SAT-based Model-Checking of Security Protocols

Luca Compagna

A dissertation submitted for the degree of
Doctor of Philosophy in
Electronics and Computer Science Engineering
under a joint programme between
Università degli Studi di Genova and the University of Edinburgh

Thesis Supervisors: Prof. Alessandro Armando[1]
Prof. Alan Smaill[2]

Committee:

Prof. David Aspinall	(University of Edinburgh)
Prof. Giorgio Delzanno	(Università degli Studi di Genova)
Prof. Paul Jackson	(University of Edinburgh)
Prof. Luca Viganò	(ETH Zürich)

(1) Dipartimento di Informatica, Sistemistica e Telematica,
Università degli Studi di Genova

(2) School of Informatics, University of Edinburgh

"The only system which is truly secure is one which is switched off and unplugged, locked in a titanium lined safe, buried in a concrete bunker, and is surrounded by nerve gas and very highly paid armed guards. Even then, I wouldn't stake my life on it."

Dr. Gene Spafford, Purdue University

Abstract

Security protocols are communication protocols that aim at providing security guarantees through the application of cryptographic primitives. Since these protocols are at the core of security-sensitive applications in a variety of domains (e.g. health-care, e-commerce, and e-government), their proper functioning is crucial as a failure may undermine the customer and, more in general, the public trust in these applications. In spite of their apparent simplicity, security protocols are notoriously error-prone. It is thus of utmost importance to have tools and specification languages that support the activity of finding flaws in protocols.

This thesis presents an high level protocol specification language accessible to protocol designers and easily translatable into a lower-level formalism for protocol insecurity problems. This thesis shows that automatic SAT-based model-checking techniques based on a reduction of protocol insecurity problems to propositional satisfiability problems (SAT) can be used effectively to find attacks on security protocols. The approach results from the combination of a reduction of protocol insecurity problems into planning problems and SAT-encoding techniques developed for planning. A model-checker, SATMC, based on these ideas has been developed and experimental results obtained by running SATMC against industrial-scale security protocols confirm the effectiveness of the approach.

Acknowledge

Firstly, I would like to thank my supervisors, Prof. Alessandro Armando and Prof. Alan Smaill for them patient and expert guidance throughout the course of the research described in this thesis.

Moreover, I would like to express sincere thanks to all the members of the AVISS and AVISPA projects for the endless, but truly interesting discussions about security protocols and related topics. You have been a remarkable source of inspiration for many ideas developed in this thesis. I am really glad to have collaborated with you all.

Finally I would like to thank my family, my girlfriend, and all my friends for their love and encouragement. Without their help and understanding, I certainly would not have been able to finish this thesis.

Preface

Declaration

I declare that this thesis was composed by myself, and that the work contained herein is my own except where explicitly stated otherwise in the text, and that this Thesis has been carried out in a joint supervision programme between the University of Genova and the University of Edinburgh as specified in the document "Agreement for the realization of a PhD in co-operation" attached. Besides this, I hereby declare that this work has not been submitted for any other degree or professional qualification except as specified.

(Luca Compagna)

Publications

Some of the work in this thesis has previously been published in [AC02, ABB+02, ACG03, AC04a, AC05, CCC+04, AC04b, ACL04, ABB+05].

For comments and suggestions, please email `compa@dist.unige.it`

Contents

1 Introduction **1**
- 1.1 Research area, motivations and objectives 1
- 1.2 An Overview of the SAT-based Model-Checking approach . . 7
- 1.3 Structure of the Thesis 8
- 1.4 Contribution of the Thesis 9

2 Formal Preliminaries **11**
- 2.1 (Multi)-Set Rewriting 11
- 2.2 Planning Problems 14
- 2.3 Propositional Satisfiability Problems (SAT) 19

3 Modelling Security Protocols **22**
- 3.1 Background 22
  - 3.1.1 Cryptography 23
  - 3.1.2 A simple example: NSPK 25
- 3.2 Protocol Insecurity Problems 27
  - 3.2.1 Honest agents 28
  - 3.2.2 Intruder 31
  - 3.2.3 Goals 33
  - 3.2.4 Optimisations 34
  - 3.2.5 A worked out example 38
- 3.3 (Un-)Decidability of Security Protocols 41
  - 3.3.1 Undecidability in the general case 41
  - 3.3.2 High-level Parameters of Security Protocols 50
  - 3.3.3 (Un-)Decidability over the Parameter Space 51
- 3.4 NPC Protocol Insecurity Problems 53

4 SAT-based Model-Checking of Security Protocols **55**
- 4.1 Protocol Insecurity Problems as Planning Problems 55
  - 4.1.1 Freshness 56
  - 4.1.2 Reduction to Planning 61
- 4.2 Planning Problems as SAT 61
  - 4.2.1 Linear Encodings 63

4.2.2 Graphplan-based Encodings 69
4.3 The overall Approach 82
4.3.1 Planning via SAT 85
4.3.2 Planning via SAT with Abstraction/Refinement . . . 90

5 Specification Languages for Security Protocols 93
5.1 The High Level Protocol Specification Language (HLPSL) . . 94
5.1.1 Preliminaries 96
5.1.2 The Language 97
5.1.3 The Semantics 111
5.2 The Intermediate Format (IF) 118
5.2.1 The Language 119
5.2.2 The Semantics 126
5.3 Experimental Modelling Results 129

6 Implementing a SAT-based Model-Checker: SATMC 132
6.1 The Architecture 133
6.2 The SATE Language 136
6.3 Running SATMC 138

7 Experimental Results 142
7.1 Comparison of the encodings in SATMC 142
7.1.1 Linear Encodings 143
7.1.2 Graphplan-Based Encodings 145
7.1.3 Linear versus Graphplan-Based Encodings 148
7.2 Comparison with other Automated Reasoning Tools 148
7.2.1 Planning Systems: blackbox 150
7.2.2 Logic Programming: CMODELS and SMODELS 152
7.2.3 AVISPA/AVISS back-ends: OFMC and CL-AtSe . . . 154

8 Related Work 162
8.1 Planning as Satisfiability. 162
8.2 Answer Set Programming. 164
8.3 Model Checking. 165
8.3.1 Bounded Model Checking. 167
8.3.2 Process Calculi. 167
8.3.3 Explicit State Model Checking. 168
8.4 The AVISPA approaches. 169
8.5 Strand Spaces approaches. 171
8.6 Protocol Specification Languages. 174

9 Conclusions and Perspectives **176**
9.1 Wrapping up 176
9.2 Future development(s) 177
9.2.1 Axioms 178
9.2.2 Abstraction for verification 179

A Reduction to planning: soundness and completeness **181**

B Experimental results: AVISPA details **183**

Bibliography **189**

List of Figures

1.1 Vulnerabilities reported by CERT between 1995 and 2003 . . 5

2.1 Propositional Logic: syntax 20
2.2 Propositional Logic: semantics 21

3.1 Caesar code 23
3.2 Encryption and Decryption, [CJ97] 23
3.3 BNF rules for facts 28
3.4 NSPK protocol insecurity problem: attack trace 40
3.5 NSPK protocol insecurity problem: honest agent rules 42
3.6 NSPK protocol insecurity problem: intruder rules 43
3.7 PCP as a Security Protocol 44
3.8 PCP as a PIP: honest agents rules 46
3.9 PCP as a PIP: concatenation rules 48

4.1 Forward search tree 73
4.2 Planning Graph 74
4.3 Approach 84
4.4 Planning via SAT 86
4.5 Planning via SAT: Init^{linear} 87
4.6 Planning via SAT: Extend^{linear} 87
4.7 Planning via SAT: Init$^{gp-\mathsf{schema}}$ 88
4.8 Planning via SAT: Extend$^{gp-\mathsf{schema}}$ 88
4.9 Planning via SAT with termination 90
4.10 Planning via SAT with the abstraction/refinement strategy . 92

5.1 HLPSL to IF framework 94
5.2 The NSPK-KS protocol 97
5.3 NSPK-KS protocol: HLPSL spec.(basic roles) 99
5.4 NSPK-KS protocol: hierarchy of roles 101
5.5 NSPK-KS protocol: HLPSL spec.(topmost-level role) 102
5.6 NSPK-KS protocol: HLPSL spec.(goals) 109
5.7 The Kerberos protocol 110
5.8 HLPSL roles: generic structure. 113
5.9 Dolev-Yao intruder knowledge formulae 116

5.10 The semantics of safety temporal formulae 117
5.11 Prelude file 120
5.12 NSPK-KS protocol: IF spec.(signature, types, and inits) . . . 121
5.13 NSPK-KS protocol: IF spec.(rules) 122
5.14 NSPK-KS protocol: IF spec.(goals) 123
5.15 Applicability of a IF rewrite rule 128
5.16 Semantics of a IF rewrite rule 129

6.1 Architecture of SATMC 133
6.2 Grammar for the SATE specification language 137
6.3 Output generated by SATMC 140
6.4 A screen-shot of SATMC within the AVISPA Tool 141

7.1 Comparison between graphplan-based and linear encodings on the number of atoms and clauses 149
7.2 Comparison between graphplan-based and linear encodings on the encoding time 150
7.3 AVISPA/AVISS Architecture 155
7.4 Comparing SATMC versions 158

8.1 NSPK protocol as parametric strands 172

List of Tables

3.1 Facts and their informal meaning 38
3.2 (Un-)Decidability Overview 52

5.1 HLPSL coverage 131

7.1 Experimental data on protocols from the Clark/Jacob library using Linear encodings 144
7.2 Experimental data on protocols from the Clark/Jacob library using Graphplan-based encodings 146
7.3 Experiments on the Clark/Jacob library: SATMC versus black-box 151
7.4 Experiments on the Clark/Jacob library: SAT versus Logic Programming 153
7.5 Experiments on the Clark/Jacob library: AVISS back-ends . 157
7.6 Experiments on the AVISPA library 159
7.7 Experiments on the NSPK-KS with increasing number of sessions 161

B.1 Experiments on the AVISPA library 185

Chapter 1

Introduction

1.1 Research area, motivations and objectives

Research area

This Thesis is about the application of *automated reasoning techniques* to formal analysis of *security protocols*. More in particular, it explores the novel combination of various techniques borrowed from and devised for *(bounded) model checking*, *AI planning*, and *propositional satisfiability* with the ultimate goal of analysing in a formal and automatic way industrial-scale security protocols.

Propositional Satisfiability (SAT). Boolean satisfiability (SAT) is the problem of deciding if there is an assignment for the variables in a propositional formula that makes the formula true. SAT is of considerable practical interest as many decision problems, such as graph colouring problems, planning problems, and scheduling problems can be encoded into SAT. As a matter of fact, unless P = NP, SAT is intractable in the worst case, as it is a NP hard problem. Namely it is the first known NP-complete problem, as proved by Stephen Cook in 1971 (see Cook's theorem for the proof, [Coo71]). However there is consolidated empirical evidence showing that many SAT instances are characterised by structural properties permitting efficient solution (e.g. see [PCK99]). Needless to say, SAT is such a fundamental problem in logic and computer science that it received a lot of attention and many different algorithms and techniques have been devised to try and solve it efficiently (see, for a good survey [GPFW97] and for more recent developments [GMW00]). The history of SAT solvers can be divided into three main periods strictly related with the creation and evolution of computers. Until 1960, SAT solving algorithms were just theoretically described, but with no real interest as they could not be used in practice. In 1960, Davis and Putnam published an algorithm (denoted by DP) which inaugu-

rated a new era.[1] DP is very inefficient at finding a satisfying assignment when one exists. This fact, and the fact that DP's memory consumption is explosive in practice, motivated the subsequent development of the Davis-Logemann-Loveland (DLL) algorithm [DLL62]. The algorithm has been used extensively to solve many kinds of problems using computers (especially from artificial intelligence and operating research). In the 90's, the interest in SAT solvers increased significantly. Computers of the 90's were able to handle large SAT instances (thousands of variables), which widened the scope of SAT solver applications to include circuit design, scheduling, and model checking. As a consequence, some researchers started studying how to improve SAT solvers in practice. Benchmarks and worldwide competition has been established and the quest for the fastest ever solver was born (see [SAT] to read the latest news on the Propositional Satisfiability domain).

AI Planning. Planning is the area of Artificial Intelligence (AI) that aims to find a sequence of actions to apply to an input problem, in order to achieve a set of predefined objectives, starting from a given initial state. Artificial Intelligence is concerned with the representation and the use of knowledge. For planning, this means developing representations that facilitate reasoning about actions and goals, and developing techniques for efficiently manipulating the entities in the representation. Planning has been a challenge area of research in AI from the early 1960s. The conceptualisation of a changing world as an alternating sequence of stable states and instantaneous actions was a natural extension of the symbolic logic representations that dominate AI, and most planning approaches have thus been built on state-based representations. The most influential planning representation has been the STRIPS formalism [FN71], because of its simplicity and efficient treatment of the frame problem [MH69]. Within this formalism, classical planning is correct and complete. During the last decade a number of researchers tried to develop algorithms and systems that combine both classical planning techniques and consolidated techniques from other areas of Artificial Intelligence. One of the most promising combination exploits recent improvements in the performances of propositional satisfiability methods. The basic idea is to compile bounded-length planning problems to SAT [KMS96, EMW97] and to use state-of-the-art SAT solvers to efficiently discover solutions. Planning techniques have been applied in numerous areas including robotics, space exploration, transportation logistics, marketing and finance, assembling parts, crisis management.

Model Checking. Model checking [CGP99] is a powerful and automatic technique for verifying finite state concurrent systems. Introduced in early

[1] The same algorithm was previously designed by Löwenheim around 1910 [HIR68].

1980s, it has been applied widely and successfully in practice to verify digital sequential circuit designs and communication protocols. However, the *state explosion problem*, wherein the number of system states grows exponentially with the number of system components, generally limited the application of model checking to designs with less than one million states (e.g. hardware circuit designs with at most 20 latches). During the 1990s, it has been devised and improved a *symbolic model checking* technique based on the idea to symbolically explore the search space through the use of *Binary Decision Diagrams* (BDDs) whose allow computation of transitions among sets of states rather than individual states [McM93]. Although the application of BDD-based symbolic model checkers allowed designers to verify systems with 10^{20} states, it remains the case that model checkers lack a certain robustness, in that they cannot consistently handle designs of typical industrial size. In late 1990s, it has been proposed a new type of model checking technique called *bounded model checking* [BCCZ99] that by exploiting the dramatic speed-up of propositional solvers it is able to analyse designs with hundreds of thousands of latches. Bounded model checking has been proved to be particularly suited in finding counter-examples, i.e. to return paths through the transition system that violate one of the specified system requirements.

