_j) = \{r' \in \Delta' \cup \Delta'_c \mid \exists r \in \theta_j, r' \in prod(r)\}$ and $R \in \theta$, then $R' \in prod(\theta_j)$. Thus, $R' \in \beta(\theta_j)$ and we can have
    $(\langle [p_j, \psi], w_j\rangle, \beta(\theta_j)) \Rightarrow_{\mathcal{BP}} \{(\langle [p_j, \psi_1], w_j\rangle, \beta(\theta_j)), (\langle [p_{j+1}, \psi], w_{j+1}\rangle, \beta(\theta_j))\}$.
    So, $\mathcal{BP}_\varphi$ has an accepting run from the configuration $(\langle [p_j, \psi], w_j\rangle, \beta(\theta_j))$.

- Case $p_j \in P_C$, for every $R = p_j \xhookrightarrow{(r_1,r_2)} p_{j+1} \in \Delta_c \cap \theta_j$, there exist $\gamma \in \Gamma$ and $w''_j \in \Gamma^*$ s.t. $w_j = \gamma w''_j$, then $R'_\perp = \langle [p_j, \psi], \gamma\rangle \xhookrightarrow{[-,-]} \{\langle [p_j, \psi_1], \gamma\rangle, \langle {p_j}^{\psi}, \gamma\rangle\} \in \Delta'$ and $R' : {p_j}^{\psi} \xhookrightarrow{(\sigma,\sigma')} [p_i, \psi] \in \Delta'_c$ where $\sigma = prod(r_1), \sigma' = prod(r_2)$ s.t. $R' \in prod(R)$.
  Then, $(\langle [p_j, \psi], w_j\rangle, \beta(\theta_j)) \Rightarrow_{\mathcal{BP}} \{(\langle [p_j, \psi_1], w_j\rangle, \beta(\theta_j)), (\langle {p_j}^{\psi}, w_j\rangle, \beta(\theta_j))\}$. Since $prod(\theta_j) = \{r' \in \Delta' \cup \Delta'_c \mid \exists r \in \theta_j, r' \in prod(r)\}$ and $R \in \theta_j$, then $R' \in prod(\theta_j)$. Thus, $R' \in \beta(\theta_j)$. Since $R, r_1 \in \theta_j$ and $\theta_{j+1} = \theta_j \backslash \{r_1\} \cup \{r_2\}$. Then $\beta(\theta_{j+1}) = prod(\theta_{j+1}) \cup \delta = prod(\theta_j \backslash \{r_1\} \cup \{r_2\}) \cup \delta$. So, $\beta(\theta_{j+1}) = \beta(\theta_j) \backslash \sigma \cup \sigma'$. Then, we have $(\langle {p_j}^{\psi}, w_j\rangle, \beta(\theta_j)) \Rightarrow_{\mathcal{BP}} \{(\langle [p_{j+1}, \psi], w_j\rangle, \beta(\theta_{j+1}))\}$. Hence, $\mathcal{BP}_\varphi$ has an accepting run from the configuration $(\langle [p_j, \psi], w_j\rangle, \beta(\theta_j))$.

Thus, $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \psi], w\rangle, \beta(\theta))$.

Case $\psi = A[\psi_1 U \psi_2]$ is similar the the case $\psi = E[\psi_1 U \psi_2]$.

Case $\psi = E[\psi_1 \widetilde{U} \psi_2]$: Since $(\langle p, w\rangle, \theta) \models_\nu E[\psi_1 \widetilde{U} \psi_2]$, then there exists a path

$$(\langle p_0, w_0\rangle, \theta_0), (\langle p_1, w_1\rangle, \theta_1), (\langle p_2, w_2\rangle, \theta_2) \cdots$$

from $(\langle p, w\rangle, \theta)$ (i.e. $(\langle p_0, w_0\rangle, \theta_0) = (\langle p, w\rangle, \theta)$) such that

1. either for every $i \geq 0, (\langle p_i, w_i\rangle, \theta_i) \models_\nu \psi_2$
2. or there exists $i \geq 0$ s.t. $(\langle p_i, w_i\rangle, \theta_i) \models_\nu \psi_1$ and for every $0 \leq j \leq i$, $(\langle p_j, w_j\rangle, \theta_j) \models_\nu \psi_2$

First let us consider item 2, it can be proved that $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \psi], w\rangle, \beta(\theta))$ by applying the induction on $i - j$ similar to the case $\psi = E[\psi_1 U \psi_2]$.

Let's consider item 1, we will show that $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \psi], w\rangle, \beta(\theta))$. Let us construct an accepting run ρ of $\mathcal{BP}_\varphi$ as follows. (Note that $(\langle [p, \psi], w\rangle, \beta(\theta)) = (\langle [p_0, \psi], w_0\rangle, \beta(\theta_0)))$.

Let $(\langle [p_0, \psi], w_0\rangle, \theta_0)$ be the root of ρ. For every $k \geq 0$,

- if $p_k \in P_N$, then we have that

  $$(\langle [p_k, \psi], w_k\rangle, \beta(\theta_k)) \Rightarrow_{\mathcal{BP}} \{(\langle [p_k, \psi_2], w_k\rangle, \beta(\theta_k)), (\langle [p_{k+1}, \psi], w_{k+1}\rangle, \beta(\theta_{k+1}))\}.$$

In this case, we let $(\langle [p_k, \psi_2], w_k\rangle, \beta(\theta_k))$ and $(\langle [p_{k+1}, \psi], w_{k+1}\rangle, \beta(\theta_{k+1}))$ be the children of $(\langle [p_k, \psi], w_k\rangle, \beta(\theta_k))$. By applying the induction hypothesis to $(\langle p_k, w_k\rangle, \theta_k) \models_\nu \psi_2$, we obtain that $\mathcal{BP}_\varphi$ has an accepting run ρ_k from $(\langle [p_k, \psi_2], w_k\rangle, \beta(\theta_k))$.

We replace the child $(\langle [p_k, \psi_2], w_k\rangle, \beta(\theta_k))$ in ρ by the run ρ_k. By the above construction, we obtain an infinite run ρ of $\mathcal{BP}_\varphi$ s.t. ρ has an infinite path

$$(\langle [p_0, \psi], w_0\rangle, \beta(\theta_0)), (\langle [p_1, \psi], w_1\rangle, \beta(\theta_1)), \cdots$$

and all the other paths infinitely often visit some accepting control locations. Since for every $k \geq 0, [p_k, \psi] \in F$, we obtain that each path of ρ infinitely often visits some accepting control locations i.e. $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \psi], w\rangle, \beta(\theta))$.

- If $p_k \in P_C$, $(\langle [p_k, \psi], w_k\rangle, \beta(\theta_k)) \Rightarrow_{\mathcal{BP}} \{(\langle [p_k, \psi_2], w_k\rangle, \beta(\theta_k)), (\langle p_k^\psi, w_k\rangle, \beta(\theta_k))\}$ and $(\langle p_k^\psi, w_k\rangle, \beta(\theta_k)) \Rightarrow_{\mathcal{BP}} \{(\langle [p_{k+1}, \psi], w_{k+1}\rangle, \beta(\theta_{k+1}))\}$. In this case, we let $(\langle [p_k, \psi_2], w_k\rangle, \beta(\theta_k))$ and $(\langle p_k^\psi, w_k\rangle, \beta(\theta_k))$ be the children of $(\langle [p_k, \psi], w_k\rangle, \beta(\theta_k))$ and $(\langle [p_{k+1}, \psi], w_{k+1}\rangle, \beta(\theta_{k+1}))$ be the child of $(\langle p_k^\psi, w_k\rangle, \beta(\theta_k))$. By applying the induction hypothesis to $(\langle p_k, w_k\rangle, \theta_k) \models_\nu \psi_2$, we obtain that $\mathcal{BP}_\varphi$ has an accepting run ρ_k from $(\langle [p_k, \psi_2], w_k\rangle, \beta(\theta_k))$. We replace the child $(\langle [p_k, \psi_2], w_k\rangle, \beta(\theta_k))$ in ρ by the run ρ_k. By the above construction, we obtain an infinite run ρ of $\mathcal{BP}_\varphi$ s.t. ρ has an infinite path

$$(\langle [p_0, \psi], w_0\rangle, \beta(\theta_0)) \cdots, (\langle p_k^\psi, w_k\rangle, \beta(\theta_k)), (\langle [p_{k+1}, \psi], w_{k+1}\rangle, \beta(\theta_{k+1})) \cdots$$

  and all the other paths infinitely often visit some accepting control locations. Since for every $k \geq 0, [p_k, \psi] \in F$, we obtain that each path of ρ infinitely often visits some accepting control locations i.e. $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \psi], w\rangle, \beta(\theta))$.

Case $\psi = A[\psi_1 \widetilde{U} \psi_2]$: it can be proved as for the case $\psi = E[\psi_1 \widetilde{U} \psi_2]$.

$(\Leftarrow)$: Suppose $\mathcal{BP}_\varphi$ has an accepting run from the configuration $(\langle [p, \psi], w\rangle, \beta(\theta))$, we show that $(\langle p, w\rangle, \theta) \models_\nu \psi$ by induction on the structure of ψ.

Case $\psi = a$: Since $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \psi], w\rangle, \beta(\theta))$, then $\forall \gamma \in \Gamma$, $\langle [p, a], \gamma\rangle \xhookrightarrow{[-]} \langle [p, a], \gamma\rangle \in \Delta' \cap \beta(\theta)$. So, $[p, a] \in F$ and $a \in \nu(p)$. Hence, $(\langle p, w\rangle, \theta) \models_\nu \psi$.

Case $\psi = \neg a$: Since $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \neg a], w\rangle, \beta(\theta))$, then $\forall \gamma \in \Gamma$, $\langle [p, \neg a], \gamma\rangle \xrightarrow{[-]} \langle [p, \neg a], \gamma\rangle \in \Delta' \cap \beta(\theta)$. So, $[p, \neg a] \in F$ and $a \notin \nu(p)$. Hence, $(\langle p, w\rangle, \theta) \models_\nu \psi$.

Case $\psi = \psi_1 \vee \psi_2$: Since $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \psi], w\rangle, \beta(\theta))$, then $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \psi_1], w\rangle, \beta(\theta))$ or $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \psi_2], w\rangle, \beta(\theta))$. By applying the induction hypothesis, we get that $(\langle p, w\rangle, \theta) \models_\nu \psi_1$ or $(\langle p, w\rangle, \theta) \models_\nu \psi_2$. This implies that $(\langle p, w\rangle, \theta) \models_\nu \psi$.

Case $\psi = \psi_1 \wedge \psi_2$: it is similar to the case $\psi = \psi_1 \vee \psi_2$.

Case $\psi = EX\psi_1$: it is similar to the case $\psi = AX\psi_1$.

Case $\psi = AX\psi_1$: Since $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p, \psi], w\rangle, \beta(\theta))$. There are two cases depending on whether $p \in P_N$ or not.

- Case $p \in P_N$. Then suppose there exists an immediate successor

  $$\{(\langle [p_1, \psi_1], w_1\rangle, \beta(\theta)), \cdots, (\langle [p_n, \psi_1], w_n\rangle, \beta(\theta))\}$$

  of $(\langle [p, \psi], w\rangle, \beta(\theta))$ in the accepting run such that

  $$(\langle [p, \psi], w\rangle, \beta(\theta)) \Rightarrow_{\mathcal{BP}} \{(\langle [p_1, \psi_1], w_1\rangle, \beta(\theta)), \cdots, (\langle [p_n, \psi_1], w_n\rangle, \beta(\theta))\}$$

  By applying the induction hypothesis, we get that $(\langle p_i, w_i\rangle, \theta_i) \models_\nu \psi_1$ for $1 \leq i \leq n$. By the construction, the immediate successors in $\mathcal{P}$ of $(\langle p, w\rangle, \theta)$ are $(\langle p_1, w_1\rangle, \theta), \cdots, (\langle p_n, w_n\rangle, \theta)$. Thus, we obtain that $(\langle p, w\rangle, \theta) \models_\nu \psi$.

- Case $p \in P_C$. Then suppose there exists a successor

  $$\{(\langle [p_1, \psi_1], w_1\rangle, \beta(\theta)), \cdots, (\langle [p_n, \psi_1], w_n\rangle, \beta(\theta))\}$$

  of $(\langle [p, \psi], w\rangle, \beta(\theta))$ in the accepting run such that

  $$\begin{aligned}(\langle [p, \psi], w\rangle, \beta(\theta)) &\Rightarrow_{\mathcal{BP}} \{(\langle p_1^\psi, w\rangle, \beta(\theta)), \cdots, (\langle p_n^\psi, w\rangle, \beta(\theta))\} \\ &\Rightarrow_{\mathcal{BP}} \{(\langle [p_1, \psi_1], w_1\rangle, \beta(\theta_1)), \cdots, (\langle [p_n, \psi_1], w_n\rangle, \beta(\theta_n))\}.\end{aligned}$$

  Then $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p_i, \psi_1], w_i\rangle, \beta(\theta_i))$ for each $i : 1 \leq i \leq n$. By applying the induction hypothesis, we get that $(\langle p_i, w_i\rangle, \theta_i) \models_\nu \psi_1$ for $1 \leq i \leq n$. By the construction of Item 6(b), there exists $R'_\perp =$

$\langle [p,\psi],\gamma\rangle \xrightarrow{[\sigma_{j_1},\cdots,\sigma_{j_m}]} \{\langle p_1^{\psi},\gamma\rangle,\cdots,\langle p_n^{\psi},\gamma\rangle\}$ s.t. $\{1,\cdots,n\} \subseteq \{j_1,\cdots,j_m\}$ where the constraint $[\sigma_{j_1},\cdots,\sigma_{j_m}]$ ensures that only the R_i's that are in θ ($R_1,\cdots,R_n$) are applied, and there exist $R'_i : p_i^{\psi} \xrightarrow{(\sigma,\sigma')} [p_i,\psi_1]$ where $\sigma = prod(r_i)$ and $\sigma' = prod(r'_i)$ ensuring that $\theta_i = (\theta \setminus \{r_i\}) \cup \{r'_i\}$. Thus, we can obtain that the set of all transition rules of the SM-PDS in $\Delta_c \cap \theta$ that have p in the left-hand side are $\{R_1 = p \xrightarrow{(r_1,r'_1)} p_1,\cdots,R_n = p \xrightarrow{(r_n,r'_n)} p_n\}$. Then, the immediate successors of $(\langle p,w\rangle,\theta)$ in $\mathcal{P}$ are $(\langle p_1,w\rangle,\theta_1),\cdots,(\langle p_n,w\rangle,\theta_n)$. Thus we obtain that $(\langle p,w\rangle,\theta) \models_\nu \psi$.

Case $\psi = E[\psi_1 U \psi_2]$: Let ρ be the accepting run from $(\langle [p,\psi],w\rangle,\beta(\theta))$. By the construction, every configuration $(\langle [p_i,\psi],w_i\rangle,\beta(\theta_i))$ in ρ has

- either two children $(\langle [p_i,\psi_1],w_i\rangle,\beta(\theta_i))$ and $(\langle q_i,w_{q_i}\rangle,\beta(\theta_i))$, where
  1. If $p_i \in P_N$, $(\langle q_i,w_{q_i}\rangle,\beta(\theta_i))$ is of the form $(\langle [p_{i+1},\psi],w_{i+1}\rangle,\beta(\theta_i))$.
  2. If $p_i \in P_C$, $(\langle q_i,w_{q_i}\rangle,\beta(\theta_i))$ is of the form $(\langle p_i^{\psi},w_i\rangle,\beta(\theta_i))$ whose child is $(\langle [p_{i+1},\psi],w_{i+1}\rangle,\beta(\theta_{i+1}))$.
- or only one child $(\langle [p_i,\psi_2],w_i\rangle,\beta(\theta_i))$

Since ρ is an accepting run, there exists a configuration $(\langle [p_n,\psi],w_n\rangle,\beta(\theta_n))$ in ρ s.t. $(\langle [p_n,\psi],w_n\rangle,\beta(\theta_n))$ has only one child $(\langle [p_n,\psi_2],w_n\rangle,\beta(\theta_n))$. In particular, there exists a path of ρ of the form $(\langle q_0,w_{q_0}\rangle,\beta(\theta_0)),\cdots,(\langle q_n,w_{q_n}\rangle,\beta(\theta_n))\cdots$ where $(\langle q_0,w_{q_0}\rangle,\beta(\theta_0)) = (\langle [p,\psi],w\rangle,\beta(\theta))$ and $(\langle q_n,w_{q_n}\rangle,\beta(\theta_i)) = (\langle [p_n,\psi],w_n\rangle,\beta(\theta_n))$. Then, $\mathcal{BP}_\varphi$ has an accepting run from $(\langle q_i,w_{q_i}\rangle,\beta(\theta_i))$ of the form $(\langle [p_i,\psi],w_i\rangle,\beta(\theta_i))$ for every $i : 0 \le i < n$.

- If $p_i \in P_N$, $(\langle [p_i,\psi],w_i\rangle,\beta(\theta_i))$ in ρ has either two children $(\langle [p_i,\psi_1],w_i\rangle,\beta(\theta_i))$ and $(\langle [p_{i+1},\psi],w_{i+1}\rangle,\beta(\theta_i))$ or has only one child $(\langle [p_i,\psi_2],w_i\rangle,\beta(\theta_i))$. Since $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p_n,\psi],w_n\rangle,\beta(\theta_n))$ in ρ and $i < n$, then $(\langle [p_i,\psi],w_i\rangle,\beta(\theta_i)) \Rightarrow_{\mathcal{BP}} \{(\langle [p_i,\psi_1],w_i\rangle,\beta(\theta_i)),(\langle [p_{i+1},\psi],w_{i+1}\rangle,\beta(\theta_i))\}$. Thus, $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p_i,\psi_1],w_i\rangle,\beta(\theta_i))$.
- If $p_i \in P_C$, $(\langle [p_i,\psi],w_i\rangle,\beta(\theta_i))$ in ρ has either two children $(\langle [p_i,\psi_1],w_i\rangle,\beta(\theta_i))$ and $(\langle p_i^{\psi},w_i\rangle,\beta(\theta_i))$ whose child is $(\langle [p_{i+1},\psi],w_{i+1}\rangle,\beta(\theta_{i+1}))$ or has only one child $(\langle [p_i,\psi_2],w_i\rangle,\beta(\theta_i))$. Since $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p_n,\psi],w_n\rangle,\beta(\theta_n))$

in ρ and $i < n$, then $(\langle[p_i, \psi], w_i\rangle, \beta(\theta_i)) \Rightarrow_{\mathcal{BP}} \{(\langle[p_i, \psi_1], w_i\rangle, \beta(\theta_i)), (\langle p_i^{\psi}, w_i\rangle, \beta(\theta_i))\}$ and $(\langle p_i^{\psi}, w_i\rangle, \beta(\theta_i)) \Rightarrow_{\mathcal{BP}} \{(\langle[p_{i+1}, \psi], w_{i+1}\rangle, \beta(\theta_{i+1}))\}$. Thus, $\mathcal{BP}_{\varphi}$ has an accepting run from $(\langle[p_i, \psi_1], w_i\rangle, \beta(\theta_i))$.