Motivations

Hardware and software are widely used in critical systems (e.g. ecommerce, communication networks, medical systems, air traffic) where a *failure* is unacceptable because its consequences can be costly (e.g. in 1994, the Pentium FDIV bug costed $475 millions to INTEL; in 1996, the explosion of Ariane 5 due to a floating point overflow has been estimated in a lost of $500 millions) and even more serious when they endanger people's safety (e.g. between 1985 and 1987, the medical machine for radiotherapy Therac-25 distributed wrong amounts of radiations to several patients because of a software bugs and two people died; in 1991 during Operation Desert Storm, a failure in the software controlling a Defence Patriot Missile resulted in 28 deaths). The huge number of reported system failures shows that conventional validation techniques based on informal arguments and/or testing are not up to the task. It is now widely recognised that only verification tools based on formal methods can provide the level of assurance required.

One of the area that has received growing attention by the formal methods community as a new, very promising and challenging application domain is that of *security protocols*. In fact, in a world strongly dependent on distributed data communication, the design of secure IT infrastructures like the Internet is a crucial task. The acceptance and continued expansion of these infrastructures depends on trust: all participants must have confidence in their security, which is integrated into the infrastructure either by means of specific security protocols or protocol families (cf. protocols like

IPSec, IKE, TLS, and SSH) aimed at satisfying general security requirements, or by augmenting protocols and applications with security sensitive parts (cf. Geopriv, SIP, etc.), i.e. security specific functional elements that satisfy application-oriented security requirements. With the spread of the Internet and network based services, and the development of new technological possibilities, the number and scale of new protocols under development is out-pacing our ability to rigorously analyse and validate them. This is an increasingly serious problem for standardisation organisations like the *Internet Engineering Task Force* (IETF), the *International Telecommunication Union* (ITU), and the *World Wide Web Consortium* (W3C) as well as for companies whose products and/or services depend on the rapid standardisation and correct functioning of these protocols. It is also a problem for users of these technologies whose rights and freedoms, e.g. the right to privacy of personal data, depend on a secure infrastructure.

Designing secure protocols is a very challenging problem. In open networks, such as the Internet, protocols should work even under worst-case assumptions, namely messages may be eavesdropped or tampered with by an *intruder* (also called the attacker or spy) or dishonest or careless principals (where we call *principals* the agents participating in a protocol execution). Surprisingly, severe attacks can be conducted even without breaking cryptography, but by exploiting weaknesses in the protocols themselves, for instance by carrying out *man-in-the-middle attacks*, where an attacker plays off one protocol participant against another, or *replay attacks*, where messages from one session (i.e. execution of an instance of the protocol) are used in another session. The history of security protocols is full of examples, where weaknesses of supposedly correct published protocols that have been implemented and deployed in real applications only to be found flawed years later. A well-known case is the Needham-Schroeder authentication protocol [NS78] that was found vulnerable to a serious attack 17 years after its publication [Low96]. An up to date and interesting amount of statistics about security violations can be found in the yearly "Computer Crime and Security Survey" [CSI04] conducted by the Computer Security Institute (San Francisco, U.S.)[2] with the participation of the San Francisco Federal Bureau of Investigation's Computer Intrusion Squad. Here we sketch a summary of the 2004 survey results that are based on the responses of 494 computer security practitioners in U.S. corporations, government agencies, financial institutions, medical institutions and Universities:

- ⋆ even if the percentage of organisations that experienced an unauthorised use of their computer systems within the last 12 months is decreasing since 2000 (mainly due to the growing use of security countermeasures technologies), such a number is still quite high (53%) as well as the dollar amount of annual financial losses resulting from security

[2]Computer Security Institute home page: `http://www.gocsi.com`

breaches ($142 millions);

- ⋆ within the 53% of respondents answering that there was unauthorised use of their organisation's computer systems in the last 12 months:
  - the 47% experienced from 1 to 5 incidents, 20% from 6 to 10, 12% experienced more than 10 security violations, and the rest (22%) do not know;
  - only the 36% reported such intrusions to public opinion in order to avoid negative publicity and further economic losses [KCZ03].

Other significant data can be found in [CER03] published by CERT/CC (CERT/Coordination Center)[3] within the Software Engineering Institute of the Carnegie Mellon University (Pittsburgh, USA). Figure 1.1 is borrowed from such a report and shows the number of vulnerabilities reported to CERT/CC between 1995 and 2003. It is immediate to see that the number of

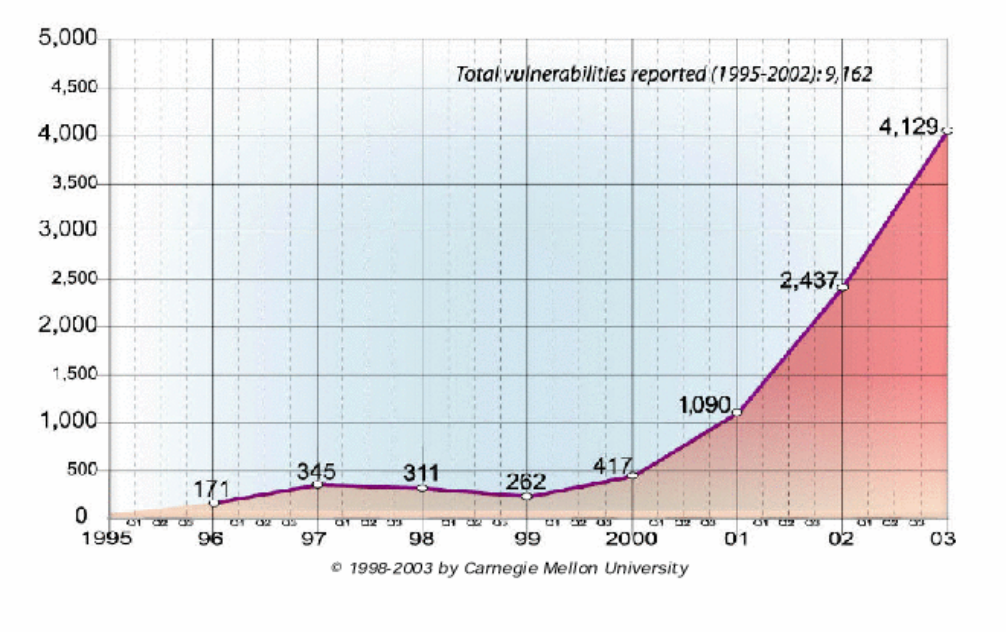

Figure 1.1: Vulnerabilities reported by CERT between 1995 and 2003

security vulnerabilities is highly growing up and it is estimated in more than 11 vulnerabilities for day (according to [Sym04] seven new vulnerabilities a day were announced in 2003).

The possibility of these violations and attacks sometimes stems from subtle misconceptions in the design of the protocols. Typically, these attacks

[3]CERT/CC home page: `http://www.cert.org`

are simply overlooked, as it is difficult for humans, even by careful inspection of simple protocols, to determine all the complex ways that different protocol sessions could be interleaved together, with possible interference of a malicious intruder.

To speed up the development of the next generation of network protocols and to improve their security and ultimately the acceptance of products based on them, it is of utmost importance to have *tools* and *specification languages* that support the activity of finding flaws in protocols or of establishing their absence. Optimally, these tools should be robust, expressive, and easily usable, so that they can be integrated into the protocol development and the standardisation process to improve the speed and quality of that process.

A variety of languages and tools (e.g. [ABB+02, BAN90, DM00, DNL99, Mea96, MD03, Pau98, SBP01]) based on different automated reasoning techniques have been devised and applied to automate the analysis of security protocols. Urged by the success of the SAT technology in domains like hardware verification, we have investigated in this thesis if similar results can be obtained by applying SAT-based model-checking to security protocols. Besides this practical reason, an important theoretical reason candidates SAT as a possible successful approach for security protocol analysis. In fact, although the general verification problem—to prove whether a security protocol satisfies its security guarantees or not—is undecidable [DLMS99, EG83], by focusing on verification of a bounded number of sessions the problem is known to be in the same complexity class of SAT i.e. NP-complete (see [RT01]). The encoding into SAT can be done in a variety of ways. However the specific nature of security protocols that make them particularly suited to be represented through rule-based formalisms [CDL+99] suggests that SAT-reduction techniques like those devised for AI planning may be particularly successful in such a context.

Objectives

Driven by the request for assistance arisen from standardisation committees for Internet protocols that mainly ask for tools and specification languages to support their activities, the objectives of this thesis have been three-fold:

1. to contribute to design a framework suited for the easy specification of industrial-scale security protocols;

2. to develop an automatic tool for security protocol analysis. More specifically, levered by the advancements of the SAT technology this thesis aims to devise a SAT-based semi-decision procedure for model checking of security protocols; and

3. to assess the effectiveness of the SAT-based Model Checking approach

against a large collection of practical relevant, industrial security protocols and therefore to determine whether this approach usually applied to finite-state transition systems and control-intensive application-domains, scales to the data and control-intensive domain of security protocols.

1.2 An Overview of the SAT-based Model-Checking approach

This thesis presents an automatic reduction of the question "Does the protocol satisfy its security requirements?" into propositional logic (SAT) thus exploiting the high performance of state-of-the-art SAT solvers for discovering attacks on protocols.

The above question is formalised into a security problem $\Xi = \langle \Gamma, \mathsf{B} \rangle$ where Γ is a state transition system representing the concurrent execution of honest agents that exchange the messages of the protocol over an insecure network controlled by the very general Dolev and Yao intruder [DY83] and B is a set of states whose reachability corresponds to a violation in the requirements of the protocol. In tackling this problem we specify Γ in a declarative language based on multiset rewriting[4] and we make the standard assumptions of perfect cryptography[5] and strong typing.[6]

The basic idea behind our SAT-based model-checking approach is that given an integer k and a security problem Ξ, it is possible to generate a propositional formula Φ_k^{Ξ} whose models correspond to attacks on the security problem. Intuitively the formula represents all the possible evolutions of the transition system described by Ξ up to depth k. Therefore the unsatisfiability of the formula, in general, does not permit to conclude that the modelled protocol satisfies its security requirements, but only that the security protocol does not suffer from any attack of length up to k. In order to provide a (semi-decision) procedure able to discover any attack on security protocols, it suffices to iterate the procedure for increasing values of k. This results in executing bounded model-checking [BCCZ99] of security protocols by means of a reduction to propositional formulae with iterative deepening on the number of steps.

Last but not least it is worth pointing out that our automatic reduction from security problems to SAT results from the combination of a translation

[4] Multiset rewriting is an action-based formalism particularly amenable to formal analysis of security protocols [CDL+99, JRV00].

[5] The perfect cryptography assumption states that an encrypted message can be decrypted only by using the appropriate decryption key.

[6] The strong typing assumption imposes that agents only react to a message if, and only if, this message has the type and the pattern expected by the receiver (i.e. type confusion is not allowed). As pointed out in [HLS00], a wide variety of type-flaw attacks can be prevented by tagging the fields of a message with information indicating its intended type.

of security problems into planning problems and SAT-encoding techniques developed for planning. This provides a significant benefit to our approach where we may exploit a decade-long research area very prolific in devising sophisticated AI planning SAT-encoding techniques.

1.3 Structure of the Thesis

This thesis is divided in the following main chapters:

Formal Preliminaries where we set the basic formal concepts of general-purpose decision problems useful in the context of this thesis. In this chapter are defined the concepts of multiset rewriting, planning, and propositional satisfiability.

Modelling Security Protocols where we define what is a security protocol and how we model the question of deciding whether the protocol enjoys its security requirements or not as a protocol insecurity problem.

SAT-based Model-Checking of Security Protocols we provide a variety of techniques for reducing protocol insecurity problems to propositional satisfiability problems and we describe in details the overall approach.

Specification Languages for Security Protocols where we present a framework for the specification of industrial-scale security protocols. Namely, a protocol and its security requirements are specified in an high-level language accessible to a human user and they are then automatically translated into a lower-level language particularly amenable to automated formal analysis.

Implementing a SAT-based Model-Checker: SATMC where we describe the implementation of an open and flexible platform for SAT-based bounded model checking of security protocols based on our ideas.

Experimental Results where we provide a large number of experimental results for comparing *(i)* the encodings in SATMC and *(ii)* our SAT-based approach with respect to others state-of-the-art automated reasoning techniques.

Related Work where we survey the most relevant works in the area of security protocols and we discuss the relationships between our SAT-based Model-Checking approach and closely related general-purpose automated reasoning techniques.

Conclusions and Perspectives where we summarise the conclusions of this thesis and future directions that may be worth exploring.

We end up this thesis with two appendices:

Appendix A where we prove that the reduction to planning presented in this Thesis is sound and complete.

Appendix B where we present in details the results obtained by running SATMC against the security problems in the AVISPA library.

1.4 Contribution of the Thesis

This thesis contributes to the research area of security protocol analysis by providing added values to the following points:

Specification Languages for Security Protocols. We have contributed to the design of a framework for the easy specification of security protocols (see Chapter 5).[7]. In such a framework security protocols are specified in the High-Level Protocol Specification Language (HLPSL) that is designed to be accessible to human users and that is easily translatable into a lower-level term-rewriting based language, called Intermediate Format (IF), well-suited to model-checking tools.

Automatic Tools for Security Protocol Analyses. We have devised an approach for analysing security protocols (see Section 4.3) that results from the combination of a reduction of security problems to planning problems (see Section 4.1) and SAT-encoding techniques developed for planning (see Section 4.2). In doing this we have investigated interesting aspects of cross-fertilisation between planning and model-checking. In fact, while the application of model-checking techniques to planning—usually referred as "Planning as Model-Checking", [FP99, AFEP97]—have been explored quite widely, the opposite direction, i.e. "Model-Checking as Planning", has not received the same attention. The SAT-based approach described in this thesis applies planning SAT-reduction techniques for bounded model checking [BCCZ99] of security protocols and thus contributes to determine whether model-checking problems may be successfully tackled through AI planning techniques.

Scaling to Industrial-Scale Security Protocols. We have developed SATMC, a SAT-based Model-Checker, based on our ideas (see Chapter 6).

[7]It must be noted that this framework is one of the result of a joint work with other researchers from various institutions—AI-Lab group at DIST of University of Genova (Genova, Italy), the CASSIS group at INRIA (Nancy, France), the Institut fur Informatik Albert-Ludwigs-Universitat (Freiburg, Germany), the Information Security Group at ETHZ (Zurich, Switzerland), and Siemens AG (Munich, Germany)—carried out in the context of the EU projects AVISS [AVI01] and AVISPA [AVI02a].