$\mathcal{BP}_{\varphi}$ has an accepting run from $(\langle[p_n, \psi_2], w_n\rangle, \beta(\theta_n))$ and $(\langle[p_i, \psi_1], w_i\rangle, \beta(\theta_i))$ for $i < n$. By applying the induction hypothesis, we get that $(\langle p_n, w_n\rangle, \theta_n) \models_{\nu} \psi_2$ and $(\langle p_i, w_i\rangle, \theta_i) \models_{\nu} \psi_1$ for $i < n$. Since $(\langle p_1, w_1\rangle, \theta_1), \cdots, (\langle p_n, w_n\rangle, \theta_n) \cdots$ is a run of $\mathcal{P}$, we obtain that $(\langle p, w\rangle, \theta) \models_{\nu} \psi$.

Case $\psi = A[\psi_1 U \psi_2]$: it is similar with the case $\psi = E[\psi_1 U \psi_2]$.

Case $\psi = E[\psi_1 \widetilde{U} \psi_2]$: Let ρ be the accepting run from $(\langle[p, \psi], w\rangle, \beta(\theta))$. By the construction, every configuration $(\langle[p_i, \psi], w_i\rangle, \beta(\theta_i))$ in ρ has two children:

1. either $(\langle[p_i, \psi_2], w_i\rangle, \beta(\theta_i))$ and $(\langle q_i, w_{q_i}\rangle, \beta(\theta_i))$ where
   - if $p_i \in P_N$, then q_i is of the form $(\langle[p_{i+1}, \psi], w_{i+1}\rangle, \beta(\theta_{i+1}))$;
   - if $p_i \in P_C$, then $(\langle q_i, w_{q_i}\rangle, \beta(\theta_i))$ is of the form $(\langle p_i^{\psi}, w_i\rangle, \beta(\theta_i)$ whose child is $(\langle[p_{i+1}, \psi], w_{i+1}\rangle, \beta(\theta_{i+1}))$.
2. or $(\langle[p_i, \psi_1], w_i\rangle, \beta(\theta_i))$ and $(\langle[p_i, \psi_2], w_i\rangle, \beta(\theta_i))$.

First, we consider Case 1.

Since ρ is an accepting run, there exists an infinite path of ρ of the form

$$(\langle q_0, w_{q_0}\rangle, \beta(\theta_0)), \cdots, (\langle q_i, w_{q_i}\rangle, \beta(\theta_i)) \cdots$$

where $(\langle q_0, w_{q_0}\rangle, \beta(\theta_0)) = (\langle[p, \psi], w\rangle, \beta(\theta))$ and $\mathcal{BP}_{\varphi}$ has an accepting run from $(\langle[p_i, \psi_2], w_i\rangle, \beta(\theta_i))$ for every $i \geq 0$.

By applying the induction hypothesis (the fact that $\mathcal{BP}_{\varphi}$ has an accepting run from $(\langle[p_i, \psi_2], w_i\rangle, \beta(\theta_i)))$, we get that $(\langle p_i, w_i\rangle, \theta_i) \models_{\nu} \psi_2$ for every $i \geq 0$. By the construction, we get that $(\langle p_0, w_0\rangle, \theta_0), \cdots, (\langle p_n, w_n\rangle, \theta_n), \cdots$ is a run of $\mathcal{P}$ with $(\langle p_0, w_0\rangle, \theta_0) = (\langle p, w\rangle, \theta)$ and $(\langle p, w\rangle, \theta) \models_{\nu} \psi$.

Let's consider Case 2. Let $(\langle[p_n, \psi], w_n\rangle, \beta(\theta_n))$ be a configuration in ρ whose children are $(\langle[p_n, \psi_1], w_n\rangle, \beta(\theta_n))$ and $(\langle[p_n, \psi_2], w_n\rangle, \beta(\theta_n))$. Then $\mathcal{BP}_{\varphi}$ has an infinite path $(\langle[p_0, \psi], w_0\rangle, \beta(\theta_0)), \cdots, (\langle[p_n, \psi], w_n\rangle, \beta(\theta_n)), (\langle[p_n, \psi_1], w_n\rangle, \beta(\theta_n)), \cdots$, where $(\langle[p_0, \psi], w_0\rangle, \beta(\theta_0)) = (\langle[p, \psi], w\rangle, \beta(\theta))$. Each configuration $(\langle[p_i, \psi], w_i\rangle, \beta(\theta_i))$

in this path has children $(\langle [p_i, \psi_2], w_i\rangle, \beta(\theta_i))$ and $(\langle [p_{i+1}, \psi], w_{i+1}\rangle, \beta(\theta_{i+1}))$ or $(\langle p_i^{\psi}, w_i\rangle, \beta(\theta_i))$ whose child is $(\langle [p_{i+1}, \psi], w_{i+1}\rangle, \beta(\theta_{i+1}))$. So $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p_n, \psi_1], w_n\rangle, \beta(\theta_n))$ and $\mathcal{BP}_\varphi$ has an accepting run from $(\langle [p_i, \psi_2], w_i\rangle, \beta(\theta_i))$ for every $1 \leq i \leq n$. By applying the induction hypothesis, $(\langle p_n, w_n\rangle, \theta_n) \models_\nu \psi_1$ and $(\langle p_i, w_i\rangle, \theta_i) \models_\nu \psi_2$ for each $i : 1 \leq i \leq n$. Thus,

$$(\langle p_0, w_0\rangle, \theta_0), (\langle p_1, w_1\rangle, \theta_1), \cdots, (\langle p_n, w_n\rangle, \theta_n) \cdots$$

is a run of $\mathcal{P}$ with $(\langle p_0, w_0\rangle, \theta_0) = (\langle p, w\rangle, \theta)$. Thus, $(\langle p, w\rangle, \theta) \models_\nu \psi$.

Case $\psi = A[\psi_1 \widetilde{U} \psi_2]$: this case is similar to the case $\psi = E[\psi_1 \widetilde{U} \psi_2]$.

□

Therefore, CTL model-checking for SM-PDSs can be reduced to the problem of determining whether a SM-ABPDS has an accepting run.

4.2 Computing the language of a SM-ABPDS

From now on, we fix a SM-ABPDS $\mathcal{BP} = (P, \Delta, \Delta_c, \Gamma, F)$. We show in this section that the set of configurations accepted by $\mathcal{BP}$ is regular and can be effectively represented by an EAMA (extended Alternating Multi-automaton). To this aim, we first characterize the set of configurations $L(\mathcal{BP})$ from which $\mathcal{BP}$ has an accepting run. Then we use this characterization to compute an EAMA that accepts it.

4.2.1 Characterizing $L(\mathcal{BP})$

Let $(X_i)_{i \geq 0}$ be the following sequence: $X_0 = P \times \Gamma^* \times 2^{\Delta \cup \Delta_c}$, and for every $i \geq 0$, $X_{i+1} = Pre^+_{\mathcal{BP}}\big(X_i \cap (F \times \Gamma^* \times 2^{\Delta \cup \Delta_c})\big)$. Let $Y_{\mathcal{BP}} = \bigcap_{i \geq 0} X_i$. We can show that $L(\mathcal{BP}) = Y_{\mathcal{BP}}$:

Theorem 4.2.1 *A SM-ABPDS $\mathcal{BP}$ has an accepting run starting from a configuration $(\langle p, w\rangle, \theta)$ iff $(\langle p, w\rangle, \theta) \in Y_{\mathcal{BP}}$.*

Proving this theorem is based on the following lemma:

Lemma 8 *$\mathcal{BP}$ has a run ρ starting from a configuration $(\langle p, w\rangle, \theta)$ s.t. each path of ρ visits configurations with control locations in F at least k times iff $(\langle p, w\rangle, \theta) \in X_k$.*

Indeed, let $c \in X_1$. Then c has a successor $C \subseteq F \times \Gamma^* \times 2^{\Delta \cup \Delta_c}$ (since $X_1 = Pre^+_{\mathcal{BP}}(X_0 \cap (F \times \Gamma^* \times 2^{\Delta \cup \Delta_c})))$. Therefore, $\mathcal{BP}$ has a run starting from c that visits some configuration $p \in F$ at least once. $X_2 = Pre^+_{\mathcal{BP}}\big(X_1 \cap (F \times \Gamma^* \times 2^{\Delta \cup \Delta_c})\big)$, thus $\forall c' \in X_2$, a run starting from c' will visit configurations in $X_1 \cap (F \times \Gamma^* \times 2^{\Delta \cup \Delta_c})$ at least once; and thus, it visits configurations with control locations in F at least twice. Thus, we can get by induction that $\forall k \geq 1$, for every configuration c in X_k, $\mathcal{BP}$ has a run that visits configurations with control locations in F at least k times.

Proof: $(\Rightarrow)$: We proceed by induction on k.

Basis. $k = 0$. In this case, we can directly obtain that $(\langle p, w \rangle, \theta) \in X_0$.

Step. $k \geq 1$. Let $(\langle p_1, w_1 \rangle, \theta_1), \cdots, (\langle p_n, w_n \rangle, \theta_n)$ be the first nodes of ρ that are visited in each path of ρ such that $p_i \in F$. Then we get (1) $(\langle p, w \rangle), \theta) \Rightarrow_{\mathcal{BP}} \{(\langle p_1, w_1 \rangle), \theta_1), \cdots, (\langle p_n, w_n \rangle, \theta_n)\}$. (2) for every $1 \leq i \leq n, p_i \in F$. (3) for every $1 \leq i \leq n$, $\mathcal{BP}$ has a run ρ_i from the configuration $(\langle p_i, w_i \rangle, \theta_i)$ s.t. all the paths of ρ_i can visit some configurations with control locations in F at least $k-1$ times.

By applying the induction hypothesis to (3), we can get that $(\langle p_i, w_i \rangle, \theta_i) \in X_{k-1}$ for each $1 \leq i \leq n$. Since $p_i \in F$, then $(\langle p_i, w_i \rangle, \theta_i) \in X_{k-1} \cap F \times \Gamma^* \times 2^{\Delta \cup \Delta_c}$. Moreover, since $X_k = Pre^+(X_{k-1} \cap F \times \Gamma^* \times 2^{\Delta \cup \Delta_c})$, we have that $(\langle p, w \rangle, \theta) \in X_k$.

$(\Leftarrow)$: In this direction, let's proceed by induction on k. It is obvious when $k = 0$. we only need to prove that $\mathcal{BP}$ has a run ρ from the configuration $(\langle p, w \rangle, \theta)$ such that each path of ρ can visit some configurations with control locations in F and least k times for $k \geq 1$.

Since $(\langle p, w \rangle, \theta) \in X_k$ for $X_k = Pre^+(X_{k-1} \cap F \times \Gamma^* \times 2^{\Delta \cup \Delta_c})$, we obtain that $(\langle p, w \rangle), \theta) \Rightarrow_{\mathcal{BP}} \{(\langle p_1, w_1 \rangle), \theta_1), \cdots, (\langle p_n, w_n \rangle, \theta_n)\}$ and $(\langle p_i, w_i \rangle, \theta_i) \in X_{k-1} \cap F \times \Gamma^* \times 2^{\Delta \cup \Delta_c}$ for every $1 \leq i \leq n$.

By applying the induction hypothesis, we can get that $\mathcal{BP}$ has a run ρ_i starting from $(\langle p_i, w_i \rangle, \theta_i)$ s.t. every path of ρ_i can visit some configurations with control locations in F at least $k-1$ times. Thus, $\mathcal{BP}$ has a run ρ from the configuration $(\langle p, w \rangle, \theta)$ such that each path of ρ can visit some configuration with the control location in F at least k times. $\square$

Then we can prove Theorem 4.2.1:

Proof: $(\Rightarrow)$: In this direction, we show that if $(\langle p, w\rangle, \theta) \notin Y_{\mathcal{BP}}$, then $\mathcal{BP}$ has no accepting run from $(\langle p, w\rangle, \theta)$.

Since $(\langle p, w\rangle, \theta) \notin Y_{\mathcal{BP}}$ and $Y_{\mathcal{BP}} = \bigcap_{i\geq 0} X_i$, there exists $k \geq 0$ s.t. $(\langle p, w\rangle, \theta) \notin X_k$. Assume $\mathcal{BP}$ has an accepting run from $(\langle p, w\rangle, \theta)$. Then, by Lemma 8, all runs from the configuration $(\langle p, w\rangle, \theta)$ can visit configurations with control location in F at least k times, we can get that $(\langle p, w\rangle, \theta) \in X_k$ which contradicts the fact that $(\langle p, w\rangle, \theta) \notin X_k$. Thus, $\mathcal{BP}$ has no accepting run from the configuration $(\langle p, w\rangle, \theta)$.

$(\Leftarrow)$: We prove that if $(\langle p, w\rangle, \theta) \in Y_{\mathcal{BP}}$, then $\mathcal{BP}$ has an accepting run from $(\langle p, w\rangle, \theta)$. Since $Y_{\mathcal{BP}} = Pre^+(Y_{\mathcal{BP}} \cap F \times \Gamma^* \times 2^{\Delta\cup\Delta_c})$ (note that $Y_{\mathcal{BP}}$ is a fix point of $Pre^+(X \cap F \times \Gamma^* \times 2^{\Delta\cup\Delta_c})$), then there exists a set of configurations $\{(\langle p_1, w_1\rangle, \theta_1), \cdots, (\langle p_n, w_n\rangle, \theta_n)\} \subseteq Y_{\mathcal{BP}} \cap F \times \Gamma^* \times 2^{\Delta\cup\Delta_c}$ such that $(\langle p, w\rangle, \theta) \Rightarrow_{\mathcal{BP}} \{(\langle p_1, w_1\rangle, \theta_1), \cdots, (\langle p_n, w_n\rangle, \theta_n)\}$. Thus, we can obtain that $(\langle p_i, w_i\rangle, \theta_i) \in Y_{\mathcal{BP}}, p_i \in F$.

Then we will construct a finite run (tree) ρ with root $(\langle p, w\rangle, \theta)$, the leaves of ρ are $\{(\langle p_1, w_1\rangle, \theta_1), \cdots, (\langle p_n, w_n\rangle, \theta_n)\}$ and the inner nodes are the successors during the derivation of $(\langle p, w\rangle, \theta) \Rightarrow_{\mathcal{BP}} \{(\langle p_1, w_1\rangle, \theta_1), \cdots, (\langle p_n, w_n\rangle, \theta_n)\}$. Every path of ρ can visit some configurations with control locations in F at least once.

Since $(\langle p_i, w_i\rangle, \theta_i) \in Y_{\mathcal{BP}}$, we can repeatedly construct a corresponding tree ρ_i for the configuration $(\langle p_i, w_i\rangle, \theta_i)$. Then the leaf $(\langle p_i, w_i\rangle, \theta_i)$ in ρ can be replaced by the tree ρ_i and we can obtain a new tree whose every path can visit some configuration with control location in F at least twice. Then we can infinitely repeat this procedure to leaves of the latest tree. Then each path of the latest tree can visit some configurations with control locations in F infinitely often. Thus, $\mathcal{BP}$ has an accepting run ρ. □

4.2.2 Computing $Y_{\mathcal{BP}}$

In this section, our goal is to compute $Y_{\mathcal{BP}}$. We show that this set can be effectively represented by an EAMA $\mathcal{A} = (Q, \Gamma, T, I, Q_f)$, where $Q \subseteq P \times 2^{\Delta\cup\Delta_c} \times \mathbb{N} \cup \{q_f\}$, $I \subseteq P \times 2^{\Delta\cup\Delta_c} \times \mathbb{N}$ is the set of initial states and q_f is the final state $(Q_f = \{q_f\})$. Following [41], we propose a saturation procedure to compute $\mathcal{A}$ iteratively. **Algorithm 1** below computes $\mathcal{A}$. Intuitively, it computes the

different X_i's iteratively. Each iteration step i computes an EAMA $\mathcal{A}_i$. States of $\mathcal{A}_i$ are of the form $(p,\theta)^i$, where $p \in P$ and $\theta \subseteq \Delta \cup \Delta_c$. There are two loops in the algorithm: the outer loop ($loop_1$) and the inner loop ($loop_2$). As will be explained later, if the sequence (X_i) is strictly decreasing, the outer loop won't terminate. So we introduce two projections to force termination as follows: for every $S \subseteq P \times 2^{\Delta\cup\Delta_c} \times \mathbb{N} \cup \{q_f\}$:

$$\pi^{-1}(S) = \begin{cases} \{q^i | q^{i+1} \in S\} \cup \{q_f\} & \text{if } q_f \in S \text{ or } \exists q^1 \in S \\ \{q^i | q^{i+1} \in S\} & \text{else.} \end{cases}$$

$$\pi^i(S) = \{q^i | \exists 1 \leq j \leq i \text{ s.t. } q^j \in S\} \cup \{q_f | q_f \in S\}$$

Algorithm 1 Computation of $Y_{\mathcal{BP}}$

1: **Initially:** $i = 0, T = \{(q_f, \gamma, \{q_f\})\}, \forall \gamma \in \Gamma, p \in P, \theta \subseteq \Delta \cup \Delta_c, (p,\theta)^0 = q_f$.
2: **Repeat** (we call this loop $loop_1$)
3: $i := i + 1$;
4: $\forall (p,\theta)^{i-1}$ in the current automaton s.t. $p \in F$,
add $(p,\theta)^i \xrightarrow{\epsilon} (p,\theta)^{i-1}$ to T
5: **Repeat** (we call this loop $loop_2$)
6: **if** $r : \langle p,\gamma \rangle \xhookrightarrow{[\sigma_1,\cdots,\sigma_n]} \{\langle p_1,w_1\rangle, \cdots, \langle p_n,w_n\rangle\} \in \Delta \cap \theta$
7: **if** $(\exists k, 1 \leq k \leq n : \sigma_k \cap \theta \neq \emptyset$ or $\forall k, 1 \leq k \leq n, \sigma_k = -)$
8: Add $(p,\theta)^i \xrightarrow{\gamma} S_Q$ to T,
where $S_Q = \{q \in Q_k | (p_k,\theta)^i \xrightarrow{w_k}_T Q_k, 1 \leq k \leq n \text{ s.t. } \sigma_k = - \text{ or } \sigma_k \cap \theta \neq \emptyset\}$.
9: **if** $r : p \xhookrightarrow{(\sigma,\sigma')} p' \in \Delta_c \cap \theta$, s.t.
10: $(p',\theta')^i \xrightarrow{\gamma}_T Q$ and $\theta' = \theta \backslash \sigma \cup \sigma'$
11: Then, add $(p,\theta)^i \xrightarrow{\gamma} Q$ to T.
12: **Until** No new transition rule can be added.
13: Remove from T the transition rules $(p,\theta)^i \xrightarrow{\epsilon} (p,\theta)^{i-1}$, for every $p \in F$.
14: Replace every $(p,\theta)^i \xrightarrow{\gamma} S$ in T by $(p,\theta)^i \xrightarrow{\gamma} \pi^i(S), \forall p \in P, \gamma \in \Gamma, S \subseteq Q$
15: **Until** $i > 1$ and for every $p \in P, \gamma \in \Gamma, \theta \in 2^{\Delta\cup\Delta_c}, S \subseteq P \times 2^{\Delta\cup\Delta_c} \times \{i\} \cup \{q_f\}, (p,\theta)^i \xrightarrow{\gamma} S \in T \iff (p,\theta)^{i-1} \xrightarrow{\gamma} \pi^{-1}(S) \in T$.

Intuitively, at each step i, every state (p,θ) is represented by state $(p,\theta)^i$ in $\mathcal{A}_i$. For every $(p,\theta) \in I$, $\mathcal{A}_i$ recognizes a configuration $(\langle p,w\rangle,\theta)$ if $(p,\theta)^i \xrightarrow{\omega}_T q_f$. $\mathcal{A}_0$ is the automaton obtained by **Line 1**. It accepts $X_0 = P \times \Gamma^* \times 2^{\Delta\cup\Delta_c}$. At

the beginning of each iteration, an ϵ-transition in the form $(p,\theta)^i \xrightarrow{\epsilon}_T (p,\theta)^{i-1}$ is added in **Line 4** for every $(p,\theta) \in F \times 2^{\Delta\cup\Delta_c}$ in the current automaton. This allows to get $L(\mathcal{A}_{i-1}) \cap (F \times \Gamma^* \times 2^{\Delta\cup\Delta_c})$. **Lines 5-12** ($loop_2$) is the saturation procedure that computes $Pre^*_{\mathcal{BP}}\big(L(\mathcal{A}_{i-1}) \cap (F \times \Gamma^* \times 2^{\Delta\cup\Delta_c})\big)$. They ensure that if θ is a phase such that $\langle p,\gamma\rangle \xrightarrow{[\sigma_1,\cdots,\sigma_n]} \{\langle p_1,w_1\rangle,\cdots,\langle p_n,w_n\rangle\} \in \Delta\cap\theta$, s.t. either $\exists k, 1\le k\le n, \sigma_k\cap\theta \ne \emptyset$ or $\forall k, 1\le k\le n, \sigma_k = -$, and for every k s.t. $\sigma_k\cap\theta\ne\emptyset$ or $\sigma_k = -$, $(\langle p_k, w_k w\rangle,\theta) \in L(\mathcal{A}_i)$ (i.e., $(p_k,\theta)^i \xrightarrow{w_k w}_T q_f$), then $(\langle p,\gamma w\rangle,\theta)$ should also be in $L(\mathcal{A}_i)$ (since it is a predecessor of $\{(\langle p_k,w_k w\rangle,\theta), 1\le k\le n\}$). I.e., T should contain the path $(p,\theta)^i \xrightarrow{\gamma w}_T q_f$. This path is added thanks to Line 8. Moreover, if θ is a phase such that $\langle p,\gamma\rangle \xhookrightarrow{(\sigma,\sigma')} p' \in \Delta_c \cap \theta$ and $(\langle p',\gamma w\rangle,\theta') \in L(\mathcal{A}_i)$ (i.e., $(p',\theta')^i \xrightarrow{\gamma w}_T q_f$), where $\theta' = \theta\backslash\sigma \cup \sigma'$; then $(\langle p,\gamma w\rangle,\theta)$ should also be in $L(\mathcal{A}_i)$ (since it is a predecessor of $(\langle p',\gamma w\rangle,\theta')$). I.e., T should contain the path $(p,\theta)^i \xrightarrow{\gamma w}_T q_f$. This path is added thanks to Line 11. **Line 13** removes the ϵ-transition added by Line 4. This leads to $Pre^+_{\mathcal{BP}}\big(L(\mathcal{A}_{i-1}) \cap (F \times \Gamma^* \times 2^{\Delta\cup\Delta_c})\big)$.

Let **Algorithm A** be **Algorithm 1** without Line 14. Then, if **Algorithm A** terminates, it computes $Y_{\mathcal{BP}}$. However, if the sequence X_i is strictly decreasing, **Algorithm A** never terminates. **Lines 14-15** are then used to force termination. Indeed, thanks to the substitution of **Line 14**, at the end of step i, states of the form $(p,\theta)^j$, for $j < i$ become useless and can be removed. **Line 15** checks then whether at step i, the transitions of $\mathcal{A}_i$ are "the same" than those of $\mathcal{A}_{i-1}$. If this is the case, the algorithm terminates. Termination of the algorithm then follows from the fact that step i adds less transitions than step $i-1$. Intuitively, this is due to the fact that $L(\mathcal{A}_i) \subseteq L(\mathcal{A}_{i-1})$, because step i computes $Pre^+_{\mathcal{BP}}\big(L(\mathcal{A}_{i-1}) \cap (F \times \Gamma^* \times 2^{\Delta\cup\Delta_c})\big)$ and $\mathcal{A}_0$ accepts $P \times \Gamma^* \times 2^{\Delta\cup\Delta_c}$. Thus, we can show that:

Theorem 4.2.2 ***Algorithm 1*** *always terminates and produces* $Y_{\mathcal{BP}}$.

To prove Theorem 4.2.2, we need the following lemma:

Lemma 9 *In* ***Algorithm 1****, for every* $\gamma\in\Gamma, w\in\Gamma^*, p\in P, S\subseteq Q$*; at each step* $i\ge 1$*, the following holds:*

(a) if $(p,\theta)^i \xrightarrow{\gamma} S \in T$*, then* $(p,\theta)^{i-1} \xrightarrow{\gamma} \pi^{-1}(\pi^i(S)) \in T$.

(b) if $(p,\theta)^i \xrightarrow{w}_T S$*, then* $(p,\theta)^{i-1} \xrightarrow{w}_T \pi^{-1}(\pi^i(S))$.

Proof: We proceed by induction on i.

Basis. $i = 1$. In this case, whenever a transition rule $(p,\theta)^1 \xrightarrow{\gamma} S$ is added to T by either the saturation procedure (Lines 2-12) or the substitution (Line 14), we can get that $\pi^{-1}(\pi^1(S)) = \{q_f\}$. Since $(p,\theta)^0 = \{q_f\}$ and $q_f \xrightarrow{\gamma} \{q_f\}$ in T, we obtain that $(p,\theta)^0 \xrightarrow{\gamma} \{q_f\}$. Hence, $(p,\theta)^0 \xrightarrow{\gamma} \pi^{-1}(\pi^1(S))$. Therefore, the statement (a) holds.
For statement (b). In this case, we can have that $\pi^{-1}(\pi^1(S)) = \{q_f\}$. If $(p,\theta)^1 \xrightarrow{w}_T S$, We need to show that $(p,\theta)^0 \xrightarrow{w}_T \pi^{-1}(\pi^1(S))$. Since $(p,\theta)^0 \xrightarrow{\gamma} \{q_f\}$ and $q_f \xrightarrow{\gamma'} \{q_f\}$ for $\gamma,\gamma' \in \Gamma$, we obtain that $(p,\theta)^0 \xrightarrow{w}_T \pi^{-1}(\pi^1(S))$ (since $\pi^{-1}(\pi^1(S)) = \{q_f\}$) for every $S \subseteq P \times 2^{\Delta\cup\Delta_c} \times \{0,1\} \cup \{q_f\}$. The ϵ-case ($w = \epsilon$) is trivial (since $(p,\theta)^0 \xrightarrow{\epsilon}_T \{(p,\theta)^0\}$ and either $(p,\theta)^1 \xrightarrow{\epsilon}_T \{(p,\theta)^0\}$ if $p \in F$ or $(p,\theta)^1 \xrightarrow{\epsilon}_T \{(p,\theta)^1\}$.) Hence, the statement (b) holds.

Step. $i > 1$. Let k be the number of transition rules added at the step i. We proceed by induction on k.

- **Basis.** $k = 0$. there is no transition rule added in the form of $(p,\theta)^i \xrightarrow{\gamma} S$ which implies that the statement (a) holds. For every $(p,\theta)^i \xrightarrow{w}_T S$, we get that there is a path $(p,\theta)^i \xrightarrow{\epsilon}_T (p,\theta)^{i-1} \xrightarrow{w}_T S$ in the automaton for some $S \subseteq P \times 2^{\Delta\cup\Delta_c} \times \{i-1\} \cup \{q_f\}$ if $p \in F$ or $(p,\theta)^i \xrightarrow{\epsilon}_T S$ with $S = \{(p,\theta)^i\}$ and $w = \epsilon$. Since $(p,\theta)^{i-1} \xrightarrow{\epsilon}_T (p,\theta)^{i-1}$ and $\pi^{-1}(\pi^i(S)) = S$, we have $(p,\theta)^{i-1} \xrightarrow{\epsilon}_T (p,\theta)^{i-1} \xrightarrow{w}_T \pi^{-1}(\pi^i(S))$. This implies the statement (b) holds.

- **Step.** $k \geq 1$.

  For statement (a). Let $t = (p,\theta)^i \xrightarrow{\gamma} S$ be the k-th transition rule added by the saturation procedure. Then there exist $1 \leq j \leq n$ and $\theta_j \subseteq \Delta \cup \Delta_c$ s.t. $(p_{j_l},\theta_{j_l})^i \xrightarrow{w_{j_l}}_T S_{j_l}$ where $\{j_1,\cdots,j_m\} \subseteq \{1,\cdots,n\}, 1 \leq l \leq m$. There are 2 cases depending on whether t is added by Line 8 or not.

  (I) if t is added by Line 8, then there exists a transition rule

  $$r : \langle p,\gamma\rangle \xrightarrow{[\sigma_1,\cdots,\sigma_n]} \{\langle p_1,w_1\rangle,\cdots,\langle p_n,w_n\rangle\} \in \Delta \cap \theta$$

  in $\mathcal{BP}$ where $\theta_j = \theta$ s.t. $\exists i : 1 \leq i \leq n,\ \sigma_i \cap \theta \neq \emptyset$ or $\sigma_i = -$ for every $1 \leq i \leq n$.

Let $\{\sigma_{j_1}, \cdots, \sigma_{j_m}\}$ be the set of σ s.t. $\sigma_{j_l} \cap \theta \neq \emptyset$ or $\sigma_{j_l} = -$ for $1 \leq l \leq m$. Then there exist $(p_{j_l}, \theta)^i \xrightarrow{w_{j_l}}_T S_{j_l}$ for every $1 \leq l \leq m$ s.t. $S = \{q \in S_x | x \in \{j_1, \cdots, j_m\}\}$ i.e. $S = \bigcup_{l=1}^m S_{j_l}$.

(II) if t is added by Line 11, then there exists a transition rule $r : p \xhookrightarrow{(\sigma,\sigma')} p_j \in \Delta_c \cap \theta$ in $\mathcal{BP}$ where $\theta_j = \theta \backslash \sigma \cup \sigma'$. In this case, $n = j = 1$. Then, there exists $(p_j, \theta_j)^i \xrightarrow{\gamma}_T S_j$ s.t. $S = S_j$. We rewrite $(p_j, \theta_j)^i \xrightarrow{\gamma}_T S_j$ of the form $(p_{j_l}, \theta_{j_l})^i \xrightarrow{w_{j_l}}_T S_{j_l}$ where $j_l = 1$, $\theta_{j_l} = \theta_j$ and $w_{j_l} = \gamma$.

By applying the induction hypothesis on i to $(p_{j_l}, \theta_{j_l})^i \xrightarrow{w_{j_l}}_T S_{j_l}$ for each $l : 1 \leq l \leq m$, we can obtain that $(p_{j_l}, \theta_{j_l})^{i-1} \xrightarrow{w_{j_l}}_T \pi^{-1}(\pi^i(S_{j_l}))$. Therefore, we only need to prove that there exists R_{j_l} s.t. $\pi^{i-1}(R_{j_l}) = \pi^{-1}(\pi^i(S_{j_l}))$ for every $1 \leq l \leq m$ and $(p_{j_l}, \theta_{j_l})^{i-1} \xrightarrow{w_{j_l}}_T R_{j_l}$ exists in the current automaton during the $(i-1)$-th iteration of $loop_1$. If the derivation of $(p_{j_l}, \theta_{j_l})^{i-1} \xrightarrow{w_{j_l}}_T \pi^{-1}(\pi^i(S_{j_l}))$ does not use any transition rule added by the substitution at Line 14, then, $(p_{j_l}, \theta_{j_l})^{i-1} \xrightarrow{w_{j_l}}_T \pi^{-1}(\pi^i(S_{j_l}))$ exists during the saturation procedure at the $(i-1)$th iteration. Otherwise, there is a transition rule $q^{i-1} \xrightarrow{\gamma'}_T R$ which is used in the derivation of $(p_{j_l}, \theta_{j_l})^{i-1} \xrightarrow{w_{j_l}}_T \pi^{-1}(\pi^i(S_{j_l}))$ and is obtained by replacing $q^{i-1} \xrightarrow{\gamma'}_T R'$ at line 14 where $R = \pi^{i-1}(R')$. Let us decompose $(p_{j_l}, \theta_{j_l})^{i-1} \xrightarrow{w_{j_l}}_T \pi^{-1}(\pi^i(S_{j_l}))$ as follows:

- $w_{j_l} = u\gamma' v$ with $u, v \in \Gamma^*$,
- $(p_{j_l}, \theta_{j_l})^{i-1} \xrightarrow{u}_T G \cup \{q^{i-1}\}$ with $G \subseteq P \times 2^{\Delta \cup \Delta_c} \times \{i-1\} \cup \{q_f\}$
- $G \xrightarrow{\gamma' v}_T G'$
- $R \xrightarrow{v}_T G''$
- $\pi^{-1}(\pi^i(S_{j_l})) = G' \cup G''$

By applying the induction hypothesis on i to $R \xrightarrow{v}_T G''$, there exists G''' s.t. $R' \xrightarrow{v}_T G'''$ is obtained by applying the saturation procedure at the $(i-1)$th iteration and $G'' = \pi^{i-1}(G''')$. Thus, there must exist R_{j_l} s.t. $\pi^{i-1}(R_{j_l}) = \pi^{-1}(\pi^i(S_{j_l}))$ and the derivation of $(p_{j_l}, \theta_{j_l})^{i-1} \xrightarrow{w_{j_l}}_T R_{j_l}$ uses transition rules added by the substitution at Line 14 less often than the derivation of $(p_{j_l}, \theta_{j_l})^{i-1} \xrightarrow{w_{j_l}}_T S_{j_l}$.

Similarly, we can apply the same reasoning to $(p_{j_l}, \theta_{j_l})^{i-1} \xrightarrow{w_{j_l}}_T R_{j_l}$ to show that there exists R'_{j_l} s.t. $(p_{j_l}, \theta_{j_l})^{i-1} \xrightarrow{w_{j_l}}_T R'_{j_l}$ holds during the saturation procedure at the $(i-1)$th iteration. Thus, the statement (a) holds. If a transition rule $(p, \theta)^i \xrightarrow{\gamma} \pi^i(S)$ is added by the substitution at line 14 due to the transition $t = (p, \theta)^i \xrightarrow{\gamma} S$, then the statement (a) still holds.

For statement (b). Let us consider the statement (b) where we show that if $(p, \theta)^{i-1} \xrightarrow{w}_T S$, then $(p, \theta)^{i-1} \xrightarrow{w}_T \pi^{-1}(\pi^i(S))$. Then, suppose $t = (p_0, \theta_0)^i \xrightarrow{\gamma} \{q_1^i, \cdots, q_n^i, q_{n+1}^{i-1}, \cdots, q_{n+n'}^{i-1}\}$ be the k-th transition rule added by either the saturation procedure or the substitution (line 14). Let x be the number of times that t is used in the path $(p, \theta)^i \xrightarrow{w}_T S$. We proceed by induction on x. In the basic case when $x = 0$, the property holds by applying the induction hypothesis on k. Let us consider the case where $x > 0$. Then, there exist $u, v \in \Gamma^*$ s.t. $w = v\gamma u$ and there exist the following path in the current automaton:

- $(p, \theta)^i \xrightarrow{v}_T G \cup \{(p_0, \theta_0)^i\}$ for some $G \subseteq Q$ where t is not used in the derivation of $(p, \theta)^i \xrightarrow{v}_T G \cup \{(p_0, \theta_0)^i\}$
- $G \xrightarrow{\gamma u} G'$.
- $q_j^i \xrightarrow{u}_T S_j$ for every $j : 1 \leq j \leq n$.
- $q_j^{i-1} \xrightarrow{u}_T S_j$ for every $j : n+1 \leq j \leq n+n'$.
- $S = G' \cup \bigcup_{j=1}^{n+n'} S_j$.