SATMC is an open and flexible platform for bounded model-checking of security protocols that can readily exploit both improvements of SAT technology and advancements in AI planning SAT-reduction. State-of-the-art taken off-the-shelf SAT-solvers (e.g. the best performers in the SAT competition, [SAT]) can be directly integrated in SATMC in a plug and play manner. Similarly, new encoding techniques for reducing planning problems into propositional satisfiability problems can be easily imported into our tool. We have thoroughly assessed (see Chapter 7) our approach by running SATMC against security protocols of growing complexity, from those in the Clark-Jacob's library [CJ97] to those, practically relevant in industrial environments, of the AVISPA library (see Chapter 5). The results obtained confirm the effectiveness of the approach and show that SAT-based Model-Checking technologies usually applied to finite-state transition systems and control-intensive application-domains, can be successfully applied also on data and control-intensive domains like those of security protocols. To the best of our knowledge no other tools, except those integrated in the AVISPA tool [AVI02a], exhibit the same level of scope and robustness while enjoying the same performance and scalability.

Chapter 2

Formal Preliminaries

The purpose of this chapter is to set the basic formal concepts of general-purpose decision problems this Thesis focuses on. The chapter consists of three sections: in the first one the concepts about (multi)-set rewriting systems and reachability problems for (multi)-set rewriting are described; in Section 2.2 the planning formalism and some related concepts are presented; finally, the third section describes well-known concepts of propositional logic.[1]

2.1 (Multi)-Set Rewriting

The theory and applications of rewriting logic have been vigorously developed by researchers all over the world during the past years. In its broad meaning, a *term rewriting system* provides a general framework for specifying and reasoning about computation.[2] It consists of transformation rules of the form $t \to t'$ (called *rewrite rules*), working over terms t and t' built out of variable and function symbols. Each rewrite rule can be interpreted as a local transition in a concurrent system. Namely, the rule explains how a local concurrent transition can take place in the system, changing the local state fragment from an instance of the term t to the corresponding instance of the term t' (see for instance Example 1).

Example 1. *Suppose we want to model a simple clock by means of a term rewriting system. In doing this we desire each transition of the rewriting*

[1] Readers familiar with some of the topics presented in this Chapter are invited to skip the correspondent section. This argument definitely applies to the last section on propositional logic. However, we would suggest to all the readers to give at least a quick look to the section on planning where some important definitions, concepts, and theoretical results are presented.

[2] In the sequel we will use the concepts of *signature*, *function*, *predicate*, *constant*, *variable*, *atom*, *(ground) term*, *(ground) atomic formula*, *(ground) substitution*, and *unification* as commonly defined in first order logic, see for instance first two chapters of [VG91].

system corresponds to a tick of the clock. This can be easily modelled through the rewrite rule $x \rightarrow s(x)$ where x is a variable ranging on the domain $\{0, s(0), s(s(0)), \ldots\}$ and s is the unary successor function. Assume the initial state of the system is represented by means of the atomic formula $time(0)$, then it is immediate to see that the only evolution of the system is given by this sequence of states: $time(0)$, $time(s(0))$, $time(s(s(0)))$, etc.

In the context of this Thesis, we focus on a specific class of rewriting systems in which the rules do not rewrite terms, but sets of atomic formulae called *facts*. Moreover, in the spirit of [CDL+99], we augment rewrite rules with existential quantification to provide a natural way of choosing new values. Intuitively, existential quantification introduces an amount of *non-determinism* particularly suited to model the generation of pseudo-random numbers that is at the core of cryptographic applications and that in reality depends on unpredictable events (e.g. mouse movements, key strokes, and similar user events). Besides this, in order to emulate the practically very low probability of generating more than one time the same pseudo-random number, we restrict our model to those non-deterministic traces in which any existential variable is always replaced by a *new* pseudo-random number.

Formally, a *(multi)-set rewriting system* is a tuple $\Gamma = \langle \mathsf{F}, \mathsf{L}, \mathsf{I}, \mathsf{R} \rangle$ where F and L are disjoint sets of ground atomic formulae of a (multi-sorted) first-order language, called *facts* and *rule labels* respectively, I is a finite subset of 2^{F} representing the initial states[3], and R is a set of labelled rewrite rules of the form[4]

$$L \xrightarrow{\ell} \exists \mathbf{y} : R$$

where $\exists \mathbf{y}$ denotes that the variables $\mathbf{y}$ are to be instantiated with terms that ought to be fresh, i.e. new values that have not been previously encountered, even if the rule is used repeatedly. Let σ be a ground substitution on $\mathbf{x}$ and $\mathbf{y}$, the above rewrite rule is such that $L\sigma$ and $R\sigma$ are finite subsets of F, and $\ell\sigma \in \mathsf{L}$ is a rule label. Moreover, we require that *(i)* the non-existential variables of the right-hand-side must be a subset of the variables in the left-hand-side, and *(ii)* if

$$\begin{aligned} &\big(L_1 \xrightarrow{\ell(\mathbf{x_1})} \exists \mathbf{y_1} : R_1\big) \in \mathsf{R}, \text{ and} \\ &\big(L_2 \xrightarrow{\ell(\mathbf{x_2})} \exists \mathbf{y_2} : R_2\big) \in \mathsf{R} \end{aligned}$$

are such that $\ell(\mathbf{x_1})\sigma_1 = \ell(\mathbf{x_2})\sigma_2$, then $L_1\sigma_1 = L_2\sigma_2$ and $R_1\sigma_1 = R_2\sigma_2$ (where σ_1 and σ_2 are two ground substitutions). This last requirement ensures that each rule label is associated with at most one ground rewrite rule.

[3] If S is a set, then 2^S denotes the powerset of S.

[4] For the sake of simplicity in the sequel we will use L, R, and ℓ to denote $L(\mathbf{x})$, $R(\mathbf{x}, \mathbf{y})$, and $\ell(\mathbf{x})$ respectively. Similarly, let i an integer, then L_i, R_i, and ℓ_i will indicate $L_i(\mathbf{x_i})$, $R_i(\mathbf{x_i}, \mathbf{y_i})$, and $\ell_i(\mathbf{x_i})$ respectively.

However, it does not prevent the non-determinism introduced by the existential quantification. In fact a rule label ℓ does not contain any reference to the existential variables occurring in the rule and therefore the application of ℓ can lead to infinitely-many states where existential variables are instantiated with different fresh terms.

Facts are used to represent the states of the transition system associated with the (multi)-set rewriting system. In particular, a state is modelled by the set of facts true in that state. More precisely, if F is a set of facts then the state represented by F is such that all the facts in F are true in it and all the facts in $\mathsf{F} \setminus F$ are false in it (closed-world assumption).

If $(L \xrightarrow{\ell} \exists \mathbf{y} : R) \in \mathsf{R}$, then we define $\mathsf{lhs}(\ell) = L$, $R(\ell) = R$, and $\mathsf{ex}(\ell) = \mathbf{y}$ to be the left-hand-side, the right-hand-side and the existential variables respectively of the rule. Moreover, if S is a set of facts, then we define $\mathsf{v}(S)$ and $\mathsf{sub}(S)$ to be the set of variables in S and the set of all the subterms occurring in each fact of S respectively.[5] (For instance, if $S = \{q(a), p(x, f(c))\}$, then $\mathsf{v}(S) = \{x\}$ and $\mathsf{sub}(S) = \{a, x, f(c), c\}$).

Let σ be a ground substitution on $\mathbf{x}$ and $\mathbf{y}$, then if $S \subseteq \mathsf{F}$ is such that $\mathsf{lhs}(\ell)\sigma \subseteq S$, then we say that the rewrite rule $\ell\sigma$ *is applicable in* S and that $S' = \mathrm{app}^{\rightarrow}_{\ell\sigma}(S) = (S \setminus \mathsf{lhs}(\ell)\sigma) \cup \mathsf{rhs}(\ell)\sigma$ *is the state resulting from the execution of* $\ell\sigma$ *in* S. A *rewriting path* is an alternating sequence of states and rule labels $S_0\ell_1\sigma_1 S_1 \ldots S_{n-1}\ell_n\sigma_n S_n$ such that for $i = 0, \ldots, n-1$

- σ_i is a ground substitution,
- $S_{i+1} = \mathrm{app}^{\rightarrow}_{\ell_{i+1}\sigma_{i+1}}(S_i)$, and
- $\big(\mathsf{ex}(\ell_{i+1}\sigma_{i+1}) \cap \mathsf{sub}(S_0 \cup \bigcup_{j=1}^{i}(\mathsf{lhs}(\ell_i)\sigma_i \cup \mathsf{rhs}(\ell_i)\sigma_i))\big) = \emptyset$.[6]

If $S_0\ell_1\sigma_1 S_1 \ldots S_{n-1}\ell_n\sigma_n S_n$ is a rewriting path, then we say that S_n *is reachable in* n *steps from* S_0 *by applying in sequence the ground rules* $\ell_1\sigma_1, \ldots, \ell_n\sigma_n$. This is denoted by $S_n = \mathrm{app}^{\rightarrow}_{\ell_1\sigma_1;\ldots;\ell_n\sigma_n}(S_0)$.

A nice feature of (multi)-set rewriting systems is that they can be easily glued together thereby facilitating the writing of modular specifications. Let $\Gamma_1 = \langle \mathsf{F}_1, \mathsf{L}_1, \mathsf{I}_1, \mathsf{R}_1 \rangle$ and $\Gamma_2 = \langle \mathsf{F}_2, \mathsf{L}_2, \mathsf{I}_2, \mathsf{R}_2 \rangle$ be (multi)-set rewriting systems with $\mathsf{L}_1 \cap \mathsf{L}_2 = \emptyset$.[7] The *composition of* Γ_1 *and* Γ_2, in symbols $\Gamma_1 \circ \Gamma_2$, is the (multi)-set rewriting system $\Gamma = \langle \mathsf{F}_1 \cup \mathsf{F}_2, \mathsf{L}_1 \cup \mathsf{L}_2, \mathsf{I}, \mathsf{R}_1 \cup \mathsf{R}_2 \rangle$ with $\mathsf{I} = \{S_1 \cup S_2 : S_1 \in \mathsf{I}_1, S_2 \in \mathsf{I}_2\}$.

Let $\Gamma = \langle \mathsf{F}, \mathsf{L}, \mathsf{I}, \mathsf{R} \rangle$ be a (multi)-set rewriting system. A *reachability problem for* Γ is a pair $\Xi = \langle \Gamma, \mathsf{B} \rangle$ where B is a finite subset of 2^{F} representing

[5] This definitions can be easily adapted to the case of terms.

[6] Notice that this condition ensures that, at each rule application, the existential variables have been instantiated with fresh terms.

[7] If L_1 and L_2 have some element in common, then it is possible to rename these elements and change the rewrite rules accordingly so to ensure that the two sets of rule labels are disjoint.

the goal (bad) states. A *solution to* $\Xi = \langle \Gamma, \mathsf{B} \rangle$ is a run $\ell_1\sigma_1, \ldots, \ell_n\sigma_n$ of Γ such that $\overrightarrow{\mathrm{app}}_{\ell_1\sigma_1;\ldots;\ell_n\sigma_n}(S) \in \mathsf{B}$ for some $S \in \mathsf{I}$.[8]

2.2 Planning Problems

Planning is a fundamental problem for many real world application domains including robotics, process planning, web-based information gathering, autonomous agents, and spacecraft mission control. Planning is the process by which we determine actions to achieve our goals (e.g. getting to the supermarket to buy a banana for lunch, finishing all our homework assignments and projects and turning them in by their deadlines, building an airplane).

Formally, a *planning system* is a tuple $\Theta = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{O} \rangle$, where $\mathcal{F}$ and $\mathcal{A}$ are disjoint sets of variable-free atomic formulae of a sorted first-order language called *fluents* and *actions* respectively, $\mathcal{I}$ is a finite subset of $2^{\mathcal{F}}$ representing the initial states, and $\mathcal{O}$ is a set of operators of the form

$$Pre \overset{Act}{\Rightarrow} Add \,;\, Del$$

where $Act \in \mathcal{A}$ and Pre, Add, and Del are finite sets of fluents such that $Add \cap Del = \emptyset$.[9] Moreover we require that if $Pre_1 \overset{a}{\Rightarrow} Add_1 \,;\, Del_1$ and $Pre_2 \overset{a}{\Rightarrow} Add_2 \,;\, Del_2$, then $Pre_1 = Pre_2$, $Add_1 = Add_2$, and $Del_1 = Del_2$. This requirement ensures that each action label is associated with at most one planning operator.

Fluents are used to represent the states of the transition system associated with the planning system. In particular, a state is modelled by the set of fluents true in that state. More precisely, if F is a set of fluents then the state represented by F is such that all the fluents in F are true in it and all the fluents in $\mathcal{F} \setminus F$ are false in it (closed-world assumption).

If $Pre \overset{a}{\Rightarrow} Add \,;\, Del$ then we define $\mathrm{pre}(a) = Pre$, $\mathrm{add}(a) = Add$, and $\mathrm{del}(a) = Del$. Moreover, if $S \subseteq \mathcal{F}$ is such that $\mathrm{pre}(a) \subseteq S$, then we say that the *action a is applicable in S* and that $S' = \mathrm{app}^{\Rightarrow}_a(S) = (S \setminus \mathrm{del}(a)) \cup \mathrm{add}(a)$ *is the state resulting from the application of a in S*. A *planning path* (*path* in short) is an alternating sequence of states and actions $S_0 a_1 S_1 \ldots S_{n-1} a_n S_n$, compactly denoted by $\pi_{a_1,\ldots,a_n}(S_0)$, such that $S_{i+1} = \mathrm{app}^{\Rightarrow}_{a_{i+1}}(S_i)$ for $i = 0, \ldots, n-1$. If $S_0 a_1 S_1 \ldots S_{n-1} a_n S_n$ is a planning path, then we say that S_n *is reachable in n steps from S_0* by applying in sequence $a_1, \ldots, a_n$; this is denoted by $S_n = \mathrm{app}^{\Rightarrow}_{a_1;\ldots;a_n}(S_0)$.

Let $\Theta = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{O} \rangle$ be a planning system. A *planning problem for* Θ is a pair $\Pi = \langle \Theta, \mathcal{G} \rangle$ where $\mathcal{G}$ is a finite subset of $2^{\mathcal{F}}$ representing the goal states. A *solution to* $\Pi = \langle \Theta, \mathcal{G} \rangle$, called *plan*, is a sequence of actions whose

[8] It must be noted that when exists a goal state that corresponds to one of the initial states, then Ξ allows for a trivial solution that we indicates by means of the empty sequence of rules ϵ.