By applying the induction hypothesis on k to $(p, \theta)^i \xrightarrow{v}_T G \cup \{(p_0, \theta_0)^i\}$, we can obtain that $(p, \theta)^{i-1} \xrightarrow{v}_T \pi^{-1}(\pi^i(G)) \cup \{(p_0, \theta_0)^{i-1}\}$. By applying the induction hypothesis on x to $G \xrightarrow{\gamma u} G'$ and $q_j^i \xrightarrow{u}_T S_j$, we get that $\pi^{-1}(\pi^i(G)) \xrightarrow{\gamma u}_T \pi^{-1}(\pi^i(G'))$ and $q_j^{i-1} \xrightarrow{u}_T \pi^{-1}(\pi^i(S_j))$ for every $j : 1 \leq j \leq n$. By applying the statement (a) to $(p_0, \theta_0)^i \xrightarrow{\gamma} \{q_1^i, \cdots, q_n^i, q_{n+1}^{i-1}, \cdots, q_{n+n'}^{i-1}\}$, we can get that $(p_0, \theta_0)^{i-1} \xrightarrow{\gamma} \{q_1^{i-1}, \cdots, q_{n+n'}^{i-1}\}$.

Since $\pi^{-1}(\pi^i(S)) = \pi^{-1}(\pi^i(G')) \cup \bigcup_{j=1}^{n} \pi^{-1}(\pi^i(S_j)) \cup \bigcup_{j=n+1}^{n+n'} S_j$, we get that $(p, \theta)^{i-1} \xrightarrow{w}_T \pi^{-1}(\pi^i(S))$. Thus, the statement (b) holds.

□

In order to show that there exists a fix-point s.t. the termination condition of $loop_1$ is true, let Algorithm C be Algorithm 1 without Line 15 i.e. without the termination condition of $loop_1$. We show that there exists a fix-point n such that $L(\mathcal{A}_n) = L(\mathcal{A}_{n+1})$.

Lemma 10 *Let $n \geq 1$ be the first number in Algorithm C s.t. for every $p \in P, \gamma \in \Gamma, S \subseteq (P \times 2^{\Delta \cup \Delta_c} \times \{n+1\}) \cup \{q_f\}$, $\theta \subseteq \Delta \cup \Delta_c$, $(p,\theta)^{n+1} \xrightarrow{\gamma} S \in T \Leftrightarrow (p,\theta)^n \xrightarrow{\gamma} \pi^{-1}(S) \in T$. For every $i \geq n$, $L(\mathcal{A}_{i+1}) = L(\mathcal{A}_n)$.*

Proof: Since $(p,\theta)^{i+1} \xrightarrow{\gamma} S$ will be replaced by $(p,\theta)^{i+1} \xrightarrow{\gamma} \pi^{i+1}(S)$ by line 14, then each path $(p,\theta)^{i+1} \xrightarrow{w}_T \{q_f\}$ only uses states in the form of $P \times 2^{\Delta \cup \Delta_c} \times \{i+1\} \cup \{q_f\}$. It is sufficient to prove that for every $(p,\theta) \in P \times 2^{\Delta \cup \Delta_c}, \gamma \in \Gamma, (p,\theta)^{i+1} \xrightarrow{\gamma} \{q_1^{i+1}, \cdots, q_m^{i+1}\} \in T \Leftrightarrow (p,\theta)^n \xrightarrow{\gamma} \{q_1^n, \cdots, q_m^n\} \in T$ by induction on i.

Basis. $i = n$. We can get directly from the condition of n that

$$(p,\theta)^{n+1} \xrightarrow{\gamma} \{q_1^{n+1}, \cdots, q_m^{n+1}\} \in T \Leftrightarrow (p,\theta)^n \xrightarrow{\gamma} \{q_1^n, \cdots, q_m^n\} \in T \qquad (0)$$

Step. $i > n$. By applying the induction hypothesis (induction on i), then we obtain that for every $(p,\theta) \in P \times 2^{\Delta \cup \Delta_c}, \gamma \in \Gamma$,

$$(p,\theta)^i \xrightarrow{\gamma} \{q_1^i, \cdots, q_m^i\} \in T \Leftrightarrow (p,\theta)^n \xrightarrow{\gamma} \{q_1^n, \cdots, q_m^n\} \in T \qquad (1)$$

Since the result of (1), $(p,\theta)^{i+1} \xrightarrow{\gamma} \{q_1^{i+1}, \cdots, q_m^{i+1}\}$ is added based on $\mathcal{A}_i$, for every $(p,\theta) \in P \times 2^{\Delta \cup \Delta_c}, \gamma \in \Gamma$, we obtain that:

$$(p,\theta)^{i+1} \xrightarrow{\gamma} \{q_1^{i+1}, \cdots, q_m^{i+1}\} \in T \Leftrightarrow (p,\theta)^{n+1} \xrightarrow{\gamma} \{q_1^{n+1}, \cdots, q_m^{n+1}\} \in T$$

From (0), we get that

$$(p,\theta)^{i+1} \xrightarrow{\gamma} \{q_1^{i+1}, \cdots, q_m^{i+1}\} \in T \Leftrightarrow (p,\theta)^n \xrightarrow{\gamma} \{q_1^n, \cdots, q_m^n\} \in T$$

□

Lemma 11 *In **Algorithm 1**, $\forall i \geq 1$, after line 13, $\mathcal{A}_i$ accepts $Pre^+\big(L(\mathcal{A}_{i-1}) \cap (F \times \Gamma^* \times 2^{\Delta \cup \Delta_c})\big)$.*

Proof:

To prove this lemma, we first show that each configuration c accepted by $\mathcal{A}_i$ after Line 13 is such that $c \in Pre^+\big(L(\mathcal{A}_{i-1}) \cap (F \times \Gamma^* \times 2^{\Delta\cup\Delta_c})\big)$, then we show that each configuration $c \in Pre^+\big(L(\mathcal{A}_{i-1}) \cap (F \times \Gamma^* \times 2^{\Delta\cup\Delta_c})\big)$ is accepted by $\mathcal{A}_i$ after Line 13.

($\Rightarrow$:) Suppose $(\langle p, w\rangle, \theta)$ is a configuration accepted by $\mathcal{A}_i$ after Line 13, we show that $(\langle p, w\rangle, \theta) \in Pre^+\big(L(\mathcal{A}_{i-1}) \cap (F \times \Gamma^* \times 2^{\Delta\cup\Delta_c})\big)$. Since there is no path of the form $(p, \theta)^i \xrightarrow{\epsilon}_T \{q_f\}$ after Line 13, then we get that $|w| \geq 1$. Then there exist $\gamma \in \Gamma$, $u \in \Gamma^+$ and $S \subseteq Q$ s.t. $w = \gamma u$, $t = (p, \theta)^i \xrightarrow{\gamma} S \in T$ and $S \xrightarrow{u}_T \{q_f\}$. There are 2 cases depending on whether t is added by Line 8 or Line 11. Let r be the transition rule used to add t.

- Case t is added by Line 8: then there exists a rule $r : \langle p, \gamma\rangle \xhookrightarrow{[\sigma_1,\cdots,\sigma_n]} \{\langle p_1, w_1\rangle \cdots \langle p_n, w_n\rangle\} \in \Delta$ s.t. $r \in \theta$, and either $\sigma_j = -$ for every $j : 1 \leq j \leq n$ or $\exists j : 1 \leq j \leq n$ s.t. $\sigma_j \cap \theta \neq \emptyset$, $S = \{q \in S_j | (p_j, \theta)^i \xrightarrow{w_j}_T S_j, 1 \leq j \leq n, \sigma_j \cap \theta \neq \emptyset \text{ or } \sigma_j = -\}$.

  This implies that $\{(\langle p_j, w_j u\rangle, \theta) | 1 \leq j \leq n, \sigma_j \cap \theta \neq \emptyset \text{ or } \sigma_j = -\} \subseteq Pre^*\big(L(\mathcal{A}_{i-1}) \cap (F \times \Gamma^* \times 2^{\Delta\cup\Delta_c})\big)$.

  Thus, we get that $(\langle p, \gamma u\rangle, \theta) \in Pre^+\big(L(\mathcal{A}_{i-1}) \cap (F \times \Gamma^* \times 2^{\Delta\cup\Delta_c})\big)$. The property holds.

- Case t is added by Line 11: then there exists a rule $r : p \xhookrightarrow{(\sigma,\sigma')} p_1 \in \Delta_c$ s.t. $r \in \theta$, $\theta_1 = \theta \backslash \sigma \cup \sigma'$. We get that $(p_1, \theta_1)^i \xrightarrow{\gamma}_T S_1$. This implies that $\{(\langle p_1, \gamma u\rangle, \theta_1)\} \subseteq Pre^*\big(L(\mathcal{A}_{i-1}) \cap (F \times \Gamma^* \times 2^{\Delta\cup\Delta_c})\big)$

  Thus, we get that $(\langle p, \gamma u\rangle, \theta) \in Pre^+\big(L(\mathcal{A}_{i-1}) \cap (F \times \Gamma^* \times 2^{\Delta\cup\Delta_c})\big)$. The property holds.

($\Leftarrow$:) Suppose $(\langle p, w\rangle, \theta) \in Pre^+\big(L(\mathcal{A}_{i-1}) \cap (F \times \Gamma^* \times 2^{\Delta\cup\Delta_c})\big)$, we show that $(\langle p, w\rangle, \theta)$ is accepted by $\mathcal{A}_i$ after Line 13. Since $Pre^+\big(L(\mathcal{A}_{i-1}) \cap (F \times \Gamma^* \times 2^{\Delta\cup\Delta_c})\big) = Pre^*\Big(Pre\big(L(\mathcal{A}_{i-1}) \cap (F \times \Gamma^* \times 2^{\Delta\cup\Delta_c})\big)\Big)$, we obtain that $Pre^+\big(L(\mathcal{A}_{i-1}) \cap (F \times \Gamma^* \times 2^{\Delta\cup\Delta_c})\big)$ is the limit of the infinite sequence $\{C_i\}_{i\geq 0}$ given by $C_0 = Pre\big(L(\mathcal{A}_{i-1}) \cap (F \times \Gamma^* \times 2^{\Delta\cup\Delta_c})\big)$ and $C_{j+1} = C_j \cup Pre(C_j)$ for every $j \geq 0$ ($C_j \subseteq C_{j+1}$ for $j \geq 0$). Thus, we only need to show that for every $j \geq 0$, $(\langle p, w\rangle, \theta) \in C_j$, there exists a path $(p, \theta)^i \xrightarrow{w}_T \{q_f\}$ in $\mathcal{A}_i$ whose derivation doesn't

use any transition rule in the form of $q^i \xrightarrow{\epsilon} \{q^{i-1}\}$. We proceed by induction on j.

Basis. $j = 0$. By applying the saturation procedure (Lines 5-12), $\mathcal{A}_i$ can accept C_0 if only the out-coming states in the form of $(p, \theta)^i$ of the added transition rules are regarded as initial states. Moreover, these transition rules are in the form of $(p, \theta)^i \xrightarrow{\gamma} Q$ for some $Q \subseteq P \times 2^{\Delta \cup \Delta_c} \times \{i-1\} \cup \{q_f\}$. Thus, for every configuration $(\langle p, w\rangle, \theta) \in C_0$, $\mathcal{A}_i$ has a path of the form $(p, \theta)^i \xrightarrow{w}_T \{q_f\}$ whose derivation doesn't use any transition rule in the form of $q^i \xrightarrow{\epsilon} \{q^{i-1}\}$.

Step. $j \geq 1$. For every configuration $(\langle p, w\rangle, \theta) \in C_j$, since $C_j = C_{j-1} \cup Pre(C_{j-1})$, we have $(\langle p, w\rangle, \theta) \in C_{j-1}$ or $(\langle p, w\rangle, \theta) \in Pre(C_{j-1})$.

If $(\langle p, w\rangle, \theta) \in C_{j-1}$, the result follows from the induction hypotheses.

If $(\langle p, w\rangle, \theta) \in Pre(C_{j-1})$, there are 2 cases depending on whether it corresponds to a self-modifying rule or not.

- If there exists a transition rule $r : \langle p, \gamma\rangle \xhookrightarrow{[\sigma_1, \cdots, \sigma_n]} \{\langle p_1, w_1\rangle, \cdots, \langle p_n, w_n\rangle\} \in \Delta$ s.t. $r \in \theta$ either $\exists k : 1 \leq k \leq n, \sigma_k \cap \theta \neq \emptyset$ or $\forall k : 1 \leq k \leq n, \sigma_k = -$, and $u \in \Gamma^*$ s.t. $w = \gamma u$ and $(\langle p_k, w_k u\rangle, \theta) \in C_{j-1}$ for $1 \leq k \leq n$ s.t. $\sigma_k \cap \theta \neq \emptyset$ or $\sigma_k = -$. By applying the induction hypothesis, we obtain that $\mathcal{A}_i$ has a path $(p_k, \theta)^i \xrightarrow{w_k}_T Q_k \xrightarrow{u}_T \{q_f\}$ for $1 \leq k \leq n, \sigma_k \cap \theta \neq \emptyset$ or $\sigma_k = -$. Applying the saturation rule, we obtain that $(p, \theta)^i \xrightarrow{\gamma} S_Q$ where $S_Q = \{q \in Q_k | (p_k, \theta)^i \xrightarrow{w_k}_T Q_k, 1 \leq k \leq n, \sigma_k \cap \theta \neq \emptyset \text{ or } \sigma_k = -\}$. This implies that $\mathcal{A}_i$ has a path $(p, \theta)^i \xrightarrow{\gamma} S_Q \xrightarrow{u}_T \{q_f\}$ whose derivation doesn't use any transition rule in the form of $q^i \xrightarrow{\epsilon} \{q^{i-1}\}$. The property holds.
- If there exists a transition rule $r : p \xhookrightarrow{(\sigma, \sigma')} p_1 \in \Delta_c \cap \theta$ s.t. $\theta_1 = \theta \backslash \sigma \cup \sigma'$ and $(\langle p_1, wu\rangle, \theta_1) \in C_{j-1}$. By applying the induction hypothesis, we obtain that $\mathcal{A}_i$ has a path $(p_1, \theta_1)^i \xrightarrow{w}_T Q_k \xrightarrow{u}_T \{q_f\}$. Applying the saturation rule, we obtain that $(p, \theta)^i \xrightarrow{\gamma} Q_1$. This implies that $\mathcal{A}_i$ has a path $(p, \theta)^i \xrightarrow{\gamma} Q_1 \xrightarrow{u}_T \{q_f\}$ whose derivation doesn't use any transition rule in the form of $q^i \xrightarrow{\epsilon} \{q^{i-1}\}$. The property holds.

Thus, $\mathcal{A}_i$ accepts $Pre^+\big(L(\mathcal{A}_{i-1}) \cap (F \times \Gamma^* \times 2^{\Delta \cup \Delta_c})\big)$. □

Lemma 12 *In* ***Algorithm C*** *(Algorithm 1 without Line 15),* $\forall i \geq 0$,

(a) *for every accepting run* ρ *of* $\mathcal{BP}$ *from* $(\langle p, w\rangle, \theta) \in P \times \Gamma^* \times 2^{\Delta \cup \Delta_c}$, *there exists a path* $(p,\theta)^i \xrightarrow{w}_T \{q_f\}$ *in* $\mathcal{A}_i$ *and for every decomposition* $(p,\theta)^i \xrightarrow{u}_T Q \xrightarrow{v}_T \{q_f\}$ *of the path* $(p,\theta)^i \xrightarrow{w}_T \{q_f\}$, *if* $Q \neq \{q_f\}$, *then for all* $(p',\theta')^i$ *or* $(p',\theta')^{i-1}$ *in* $Q \backslash \{q_f\}$, *some path of the run* ρ *will reach* $(\langle p', v\rangle, \theta')$ *i.e.* $(\langle p, w\rangle, \theta) \Rightarrow^* (\langle p', v\rangle, \theta')$

(b) $Y_{\mathcal{BP}} \subseteq L(\mathcal{A}_i)$ *after substitution at line 14.*

Proof: We proceed by induction on i.

Basis. $i = 0$. The statement (a) holds directly from the fact that for every configuration $(\langle p, w\rangle, \theta) \in P \times \Gamma^* \times 2^{\Delta \cup \Delta_c}$, there exists a path $(p,\theta)^0 \xrightarrow{w}_T \{q_f\}$ in the initial automaton $\mathcal{A}_0$ and for every decomposition $(p,\theta)^0 \xrightarrow{u}_T Q \xrightarrow{v}_T \{q_f\}, Q = \{q_f\}$. Then, statement (b) holds from the fact that $Y_{\mathcal{BP}} \subseteq P \times \Gamma^* \times 2^{\Delta \cup \Delta_c} = L(\mathcal{A}_0)$.

Step. $i \geq 1$. For the statement (a). Let $H(\rho)$ be the maximum number of steps required by the paths of ρ from root $((\langle p, w\rangle, \theta))$ to reach some configuration with control locations in F. We apply a nested induction on $H(\rho)$.

- **Basis.** $H(\rho) = 0$. Since the root of ρ is $(\langle p, w\rangle, \theta)$, we can obtain that $(\langle p, w\rangle, \theta) \in F \times \Gamma^* \times 2^{\Delta \cup \Delta_c}$. By the transition rule added to the automaton during the i-th iteration by line 4, we can get that $(p,\theta)^i \xrightarrow{\epsilon} \{(p,\theta)^{i-1}\}$ is a transition rule of $\mathcal{A}_i$. Then, by applying the induction hypothesis on i, the result immediately follows.

- **Step.** $H(\rho) \geq 1$. Let $\rho^1, \cdots, \rho^n$ be the sub-trees of ρ whose root is the children of the root $(\langle p, w\rangle, \theta)$. Let $p_1, \cdots, p_n \in P$, $w_1, \cdots, w_n \in \Gamma^*$, $\gamma \in \Gamma$ and $\theta_1 \subseteq \Delta \cup \Delta_c$ be such that $w = \gamma w'$ and the roots of the sub-trees of ρ are $(\langle p_1, w_1 w'\rangle, \theta_1), \cdots, (\langle p_n, w_n w'\rangle, \theta_n)$. There are 2 cases depending on whether $(\theta = \theta_j)$ for every $j : 1 \leq j \leq n$ or not.

  - Case $\theta_j = \theta$ for every $j : 1 \leq j \leq n$. Then $\rho^1, \cdots, \rho^n$ are the accepting runs of $\mathcal{BP}$ from configurations $(\langle p_1, w_1 w'\rangle, \theta), \cdots, (\langle p_n, w_n w'\rangle, \theta)$ and there exists $r : \langle p, \gamma\rangle \xhookrightarrow{[\sigma_{j_1}, \cdots, \sigma_{j_m}]} \{\langle p_{j_1}, w_{j_1}\rangle, \cdots, \langle p_{j_m}, w_{j_m}\rangle\} \in \Delta$ s.t. $r \in \theta$, $\{1, \cdots, n\} \subseteq \{j_1, \cdots, j_m\}$ and for every $1 \leq j \leq n$, $\sigma_j \cap \theta \neq \emptyset$ or $\sigma_j = -$. Note that for the constraint $[\sigma_{j_1}, \cdots, \sigma_{j_m}]$, either $\exists 1 \leq l \leq m$, $\sigma_{j_l} \cap \theta \neq \emptyset$ or $\forall 1 \leq l \leq m$, $\sigma_{j_l} = -$. Since $H(\rho) \geq 1$ ($p \notin F$), we can get $H(\rho^j) < H(\rho)$ for $1 \leq j \leq n$. Thus we apply the nested induction

hypothesis on $H(\rho^j)$, we can get that there exists a path $(p_j,\theta)^i \xrightarrow{w_j w'}_T \{q_f\}$ in $\mathcal{A}_i$ and for every decomposition $(p_j,\theta)^i \xrightarrow{u}_T Q \xrightarrow{v}_T \{q_f\}$ of the path $(p_j,\theta)^i \xrightarrow{w_j w'}_T \{q_f\}$, if $Q \neq \{q_f\}$, then for all $(p',\theta')^k \in Q\backslash\{q_f\}$ with $k \in \{i, i-1\}$, some path of the run ρ^j will reach the configuration $(\langle p', v\rangle, \theta')$.

Moreover, there exists a path $(p_j,\theta)^i \xrightarrow{w_j}_T Q_j \xrightarrow{w'}_T \{q_f\}$ in $\mathcal{A}_i$ for every $j : 1 \leq j \leq n$ and by applying the saturation procedure, we get that $(p,\theta)^i \xrightarrow{\gamma} \bigcup_{j=1}^{n} Q_j \xrightarrow{w'}_T \{q_f\}$ in $\mathcal{A}_i$. Then for every decomposition $(p,\theta)^i \xrightarrow{u}_T Q \xrightarrow{v}_T \{q_f\}$ of the path $((p,\theta)^i \xrightarrow{\gamma w'}_T \{q_f\})$, if $Q \neq \{q_f\}$, then for all $(p',\theta')^k \in Q\backslash\{q_f\}$ with $k \in \{i, i-1\}$, some path of the run ρ will reach the configuration $(\langle p', v\rangle, \theta')$. Thus, we can have $(\langle p, w\rangle, \theta) \Rightarrow^* (\langle p', v\rangle, \theta')$.

- Case $\theta_j \neq \theta$, then there exists a transition rule r in θ, $r : p \xhookrightarrow{(\sigma,\sigma')} p_j \in \Delta_c$ and $\theta_j = \theta\backslash\sigma \cup \sigma'$. In this case, $w_j = \gamma$ and $j = 1$. Thus, the root of ρ^j is $(\langle p_j, w_j w'\rangle, \theta_j)$. Since $H(\rho) \geq 1$ ($p \notin F$), we can get $H(\rho^j) < H(\rho)$. Thus we apply the induction hypothesis on $H(\rho^j)$, we can get that there exists a path $(p_j,\theta_j)^i \xrightarrow{w_j w'}_T \{q_f\}$ in $\mathcal{A}_i$ and for every decomposition $(p_j,\theta_j)^i \xrightarrow{u}_T Q \xrightarrow{v}_T \{q_f\}$ of the path $(p_j,\theta_j)^i \xrightarrow{w_j w'}_T \{q_f\}$ where $uv = w_j w'$, if $Q \neq \{q_f\}$, then for all $(p',\theta')^k \in Q\backslash\{q_f\}$ with $k \in \{i, i-1\}$, some path of the run ρ^j will reach the configuration $(\langle p', v\rangle, \theta')$.

  Moreover, there exists a path $(p_j,\theta_j)^i \xrightarrow{w_j}_T Q_j \xrightarrow{w'}_T \{q_f\}$ in $\mathcal{A}_i$ and by applying the saturation procedure, we get that $(p,\theta)^i \xrightarrow{\gamma} Q_j \xrightarrow{w'}_T \{q_f\}$ in $\mathcal{A}_i$. Thus, for every decomposition $(p,\theta)^i \xrightarrow{u}_T Q \xrightarrow{v}_T \{q_f\}$ of the path $(p,\theta)^i \xrightarrow{\gamma w'}_T \{q_f\}$, if $Q \neq \{q_f\}$, then for all $(p',\theta')^k \in Q\backslash\{q_f\}$ with $k \in \{i, i-1\}$, some path of the run ρ will reach the configuration $(\langle p', v\rangle, \theta')$ i.e. $(\langle p, w\rangle, \theta) \Rightarrow^* (\langle p', v\rangle, \theta')$.

For the statement (b). Since $Y_{\mathcal{BP}} = Pre^+(Y_{\mathcal{BP}} \cap F \times \Gamma^* \times 2^{\Delta\cup\Delta_c})$ and by the induction hypothesis $Y_{\mathcal{BP}} \subseteq L(\mathcal{A}_{i-1})$, we get that

$$Y_{\mathcal{BP}} \subseteq Pre^+(L(\mathcal{A}_{i-1}) \cap F \times \Gamma^* \times 2^{\Delta\cup\Delta_c}) \quad (1)$$

By Lemma 11, we get that just before the substitution at Line 14, $\mathcal{A}_i$ accepts

$Pre^+(L(\mathcal{A}_{i-1}) \cap F \times \Gamma^* \times 2^{\Delta \cup \Delta_c})$. Thus, it is sufficient to prove that for every configuration $(\langle p, w\rangle, \theta) \in Y_{\mathcal{BP}}$, A_i accepts $(\langle p, w\rangle, \theta)$ after the substitution at Line 14. Let n be the number of transition rules substituted at Line 14. For all $m \leq n$, let $\mathcal{A}_i^m$ be the automaton obtained by substituting m transition rules. We show that $Y_{\mathcal{BP}} \subseteq L(\mathcal{A}_i^m)$ by induction on m.

- **Basis.** $m = 0$. We directly get that $Y_{\mathcal{BP}} \subseteq L(\mathcal{A}_i^0)$.

- **Step.** $m \geq 1$. By applying the induction hypothesis, we can get that $Y_{\mathcal{BP}} \subseteq L(\mathcal{A}_i^{m-1})$. If $L(\mathcal{A}_i^{m-1}) \subseteq L(\mathcal{A}_i^m)$, the result follows from the fact that $Y_{\mathcal{BP}} \subseteq L(\mathcal{A}_i^{m-1})$. Otherwise, if $L(\mathcal{A}_i^{m-1}) \backslash L(\mathcal{A}_i^m) \neq \emptyset$, let $(\langle p, w\rangle, \theta) \in L(\mathcal{A}_i^{m-1}) \backslash L(\mathcal{A}_i^m)$ be some configuration s.t. $|w|$ is the minimum of $\{|w'| \mid (\langle p', w'\rangle, \theta') \in L(\mathcal{A}_i^{m-1}) \backslash L(\mathcal{A}_i^m)\}$ s.t. $(\langle p, w\rangle, \theta) \in Y_{\mathcal{BP}}$. Then we prove by contradiction that $(\langle p, w\rangle, \theta)$ should not be in $Y_{\mathcal{BP}}$.

  For every path of the form $(p, \theta)^i \xrightarrow{w}_T \{q_f\}$ in $\mathcal{A}_i^{m-1}$, there exist $u \in \Gamma^+, v \in \Gamma^*$ and $q \in P$ such that $w = uv$ and $(p, \theta)^i \xrightarrow{u}_T Q \cup \{\ (q, \theta')^{i-1}\} \xrightarrow{v}_T \{q_f\}$ in $\mathcal{A}_i^{m-1}$ and $\mathcal{A}_i^m$ doesn't have $\{(q, \theta')^i\} \xrightarrow{v}_T \{q_f\}$. (Otherwise, $(\langle p, w\rangle, \theta) \in L(\mathcal{A}_i^m)$). By the statement(a), for each accepting run ρ of $\mathcal{BP}$ starting from $(\langle p, w\rangle, \theta)$, one path of this run ρ will reach such a configuration $(\langle q, v\rangle, \theta')$. It is sufficient to show that $(\langle q, v\rangle, \theta') \notin Y_{\mathcal{BP}}$.

  Now let us show that $(\langle q, v\rangle, \theta') \notin Y_{\mathcal{BP}}$. If $(\langle q, v\rangle, \theta') \notin L(\mathcal{A}_i^{m-1})$, by applying the induction hypothesis on m, we get that $(\langle q, v\rangle, \theta') \notin Y_{\mathcal{BP}}$. If $(\langle q, v\rangle, \theta') \in L(\mathcal{A}_i^{m-1})$, then $(\langle q, v\rangle, \theta') \in L(\mathcal{A}_i^{m-1}) \backslash L(\mathcal{A}_i^m)$. If $(\langle q, v\rangle, \theta') \in Y_{\mathcal{BP}}$, then $|v| < |w|$ which contradicts the fact that $|w|$ is the minimum of $\{|w'| \mid (\langle p', w'\rangle, \theta') \in L(\mathcal{A}_i^{m-1}) \backslash L(\mathcal{A}_i^m)\}$ s.t. $(\langle p, w\rangle, \theta) \in Y_{\mathcal{BP}}$. Thus, we can obtain that $(\langle q, v\rangle, \theta') \notin Y_{\mathcal{BP}}$.

□

Then, we can prove Theorem 4.2.2:

Proof: We prove termination and correctness.

Termination: There are two loops in **Algorithm 1**. Thus, we will prove those two loops both terminate.

Loop$_2$: Suppose *loop*$_2$ is in the ith iteration of *loop*$_1$. Since only states of the form $(p,\theta)^i \in P \times 2^{\Delta\cup\Delta_c} \times \{i\}$ can be added into $\mathcal{A}$ at the ith iteration, we obtain that *Loop*$_2$ only add a finite number of transition rules at the ith iteration. This implies that $\forall i \geq 1$, *loop*$_2$ always terminates at the i-th iteration.

Loop$_1$: Now we consider the termination of *Loop*$_1$. For every $i \geq 1$, Line 14 ensures that at the end of the ith iteration, every transition rule in the current automaton is in the form of $(p,\theta)^j \xrightarrow{\gamma} S$ for every $j \leq i$, $S \subseteq (P \times 2^{\Delta\cup\Delta_c} \times \{j\}) \cup \{q_f\}$. Thus, by the Lemma 9(a), at $(i+1)$th iteration with $i \geq 1$, either the termination condition at Line 15 is satisfied or the number of transitions is strictly smaller than in the ith iteration. Therefore, **Algorithm 1** terminates.

Correctness: Let $n > 1$ be the fix-point of **Algorithm 1** s.t. for every $p \in P, \gamma \in \Gamma, R \subseteq P \times 2^{\Delta\cup\Delta_c} \times \{n\} \cup \{q_f\}, (p,\theta)^n \xrightarrow{\gamma} R \in T \iff (p,\theta)^{n-1} \xrightarrow{\gamma} \pi^{-1}(R) \in T$ holds. Then $L(\mathcal{A}_n) = L(\mathcal{A}_{n-1})$. We will show that $L(\mathcal{A}_n) = Y_{\mathcal{BP}}$.

If we remove the termination condition of *loop*$_1$ i.e. if we consider **Algorithm C**, by Lemma 10 and Lemma 9(b) and the fact that $L(\mathcal{A}_0) = P \times \Gamma^* \times 2^{\Delta\cup\Delta_c}$, we have that for all $i \geq n$:

$$L(\mathcal{A}_i) = L(\mathcal{A}_{i-1}) =, \cdots, = L(\mathcal{A}_n) \subseteq L(\mathcal{A}_{n-1}) \subseteq L(\mathcal{A}_0). \tag{1}$$

- Let us first show that $L(\mathcal{A}_n) \subseteq Y_{\mathcal{BP}}$: Since $Y_{\mathcal{BP}} = \bigcap_{i\geq 0} X_i$ and $X_{i+1} = Pre^+\big(X_i \cap (F \times \Gamma^* \times 2^{\Delta\cup\Delta_c})\big)$, then it is sufficient to prove that $L(\mathcal{A}_i) \subseteq X_i$ for every $i \geq 0$. We proceed by induction on i.

  - **Basis.** $i = 0$. $L(\mathcal{A}_0) \subseteq X_0$ always holds.
  - **Step.** $i > 0$. By applying the induction hypothesis (induction on i), we get that $L(\mathcal{A}_{i-1}) \subseteq X_{i-1}$. By the definition of $X_i = Pre^+(X_{i-1} \cap (F \times \Gamma^* \times 2^{\Delta\cup\Delta_c}))$, we can have that

    $$Pre^+\big(L(\mathcal{A}_{i-1}) \cap (F \times \Gamma^* \times 2^{\Delta\cup\Delta_c})\big) \subseteq X_i \tag{2}$$

    By Lemma 11, $\forall i \geq 1$, $\mathcal{A}_i$ accepts $Pre^+\big(L(\mathcal{A}_{i-1}) \cap (F \times \Gamma^* \times 2^{\Delta\cup\Delta_c})\big)$ before Line 14 of the algorithm. By Lemma 9(b), Line 14 only removes configurations from $\mathcal{A}_i$ (Line 15 can only reduce the language of $\mathcal{A}_i$), we can obtain that:

    $$L(\mathcal{A}_i) \subseteq Pre^+\big(L(\mathcal{A}_{i-1}) \cap (F \times \Gamma^* \times 2^{\Delta\cup\Delta_c})\big) \tag{3}$$

    From (2) and (3), we can get that $L(\mathcal{A}_i) \subseteq X_i$.

- Now, we show that $Y_{\mathcal{BP}} \subseteq L(\mathcal{A}_n)$: It directly follows from Lemma 12(b).

Therefore, we show that $Y_{\mathcal{BP}} = L(\mathcal{A}_n)$.

□

Thus, it follows from Theorems 4.1.1, 4.2.1 and 4.2.2 that:

Corollary 1 *Let $\mathcal{P}$ be a SM-PDS, $\nu : P \to 2^{At}$ be a labelling function, and φ be a CTL formula over At. Then, we can compute an EAMA $\mathcal{A}$ that characterizes the set of configurations $(\langle p, w\rangle, \theta)$ of $\mathcal{P}$ such that $(\langle p, w\rangle, \theta) \models_\nu \varphi$.*

4.3 Experiments

4.3.1 Our algorithm vs. standard CTL on PDSs

We implemented our algorithm in a tool and we compared its performance with the approach that consists in translating the SM-PDS to an equivalent standard PDS, and then applying the standard CTL model checking algorithm implemented in the PDS model-checker tool PuMoC [40]. All our experiments were run on Ubuntu 16.04 with a 2.7 GHz CPU, 2GB of memory. To perform this comparision, we randomly generate several SM-PDSs and CTL formulas. Our results (CPU Execution time) are shown in Table 4.1. **Column** $|\Delta|+|\Delta_c|$ indicates the size of the transition rules. **Column** formula size shows the size of the CTL formula. **Column** SM-PDS is the cost of our direct algorithm. **Column** To PDS reports the cost it takes to get the equivalent PDS from the SM-PDS. **Column** PDS is the cost used to run standard CTL model checking for the equivalent PDS in PuMoC. **Column** Total Time is the whole cost it takes to translate the SM-PDS into a PDS, and then apply the PDS CTL model-checking algorithm of PuMoC [40] (Total Time= To PDS + PDS). **Column** *Result*1 is the result of our approach and *Result*2 is the result of PuMoC [40], where Y means yes the formula is satisfied and N means no, the formula is not satisfied. "-" means out of memory. It can be seen that our direct approach is much more efficient, and that it terminates in all the cases, whereas going through CTL model-checking of PDSs gets out of memory in most of the cases. **Translating the SM-PDS**

to a standard PDS may take more than 24 days, whereas our direct algorithm takes only a few seconds.

4.3.2 Malicious Behavior Detection on Self-Modifying Code

4.3.2.1 Specifying malicious behaviors using CTL.

We applied our tool to detect several self-modifying malwares. Indeed, as shown in [40], several malicious behaviors can be described by CTL formulas. We give in what follows an example of such a malicious behavior.

Spyware (Scanning the Disk). The aim of a spyware is to steal information from the host. To do this, it has to scan the disk of the host in order to find the interesting file that he wants to steal. If a file is found, it will run a payload to steal it, then continues searching the next file. If a directory is found, it will enter this path and continues scanning. This malicious behaviour is present e.g. in the notorious spyware Flame: It first calls the function FindFirstFileW to search the first object in the given path, then, it will check whether the function call succeeds or not. If the function call fails, it will call the function GetLastError. Otherwise it will call either the function FindFirstFileW again if it finds a directory or the function FindNextFileW to search for the next object. We can specify this behavior in CTL as follows:

$$\phi_{spy} = \mathbf{EF}\ \Big(call\ FindFirstFileW \wedge \mathbf{AF}\ \big(call\ GetLastError \vee call\ FindFirstFileW \vee call\ FindNextFileW \big) \Big)$$

This formula states that there exists a path where the function $FindFirstFileW$ is called, then, in all the future paths, the program either calls $GetLastError$ (if $FindFirstFileW$ failed) or calls $FindFirstFileW$ (if a directory is found) or calls $FindNextFileW$ (to search for the next file). Scanning a disk can be a behavior of a benign program. To avoid false alarms, we can combine this CTL formula with other formulas describing other malicious behaviors expressing the payload (such as sending a file) to determine whether the binary code is a malware or not. Note that, the formula is branching time and cannot be described as a LTL formula.