[9] Notice that the triple $\langle \mathcal{F}, \mathcal{A}, \mathcal{O} \rangle$ is also referred to as the *planning domain* [Neb00].

execution leads from the initial state to a final state and the preconditions of each action appears in the state to which it applies. Formally, $a_1, \ldots, a_n$ is a plan of length n of Π if and only if $\mathrm{app}^{\Rightarrow}_{a_1;\ldots;a_n}(S) \in \mathcal{G}$ for some $S \in \mathcal{I}$.[10]

Definition 1 (Parallel Composition). *Let $a_i \in \mathcal{A}$ for $i = 1, \ldots, n$. We say that $a_1, \ldots, a_n$ are composable if and only if $\bigcup_{i=1}^{n} \mathrm{add}(a_i) \cap \bigcup_{i=1}^{n} \mathrm{del}(a_i) = \emptyset$, i.e. the effects of the actions in $a_1, \ldots, a_n$ do not contradict each other. If $a_1, \ldots, a_n$ are composable actions, then the (parallel) composition of $a_1, \ldots, a_n$, in symbols $a_1 \| \cdots \| a_n$, is the action a associated with the planning operator $\bigcup_{i=1}^{n} \mathrm{pre}(a_i) \overset{a}{\Rightarrow} \bigcup_{i=1}^{n} \mathrm{add}(a_i)\ ; \bigcup_{i=1}^{n} \mathrm{del}(a_i)$.*[11]

If we extend the definition of planning problem so to consider also the application of the actions obtained from $\mathcal{O}$ by parallel composition, then we may get shorter solutions. Formally, let $\Pi = \langle\langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{O}\rangle, \mathcal{G}\rangle$, we define $\hat{\Pi} = \langle\langle \mathcal{F}, \hat{\mathcal{A}}, \mathcal{I}, \hat{\mathcal{O}}\rangle, \mathcal{G}\rangle$ where $\hat{\mathcal{O}}$ and $\hat{\mathcal{A}}$ are obtained from $\mathcal{O}$ and $\mathcal{A}$ by adding all the parallel composition of actions, i.e. $\hat{\mathcal{O}} = \{a_1 \| \cdots \| a_n : \{a_1, \ldots, a_n\} \subseteq \mathcal{O}\}$, and $\hat{\mathcal{A}} = \{a : \{a_1, \ldots, a_n\} \subseteq \mathcal{A}, a_1 \| \cdots \| a_n, \text{and } a \text{ is a unique identifier for } a_1 \| \cdots \| a_n\}$. It is easy to see that $\hat{\Pi}$ may have shorter solutions than Π. To illustrate, consider Example 2.

Example 2. *Given the planning problem $\Pi = \langle\langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{O}\rangle, \mathcal{G}\rangle$ where:*

$$\begin{aligned}
\mathcal{F} &= \{a_1, \ldots, a_n, b_1, \ldots, b_n\} \\
\mathcal{A} &= \{l_1, \ldots, l_n\} \\
\mathcal{I} &= \{\{a_1, \ldots, a_n\}\} \\
\mathcal{O} &= \{a_i \overset{l_1}{\Rightarrow} b_i\ ; \neg a_i : i = 1, \ldots, n\} \\
\mathcal{G} &= \{\{b_1, \ldots, b_n\}\}
\end{aligned}$$

The only solutions to Π are all the permutations of $l_1, \ldots, l_n$, whereas $l_1 \| \cdots \| l_n$ is a solution to $\hat{\Pi}$ of length 1 concisely representing the effect of applying the actions $l_1, \ldots, l_n$ in any order. In the real world, each a_i and b_i could be two possible configurations of the i-th object of the system and l_i a switch that, if pressed, allows the i-th object to move from a_i to b_i. By means of parallel composition we can model the situation in which all the n switches are activated simultaneously.

However a solution to $\hat{\Pi}$ does not necessarily correspond to a solution to Π (see for instance Example 3).

[10] It must be noted that when there exists a goal state that corresponds to one of the initial states, then Π allows for a trivial solution that we indicates by means of the empty sequence of actions ϵ.

[11] If Λ is a set of composable actions, then $\|(\Lambda)$ denotes the parallel composition of the actions in Λ.

Example 3. *For instance, the planning problem* $\Pi = \langle\langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{O}\rangle, \mathcal{G}\rangle$ *with*

$$\begin{array}{lcl} \mathcal{F} & = & \{a, b, c\} \\ \mathcal{A} & = & \{l_1, l_2\} \\ \mathcal{I} & = & \{\{a\}\} \\ \mathcal{O} & = & \{(a \overset{l_1}{\Rightarrow} b\ ;\neg a),\ (a \overset{l_2}{\Rightarrow} c\ ;\neg a)\} \\ \mathcal{G} & = & \{\{b, c\}\} \end{array}$$

has no solution, whereas $\hat{\Pi} = \langle\langle \mathcal{F}, \hat{\mathcal{A}}, \mathcal{I}, \hat{\mathcal{O}}\rangle, \mathcal{G}\rangle$ *is such that* $(a \overset{l_1 \| l_2}{\Rightarrow} b, c\ ;\neg a) \in \hat{\mathcal{O}}$ *and therefore* $l_1 \| l_2$ *is a solution to* $\hat{\Pi}$.

We are thus interested in only those solutions to $\hat{\Pi}$ that correspond to solutions to Π.

Definition 2 (Linearization). *Let* $a_1, \ldots, a_n$ *be composable actions. A linearization of* $a_1 \| \cdots \| a_n$ *is a permutation* $a = a_{k_1}, \ldots, a_{k_n}$ *of* $a_1, \ldots, a_n$ *such that for all* $S_0 \supseteq \bigcup_{i=1}^{n} \mathrm{pre}(a_i)$ *there exists a planning path* $\pi_{a_1,\ldots,a_n}(S_0)$. *We say that* $a_1 \| \cdots \| a_n$ *is linearizable if and only if there exists a linearization of* $a_1 \| \cdots \| a_n$.

Lemma 1. *Let* $a_1, \ldots, a_n$ *be composable actions, let* $a_{k_1}, \ldots, a_{k_n}$ *be a linearization of* $a_1 \| \cdots \| a_n$, *and let* $S_0 \supseteq \bigcup_{i=1}^{n} \mathrm{pre}(a_i)$, *then* $\mathrm{app}^{\Rightarrow}_{a_{k_1};\cdots;a_{k_n}}(S_0) = \mathrm{app}^{\Rightarrow}_{a_1\|\cdots\|a_n}(S_0)$.

Proof. The proof is by induction on n. Since the base case is trivial, here we focus on the step case. Then

$$\begin{array}{ll} & \mathrm{app}^{\Rightarrow}_{a_{k_1};\cdots;a_{k_n}}(S) \\ = & \mathrm{app}^{\Rightarrow}_{a_{k_n}}(\mathrm{app}^{\Rightarrow}_{a_{k_1};\cdots;a_{k_{n-1}}}(S)) & \text{[by definition]} \\ = & \mathrm{app}^{\Rightarrow}_{a_{k_n}}(\mathrm{app}^{\Rightarrow}_{a_{k_1}\|\cdots\|a_{k_{n-1}}}(S)) & \text{[by induction hypothesis]} \\ = & (((S \setminus \bigcup_{i=1}^{n-1} \mathrm{del}(a_{k_i})) \cup \bigcup_{i=1}^{n-1} \mathrm{add}(a_{k_i})) \setminus \mathrm{del}(a_{k_n})) \cup \mathrm{add}(a_{k_n}) & \text{[by def.]} \\ = & (((S \setminus \bigcup_{i=1}^{n-1} \mathrm{del}(a_{k_i})) \setminus \mathrm{del}(a_{k_n})) \cup \bigcup_{i=1}^{n-1} \mathrm{add}(a_{k_i})) \cup \mathrm{add}(a_{k_n}) & \\ & \qquad [a_1, \ldots, a_n \text{ are composable and thus } \mathrm{del}(a_{k_n}) \cap \bigcup_{i=1}^{n-1} \mathrm{add}(a_{k_i}) = \emptyset] & \\ = & (S \setminus \bigcup_{i=1}^{n} \mathrm{del}(a_{k_i})) \cup \bigcup_{i=1}^{n} \mathrm{add}(a_{k_i}) & \text{[by simplification]} \\ = & \mathrm{app}^{\Rightarrow}_{a_1\|\cdots\|a_n}(S) & \text{[by def.]} \end{array}$$

□

Let $\pi = S_0 a_1 S_1 \cdots S_{m-1} a_m S_m$ and $\pi' = S'_0 a'_1 S'_1 \cdots S'_{n-1} a'_n S'_n$ be planning paths such that $S'_0 = S_m$. The *concatenation of* π *and* π', in symbols $\pi\pi'$, is the planning path $S_0 a_1 S_1 \cdots S_{m-1} a_m S_m a'_1 S'_1 \cdots S'_{n-1} a'_n S'_n$.

Lemma 2. *Let $\Theta = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{O} \rangle$ and $\hat{\Theta} = \langle \mathcal{F}, \hat{\mathcal{A}}, \mathcal{I}, \hat{\mathcal{O}} \rangle$, where $\hat{\mathcal{O}}$ and $\hat{\mathcal{A}}$ are obtained from $\mathcal{O}$ and $\mathcal{A}$ by adding all linearizable compound actions. If $S_0 a_1 S_1 \cdots S_{m-1} a_m S_m$ is a path of $\hat{\Theta}$, then $\pi_{\overline{a}_1}(S_0)\, \pi_{\overline{a}_2}(S_1) \cdots \pi_{\overline{a}_m}(S_m)$ is a path of Θ where $\overline{a}_1, \ldots, \overline{a}_m$ are linearizations of $a_1, \ldots, a_m$ respectively. In such cases we say that the planning path $S_0 a_1 S_1 \cdots S_{m-1} a_m S_m$ is linearizable.*

Proof. Simple consequence of Lemma 1. □

It is thus important to identify conditions that guarantee the linearizability of parallel planning paths. A simple approach is based on the idea of requiring that all component actions must be executed in any order and must lead to the same result.

Definition 3 (Interference). *We say that a_1 interferes with a_2, in symbols $a_1 \oplus a_2$, if and only if* (i) *$a_1 \neq a_2$ and* (ii) *$\mathrm{del}(a_1) \cap \mathrm{pre}(a_2) \neq \emptyset$ or $\mathrm{del}(a_2) \cap \mathrm{pre}(a_1) \neq \emptyset$, i.e. a_1 is different from a_2 and the execution of a_1 does not prevent the execution of a_2 and vice versa.*

Lemma 3. *Let $a_1, \ldots, a_n$ be composable actions. If a_i does not interfere with a_j for $i, j = 1, \ldots, n$ with $i \neq j$, then all permutations of $a_1, \ldots, a_n$ are linearizations of $a_1 \| \cdots \| a_n$.*

Proof. Let $a_{k_1}, \ldots a_{k_n}$ be a generic permutation of $a_1, \ldots, a_n$. We must show that $a_{k_{i+1}}$ is applicable in S_i for $i = 0, \ldots, n-1$, where $S_0 \supseteq \bigcup_{i=1}^{n} \mathrm{pre}(a_i)$ and $S_{i+1} = \mathrm{app}^{\rightarrow}_{a_{k_{i+1}}}(S_i)$ for $i = 0, \ldots, n-1$. To this end we prove a more general fact, namely that a_{k_j} is applicable in S_i for all $j = i+1, \ldots, n$ and all $i = 0, \ldots, n-1$, from which the above fact can be readily inferred by considering the cases in which $j = i+1$. The proof is by induction on i. The base case (i.e. $i = 0$) boils down to showing that a_{k_j} is applicable in S_0 for all $j = 1, \ldots, n$. This trivially holds since $S_0 \supseteq \bigcup_{i=1}^{n} \mathrm{pre}(a_i) = \bigcup_{j=1}^{n} \mathrm{pre}(a_{k_j})$. For the step case we must show that a_{k_j} is applicable in S_{i+1} for all $j = i+2, \ldots, n$. Let j be such that $i+1 < j \leq n$. By induction hypothesis it readily follows that a_{k_j} is applicable in S_i or, in other words, that $\mathrm{pre}(a_{k_j}) \subseteq S_i$. Since, by hypothesis, a_{k_j} and $a_{k_{i+1}}$ do not interfere, we know that $\mathrm{pre}(a_{k_j}) \cap \mathrm{del}(a_{k_{i+1}}) = \emptyset$. From this we can conclude that $\mathrm{pre}(a_{k_j}) \subseteq S_i \setminus \mathrm{del}(a_{k_{i+1}})$ and therefore also that $\mathrm{pre}(a_{k_j}) \subseteq (S_i \setminus \mathrm{del}(a_{k_{i+1}})) \cup \mathrm{add}(a_{k_{i+1}}) = S_{i+1}$. Thus a_{k_j} is applicable in S_i and this concludes the proof. □

Theorem 1. *Let $\Theta = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{O} \rangle$ and let $\hat{\Theta} = \langle \mathcal{F}, \hat{\mathcal{A}}, \mathcal{I}, \hat{\mathcal{O}} \rangle$ where $\hat{\mathcal{O}}$ and $\hat{\mathcal{A}}$ are obtained from $\mathcal{O}$ and $\mathcal{A}$ by adding all compound actions $a_1 \| \cdots \| a_n$ for all pairwise non-interfering set of actions $\{a_1, \ldots, a_n\} \subseteq \mathcal{O}$. Given this, if $S_0 a_1 S_1 \cdots S_{m-1} a_m S_m$ is a path of $\hat{\Theta}$, then $\pi_{\overline{a}_1}(S_0)\, \pi_{\overline{a}_2}(S_1) \cdots \pi_{\overline{a}_m}(S_m)$ is a path of Θ where $\overline{a}_1, \ldots, \overline{a}_m$ are linearizations of $a_1, \ldots, a_m$ respectively. In such cases we also say that $S_0 a_1 S_1 \cdots S_{m-1} a_m S_m$ is a partial-order planning path of Θ.*

Proof. Simple consequence of Lemma 1 and Lemma 3. □

An immediate consequence of Theorem 1 is that any solution to $\hat{\Pi} = \langle \hat{\Theta}, \mathcal{G} \rangle$ compactly represents a finite set of solutions (plans) to $\Pi = \langle \Theta, \mathcal{G} \rangle$ and it is thus referred to as a *partial-order plan* of Π. Moreover, since by definition the planning paths of $\hat{\Theta}$ are a superset of those of Θ, then for each solution to Π there exists a correspondent (maybe shorter) solution to $\hat{\Pi}$ and vice versa.