	$\|\Delta\| + \|\Delta_c\|$	formula	SM-PDS	To PDS	PDS	Total Time	$Result1$	$Result2$
CTL Model Checking	$5 + 2$	2	**0.27s**	0.09s	0.25s	0.34s	Y	Y
	$6 + 4$	5	**0.36s**	0.21s	0.45s	0.66s	Y	Y
	$8 + 4$	12	**2.88s**	0.35s	3.41s	3.76s	Y	Y
	$10 + 4$	18	**3.71s**	0.39s	3.85s	4.24s	Y	Y
	$20 + 4$	15	**3.84s**	0.62s	3.94s	4.56s	N	N
	$30 + 4$	8	**4.01s**	2.20s	4.79s	6.99s	Y	Y
	$35 + 4$	20	**5.13s**	2.36s	6.53s	8.89s	Y	Y
	$50 + 8$	6	**7.86s**	4.92s	8.04s	12.96s	N	N
	$80 + 8$	15	**8.46s**	5.06s	10.31s	15.37s	Y	Y
	$80 + 8$	20	**9.57s**	5.06s	10.79s	15.85s	Y	Y
	$110 + 8$	6	**8.83s**	5.25s	11.42s	16.64s	N	N
	$110 + 8$	15	**9.01s**	5.25s	12.98s	18.13s	N	N
	$110 + 8$	20	**10.24s**	5.25s	13.44s	18.69s	Y	Y
	$120 + 10$	10	**9.59s**	5.70s	12.32s	18.02s	N	N
	$120 + 10$	20	**11.48s**	5.70s	14.87s	20.57s	Y	Y
	$250 + 8$	6	**13.22s**	9.13s	18.94s	28.07s	N	N
	$250 + 8$	15	**18.37s**	9.13s	21.11s	30.24s	Y	Y
	$500 + 8$	6	**20.51s**	17.02s	29.25s	46.27s	N	N
	$600 + 9$	8	**23.34s**	295.24s	57.79s	353.03s	Y	Y
	$600 + 9$	15	**28.88s**	295.24s	63.16s	358.40s	Y	Y
	$600 + 9$	25	**35.39s**	295.24s	69.82s	365.06s	Y	Y
	$1000 + 10$	6	**35.11s**	3251.02s	7127.41s	10378.43s	N	N
	$1100 + 10$	8	**37.34s**	3251.02s	7319.82s	10570.84s	Y	Y
	$1100 + 10$	45	**83.63s**	3251.02s	-	-	N	-
	$1500 + 8$	30	**60.71s**	2182.78s	13821.34s	16004.12s	N	N
	$2000 + 10$	18	**49.48s**	5529.30s	-	-	Y	-
	$2000 + 10$	36	**61.13s**	5529.30s	-	-	N	-
	$2100 + 10$	15	**60.74s**	5544.69s	-	-	Y	-
	$2500 + 8$	30	**68.55s**	3981.93s	-	-	N	-
	$3000 + 7$	10	**65.84s**	5167.27s	-	-	Y	-
	$3000 + 7$	22	**78.51s**	5167.27s	-	-	N	-
	$3500 + 8$	6	**70.83s**	6105.60s	-	-	N	-
	$3500 + 10$	6	**75.91s**	9219.18s	-	-	N	-
	$3500 + 10$	20	**93.37s**	9219.18s	-	-	Y	-
	$3800 + 10$	30	**99.06s**	9295.24s	-	-	N	-
	$3850 + 10$	8	**93.20s**	9308.01s	-	-	Y	-
	$3850 + 10$	30	**115.52s**	9308.01s	-	-	N	-
	$4000 + 10$	20	**125.81s**	10002.28s	-	-	N	-
	$4200 + 8$	15	**121.16s**	9599.37s	-	-	Y	-
	$4500 + 8$	23	**136.72s**	9881.85s	-	-	Y	-
	$4500 + 11$	5	**139.95s**	40290.27s	-	-	Y	-
	$4800 + 11$	10	**142.13s**	42184.85s	-	-	Y	-
	$4800 + 11$	15	**153.22s**	42184.85s	-	-	Y	-
	$5500 + 10$	20	**196.46s**	45745.44s	-	-	Y	-

Table 4.1: Our approach vs. standard algorithms for PDSs for CTL model checking

4.3.2.2 Applying our tool for malware detection.

We applied our tool to detect several malwares. We consider 400 email-worms, 30 worms and 100 viruses from VX heaven [48] and 260 new malwares generated by NGVCK. We also choose 19 benign samples from Windows XP system (win32). We consider self-modifying versions of these programs. In these versions, the malicious behaviors are unreachable if the semantics of the self-modifying instructions are not taken into account, i.e., if the self-modifying instructions are considered as "standard" instructions that do not modify the code, then the malicious behaviors cannot be reached. As previously, first, we abstract away the semantics of the self-modifying instructions and model such programs as standard PDSs as described in [40], and perform CTL model-checking for PDSs to determine whether the programs contain any malicious behavior. In this case, *none* of the programs was declared as malicious. Then, we use SM-PDSs to model these programs, thus, taking self-modifying instructions into consideration. Then, we check whether these SM-PDSs satisfy any malicious CTL formula in our database. If yes, the program is declared as malicious. If not, it is declared as benign. In our experiments (we have 790 malwares), our tool was able to detect all these programs as malicious (whereas when we model these programs using standard PDSs and abstract away self-modifying instructions, none of these programs was detected as malicious). Our tool was also able to determine that benign programs are benign. We report in Tables 4.2, 4.3 and 4.4 the results we obtained. **Column** Size gives the number of control locations, **Column** Result shows the result of our algorithm: Yes means malicious and No means benign; and **Column** cost gives the cost in seconds. You can see that our CTL model checking approach allows to detect all the malicious programs in a few seconds.

Example	Size	Result	*cost*	Example	Size	Result	*cost*	Example	Size	Result	*cost*
calculation.exe	9952	No	76.34s	cisvc.exe	4105	No	31.22s	simple.exe	52	No	3.17s
shutdown.exe	2529	No	23.52s	loop.exe	529	No	11.78s	cmd.exe	1324	No	19.36s
notepad.exe	10529	No	68.77s	java.exe	800	No	19.17s	java.exe	21324	No	122.07s
sort.exe	8529	No	74.12s	bibDesk.exe	32800	No	243.79s	interface.exe	1005	No	18.25s
ipv4.exe	968	No	24.43s	TextWrangler.exe	14675	No	65.09s	sogou.exe	45219	No	301.14s
game.exe	34325	No	234.14s	cycle.tex	9014	No	75.44s	calender.exe	892	No	25.39s
Netsky.a	45	Yes	19.12s	Mydoom.c	155	Yes	4.14s	MyDoom-N	16980	Yes	343.93s
Netsky.x	55	Yes	21.85s	Netsky.y	68	Yes	29.06s	Netsky.z	115	Yes	43.37s
Netsky.gen	5508	Yes	59.24s	Netsky.p	6015	Yes	76.32s	Netsky.m	6805	Yes	73.77s
Netsky.r	230	Yes	8.83s	Netsky.k	6115	Yes	79.79s	Netsky.e	6245	Yes	79.44s
Mydoom.y	26902	Yes	452.77s	Mydoom.j	22355	Yes	211.93s	klez-N	6281	Yes	63.07s
klez.c	30	Yes	2.79s	Mydoom.v	5965	Yes	283.11s	Netsky.b	45	Yes	29.51s
Repah.b	221	Yes	12.76s	Gibe.b	5358	Yes	37.01s	Magistr.b	4670	Yes	43.59s
Netsky.d	45	Yes	1.87s	Ardurk.d	1913	Yes	12.08s	klez.f	27	Yes	0.73s
Kelino.l	495	Yes	21.01s	Kipis.t	20378	Yes	121.11s	klez.d	31	Yes	0.95s
Kelino.g	470	Yes	22.08s	Plage.b	395	Yes	1.96s	Urbe.a	123	Yes	9.17s
klez.e	27	Yes	3.94s	Magistr.b	4670	Yes	231.97s	Magistr.a.poly	36989	Yes	469.63s
Mydoom.M	5965	Yes	75.19s	MyDoom.54464	5935	Yes	45.78s	Mydoom.e	138	Yes	46.53s
Mydoom.R	230	Yes	30.22s	Mydoom.dlnpqi	235	Yes	1.99s	Mydoom.o	235	Yes	2.01s
Sramota.avf	240	Yes	11.01s	Mydoom	238	Yes	2.01s	Mydoom.288	248	Yes	3.12s
Mydoom.ACQ	19210	Yes	439.57s	Mydoom.ba	19423	Yes	238.77s	Mydoom.ftde	19495	Yes	339.29s
LdPinch.by	970	Yes	42.92s	Generic.2026199	433	Yes	32.83s	LdPinch.arr	1250	Yes	49.84s
Generic.12861	30183	Yes	188.94s	Generic.18017273	267	Yes	9.19s	LdPinch.mg	5957	Yes	69.77s
LDPinch.400	1783	Yes	54.93s	PSW.LdPinch.plt	1808	Yes	55.88s	PSW.Pinch.1	1905	Yes	57.07s
Newapt.F	11785	Yes	211.24s	Newapt.A	11715	Yes	205.79s	Newapt.E	11797	Yes	252.49s
LdPinch.bb	8145	Yes	63.13s	LdPinch.br	3645	Yes	33.52s	LdPinch.hb	1645	Yes	21.08s
LdPinch.v	7235	Yes	51.69s	LdPinch.fk	4906	Yes	47.11s	LdPinch.awp	195	Yes	17.97s
LdPinch.aaz	4145	Yes	41.05s	LdPinch.c0	8230	Yes	65.17s	LdPinch.ee	6501	Yes	71.30s
Bagle.m	5111	Yes	39.92s	Bagle.k	35	Yes	1.92s	Bagle.t	3345	Yes	45.64s
Newapt.C	11730	Yes	924.92s	Krynos.b	18370	Yes	893.45s	Jeans.a	6490	Yes	188.36s
Atak.f	2005	Yes	11.35s	Atak.g	2498	Yes	16.69s	Atak.l	1914	Yes	10.37s
Bagle.ab	5690	Yes	89.42s	Bagle.ef	995	Yes	54.11s	Bagle.eg	380	Yes	25.49s

Table 4.2: Experimental Results (part 1)

Example	Size	Result	*cost*	Example	Size	Result	*cost*	Example	Size	Result	*cost*
calculation.exe	9952	No	76.34s	cisvc.exe	4105	No	31.22s	simple.exe	52	No	3.17s
shutdown.exe	2529	No	23.52s	loop.exe	529	No	11.78s	cmd.exe	1324	No	19.36s
notepad.exe	10529	No	68.77s	java.exe	800	No	19.17s	java.exe	21324	No	122.07s
sort.exe	8529	No	74.12s	bibDesk.exe	32800	No	243.79s	interface.exe	1005	No	18.25s
ipv4.exe	968	No	24.43s	TextWrangler.exe	14675	No	65.09s	sogou.exe	45219	No	301.14s
game.exe	34325	No	234.14s	cycle.tex	9014	No	75.44s	calender.exe	892	No	25.39s
Adson.1651	39	Yes	0.44s	Adson.1734	42	Yes	0.43s	Alcaul.d	40	Yes	0.48s
Adon.1703	37	Yes	0.39s	Adon.1559	37	Yes	0.35s	Alcaul.i	48	Yes	0.44s
Alcaul.o	33	Yes	0.29s	Alcaul.d	845	Yes	0.165s	Alaul.c	355	Yes	0.109s
Alcaul.j	45	Yes	0.56s	Alcaul.m	23	Yes	0.19s	Evol.a	53	Yes	7.09s
Alcaul.e	32	Yes	1.93s	Alcaul.h	34	Yes	3.95s	Alcaul.g	25	Yes	4.18s
Alcaul.b	19	Yes	0.12s	Alcaul.f	23	Yes	1.99s	Alcaul.k	28	Yes	2.31s
Alcaul.l	27	Yes	0.95s	Klinge	45	Yes	64.15s	Akez.Win32.5	490	Yes	53.18s
Oroch.3982	31	Yes	1.49s	Anar.a	22	Yes	1.27s	Anar.b	25	Yes	1.58s
Bagle.dp	235	Yes	3.52s	Bagle.dv	175	Yes	6.14s	Bagle.ds	328	Yes	7.29s
Bagle.e	30	Yes	1.52s	Bagle.eb	185	Yes	3.87s	Bagle.ee	198	Yes	3.88s
Bagle.dn	198	Yes	6.07s	Bagle.ej	188	Yes	5.11s	Bagle.ff	355	Yes	8.07s
Bagle.ex	197	Yes	5.96s	Bagle.ev	183	Yes	6.50s	Bagle.en	192	Yes	9.99s
Predec.h	2650	Yes	58.34s	Predec.i	2855	Yes	63.58s	Predec.j	2835	Yes	62.37s
Predec.b	2830	Yes	61.77s	Predec.c	2858	Yes	64.02s	Predec.d	2826	Yes	61.11s
Predec.e	2850	Yes	67.97s	Predec.f	2895	Yes	69.50s	Predec.g	2829	Yes	66.59s
Haharin	355	Yes	13.52s	Ditex.a	25	Yes	1.46s	Oroch.5420	75	Yes	9.42s
Adson.1651	39	Yes	0.44s	Adson.1734	42	Yes	0.43s	Alcaul.d	40	Yes	0.48s
Adon.1703	37	Yes	0.39s	Adon.1559	37	Yes	0.35s	Alcaul.i	48	Yes	0.44s
Alcaul.o	33	Yes	0.29s	Alcaul.d	845	Yes	0.165s	Alaul.c	355	Yes	0.109s
Alcaul.j	45	Yes	0.56s	Alcaul.m	23	Yes	0.19s	Evol.a	53	Yes	7.09s
Alcaul.e	32	Yes	1.93s	Alcaul.h	34	Yes	3.95s	Alcaul.g	25	Yes	4.18s
Alcaul.b	19	Yes	0.12s	Alcaul.f	23	Yes	1.99s	Alcaul.k	28	Yes	2.31s
Alcaul.l	27	Yes	0.95s	Klinge	45	Yes	64.15s	Akez.Win32.5	490	Yes	53.18s
Oroch.3982	31	Yes	1.49s	Anar.a	22	Yes	1.27s	Anar.b	25	Yes	1.58s
Bagle.dp	235	Yes	3.52s	Bagle.dv	175	Yes	6.14s	Bagle.ds	328	Yes	7.29s
Bagle.e	30	Yes	1.52s	Bagle.eb	185	Yes	3.87s	Bagle.ee	198	Yes	3.88s
Bagle.dn	198	Yes	6.07s	Bagle.ej	188	Yes	5.11s	Bagle.ff	355	Yes	8.07s
Bagle.ex	197	Yes	5.96s	Bagle.ev	183	Yes	6.50s	Bagle.en	192	Yes	9.99s

Table 4.3: Experimental Results(part 2)

Example	Size	Result	*cost*	Example	Size	Result	*cost*	Example	Size	Result	*cost*
Netsky.a	45	Yes	19.12s	Mydoom.c	155	Yes	4.14s	MyDoom-N	16980	Yes	343.93s
Netsky.x	55	Yes	21.85s	Netsky.y	68	Yes	29.06s	Netsky.z	115	Yes	43.37s
Netsky.gen	5508	Yes	59.24s	Netsky.p	6015	Yes	76.32s	Netsky.m	6805	Yes	73.77s
Netsky.r	230	Yes	8.83s	Netsky.k	6115	Yes	79.79s	Netsky.e	6245	Yes	79.44s
Mydoom.y	26902	Yes	452.77s	Mydoom.j	22355	Yes	211.93s	klez-N	6281	Yes	63.07s
klez.c	30	Yes	2.79s	Mydoom.v	5965	Yes	283.11s	Netsky.b	45	Yes	29.51s
Repah.b	221	Yes	12.76s	Gibe.b	5358	Yes	37.01s	Magistr.b	4670	Yes	43.59s
Netsky.d	45	Yes	1.87s	Ardurk.d	1913	Yes	12.08s	klez.f	27	Yes	0.73s
Kelino.l	495	Yes	21.01s	Kipis.t	20378	Yes	121.11s	klez.d	31	Yes	0.95s
Kelino.g	470	Yes	22.08s	Plage.b	395	Yes	1.96s	Urbe.a	123	Yes	9.17s
klez.e	27	Yes	3.94s	Magistr.b	4670	Yes	231.97s	Magistr.a.poly	36989	Yes	469.63s
Mydoom.M	5965	Yes	75.19s	MyDoom.54464	5935	Yes	45.78s	Mydoom.e	138	Yes	46.53s
Mydoom.R	230	Yes	30.22s	Mydoom.dlnpqi	235	Yes	1.99s	Mydoom.o	235	Yes	2.01s
Sramota.avf	240	Yes	11.01s	Mydoom	238	Yes	2.01s	Mydoom.288	248	Yes	3.12s
Mydoom.ACQ	19210	Yes	439.57s	Mydoom.ba	19423	Yes	238.77s	Mydoom.ftde	19495	Yes	339.29s
LdPinch.by	970	Yes	42.92s	Generic.2026199	433	Yes	32.83s	LdPinch.arr	1250	Yes	49.84s
Generic.12861	30183	Yes	188.94s	Generic.18017273	267	Yes	9.19s	LdPinch.mg	5957	Yes	69.77s
LDPinch.400	1783	Yes	54.93s	PSW.LdPinch.plt	1808	Yes	55.88s	PSW.Pinch.1	1905	Yes	57.07s
Newapt.F	11785	Yes	211.24s	Newapt.A	11715	Yes	205.79s	Newapt.E	11797	Yes	252.49s
LdPinch.bb	8145	Yes	63.13s	LdPinch.br	3645	Yes	33.52s	LdPinch.hb	1645	Yes	21.08s
LdPinch.v	7235	Yes	51.69s	LdPinch.fk	4906	Yes	47.11s	LdPinch.awp	195	Yes	17.97s
LdPinch.aaz	4145	Yes	41.05s	LdPinch.c0	8230	Yes	65.17s	LdPinch.ee	6501	Yes	71.30s
Bagle.m	5111	Yes	39.92s	Bagle.k	35	Yes	1.92s	Bagle.t	3345	Yes	45.64s
Newapt.C	11730	Yes	924.92s	Krynos.b	18370	Yes	893.45s	Jeans.a	6490	Yes	188.36s
Atak.f	2005	Yes	11.35s	Atak.g	2498	Yes	16.69s	Atak.l	1914	Yes	10.37s
Bagle.ab	5690	Yes	89.42s	Bagle.ef	995	Yes	54.11s	Bagle.eg	380	Yes	25.49s
Predec.h	2650	Yes	58.34s	Predec.i	2855	Yes	63.58s	Predec.j	2835	Yes	62.37s
Predec.b	2830	Yes	61.77s	Predec.c	2858	Yes	64.02s	Predec.d	2826	Yes	61.11s
Predec.e	2850	Yes	67.97s	Predec.f	2895	Yes	69.50s	Predec.g	2829	Yes	66.59s
Haharin	355	Yes	13.52s	Ditex.a	25	Yes	1.46s	Oroch.5420	75	Yes	9.42s

Table 4.4: Experimental Results(part 3)

5

SMODIC: A Model Checker for Self Modifying Code

In this chapter, we present SMODIC, a model checker for self-modifying binary code that use self-modifying **mov** instructions. In SMODIC, such binary code is modeled using Self Modifying Pushdown Systems (SM-PDS). SMODIC takes as input either a self-modifying binary code or a self modifying pushdown system. It can then perform reachability analysis and LTL/CTL model-checking for these models. SMODIC first adapts the tool Jakstab [22] to get the Control Flow Graph from the binary code. Then, it translates this CFG into a SM-PDS. It then implements the algorithms presented in the previous chapters to perform reachability analysis and LTL/CTL model-checking for this model.

We successfully used SMODIC to model-check more than 900 self-modifying binary codes. In particular, we applied SMODIC for malware detection. In our experiments, SMODIC was able to detect 895 malwares and to prove that 19 benign programs were benign. SMODIC was also able to detect several malwares that well-known antiviruses such as Bit-Defender, Kinsoft, Avira, eScan, Kaspersky, Avast, and Symantec failed to detect. SMODIC can be found in

`https://lipn.univ-paris13.fr/~xin/smodic/index.html`.

5.1 Architecture

The Architecture of SMODIC is shown in Figure. 5.1. SMODIC takes as input either a binary program or a SM-PDS. SMODIC can perform both reachability analysis and LTL/CTL model checking.

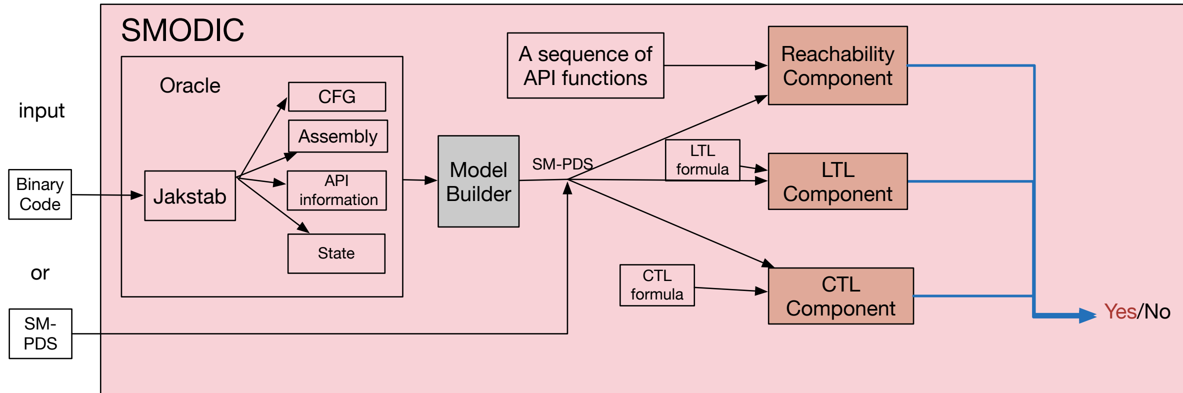

Figure 5.1: Architecture of SMODIC

If the input of SMODIC is a binary program, it is passed to the component **Oracle**. This component is based on the disassembler Jakstab [22]. It takes as input a binary program, and outputs its corresponding assembly program, its corresponding Control Flow Graph (CFG) equipped with the assembly instruction corresponding to each edge, together with informations about the called API functions, and the different values of the registers and memory addresses at each control point (state). All these outputs are fed to the component **Model Builder** that will compute the corresponding SM-PDS.

The **Reachability** component takes as input a SM-PDS, and a sequence of API functions, and applies the reachability algorithms of Chapter 2 to check whether the SM-PDS has a run that calls these API functions in this order. For example, if we consider the sequence f_1, f_2, f_3, then **Reachability** component checks whether the SM-PDS has a run that calls first f_1, then f_2, then f_3.

The **LTL** component takes as input a SM-PDS and an LTL formula, and applies the algorithms of Chapter 3 to check whether the SM-PDS satisfies the LTL formula. Similarly, the **CTL** component takes as input a SM-PDS and a CTL formula, and applies the algorithms of Chapter 4 to check whether the SM-PDS satisfies the CTL formula.

SMODIC	McAfee	Norman	BitDefender	Kinsoft	Avira	eScan	Kaspersky	Qihoo360	Avast	Symantec
100%	27.6%	22.1%	33.1%	14.4%	28.3%	21.4%	56.2 %	35.9%	50.7%	77.9%

Table 5.1: SMODIC vs. Well-known Anti-viruses

5.2 Experiments

5.2.1 Analysing Self-modifying Binary Code

We successfully applied SMODIC to perform reachability analysis and LTL/CTL model-checking for binary code and for Self-Modifying Pushdown systems. The results are summarized in Tables 2.1, 2.2, 3.1, 3.2 and 4.1. SMODIC was also able to detect 895 malwares and to prove that 19 benign programs are benign. The experimental results are summarized in Tables 3.5, 3.6, 3.7, 3.8, 4.2, 4.3 and 4.4.

5.2.2 Comparison with Well-known Anti-viruses

We also compare our tool against well-known and widely used antiviruses. In order to have a fair comparision, we need to consider new malwares, since the anti-viruses know the signatures of all the known malwares. Thus, the challenge for the anti-viruses is to detect *new* malwares. To this aim, we use the sophisticated malware generator NGVCK available at VX Heavens [48] to generate new malwares. Then we obfuscate these malwares with self-modifying code. Then, we feed these malwares to SMODIC and to well-known antiviruses such as BitDefender, Kinsoft, Avira, eScan, Kaspersky, Qihoo-360, Avast, and Symantec to detect them. Our tool was able to detect all these programs as malicious, whereas none of the well-known antiviruses was able to detect all these malwares. Table 5.1 reports the detection rates of our tool and the well-known anti-viruses.

5.3 Description of SMODIC

Let us show how to use SMODIC. The commands to launch the tool are as follows:

- SMODIC <option1> < modelfile> < option2 > <formula>

Option1 specifies the input file of SMODIC:

- M: the input is a SM-PDS model.
- B: the input is a binary program

Option2 specifies the model checking strategy:

- L: use the LTL model checking algorithm
- C: use the CTL model checking algorithm
- R1: perform the Reachability Analysis using pre^*
- R2: perform the Reachability Analysis using $post^*$

The model file can be either a binary program or a SM-PDS (.smpds file). The output have three files: one for the Control Flow Graph, one for assembly codes, and one for the generated SM-PDS. A SM-PDS consists of four parts: a finite set of standard PDS transition rules, a finite set of self-modifying transition rules, an initial phase (the initial set of transition rules) and an initial configuration (initial control location equipped with the stack contents).

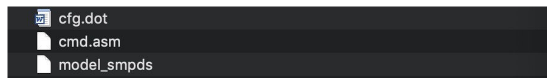

Figure 5.2: The Output of SMODIC

```
0x4ad0161a:     pushl   $0x4ad1f1b2     ; from: 0x4ad0504d(MAY),
0x4ad03fe4(MAY)
0x4ad0161f:     movl    %fs:0, %eax
0x4ad01625:     pushl   %eax
0x4ad01626:     movl    0x10(%esp), %eax
0x4ad0162a:     movl    %ebp, 0x10(%esp)
0x4ad0162e:     leal    0x10(%esp), %ebp
0x4ad01632:     subl    %eax, %esp
0x4ad01634:     pushl   %ebx
0x4ad01635:     pushl   %esi
0x4ad01636:     pushl   %edi
0x4ad01637:     movl    -8(%ebp), %eax
0x4ad0163a:     movl    %esp, -24(%ebp)
0x4ad0163d:     pushl   %eax
0x4ad0163e:     movl    -4(%ebp), %eax
0x4ad01641:     movl    $0xffffffff, -4(%ebp)
0x4ad01648:     movl    %eax, -8(%ebp)
0x4ad0164b:     leal    -16(%ebp), %eax
0x4ad0164e:     movl    %eax, %fs:0
0x4ad01654:     ret     ; targets: 0x4ad03fe9(MAY), 0x4ad05052(MAY)

0x4ad0165a:     movl    0x4ad33b90, %eax        ; from: 0x4ad03fef(MAY)
0x4ad0165f:     testl   %eax, %eax
0x4ad01661:     je      0x4ad04d48      ; targets: 0x4ad04d48(MAY),
0x4ad01667(MAY)
0x4ad01667:     pushl   $0x0    ; from: 0x4ad01661(MAY), 0x4ad04d75(MAY)
0x4ad01669:     call    %eax    ; targets: unresolved
0x4ad0166b:     ret     $0x4    ; targets: 0x4ad03ff4(MAY)      ; from:
0x4ad11d36(MAY)
```

Figure 5.3: A Segment of Disassembly Codes

In order to show this, we will use the following command to check whether the program cmd.exe can eventually call the API function `GetModuleA` or not. For this case, we execute the following command:

- SMODIC B malware/cmd.exe L <>getmodulea

Figure 5.4 is the snapshot of the command to start SMODIC. In this command, “B” is Option1 specifying that the input is a binary program. “L” specifies that the strategy of model checking is LTL. <> getmodulea is the LTL formula **F** (*call GetModuleA*).

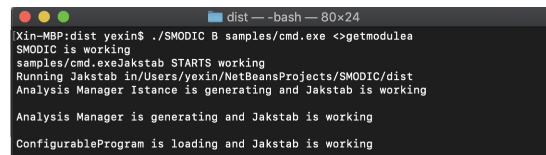

Figure 5.4: An Example to Run SMODIC

The output have three files: cfg.dot contains the control flow graph (Figures 5.5 and 5.6 are two segments of cfg.dot: the control locations corresponding to the instructions are given in Figure 5.5, and the edges between control locations are given in Figure 5.6), cmd.asm contains the assembly code equipped with informations about the API functions (Figure 5.3 is a fragment of this file), and model.smpds contains the SM-PDS (Figure 5.7 is a segment of the SM-PDS transition rules). This file contains in addition an initial configuration (the initial control point with the stack contents and the initial set of transition rules). The three files are shown in Fig. 5.2.

```
162  "0x4ad050cc"[label="0x4ad050cc\nmovl 0x4ad34874, %ecx"];
163  "0x4ad04e9a"[label="0x4ad04e9a\npushl $0x4ad04eb8"];
164  "0x4ad04e9a"[label="0x4ad04e9a\npushl $0x4ad04eb8"];
165  "0x4ad03fe4"[label="0x4ad03fe4\ncall 0x4ad0161a"];
166  "0x4ad01637"[label="0x4ad01637\nmovl -8(%ebp), %eax"];
167  "0x4ad0511e"[label="0x4ad0511e\npushl %eax"];
168  "0x4ad0511e"[label="0x4ad0511e\npushl %eax"];
169  "0x4ad04e89"[label="0x4ad04e89\nmovl 0x8(%ebp), %esi"];
170  "0x4ad050aa"[label="0x4ad050aa\norl $0xffffffff, 0x4ad2fa50"];
171  "0x4ad050aa"[label="0x4ad050aa\norl $0xffffffff, 0x4ad2fa50"];
172  "0x4ad03fe4"[label="0x4ad03fe4\ncall 0x4ad0161a"];
173  "0x4ad0514f"[label="0x4ad0514f\ncall 0x4ad03fdd"];
174  "0x4ad04d70"[label="0x4ad04d70\nmovl %eax, 0x4ad33b90"];
175  "0x4ad0510d"[label="0x4ad0510d\nleal -36(%ebp), %eax"];
176  "0x4ad0514f"[label="0x4ad0514f\ncall 0x4ad03fdd"];
177  "0x4ad01626"[label="0x4ad01626\nmovl 0x10(%esp), %eax"];
178  "0x4ad03ff4"[label="0x4ad03ff4\nmovl %ebx, -4(%ebp)"];
179  "0x4ad19049"[label="0x4ad19049\ncmpl $0xe, 0x84(%ecx)"];
180  "0x4ad0513e"[label="0x4ad0513e\nmovl __initenv@msvcrt.dll, %ecx"];
181  "0x4ad19049"[label="0x4ad19049\ncmpl $0xe, 0x84(%ecx)"];
182  "0x4ad01636"[label="0x4ad01636\npushl %edi"];
183  "0x4ad050db"[label="0x4ad050db\nmovl %eax, 0x4ad2fa54"];
184  "0x4ad01636"[label="0x4ad01636\npushl %edi"];
185  "0x4ad0506b"[label="0x4ad0506b\ncmpl $0x4550, (%ecx)"];
186  "0x4ad04e8c"[label="0x4ad04e8c\nandl $0x0, (%esi)"];
```

Figure 5.5: Control Locations with Instructions

5.3.1 Reachability Analysis in SMODIC

Let us show how to use SMODIC to perform reachability analysis on SMPDSs

```
316 "0x4ad05080" -> "0x4ad1903e" [color="#000000",label="T"];
317 "0x4ad0512c" -> "0x4ad05131" [color="#000000"];
318 "0x4ad05184" -> "0x4ad05189" [color="#000000"];
319 "0x4ad04d52" -> "0x4ad04d57" [color="#000000"];
320 "0x4ad04e93" -> "0x4ad04e98" [color="#000000"];
321 "0x4ad05071" -> "0x4ad05077" [color="#000000",label="F"];
322 "0x4ad0161a" -> "0x4ad0161f" [color="#000000"];
323 "0x4ad01661" -> "0x4ad01667" [color="#000000",label="F"];
324 "0x4ad05098" -> "0x4ad0509b" [color="#000000"];
325 "0x4ad11d2b" -> "0x4ad11d30" [color="#000000"];
326 "0x4ad04e87" -> "0x4ad04e88" [color="#000000"];
327 "0x4ad0514f" -> "0x4ad03fdd" [color="#000000"];
328 "0x4ad0505b" -> "0x4ad05060" [color="#000000"];
329 "0x4ad050a9" -> "0x4ad050aa" [color="#000000"];
330 "0x4ad03ffa" -> "0x4ad03ffb" [color="#000000"];
331 "0x4ad04d50" -> "0x4ad04d62" [color="#000000",label="T"];
332 "0x4ad05129" -> "0x4ad0512c" [color="#000000"];
333 "0x4ad19068" -> "0xff0002e0" [color="#000000"];
334 "0x4ad01626" -> "0x4ad0162a" [color="#000000"];
335 "0xff000670" -> "0x4ad04d5d" [color="#000000"];
336 "0x4ad03feb" -> "0x4ad03fee" [color="#000000"];
337 "0x4ad01641" -> "0x4ad01648" [color="#000000"];
338 "0xff000610" -> "0x4ad11d36" [color="#000000"];
339 "0x4ad0164b" -> "0x4ad0164e" [color="#000000"];
340 "0x4ad01667" -> "0x4ad01669" [color="#000000"];
341 "0x4ad03fe9" -> "0x4ad03feb" [color="#000000"];
342 "0x4ad0164e" -> "0x4ad01654" [color="#000000"];
```

Figure 5.6: A Segment of Edges between Locations

and self-modifying code. To start the reachability analysis, we need to specify the options. Let us consider Option1 B, and Option2 R2(or R1). We also need to specify the sequence of API functions. For example, to perform the reachability analysis on the sequence of API functions "Call GetModuleA", "Call CopyFile" , "call SendFile", we put the names of the functions in lowercase and use the symbol ";" to separate the names. To use the *post** approach to check whether the above sequence of API functions can be reached or not, we use the following command (see Fig. 5.8):

- ./SMODIC B malware/cmd.exe R2 getmodulea;copyfile;sendfile

The result is shown in Fig. 5.9.

We also can run reachability analysis on SM-PDSs. Then, we need to specify the options. We make Option1 M, and Option2 R2(or R1). We also need to specify the target configuration. For example, we can execute reachability analysis using the *post** approach to check if configuration $\langle p_0, r_0 \rangle$ can be reached or not by the following command:

- ./SMODIC M input.smpds R2 p0:r0

The output of SMODIC is the automaton representing the set of reachable configurations. SMODIC also tells whether the target configuration is reached or not.

5.3.2 LTL and CTL in SMODIC

First, we will introduce the syntax of LTL/CTL used in SMODIC. To be able

```
model_smpds
R03718:<p3,r22><p92,$0x4ad1f1b2r22>
R03717:<p3,r26><p92,$0x4ad1f1b2r26>
R03719:<p3,r25><p92,$0x4ad1f1b2r25>
R03710:<p3,r30><p92,$0x4ad1f1b2r30>
R03712:<p3,r11><p92,$0x4ad1f1b2r11>
R03711:<p3,r5><p92,$0x4ad1f1b2r5>
R03714:<p3,r40><p92,$0x4ad1f1b2r40>
R03713:<p3,r35><p92,$0x4ad1f1b2r35>
R03716:<p3,r18><p92,$0x4ad1f1b2r18>
R03715:<p3,r1><p92,$0x4ad1f1b2r1>
R03707:<p3,r17><p92,$0x4ad1f1b2r17>
R03706:<p3,r15><p92,$0x4ad1f1b2r15>
R03709:<p3,r3><p92,$0x4ad1f1b2r3>
R03708:<p3,r31><p92,$0x4ad1f1b2r31>
R03701:<p3,r2><p92,$0x4ad1f1b2r2>
R03700:<p3,r0><p92,$0x4ad1f1b2r0>
R03703:<p3,r34><p92,$0x4ad1f1b2r34>
R03702:<p3,r19><p92,$0x4ad1f1b2r19>
R03705:<p3,r39><p92,$0x4ad1f1b2r39>
R03704:<p3,r20><p92,$0x4ad1f1b2r20>
R02409:<p37,r38><p38,r38>
R02408:<p37,r16><p38,r16>
R03739:<p3,r13><p92,$0x4ad1f1b2r13>
R02401:<p147,r14><p140,r14>
R03732:<p3,r28><p92,$0x4ad1f1b2r28>
R02400:<p147,r4><p140,r4>
R03731:<p3,r36><p92,$0x4ad1f1b2r36>
R02403:<p147,r27><p140,r27>
R03734:<p3,r14><p92,$0x4ad1f1b2r14>
R02402:<p147,r7><p140,r7>
R03733:<p3,r4><p92,$0x4ad1f1b2r4>
R02405:<p147,r34><p140,r34>
R03736:<p3,r27><p92,$0x4ad1f1b2r27>
R02404:<p147,r8><p140,r8>
R03735:<p3,r7><p92,$0x4ad1f1b2r7>
R02407:<p147,r23><p140,r23>
R03738:<p3,r34><p92,$0x4ad1f1b2r34>
```

Figure 5.7: A Segment of SM-PDS Transition Rules

```
dist — -bash — 80×24
Xin-MBP:dist yexin$ ./SMODIC B samples/cmd.exe R2 getmodulea;copyfile;sendfile
SMODIC is working
samples/cmd.exeJakstab STARTS working
Running Jakstab in/Users/yexin/NetBeansProjects/SMODIC/dist
Analysis Manager Istance is generating and Jakstab is working

Analysis Manager is generating and Jakstab is working

ConfigurableProgram is loading and Jakstab is working

/Users/yexin/NetBeansProjects/SMODIC/dist/org/jakstab/analysis
```

Figure 5.8: An Example to Start SMODIC for Reachability Analysis

to use SMODIC, you need to be familiar with the syntax of the logics LTL and CTL. The implementation of LTL and CTL operators in SMODIC is as follows:

For LTL:

- Propositional Symbols: true, false and any lowercase string.