Notationally, if $\|(\Lambda_1), \ldots, \|(\Lambda_k)$ is a solution to $\hat{\Pi}$, then the sequence of sets of actions $\Lambda_1, \ldots, \Lambda_k$ denotes a partial-order plan of Π. Moreover, let $a_{i,1}, \ldots, a_{i,|\Lambda_i|}$ be a permutation (without repetition) of the actions in Λ_i for $i = 1, \ldots, k$, then any sequence—there are $|\Lambda_1|!|\Lambda_2|! \cdots |\Lambda_k|!$ possible sequences—of actions $a_{1,1}, \ldots, a_{1,|\Lambda_1|}, \ldots, a_{k,1}, \ldots, a_{k,|\Lambda_k|}$ is a plan of Π (see Example 4).

Example 4. *Given the planning problem* $\Pi = \langle \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{O} \rangle, \mathcal{G} \rangle$ *where:*

$$
\begin{array}{rcl}
\mathcal{F} & = & \{at(home), at(sm), taken(toy), taken(milk), own(toy), own(milk)\} \\
\mathcal{A} & = & \{move(home, sm), move(sm, home), take(toy), take(milk), \\
 & & buy(toy), buy(milk)\} \\
\mathcal{I} & = & \{\{at(home)\}\} \\
\mathcal{O} & = & \{ \\
 & & \quad at(home) \overset{move(home,sm)}{\Rightarrow} at(sm)\ ; \neg at(home), \\
 & & \quad at(sm) \overset{move(sm,home)}{\Rightarrow} at(home)\ ; \neg at(sm), \\
 & & \quad at(sm) \overset{take(toy)}{\Rightarrow} taken(toy)\ ;, \\
 & & \quad at(sm) \overset{take(milk)}{\Rightarrow} taken(milk)\ ;, \\
 & & \quad at(sm), taken(toy) \overset{buy(toy)}{\Rightarrow} own(toy)\ ;, \\
 & & \quad at(sm), taken(milk) \overset{buy(milk)}{\Rightarrow} own(milk)\ ; \\
 & & \} \\
\mathcal{G} & = & \{\{at(sm), own(milk), own(toy)\}\}
\end{array}
$$

The sequence of sets of actions

$$\{move(home, sm)\}, \{take(milk), take(toy)\}, \{buy(milk), buy(toy)\}\}$$

is a partial-order plan of Π *compactly representing all the following four plans of* Π*:*

(1) $move(home, sm), take(milk), take(toy), buy(milk), buy(toy)$
(2) $move(home, sm), take(milk), take(toy), buy(toy), buy(milk)$
(3) $move(home, sm), take(toy), take(milk), buy(toy), buy(milk)$
(4) $move(home, sm), take(milk), take(toy), buy(toy), buy(milk)$

Intuitively, in order to own the toy and the milk firstly we have to reach the supermarket sm, secondly we have to take the toy and the milk (in any order), and finally we have to pay them (in any order) at the cash-desk.

Finally it is worth pointing out that, in general, planning problems are semi-decidable problems [ENS95]. Basically it is possible to devise a procedure that halts with a correct plan when a plan exists, but otherwise may not terminate. When the planning problem consists of a finite set of propositional fluents and a finite set of operators, then the associated search space becomes finite and the planning problem is decidable. Namely, such a family of planning problems, referred to as Classical planning, belongs to the PSPACE-complete complexity class [ENS95, Neb00]. Finally, it must be noted that Classical planning is NP-complete for plans of polynomially-bounded length [Tur02].

2.3 Propositional Satisfiability Problems (SAT)

The Boolean satisfiability problem (SAT) is a decision problem considered in complexity theory. The class of satisfiable propositional formulae is NP-complete, as is that of its variant 3-SAT. An instance of the problem is defined by a propositional expression written using only standard logical connectives (such as conjunction, disjunction, and negation), and Boolean variables. The question is: given the expression, is there some assignment of TRUE and FALSE values[12] to the Boolean variables that will make the entire expression true?

Formally, a *proposition* (or *Boolean variable*) is a symbol from the set $P = \{p_1, \ldots, p_n\}$, our *propositional alphabet*. As specified by the (unambiguous) grammar rule *Formula* depicted in Figure 2.1, a *propositional formula* is built out of propositions, logical connectives, and parenthesis. A *literal* (see grammar rule *Literal* in Figure 2.1) is either a proposition (say, p) or the complement of one (denoted by $\neg p$); in the first case, we say that l is a *positive literal*, and in the second, we say that l is a *negative literal*. A *clause* (see grammar rule *Clause* in Figure 2.1) is a finite disjunction of literals. A propositional formula in *conjunctive normal form* (*CNF*) (see grammar rule *CNFFormula* in Figure 2.1) is a finite conjunction of clauses. Any propositional formula can be converted to an equi-satisfiable CNF: by allowing the introduction of auxiliary variables (see, for example, [PG86]) the worst case computation time remains polynomial in the size of the original formula.

The purpose of a propositional formula is to create meanings of statements given meanings of atomic statements. The semantics of a formula ϕ with propositional symbols $p_1, \ldots, p_n$ is a mapping associating to each truth assignment v to $p_1, \ldots, p_n$ a truth value (FALSE or TRUE) for ϕ. In other words, a *truth assignment* v is a partial function from the set of propositions to $\{\text{TRUE}, \text{FALSE}\}$ ($v : P \mapsto \{\text{TRUE}, \text{FALSE}\}$). Given a truth assignment v, the satisfiability of a propositional formula ϕ, denoted by $v \models \phi$, is in-

[12] Notice that TRUE and FALSE denote logical truth and falsehood, respectively.

$$
\begin{array}{rcl}
\mathit{Formula} & ::= & \mathit{Atom} \\
& | & \neg \mathit{Formula} \\
& | & (\mathit{Formula} \wedge \mathit{Formula}) \\
& | & (\mathit{Formula} \vee \mathit{Formula}) \\
& | & (\mathit{Formula} \supset \mathit{Formula}) \\
& | & (\mathit{Formula} \equiv \mathit{Formula}) \\
\mathit{Atom} & ::= & \mathit{Proposition} \\
& | & \textsc{true} \\
& | & \textsc{false} \\
\mathit{Proposition} & ::= & p_1 \mid p_2 \mid \cdots \mid p_n \\
 & & \\
\mathit{Literal} & ::= & \mathit{Proposition} \\
& | & \neg \mathit{Proposition} \\
\mathit{Clause} & ::= & \mathit{Atom} \\
& | & (\mathit{Clause} \vee \mathit{Clause}) \\
\mathit{CNFFormula} & ::= & \mathit{Clause} \\
& | & (\mathit{Clause} \wedge \mathit{Clause})
\end{array}
$$

Figure 2.1: Propositional Logic: syntax

ductively determined according to Figure 2.2.[13] We represent truth value assignments as sets of propositional letters assigned to TRUE. We say that an assignment M *satisfies (or is a model of) a propositional formula* ϕ if and only if $M \cup \{\neg p : p \in P, p \notin M\} \models \phi$ where P is the alphabet of the propositions, and $\models$ is the aforementioned entailment relation.

In terms of notation we use also $\bigwedge\{\phi_1, \ldots, \phi_m\}$ and $\bigvee\{\phi_1, \ldots, \phi_m\}$ to indicate the conjunction and the disjunction respectively of the SAT formulae in the set $\{\phi_1, \ldots, \phi_m\}$. Such conjunction and disjunction of formulae are also denoted by:

$$
\begin{array}{cc} \wedge & \phi_1 \\ \vdots & \vdots \\ \wedge & \phi_m \end{array} \quad \text{and} \quad \begin{array}{cc} \vee & \phi_1 \\ \vdots & \vdots \\ \vee & \phi_m \end{array}
$$

respectively.

Follow two simple examples of propositional formulae and their related satisfiability results.

[13] In there, p is a proposition, v is a truth assignment, $\models$ is the entailment relation, and ϕ_1, ϕ_2 are propositional formulae.

$$
\begin{array}{lll}
v \models p & \text{iff} & v(p) = \text{TRUE with } p \in P \\
v \models \neg\phi_1 & \text{iff} & v \not\models \phi_1 \\
v \models \phi_1 \vee \phi_2 & \text{iff} & v \models \phi_1 \text{ or } v \models \phi_2 \\
v \models \phi_1 \wedge \phi_2 & \text{iff} & v \models \phi_1 \text{ and } v \models \phi_2 \\
v \models \phi_1 \supset \phi_2 & \text{iff} & v \not\models \phi_1 \text{ or } v \models \phi_2 \\
v \models \phi_1 \equiv \phi_2 & \text{iff} & v \models \phi_1 \supset \phi_2 \text{ and } v \models \phi_2 \supset \phi_1 \\
 & & \\
v \models \text{TRUE} & & \text{for each } v \\
v \not\models \text{FALSE} & & \text{for each } v
\end{array}
$$

Figure 2.2: Propositional Logic: semantics

Example 5. *For instance, the propositional formula*

$$
\begin{array}{rcl}
\Phi = & \wedge & a \supset (\neg b \wedge c) \\
 & \wedge & c \supset a \\
 & \wedge & c
\end{array}
$$

built out of the propositions $\{a, b, c\}$ *allows for only one model in which* a *and* b *must be assigned to* TRUE*, and* b *to* FALSE*. Formally* $\{a, c\} \models \Phi$*. Furthermore the formula*

$$
\begin{array}{rcl}
\Phi_{CNF} = & \wedge & \neg a \vee \neg b \\
 & \wedge & \neg a \vee c \\
 & \wedge & \neg c \vee a \\
 & \wedge & c
\end{array}
$$

is the (equi-satisfiable) CNF conversion of Φ*.*

Example 6. *For instance, the propositional formula*

$$
\begin{array}{rcll}
\Phi = & \wedge & a \supset (\neg b \wedge c) & (2.1) \\
 & \wedge & c \supset a & (2.2) \\
 & \wedge & c & (2.3) \\
 & \wedge & b & (2.4)
\end{array}
$$

built out of the propositions $\{a, b, c\}$ *is unsatisfiable: (2.3) and (2.4) would impose* c *and* b *to be both assigned to* TRUE*; (2.2) would then impose* a *to be assigned to* TRUE *too, but this would cause a contradiction with respect to (2.1).*

Chapter 3

Modelling Security Protocols

In this chapter we describe how to model the problem of establishing if a security protocol guarantees its desiderata or not. We start providing in Section 3.1 background informations on security protocols (cryptographic concepts, and the well-know example of the Needham-Schroeder Public-Key protocol discovered to be flawed 17 years after its publication). After this basic and informal notions we introduce in Section 3.2 the concept of *protocol insecurity problem* to formally describe how we model security protocols, their security properties, and the question, in general undecidable, of deciding whether a protocol satisfies its security requirements or not. We conclude in Section 3.3 by providing an overview of (un)-decidability results on the domain of security protocols.

3.1 Background

Communication protocols specify an exchange of messages between principals, i.e. the agents participating in a protocol execution (e.g. users, hosts, or processes). Messages are sent over open networks, such as the Internet, that cannot be considered secure. As a consequence, protocols should be designed "robust" enough to work even under worst-case assumptions, namely messages may be eavesdropped or tampered with by an *intruder* or dishonest or careless principals. A specific category of protocols has been devised with the purpose of securing communications over insecure networks: *security* (or cryptographic) *protocols* are communication protocols that aim at providing *security guarantees* such as authentication of principals or secrecy of some piece of information through the application of *cryptographic primitives*.

3.1.1 Cryptography

Cryptography is an ancient science that studies how to convert a *plaintext* P into a *ciphertext* C (and vice versa) that is unintelligible to anyone monitoring the network. The first documented example of written cryptography is dated around 1,900 B.C., when an Egyptian scribe used non-standard hieroglyphs in an inscription. Indeed more famous, the example of Julius Caesar (around 60 B.C.) who used a simple substitution with the normal alphabet, just shifting the letters a fixed amount, to secure government communications with his army. Figure 3.1 shows the Caesar code for a shifting fixed to 5. According to this figure the ciphertext "Ymnx xjsyjshj nx ns hqjfw" corresponds to the plaintext "This sentence is in clear".

Caesar Code: shifting fixed to 5													
Plain letter	a	b	c	d	e	f	g	h	i	j	k	l	m
Code letter	f	g	h	i	j	k	l	m	n	o	p	q	r
Plain letter	n	o	p	q	r	s	t	u	v	w	x	y	z
Code letter	s	t	u	v	w	x	y	z	a	b	c	d	e

Figure 3.1: Caesar code

The process of converting P into C is called *encryption*, while the reverse procedure is called *decryption*. The main feature of modernday encryption is that the precise form of the ciphertext C corresponding to a plaintext P depends on an additional parameter K known as the encryption key. In order to recover the original plaintext the intended receiver should use a second key K^{-1} called the inverse key.[1] The cryptographic mechanism is depicted in Figure 3.2.

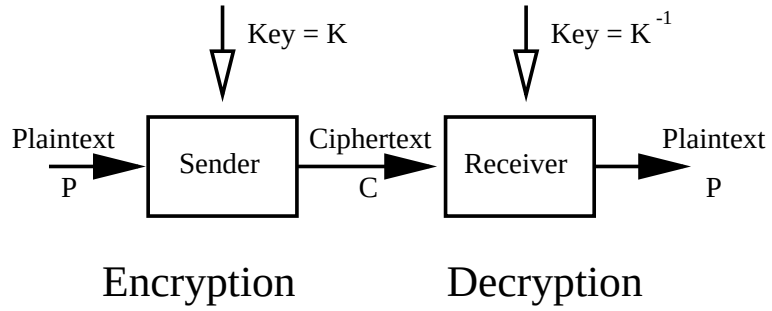

Figure 3.2: Encryption and Decryption, [CJ97]

[1]Even if we indicate the inverse key as K^{-1} there is no way to compute it from K (and vice versa).

The most known cryptographic schemes are the symmetric key cryptosystem and the asymmetric key cryptosystem. In symmetric key cryptography, the decryption key and the encryption key are assumed to be identical, i.e. $K = K^{-1}$. In more details, each pair of principals (e.g. A and B) will have their own key (e.g. Kab) for communicating with each other. In asymmetric key cryptography (also called public key cryptography), each principal A has a public key, Ka, known to all users of the network, and a private key Ka^{-1}, known only to herself. Knowledge of a user's public key provides no help in finding her private key. Under many public key schemes public keys and private keys can be used for both encryption and decryption. On one side, all users of the network can encrypt messages for A by using A's public key, and these messages will then be unintelligible to everyone except A. On the other side, a principal A can encrypt her transmission with her own private key, Ka^{-1}. This creates a message readable by everyone (they can decrypt it using A's public key), with the property that it must have been written by someone in possession of A's private key (ideally, only A should know Ka^{-1}). This is also at the base of digital signature that exploits private key encryption to authenticate the identity of the sender of a message (or the signer of a document), and possibly ensure that the original content of the message (or document) sent is unchanged.

In term of notation, given a message M and a key K, we indicate the process of either encryption or decryption by means of $\{M\}_K$. Independently from the cryptosystem used, the following equations hold:

$$\{\{M\}_K\}_{K^{-1}} = M \tag{3.1}$$

$$\{\{M\}_{K^{-1}}\}_K = M \tag{3.2}$$

In natural language, equation 3.1 (3.2) states that by decrypting with K^{-1} (K) a message already encrypted with K (K^{-1}) it is possible to recover M.

As a matter of fact cryptographic algorithms ensure a high degree of confidence in exchanging messages over insecure communication channels. The best-known cryptographic algorithms for symmetric and asymmetric key schemes are the DES (Digital Encryption Standard, [FIP77]),[2] and the RSA (Rivest, Shamir, and Adleman, [RSA78]) algorithms, respectively. The security of cryptographic algorithms relies in the difficulty of breaking them by performing a brute-force search of the key space. Hence the use of keys sufficiently long to prevent a brute force attack in a reasonable time entirely justifies the standard assumption adopted in formal analysis of security protocols and called *perfect cryptography*. The idea underlying such an assumption is that an encrypted message can be decrypted only by using the appropriate decryption key, i.e. it is possible to retrieve M from $\{M\}_K$ only by using K^{-1} as decryption key (see for instance Example 7).

[2]DES has been adopted as an encryption standard by the US government in 1976. Since 2001 the AES (Advanced Encryption Standard) algorithm replaced DES in such a role and it is expected to be used worldwide and analysed extensively.

Example 7. *Suppose that under a symmetric key cryptosystem the agent A knows K_1, but not K_2 and K_3. Given the message $\{\{M\}_{K_3}\}_{K_2}, K_2\}_{K_1}$, the agent A can deduce the terms K_2 and $\{M\}_{K_3}$, but not M.*

Informally, the scenario we are interesting in involves a set of honest agents that, according to a security protocol, exchange messages over insecure communication channels controlled by a malicious agent called intruder with the ultimate goal of achieving some security requirements (e.g. authentication of principals, secrecy of some data). Specifically, we analyse such a scenario in the attempt of establishing if the security protocol satisfies its design criteria or not.

3.1.2 A simple example: NSPK

Security protocols are usually presented by means of the common and informal Alice&Bob notation, [CJ97]. For instance, the well-known Needham-Schroeder Public-Key (NSPK) authentication protocol [NS78] looks like follow:

$$\begin{array}{llllll} (1) & A & \rightarrow & B & : & \{A, Na\}_{Kb} \\ (2) & B & \rightarrow & A & : & \{Na, Nb\}_{Ka} \\ (3) & A & \rightarrow & B & : & \{Nb\}_{Kb} \end{array}$$

where A and B are the roles involved in the protocol; Ka and Kb are the public keys of A and B, respectively; Na and Nb are nonce[3] challenges generated by A and B, respectively. Step (1) of the protocol denotes A sending B a message comprising the identity of A and the nonce Na encrypted with Kb. Since $\{A, Na\}_{Kb}$ can only be deciphered by means of the private key Kb^{-1} and the latter is (by assumption) only known by B, then the effect of Step (1) is that only B can possibly learn the value of Na. In Step (2) agent B proves to A his participation in the protocol and, by sending Nb, asks A to do the same. In Step (3) agent A concludes by proving to B her own participation in the protocol. The above high level protocol specification describes a kind of template $NSPK(A, B, Ka, Kb, Na, Nb)$ parametrised by some free variables[4] appearing in it. A ground instance of the security protocol template represents a session of the protocol. Successful execution of the NSPK protocol should convince both A and B that they have been talking to each other. The rationale is that only B and A could have formed the appropriate response to the nonce challenges issued in (1) and in (2), respectively. In fact, a malicious agent I can deceit *bob* (an instance of B) into believing that he is talking with *alice* (instance of A) whereas he

[3] *Nonces* are numbers randomly generated by principals and are intended to be used *only once*.

[4] Free variables are indicated by means of capital letters excepted I that is used to indicate the malicious agent.

is talking with I. This is achieved by executing concurrently two sessions $NSPK(alice, I, ka, ki, na, ni)$ and $NSPK(alice, bob, ka, kb, na2, nb)$ of the protocol and using messages from one session to form messages in the other as illustrated by the following protocol trace:[5]

$$\begin{array}{llcll} (1.1) & alice & \rightarrow & I & : & \{alice, na\}_{ki} \\ (2.1) & I(alice) & \rightarrow & bob & : & \{alice, na\}_{kb} \\ (2.2) & bob & \rightarrow & I(alice) & : & \{na, nb\}_{ka} \\ (1.2) & I & \rightarrow & alice & : & \{na, nb\}_{ka} \\ (1.3) & alice & \rightarrow & I & : & \{nb\}_{ki} \\ (2.3) & I(alice) & \rightarrow & bob & : & \{nb\}_{kb} \end{array}$$

where $I(alice)$ indicates the intruder pretending to be *alice*. At the end of the above trace *bob* believes he has been talking with *alice*, but this is obviously not the case. As suggested by Lowe the successful execution of the above trace in a scenario where *bob* is a banker would authorise the intruder to send the following request to the bank:

$$I(alice) \rightarrow bob : \{alice, na, nb, \text{move 10,000 euro from my account to } I\text{'s}\}_{kb}$$

In natural language such a message can be read as:

> Hello *bob*, this is *alice* and these are my authentication credentials *na* and *nb*. Please transfer 10,000 euro from my bank account to that of my friend I.

The problem underlying the NSPK protocol is that the second message does not provide guarantees of the identity of the sender. It is fairly easy to change the protocol so as to prevent the attack by including B's identity in the second message of the protocol:

$$B \rightarrow A : \{B, Na, Nb\}_{Ka}$$

After such a correction, step (2.2) of the above attack trace would become

$$bob \rightarrow I(alice) : \{bob, na, nb\}_{ka}$$

and therefore the intruder can not successfully replay this message in step (1.2), because *alice* is expecting to receive a message containing I's identity.

A problem with the Alice&Bob notation used above to describe security protocols is that it leaves implicit many important details such as the shared information and how the principals should react to messages of an

[5]Notice that, such an attack has been discovered by Gavin Lowe 17 years after the publication of the NSPK protocol. In the sequel we will refer to this attack as the Lowe's attack on the NSPK.

unexpected form. This kind of description is usually supplemented with explanations in natural language which in our case explain that Na and Nb are nonces, and that Ka and Kb are A's and B's public key, respectively. To cope with the above difficulties and pave the way to the formal analysis of protocols a set of models and specification formalisms have been put forward [CDL+00, Low98, Pau98]. In Chapter 5 we provide an elegant and concrete framework to specify security protocols.

At a more abstract level we introduce in the next Section the concept of *protocol insecurity problem* to formally describe how we model security protocols, their security properties, and the question, in general undecidable [DLMS99, EG83], of establishing whether a protocol satisfies its security requirements or not.

3.2 Protocol Insecurity Problems

We formalise protocol insecurity problems as reachability problems associated with a transition system modelling the behaviour of the honest agents as well as the behaviour of the intruder. Besides this, we make the standard assumptions of the aforementioned *perfect cryptography*, and of *strong typing* i.e. agents only accept type-correct messages and therefore type confusion is not allowed (see for instance Example 8).[6]

Example 8. *Let us consider the one-step protocol:*

$$A \rightarrow B : \{A, N\}_{Kb}$$

where Kb and N are B's public key and a nonce freshly generated by A, respectively. Under the strong typing assumption, honest agents are expected to be able to recognise the type and the pattern of messages. As a consequence, B will ignore any message whose type pattern differs from $\{agent, text\}_{public\_key}$, i.e. an encryption under a public key of an agent paired with a nonce. For instance messages like $\{A, A\}_{Kb}$ or $\{N\}_{Kb}$ will not be accepted by B.

Formally, a *protocol insecurity problem* is a reachability problem $\Xi = \langle \Gamma_H \circ \Gamma_I, \mathsf{B} \rangle$, where Γ_H and Γ_I are (multi)-set rewriting systems specifying the behaviour of the honest agents and of the intruder respectively and B is a set of states whose reachability implies the violation of a security property that the protocol is intended to guarantee.

In the rest of this section we describe each of these components by focusing on the aforementioned NSPK running example. Namely, we will present how we model the two concurrent sessions of the protocol leading to the

[6] As pointed out in [HLS00], a wide variety of type-flaw attacks can be prevented by tagging the fields of a message with information indicating its intended type.

$$
\begin{array}{rcl}
\mathit{Fact} & ::= & \mathit{PrincipalFact} \mid \\
 & & \mathit{IntruderFact} \\
\mathit{PrincipalFact} & ::= & \mathtt{state}(\mathit{StepId}, \mathit{Agent}, \mathit{Agent}, \mathit{Msgs}, \mathit{SessId}) \mid \\
 & & \mathtt{msg}(\mathit{StepId}, \mathit{Agent}, \mathit{Agent}, \mathit{Msg}) \mid \\
 & & \mathtt{secret}(\mathit{Msg}, \mathit{Agent}) \mid \\
 & & \mathtt{witness}(\mathit{Agent}, \mathit{Agent}, \mathit{IndSym}, \mathit{Msg}) \mid \\
 & & \mathtt{request}(\mathit{Agent}, \mathit{Agent}, \mathit{IndSym}, \mathit{Msg}) \\
\mathit{IntruderFact} & ::= & \mathtt{ik}(\mathit{Msg}) \mid \\
 & & \mathtt{msg}(\mathit{StepId}, \mathit{Agent}, \mathit{Agent}, \mathit{Msg}) \\
\mathit{Msgs} & ::= & [\,] \mid [\mathit{Msg}(,\mathit{Msg})^*] \\
\mathit{Msg} & ::= & \mathit{Agent} \mid \mathit{NumSym} \mid \mathit{Key} \mid \{\mathit{Msg}\}_{\mathit{Key}} \mid \langle \mathit{Msg}, \mathit{Msg} \rangle \\
\mathit{Key} & ::= & \mathit{IndSym} \mid \mathit{IndSym}^{-1} \mid \mathit{Msg} \\
\mathit{Agent} & ::= & \mathit{IndSym} \\
\mathit{NumSym} & ::= & \mathit{IndSym} \\
\mathit{IndSym} & ::= & \mathit{IndConst} \mid \mathit{IndVar} \\
\mathit{StepId} & ::= & 0 \mid 1 \mid 2 \mid \cdots \\
\mathit{SessId} & ::= & 0 \mid 1 \mid 2 \mid \cdots
\end{array}
$$

Figure 3.3: BNF rules for facts

well-known Lowe's attack on NSPK (see Section 3.1.2). We start presenting in Section 3.2.1 and Section 3.2.2 the (multi)-set rewriting systems Γ_H and Γ_I. Section 3.2.3 is devoted to the description of the goal predicates we use to specify some of the best-known security requirements. In Section 3.2.4 we present how to specialise the intruder with respect to the considered protocol and under the scenario analysed. Finally, Section 3.2.5 summaries the previous sections in the presentation of the whole protocol insecurity problem associated to the two concurrent sessions of the NSPK protocol leading to the well-known Lowe's attack.

3.2.1 Honest agents

The (multi)-set rewriting system $\Gamma_H = \langle \mathsf{F}_H, \mathsf{L}_H, \mathsf{I}_H, \mathsf{R}_H \rangle$ models the behaviour of the honest agents involved in the protocol.

Facts. The set of facts F_H are formed according to the BNF (grammar) rule *PrincipalFact* of Figure 3.3. The non terminal symbols *IndConst* and *IndVar* denote a set of individual constants and variables respectively. Numerals are used to denote the different steps (*StepId*) and sessions (*SessId*) of the protocol being modelled. Messages (*Msg*) are built out of agent

names (*Agent*), numeric values (*NumSym*), and keys (*Key*) using pairing ($\langle Msg, Msg\rangle$) and encryption ($\{Msg\}_{Key}$) to form compound messages.[7] Keys (*Key*) are either public (*IndSym*), private ($IndSym^{-1}$), or symmetric (*Msg*). Thus, if K is an individual symbol denoting a public key, then K^{-1} denotes the private key associated to K. Notice that any message can be used as a symmetric key: this allows us to model protocols of real-world complexity (e.g. the authentication protocol used in Kerberos [NS78]) in which symmetric keys are built out of several pieces of information.

A fact of the form $\mathtt{msg}(J, S, R, M)$ states that principal S has supposedly[8] sent message M to principal R at protocol step J. The state of execution of principal R at step J of session C is represented by a fact of the form $\mathtt{state}(J, S, R, [M_1, \ldots, M_{\mathsf{h}}], C)$; in particular it means that R knows the terms $M_1, \ldots, M_{\mathsf{h}}$ at step J of session C, and—if $J \neq 0$—also that a message from S to R is awaited for step J of session C to be executed. A fact of the form $\mathtt{secret}(M, S)$ means that message M is a secret shared between a group of principals including S. Facts of the form $\mathtt{witness}(S, R, O, M)$ and $\mathtt{request}(R, S, O, M)$ play key role in the specification of the authentication security property. In particular, $\mathtt{witness}(S, R, O, M)$ means that honest agent S wants to execute the protocol with agent R by using M as value for the authentication identifier O, whereas $\mathtt{request}(R, S, O, M)$ means that agent R accepts the value M and now relies on the guarantee that agent S exists and agrees with him on this value for the authentication identifier O. Thus, the situation in which $\mathtt{request}(R, S, O, M)$ holds and the corresponding witness fact (i.e. $\mathtt{witness}(S, R, O, M)$) does not hold represents a violation of an authentication property.