- Boolean operators: !(negation), -> (implication), <->(equivalence), && (and), || (or).

- Temporal operators: $[]p$(p always holds), $<> p$ (eventually p holds), pUq (p holds until q holds), and Xp (p holds next time).

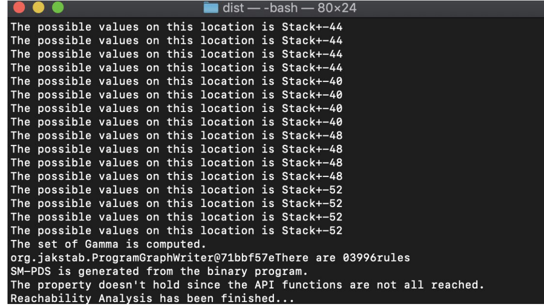

Figure 5.9: The Result of the Example for Reachability Analysis

For CTL:

- Propositional Symbols: tt(true),ff(false) and any lowercase string.
- Boolean operators: !(negation), -> (implication), <->(equivalence), && (and), || (or).
- Path quantifiers: A (for all paths) and E (there exists a path).
- Temporal operators: Xp (p holds next time), pRq (p holds until q does't hold), pUq (p holds until q holds).

5.4 Applying SMODIC for Malware Detection

We show how to apply LTL/CTL model checking to malware detection. Let us take a spy worm as example. Such a worm can record data and send it using the Socket API functions. For example, Keylogger is a spy worm that can record the keyboard states by calling the API functions GetAsyKeyState and GetKeyState and send this to the specific server by calling the socket function sendto. This behavior can be specified by the following LTL formula:

$\phi_{sw} = \mathbf{F}\big((call\ GetAsyncKeyState \vee call\ GetRawInputData) \wedge \mathbf{F}(call\ sendto \vee call\ send)\big)$.

To check whether the program cmd.exe satisfies this formula or not, first, we need to rewrite this formula to the form supported by our tool SMODIC. Because all the propositions are lowercase strings, we rewrite API function calls (like call GetAsyncKeyState) by removing the word "*call*" and changing the name of the

function in lowercase string. The operators are in spin syntax. Thus, formula ϕ_{sw} is rewritten as:

$$<>\big((getasynckeystate||getrawinputdata)\&\&<>(sendto||send)\big)$$

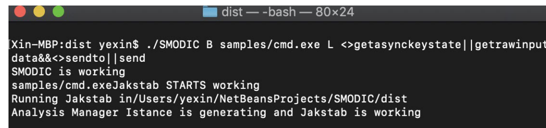

Figure 5.10: Command for LTL Model Checking

So, we can check whether the program cmd.exe satisfies this formula or not by using the following command (shown in Figure 5.10):

./SMODIC B malware/cmd.exe L $<>\big((getasynckeystate||$ $getrawinputdata)\&\&<>(sendto||send)\big)$.

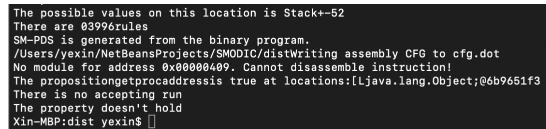

Figure 5.11: Result of LTL Model Checking

The result is shown in Figure 5.11. The result of the computation is that there is no accepting run. The output of the tool is No, i.e. **cmd.exe** is not a spyware.

6

Conclusion

6.1 Summary

In this thesis, we propose a new formal model for self-modifying code called Self Modifying Pushdown System(SM-PDS). It is an extension of standard Pushdown Systems (PDS) with self-modifying transition rules that modify the set of the rules of the PDS during the execution. This allows to represent the self-modifying instructions of the program. We also proposed several corresponding model-checking algorithms for this SM-PDS model and implemented them in a tool: SMODIC to perform the analysis of self-modifying code and malware detection.

Modeling Self-modifying Code: In Chapter 2, we introduce our new model: SM-PDS. This new model allows us to present the self-modifying code by self-modifying transition rules. A SM-PDS is a Pushdown System that can dynamically modify its set of rules during the execution time: rules that are not self-modifying rules are standard PDS transition rules, while self-modifying rules modify the current set of transition rules. We show how SM-PDSs can be used to naturally represent self-modifying programs. It turns out that SM-PDSs are equivalent to standard PDSs. We show how to translate a SM-PDS to a standard PDS. This translation is exponential. Thus, performing the model-checking analysis on the equivalent PDS is not efficient. We propose then in this thesis direct algorithms to perform reachability and LTL/CTL model checking on SM-PDSs.

Rechability Analysis of Self-Modifying Code: In Chapter 2, we propose *direct* algorithms to compute the forward ($post^*$) and backward (pre^*) reachabil-

ity sets for SM-PDSs. Our algorithms are based on representing regular sets of configurations of SM-PDSs using finite state automata, and applying saturation procedures on these automata.

LTL Model Checking of Self-modifying Code: In Chapter 3, we propose a **direct** LTL model checking algorithm for SM-PDSs. Our algorithm is based on reducing the LTL model checking problem to the emptiness problem of Self Modifying Büchi Pushdown Systems (SM-BPDSs). Intuitively, we obtain this SM-BPDS by taking the product of the SM-PDS with a Büchi automaton accepting an LTL formula φ. Then, we solve the emptiness problem of an SM-BPDS by computing its repeating heads. This computation is based on computing labelled pre^* configurations by applying a saturation procedure on labelled finite automata.

CTL Model Checking of Self-modifying Code: In Chapter 4, we consider the CTL model-checking problem for SM-PDSs. This allows to detect CTL-like malicious behaviors on self-modifying code. We reduce this problem to the emptiness checking of Self-modifying Alternating Büchi Pushdown Systems (SM-ABPDS), and we propose an algorithm that computes a finite automaton that characterizes the set of configurations accepted by the SM-ABPDS.

SMODIC: A Model Checker for Self-modifying Code: we implemented our techniques in a tool for self-modifying code analysis called SMODIC. We successfully used SMODIC to model-check more than 900 self-modifying binary codes. In particular, we applied SMODIC for malware detection, since malwares usually use self-modifying instructions, and since malicious behaviors can be described by LTL or CTL formulas. In our experiments, SMODIC was able to detect 895 malwares and to prove that 19 benign programs were benign. SMODIC was also able to detect several malwares that well-known antiviruses such as Bit-Defender, Kinsoft, Avira, eScan, Kaspersky, Avast, and Symantec failed to detect. SMODIC can be found in
`https://lipn.univ-paris13.fr/~xin/smodic/index.html`.

6.2 Future Work

The results presented in this thesis can be extended in several ways :

Precise Model for Self-modifying Code: As described in Section 2.3, during the construction of the SM-PDS, we need to assume that instructions i_1 and i_2 have the same number of operands where i_1 is replaced by i_2 because of some self-modifying instructions. If instructions i_1 and i_2 do not have the same number of operands, then the corresponding self-modifying instruction, in addition to replacing i_1 by i_2, changes several other instructions that follow i_1. As mentioned in Section 2.3, our translation from a self-modifying binary program to a SM-PDS can only handle the case where i_1 and i_2 have the same number of operands. In the future, we plan to improve our SM-PDS model so that it can handle the case where i_1 and i_2 do not have the same number of operands.

Precise Control Flow Reconstruction: In our implementation, the control flow reconstruction is not very precise. This step is based on the tool Jakstab [22] as disassembler. But Jakstab will sometimes ran into some situations where the value set analysis cannot be processed and the reconstruction of the control flow will stop. This holds, in many cases such as: (1) some values of the registers and possible values of memory addresses are unknown and are represented by any possible values (the $\top$ value); or (2) the destination of some indirect jumps cannot be computed. In the future, we plan to come up with new approaches to construct more precise control flow graphs from binary code to make the procedure of disassembling more precise.

Precise Malicious Behavior Description: In our experiments, we use standard LTL/CTL formulas to describe malicious behaviors. It was shown in [13, 14] that SCTPL and SLTPL are more precise and concise to represent malicious behaviors. SCTPL and SLTPL are logics that extend LTL and CTL with variables, quantifiers and predicates over the stack. In the future, we plan to propose SCTPL/SLTPL model checking algorithms for SM-PDS. This would allow to have more precise and concise algorithms for self-modifying code analysis and malware detection.

References

[1] A.Bertrand, M.Matias, and D.Koen. A model for self-modifying code. In International Workshop on Information Hiding, 2006.

[2] Gogul Balakrishnan, Radu Gruian, Thomas Reps, and Tim Teitelbaum. Codesurfer/x86-a platform for analyzing x86 executables*. In Compiler Construction: 14th International Conference, CC 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005. Proceedings, volume 3443, page 250. Springer, 2005.

[3] Sébastien Bardin, Philippe Herrmann, Jérôme Leroux, Olivier Ly, Renaud Tabary, and Aymeric Vincent. The bincoa framework for binary code analysis. In International Conference on Computer Aided Verification, pages 165–170. Springer, 2011.

[4] Jean Bergeron, Mourad Debbabi, Jules Desharnais, Mourad M Erhioui, Yvan Lavoie, Nadia Tawbi, et al. Static detection of malicious code in executable programs. Int. J. of Req. Eng, 2001(184-189):79, 2001.

[5] Guillaume Bonfante, Jose Fernandez, Jean-Yves Marion, Benjamin Rouxel, Fabrice Sabatier, and Aurélien Thierry. Codisasm: medium scale concatic disassembly of self-modifying binaries with overlapping instructions. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pages 745–756. ACM, 2015.

[6] A. Bouajjani, J. Esparza, and O. Maler. Reachability Analysis of Pushdown Automata: Application to Model Checking. In International Conference on Concurrency Theory, 1997.

[7] Laura Bozzelli. Complexity results on branching-time pushdown model checking. Theoretical computer science, 379(1-2):286–297, 2007.

[8] David Brumley, Cody Hartwig, Min Gyung Kang, Zhenkai Liang, James Newsome, Pongsin Poosankam, Dawn Song, and Heng Yin. Bitscope: Automatically dissecting malicious binaries. Technical report, Technical Report CS-07-133, School of Computer Science, Carnegie Mellon, 2007.

[9] David Brumley, Ivan Jager, Thanassis Avgerinos, and Edward J Schwartz. Bap: A binary analysis platform. In International Conference on Computer Aided Verification, pages 463–469. Springer, 2011.

[10] Danilo Bruschi, Lorenzo Martignoni, and Mattia Monga. Detecting self-mutating malware using control-flow graph matching. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pages 129–143. Springer, 2006.

[11] Robin David, Sébastien Bardin, Thanh Dinh Ta, Laurent Mounier, Josselin Feist, Marie-Laure Potet, and Jean-Yves Marion. Binsec/se: A dynamic symbolic execution toolkit for binary-level analysis. In IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), volume 1, pages 653–656. IEEE, 2016.

[12] Javier Esparza, Antonín Kučera, and Stefan Schwoon. Model-checking ltl with regular valuations for pushdown systems. In International Symposium on Theoretical Aspects of Computer Software, pages 316–339. Springer, 2001.

[13] F.Song and T.Touili. Efficient malware detection using model-checking. In International Symposium on Formal Methods, 2012.

[14] F.Song and T.Touili. Ltl model-checking for malware detection. In International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2013.

[15] G.Balakrishnan, T.W. Reps, N.Kidd, A.Lal, J.Lim, et al. Model checking x86 executables with codesurfer/x86 and WPDS++. In International Conference on Computer Aided Verification, 2005.

[16] G.Bonfante, J.Marion, and D.Reynaud-Plantey. A computability perspective on self-modifying programs. In 2009 Seventh IEEE International Conference on Software Engineering and Formal Methods, 2009.

[17] Nguyen Minh Hai, O Mizuhito, and Quan Thanh Tho. Pushdown model generation of malware. Technical report, Technical report, Japan Advanced Institute of Science and Technology, Japan, 2014.

[18] H.Cai, Z.Shao, and A.Vaynberg. Certified self-modifying code. ACM SIGPLAN Notices, 42(6), 2007.

[19] J.Bergeron], M.Debbabi, et al. Static detection of malicious code in executable programs. Int. J. of Req. Eng, 2001(184-189), 2001.

[20] J.Esparza, D.Hansel, P.Rossmanith, and S.Schwoon. Efficient algorithms for model checking pushdown systems. In International Conference on Computer Aided Verification, 2000.

[21] J.Kinder, S.Katzenbeisser, C.Schallhart, and H.Veith. Detecting malicious code by model checking. In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, 2005.

[22] H.Veith J.Kinder. Jakstab: A static analysis platform for binaries. In International Conference on Computer Aided Verification, 2008.

[23] K.Coogan, S.Debray, T.Kaochar, and G.Townsend. Automatic static unpacking of malware binaries. In 16th Working Conference on Reverse Engineering, 2009.

[24] K.Dam and T.Touili. Malware detection based on graph classification. In International Conference on Information Systems Security and Privacy, 2017.

[25] K.Dam and T.Touili. Learning malware using generalized graph kernels. In Proceedings of the 13th International Conference on Availability, Reliability and Security, 2018.

[26] K.Dam and T.Touili. Precise extraction of malicious behaviors. In IEEE 42nd Annual Computer Software and Applications Conference, 2018.

[27] K.Gyung et al. Renovo: A hidden code extractor for packed executables. In Proceedings of the 2007 ACM workshop on Recurring malcode, 2007.

[28] K.Roundy and B.Miller. Hybrid analysis and control of malware. In International Workshop on Recent Advances in Intrusion Detectio, 2010.

[29] M.Christodorescu, S.Jha, S.Seshia, D.Song, and R.Bryant. Semantics-aware malware detection. In Security and Privacy, 2005 IEEE Symposium on.

[30] M.Vardi and P.Wolper. Reasoning about infinite computations. Inf. Comput., 115(1), 1994.

[31] P.Gastin and D.Oddoux. Fast ltl to büchi automata translation. In International Conference on Computer Aided Verification, 2001.

[32] P.Royal, M.Halpin, et al. Polyunpack: Automating the hidden-code extraction of unpack-executing malware. In 22nd Annual Computer Security Applications Conference (ACSAC'06), 2006.

[33] P.Singh and A.Lakhotia. Static verification of worm and virus behavior in binary executables using model checking. In IEEE Systems, Man and Cybernetics SocietyInformation Assurance Workshop, 2003.

[34] S.Blazy, V.Laporte, and D.Pichardie. Verified abstract interpretation techniques for disassembling low-level self-modifying code. Journal of Automated Reasoning, 56(3), 2016.

[35] S.Cutler. malshare. `https://malshare.com`.

[36] S.Debray, K.Coogan, and G.Townsend. On the semantics of self-unpacking malware code. Tech. rep. University of Arizona, Computer Science, 2008.

[37] Axel Simon and Julian Kranz. The gdsl toolkit: Generating frontends for the analysis of machine code. In Proceedings of ACM SIGPLAN on Program Protection and Reverse Engineering Workshop 2014, page 7. ACM, 2014.

[38] Dawn Song, David Brumley, Heng Yin, Juan Caballero, Ivan Jager, Min Gyung Kang, Zhenkai Liang, James Newsome, Pongsin Poosankam,

and Prateek Saxena. Bitblaze: A new approach to computer security via binary analysis. In International Conference on Information Systems Security, pages 1–25. Springer, 2008.

[39] Fu Song and Tayssir Touili. Efficient CTL model-checking for pushdown systems. In CONCUR 2011 - Concurrency Theory - 22nd International Conference, CONCUR 2011, Aachen, Germany, September 6-9, 2011. Proceedings, pages 434–449, 2011.

[40] Fu Song and Tayssir Touili. Pumoc: a ctl model-checker for sequential programs. In 2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, pages 346–349, 2012.

[41] Fu Song and Tayssir Touili. Efficient ctl model-checking for pushdown systems. Theoretical Computer Science, 549:127–145, 2014.

[42] S.Schwoon. Model-checking pushdown systems. PhD thesis, Technische Universität München, Universitätsbibliothek, 2002.

[43] EnSilo Research Team. Self-modifying code unpacking tool using dynamorio. `https://github.com/BreakingMalware/Selfie`.

[44] Aditya Thakur, Junghee Lim, Akash Lal, Amanda Burton, Evan Driscoll, Matt Elder, Tycho Andersen, and Thomas Reps. Directed proof generation for machine code. In International Conference on Computer Aided Verification, pages 288–305. Springer, 2010.

[45] Unpacker Tool. Automated unpacking: A behaviour based approach. `https://github.com/malwaremusings/unpacker`.

[46] T.Touili and X.Ye. Reachability analysis of self modifying code. In 22nd International Conference on Engineering of Complex Computer Systems (ICECCS), 2017.

[47] T.Touili and X.Ye. Ltl model checking for self modifying code. In 24th International Conference on Engineering of Complex Computer Systems (ICECCS), 2019.

[48] V.Heaven. V.heavens. `http://vxer.org/lib/`.

[49] VirusShare. vxshare. https://virusshare.com.