Initial States. To simplify notation if f is a fact and fs is a set of facts, then we write $\mathsf{f}.\mathsf{fs}$ in place of $\{\mathsf{f}\} \cup \mathsf{fs}$ and, when no confusion arises, we write f in place of $\{\mathsf{f}\}$.[9] Thus, for example, the following expression represents the initial state S_0 of the honest agents involved in two concurrent sessions of the NSPK:[10]

$$\mathtt{state}(0, a, a, [a, i, ka, ka^{-1}, ki], 1) \tag{3.3}$$

$$.\,\mathtt{state}(0, a, a, [a, b, ka, ka^{-1}, kb], 2)\,.\,\mathtt{state}(1, a, b, [b, a, kb, kb^{-1}, ka], 2) \tag{3.4}$$

Fact (3.3) represents the initial state of the honest agent a playing the role of initiator in session 1 and knowing her own identity, the identity of intruder (the agent she would like to talk with), her own public and private keys,

[7] Notice that, we can easily extend this model of messages to include other operators like for instance non-interpreted hash functions.

[8] As we will see, since the intruder may fake other principals' identity, the message might have been sent by the intruder.

[9] Notice that the set constructor . is thus associative and commutative.

[10] It must be noted that in such a case I_H is the singleton set $\{\mathsf{S}_0\}$.

and the public key of the intruder. Facts (3.4) represent the initial state of the honest agents a and b involved as initiator and responder (respectively) in session 2.

Rewrite Rules. Honest participants strictly behave according to the protocol. We recall that a rewrite rule with variables specifies a set of possible transitions of the transition system associated with the (multi)-set rewriting system. In the rest of this paragraph we describe the set of rewrite rule, R_H, used to model the NSPK example.[11]

The following rewrite rule models the first step of the NSPK protocol that any agent A can carry out when acting as initiator:

$$\begin{array}{l} \mathtt{state}(0, A, A, [A, B, Ka, Ka^{-1}, Kb], S) \\ \quad \xrightarrow{step_1(A,B,Ka,Kb,S)} \\ \exists Na : \\ \ \mathtt{state}(2, B, A, [Na, A, B, Ka, Ka^{-1}, Kb], S) \\ \ \mathtt{.msg}(1, A, B, \{\langle A, Na\rangle\}_{Kb}) \\ \ \mathtt{.witness}(A, B, na, Na) \end{array} \tag{3.5}$$

It is easy to see that rule (3.5) is applicable to the state comprising facts (3.3) and (3.4) when, for instance, the substitution $[A/a, B/i, Ka/ka, Kb/ki, S/1]$ is applied (i.e. the variables A, B, Ka, Kb, and S are instantiated with a, i, ka, ki, and 1, respectively). The execution of such instantiated rule, $step_1(a, i, ka, kb, 1)$, leads to the state:

$$\begin{array}{l} \mathtt{state}(2, i, a, [na0, a, i, ka, ka^{-1}, ki], 1) \\ \mathtt{.msg}(1, a, i, \{\langle a, na0\rangle\}_{ki})\ \mathtt{.witness}(a, i, na, na0) \\ \mathtt{.state}(0, a, a, [a, b, ka, ka^{-1}, kb], 2)\ \mathtt{.state}(1, a, b, [b, a, kb, kb^{-1}, ka], 2) \end{array}$$

where $na0$ is a constant freshly generated in place of the existential variable Na. Notice that, as effect of the application of the rule, message $\{\langle a, na0\rangle\}_{ki}$ has been sent by a to the intruder, a extend her knowledge with the nonce $na0$ and she is now ready to execute step 2 of the protocol, fact $\mathtt{witness}(a, i, na, na0)$ gives evidence that a wants to execute the protocol with the intruder by using $na0$ as value for the authentication identifier na, and all the other facts of the initial state are automatically derived from the previous state by inertia.

The second step of the NSPK protocol in which the responder receives the challenge generated by the initiator and replies with his own nonce, is

[11] Notice that the set of rule labels L_H readily follow from the sets of rewrite rules and of facts.

modelled by:

$$\begin{array}{l}\texttt{msg}(1, A, B, \{\langle A, Na\rangle\}_{Kb})\,.\,\texttt{state}(1, A, B, [B, A, Kb, Kb^{-1}, Ka], S) \\ \quad \xrightarrow{step_2(A,B,Ka,Kb,Na,S)} \\ \exists Nb : \\ \quad \texttt{state}(3, A, B, [Na, Nb, B, A, Kb, Kb^{-1}, Ka], S) \\ \quad .\texttt{msg}(2, B, A, \{\langle Na, Nb\rangle\}_{Ka}) \\ \quad .\texttt{witness}(B, A, nb, Nb)\end{array} \tag{3.6}$$

The state of the responder is updated by increasing the protocol step and by extending the knowledge with the acquired informations, namely the nonce sent by the initiator and the nonce freshly generated by the responder itself. The message $\{\langle Na, Nb\rangle\}_{Ka}$ is sent to A, and the fact $\texttt{witness}(B, A, nb, Nb)$ is asserted.

The third step of the protocol is modelled by:

$$\begin{array}{l}\texttt{msg}(2, B, A, \{\langle Na, Nb\rangle\}_{Ka})\,.\,\texttt{state}(2, B, A, [Na, A, B, Ka, Ka^{-1}, Kb], S) \\ \quad \xrightarrow{step_3(A,B,Ka,Kb,Na,Nb,S)} \\ \texttt{state}(4, B, A, [Nb, Na, A, B, Ka, Ka^{-1}, Kb], S) \\ .\texttt{msg}(3, A, B, \{Nb\}_{Kb}) \\ .\texttt{request}(A, B, nb, Nb)\end{array} \tag{3.7}$$

The state of the initiator is updated by increasing the protocol step and by extending the knowledge with the nonce sent by the responder. The message $\{Nb\}_{Kb}$ is sent to B, and the fact $\texttt{request}(A, B, nb, Nb)$ is asserted to give evidence that A accepts B's identity as authentic with respect to the value Nb used for the authentication identifier nb.

The final step of the protocol is modelled by:

$$\begin{array}{l}\texttt{msg}(3, A, B, \{Nb\}_{Kb})\,.\,\texttt{state}(3, A, B, [Na, Nb, B, A, Kb, Kb^{-1}, Ka], S) \\ \quad \xrightarrow{step_4(A,B,Ka,Kb,Na,Nb,S)} \\ \texttt{state}(5, A, B, [Na, Nb, B, A, Kb, Kb^{-1}, Ka], S) \\ .\texttt{request}(B, A, na, Na)\end{array} \tag{3.8}$$

The state of the responder is updated by increasing the protocol step, and the fact $\texttt{request}(B, A, na, Na)$ is asserted to give evidence that B accepts A's identity as authentic with respect to the value Na used for the authentication identifier na.

3.2.2 Intruder

The (multi)-set rewriting system $\Gamma_I = \langle \mathsf{F}_I, \mathsf{L}_I, \mathsf{I}_I, \mathsf{R}_I \rangle$ models the behaviour of the intruder, i.e. a malicious agent that composes and sends fraudulent

messages over the network in the attempt of violating some of the security property the protocol is designed to satisfy.

Facts. The set of facts F_I are formed according to the BNF rule *IntruderFact* of Figure 3.3. The intuitive meaning of facts of the form $\mathtt{msg}(J, S, R, M)$ has been already described in Section 3.2.1. The knowledge of the intruder is modelled by facts of the form $\mathtt{ik}(M)$ stating that the intruder knows message M.

Initial States. The state of the intruder is completely described by its knowledge. When an asymmetric protocol is considered, the intruder is usually allowed to know from the beginning his own identity, his own public and private keys, and the identities of the honest agents involved in the protocol together with their public keys. For instance, in the case of the NSPK protocol the set of initial states of the intruder is the singleton set whose single element is given by the following set of facts:

$$\mathtt{ik}(i) \centerdot \mathtt{ik}(a) \centerdot \mathtt{ik}(b) \centerdot \mathtt{ik}(ki) \centerdot \mathtt{ik}(ki^{-1}) \centerdot \mathtt{ik}(ka) \centerdot \mathtt{ik}(kb) \tag{3.9}$$

where ki is the public key of the intruder.

Rewrite Rules. Contrary to honest agents that execute faithfully each statement specified by the protocol, the Dolev-Yao intruder has many degrees of freedom. By observing all the traffic in the network, it extends its knowledge and from such a knowledge it can compose and send fraudulent messages to honest participants. In the rest of this paragraph we describe the set of rewrite rule, R_I, used to model the DY intruder.[12]

The following rule models the ability of the intruder of diverting the information exchanged by the honest participants:

$$\mathtt{msg}(I, A, B, M) \xrightarrow{divert(A,B,I,M)} \mathtt{ik}(M) \tag{3.10}$$

It states that, if a message has been sent on the communication channel, then the intruder can read its content and remove it from the channel.

The ability of compose messages is modelled by:

$$\mathtt{ik}(M) \centerdot \mathtt{ik}(K) \xrightarrow{encrypt(K,M)} \mathtt{ik}(M) \centerdot \mathtt{ik}(K) \centerdot \mathtt{ik}(\{M\}_K) \tag{3.11}$$

$$\mathtt{ik}(M_1) \centerdot \mathtt{ik}(M_2) \xrightarrow{pairing(M_1,M_2)} \mathtt{ik}(\langle M_1, M_2 \rangle) \tag{3.12}$$

These rules specify how the intruder performs encryption and pairing of messages, respectively.

[12] Notice that the set of rule labels L_I readily follow from the set of rewrite rules and of facts.

The messages gleaned from the observed traffic can be decomposed by means of:

$$\mathtt{ik}(\{M\}_K)\,.\,\mathtt{ik}(K^{-1}) \xrightarrow{decrypt(K,M)} \mathtt{ik}(\{M\}_K)\,.\,\mathtt{ik}(K^{-1})\,.\,\mathtt{ik}(M) \quad (3.13)$$

$$\mathtt{ik}(\langle M_1, M_2\rangle) \xrightarrow{decompose(M_1,M_2)} \mathtt{ik}(M_1)\,.\,\mathtt{ik}(M_2) \quad (3.14)$$

The first rule states that if the intruder knows both a message encrypted (in an asymmetric cryptosystem) with the key K and the decryption key K^{-1}, then the (multi)-set rewriting system can move in a state where the knowledge of the intruder is extended with the content M of the message. Similarly, the second rule states that if the intruder knows the pairing message $\langle M_1, M_2\rangle$, then the rule can be applied to reach a state in which the knowledge of the intruder is extended with the messages M_1 and M_2.

Finally, the intruder can send arbitrary messages possibly faking somebody else's identity in doing so:

$$\mathtt{ik}(M)\,.\,\mathtt{ik}(A)\,.\,\mathtt{ik}(B) \xrightarrow{fake_1(A,B,M)} \mathtt{ik}(M)\,.\,\mathtt{ik}(A)\,.\,\mathtt{ik}(B)\,.\,\mathtt{msg}(1, A, B, M) \quad (3.15)$$

$$\mathtt{ik}(M)\,.\,\mathtt{ik}(A)\,.\,\mathtt{ik}(B) \xrightarrow{fake_2(A,B,M)} \mathtt{ik}(M)\,.\,\mathtt{ik}(A)\,.\,\mathtt{ik}(B)\,.\,\mathtt{msg}(2, A, B, M) \quad (3.16)$$

$$\mathtt{ik}(M)\,.\,\mathtt{ik}(A)\,.\,\mathtt{ik}(B) \xrightarrow{fake_3(A,B,M)} \mathtt{ik}(M)\,.\,\mathtt{ik}(A)\,.\,\mathtt{ik}(B)\,.\,\mathtt{msg}(3, A, B, M) \quad (3.17)$$

The above rewrite rules, (3.10)–(3.17), are enough to model all the capabilities of the standard DY intruder. It is worth pointing out that different intruder models can be readily specified by providing a different set of rewriting rules. This contrasts with other approaches whereby the intruder is usually hardwired in the analysis tool.

3.2.3 Goals

Bad states are states whose reachability implies a violation of a security property the protocol should guarantee. Both secrecy and authentication security properties can be readily specified in this way. For instance, the reachability of a state containing the facts $\mathtt{secret}(T, A)$, and $\mathtt{ik}(T)$ but not containing $\mathtt{secret}(T, i)$ means that a secrecy property of the protocol is violated i.e. the intruder, i, knows a term, T, that he should not share with any other agent. Moreover, as said before, the reachability of a state containing a fact of the form $\mathtt{request}(R, S, O, T)$ but not containing a "matching" witness fact (i.e. $\mathtt{witness}(S, R, O, T)$) means that an authentication property of the protocol has been violated.

3.2.4 Optimisations

The model outlined in the previous section is accurate but not adequate to carry out automatic analysis in an efficient way. The main problem rests with the specification of the intruder that allows for the forging of messages that cannot be possibly be used to mount an attack on the protocol. This can be avoided by providing a refined model of the intruder that forges a message only if it can be accepted by a honest participant. Moreover it is possible to drastically reduce the number of transitions (while retaining all the possible attacks on the protocol) by merging some intruder rules with protocols steps carried out by the honest participants. In this section we present two optimisations based on the above ideas: impersonate and step compression.

Impersonate

Many of the messages forged by the intruder can be of no interest for the analysis of the protocol. In fact, as previously mentioned, we assume that honest agents only react to a message if, and only if, this message has the type and the pattern expected by the receiver (i.e. strong typing assumption).[13] For instance, the following partially instantiated rule modeling the ability of the Dolev-Yao intruder of pairing messages (see rule (3.12))

$$\mathtt{ik}(a)\,.\,\mathtt{ik}(M) \xrightarrow{pairing(a,M)} \mathtt{ik}(a)\,.\,\mathtt{ik}(M)\,.\,\mathtt{ik}(\langle a, M\rangle)$$

can be applied in the initial state of the NSPK by substituting M with a and in every successor state extending the knowledge of the intruder with messages of the form $\langle a, \langle a, \langle a, \ldots\rangle\rangle\rangle$. However, such messages are not "interesting" for the analysis of our example and, therefore, also the rules applied for composing them are useless. As a matter of fact, *diverting*, *impersonating*, and *decomposing* rules are sufficient to model the Dolev-Yao intruder and it is easy to see that this optimisation is correct as it preserves the existing attacks and does not introduce new ones, [JRV00].

Diverting (see rewrite rule (3.10)) and decomposing (see rules (3.13) and (3.14)) rules have been already presented in Section 3.2.2. Let us describe the basic idea of impersonating rules.

The observation that most of the messages generated by a strictly compliant Dolev-Yao intruder are rejected by the receiver as non-expected or ill-formed suggests to restrict these rules so that the intruder sends only messages matching the patterns expected by the receiver. For each protocol rule of the form

$$\ldots\,.\,\mathtt{msg}(J, A, B, M)\,.\,\mathtt{state}(J, A, B, Knw, S) \xrightarrow{step_J(\ldots)} \ldots$$

[13] Notice that when we focus on reachability problems with a bounded number of parallel sessions in which each honest agent executes a bounded number of steps, then the set of such "interesting" messages is finite.

and for each possible least set of messages $\{M_{1,\mathsf{l}}, \ldots, M_{\mathsf{j_l},\mathsf{l}}\}$ (let m be the number of such sets, then $\mathsf{l} = 1, \ldots, \mathsf{m}$ and $\mathsf{j_l} > 0$) from which the Dolev-Yao intruder would be able to build a message M' that unifies (and complies with the type of) M, we add a new rule of the form

$$\begin{aligned}&\mathtt{state}(J, A, B, Knw, S) \,.\, \mathtt{ik}(A) \,.\, \mathtt{ik}(B) \,.\, \mathtt{ik}(M_{1,\mathsf{l}}) \,.\, \ldots \,.\, \mathtt{ik}(M_{\mathsf{j_l},\mathsf{l}}) \\ &\quad \xrightarrow{impersonate_{J,\mathsf{l}}(\ldots)} \\ &\mathtt{msg}(J, A, B, M') \,.\, \mathtt{state}(J, A, B, Knw, S) \,.\, \mathtt{ik}(A) \,.\, \mathtt{ik}(B) \\ &\,.\, \mathtt{ik}(M_{1,\mathsf{l}}) \,.\, \ldots \,.\, \mathtt{ik}(M_{\mathsf{j_l},\mathsf{l}}) \,.\, \mathtt{ik}(M')\end{aligned}$$

This rule states that if agent B is waiting for a message M from A and the intruder is able to compose a message M' unifying (and complying with the type of) M, then the intruder can impersonate A and send M' (see for instance Example 9). This optimisation, first introduced in [JRV00], often reduces the number of rule instances in a dramatic way (for more details see [AC02]).

Example 9. *By considering the second rule of NSPK (see rule (3.6)) it is immediate to see that the message* $\{\langle A, Na\rangle\}_{Kb}$ *can be composed by the intruder provided that its knowledge contains at least one of the three sets of messages:*

$$\{\{\langle A, Na\rangle\}_{Kb}\} \tag{3.18}$$
$$\{\langle A, Na\rangle, Kb\} \tag{3.19}$$
$$\{A, Na, Kb\} \tag{3.20}$$

As a consequence three correspondent rewrite rules are generated for impersonating the initiator A in sending the first message of NSPK:

$$\begin{aligned}&\mathtt{state}(1, A, B, [B, A, Kb, Kb^{-1}, Ka], S) \,.\, \mathtt{ik}(A) \,.\, \mathtt{ik}(B) \,.\, \mathtt{ik}(\{\langle A, Na\rangle\}_{Kb}) \\ &\quad \xrightarrow{impersonate_{2,1}(A,B,Ka,Kb,Na,S)} \\ &\mathtt{state}(1, A, B, [B, A, Kb, Kb^{-1}, Ka], S) \\ &\,.\, \mathtt{msg}(1, A, B, \{\langle A, Na\rangle\}_{Kb}) \\ &\,.\, \mathtt{ik}(A) \,.\, \mathtt{ik}(B) \,.\, \mathtt{ik}(\{\langle A, Na\rangle\}_{Kb})\end{aligned} \tag{3.21}$$

$$\begin{aligned}&\mathtt{state}(1, A, B, [B, A, Kb, Kb^{-1}, Ka], S) \,.\, \mathtt{ik}(A) \,.\, \mathtt{ik}(B) \,.\, \mathtt{ik}(\langle A, Na\rangle) \,.\, \mathtt{ik}(Kb) \\ &\quad \xrightarrow{impersonate_{2,2}(A,B,Ka,Kb,Na,S)} \\ &\mathtt{state}(1, A, B, [B, A, Kb, Kb^{-1}, Ka], S) \\ &\,.\, \mathtt{msg}(1, A, B, \{\langle A, Na\rangle\}_{Kb}) \\ &\,.\, \mathtt{ik}(A) \,.\, \mathtt{ik}(B) \,.\, \mathtt{ik}(\langle A, Na\rangle) \,.\, \mathtt{ik}(Kb)\end{aligned} \tag{3.22}$$

$$\begin{array}{l}\mathtt{state}(1, A, B, [B, A, Kb, Kb^{-1}, Ka], S) \mathbin{.} \mathtt{ik}(A) \mathbin{.} \mathtt{ik}(B) \mathbin{.} \mathtt{ik}(Na) \mathbin{.} \mathtt{ik}(Kb) \\ \quad \xrightarrow{impersonate_{2,3}(A,B,Ka,Kb,Na,S)} \\ \mathtt{state}(1, A, B, [B, A, Kb, Kb^{-1}, Ka], S) \\ \mathbin{.} \mathtt{msg}(1, A, B, \{\langle A, Na\rangle\}_{Kb}) \\ \mathbin{.} \mathtt{ik}(A) \mathbin{.} \mathtt{ik}(B) \mathbin{.} \mathtt{ik}(Na) \mathbin{.} \mathtt{ik}(Kb)\end{array} \tag{3.23}$$

Intuitively rule (3.21) *can be applied any time the intruder knows the whole message* (3.18) *the responder* B *is waiting for. Similarly, rules* (3.22) *and* (3.23) *can be executed when the intruder knows messages* (3.19) *and* (3.20) *respectively from which it can promptly compose the message that* B *is waiting.*

Step Compression

A very effective, sound and complete, optimisation, called *step compression* has been proposed in [BMV03a]. It consists of the idea of merging intruder rewrite rules with honest agents rules. In particular, an impersonate rule:

$$\begin{array}{l}\mathtt{state}(J, X_1, X_2, X_3, X_4) \mathbin{.} \mathtt{ik}(X_1) \mathbin{.} \mathtt{ik}(X_2) \mathbin{.} \mathtt{ik}(X_5) \\ \quad \xrightarrow{impersonate_J(\ldots)} \\ \mathtt{msg}(J, X_1, X_2, X_5) \mathbin{.} \mathtt{state}(J, X_1, X_2, X_3, X_4) \\ \mathbin{.} \mathtt{ik}(X_1) \mathbin{.} \mathtt{ik}(X_2) \mathbin{.} \mathtt{ik}(X_5)\end{array} \tag{3.24}$$

a generic honest agent step rule:

$$\begin{array}{l}\mathtt{state}(J, Y_1, Y_2, Y_3, Y_4) \mathbin{.} \mathtt{msg}(J, Y_1, Y_2, Y_5) \\ \quad \xrightarrow{step_J(\ldots)} \\ \exists W : \mathtt{state}(K, Y_1, Y_2, Y_7, Y_4) \mathbin{.} \mathtt{msg}(J+1, Y_2, Y_1, Y_8)\end{array} \tag{3.25}$$

and an intruder divert rule:

$$\mathtt{msg}(J+1, Z_1, Z_2, Z_3) \xrightarrow{divert(\ldots)} \mathtt{ik}(Z_3) \tag{3.26}$$

can be merged together and replaced by the following step-compressed rule:

$$\begin{array}{l}\mathtt{state}(J, X_1, X_2, X_3, X_4)\sigma \mathbin{.} \mathtt{ik}(X_1)\sigma \mathbin{.} \mathtt{ik}(X_2)\sigma \mathbin{.} \mathtt{ik}(X_5)\sigma \\ \quad \xrightarrow{step\_compressed_J(\ldots)\sigma} \\ \exists W : \mathtt{state}(K, Y_1, Y_2, Y_7, Y_4)\sigma \mathbin{.} \mathtt{ik}(Z_1)\sigma \mathbin{.} \mathtt{ik}(Z_2)\sigma \mathbin{.} \mathtt{ik}(Z_3)\sigma\end{array}$$

where $\sigma = \sigma_1 \bullet \sigma_2$ with[14]

$$\sigma_1 = mgu(\{\ \mathtt{msg}(J, X_1, X_2, X_5) = \mathtt{msg}(J, Y_1, Y_2, Y_5),\\ \mathtt{state}(J, X_1, X_2, X_3, X_4) = \mathtt{state}(J, Y_1, Y_2, Y_3, Y_4)\ \})$$
$$\sigma_2 = mgu(\{\ \mathtt{msg}(J+1, Y_2, Y_1, Y_8) = \mathtt{msg}(J+1, Z_1, Z_2, Z_3)\ \})$$

The rationale of this optimisation (see for instance Example 10) is that we can safely restrict our attention to rewriting paths where (3.24), (3.25), and (3.26) are executed in this sequence without any interleaved action in between.

Example 10. *For instance, given the ground impersonate rule:*

$$\mathtt{state}(1, a, b, [b, a, kb, kb^{-1}, ka], 2) \mathbin{.} \mathtt{ik}(a) \mathbin{.} \mathtt{ik}(b) \mathbin{.} \mathtt{ik}(\{\langle a, na0\rangle\}_{kb})$$
$$\xrightarrow{impersonate_{2,1}(a,b,ka,kb,na0,2)}$$
$$\mathtt{state}(1, a, b, [b, a, kb, kb^{-1}, ka], 2)$$
$$\mathbin{.} \mathtt{msg}(1, a, b, \{\langle a, na0\rangle\}_{kb})$$
$$\mathbin{.} \mathtt{ik}(a) \mathbin{.} \mathtt{ik}(b) \mathbin{.} \mathtt{ik}(\{\langle a, na0\rangle\}_{kb})$$

the honest agent rule:

$$\mathtt{msg}(1, A, B, \{\langle A, Na\rangle\}_{Kb}) \mathbin{.} \mathtt{state}(1, A, B, [B, A, Kb, Kb^{-1}, Ka], S)$$
$$\xrightarrow{step_2(A,B,Ka,Kb,Na,S)}$$
$$\exists Nb:$$
$$\mathtt{state}(3, A, B, [Na, Nb, B, A, Kb, Kb^{-1}, Ka], S)$$
$$\mathbin{.} \mathtt{msg}(2, B, A, \{\langle Na, Nb\rangle\}_{Ka})$$
$$\mathbin{.} \mathtt{witness}(B, A, nb, Nb)$$

and the divert rule:

$$\mathtt{msg}(J, Z_1, Z_2, Z_3) \xrightarrow{divert(J,Z_1,Z_2,Z_3)} \mathtt{ik}(Z_3)$$

can be merged together in the step-compressed rule:

$$\mathtt{state}(1, a, b, [b, a, kb, kb^{-1}, ka], 2) \mathbin{.} \mathtt{ik}(a) \mathbin{.} \mathtt{ik}(b) \mathbin{.} \mathtt{ik}(\{\langle a, na0\rangle\}_{kb})$$
$$\xrightarrow{step\_compressed_1(a,b,ka,kb,na0,2)}$$
$$\exists Nb:$$
$$\mathtt{state}(3, a, b, [na0, Nb, b, a, kb, kb^{-1}, ka], 2)$$
$$\mathbin{.} \mathtt{ik}(a) \mathbin{.} \mathtt{ik}(b) \mathbin{.} \mathtt{ik}(\{\langle a, na0\rangle\}_{kb})$$
$$\mathbin{.} \mathtt{ik}(\{\langle na0, Nb\rangle\}_{ka})$$

where the substitution $\sigma = [A/a, B/b, J/2, Ka/ka, Kb/kb, Na/na0, S/2, Z_1/b, Z_2/a, Z_3/\{\langle na0, Nb\rangle\}_{ka}]$ *has been applied.*

[14]Notice that $\bullet$ is the operator for composing substitutions, and *mgu* is the most general unifier.

Table 3.1: Facts and their informal meaning

Fact	Meaning
$\mathtt{ik}(M)$	The intruder knows M.
$\mathtt{msg}(J, S, R, M)$	Principal S has supposedly sent message M to principal R at protocol step J.
$\mathtt{state}(J, S, R, [M_1, \ldots, M_{\mathsf{h}}], C)$	Principal R is ready to execute step J of session C of the protocol, knows the messages $M_1, \ldots, M_{\mathsf{h}}$, and—if $J \neq 0$—a message from S to R is awaited for the step to be executed.
$\mathtt{secret}(M, S)$	Message M is a secret shared between a group of agents containing principal S.
$\mathtt{witness}(S, R, O, M)$	Honest agent S wants to execute the protocol with agent R by using M as value for the authentication identifier O.
$\mathtt{request}(R, S, O, M)$	Honest agent R accepts the value M and now relies on the guarantee that agent S exists and agrees with him on this value for the authentication identifier O.

3.2.5 A worked out example

Here we describe in details the whole protocol insecurity problem Ξ^{NSPK} associated to the two concurrent sessions of the NSPK protocol leading to the well-known Lowe's attack. This is merely a wrapping up of the previous sections to give a global view of the underlying model.

Let $\Gamma_H^{\text{NSPK}} = \langle \mathsf{F}_H, \mathsf{L}_H, \mathsf{I}_H, \mathsf{R}_H \rangle$ and $\Gamma_I^{\text{NSPK}} = \langle \mathsf{F}_I, \mathsf{L}_I, \mathsf{I}_I, \mathsf{R}_I \rangle$ be the (multi)-set rewriting systems modelling the behaviour of the honest agents and of the intruder respectively and let B^{NSPK} be set of states whose reachability implies the violation of the authentication property that the NSPK protocol is intended to guarantee. Given this, we recall that $\Xi^{\text{NSPK}} = \langle \Gamma_H^{\text{NSPK}} \circ \Gamma_I^{\text{NSPK}}, \mathsf{B}^{\text{NSPK}} \rangle$ and that, by definition, $\Gamma_H^{\text{NSPK}} \circ \Gamma_I^{\text{NSPK}} = \langle \mathsf{F}_H \cup \mathsf{F}_I, \mathsf{L}_H \cup \mathsf{L}_I, \mathsf{I}, \mathsf{R}_H \cup \mathsf{R}_I \rangle$ where $\mathsf{I} = \{S_1 \cup S_2 : S_1 \in \mathsf{I}_H, S_2 \in \mathsf{I}_I\}$.

Facts. The facts, $\mathsf{F}_H \cup \mathsf{F}_I$, of the protocol insecurity problem Ξ^{NSPK} are formed according to the BNF rule *Fact* of Figure 3.3 and their informal semantics are summarised in Table 3.1.

Initial States. With respect to the initial states provided in Section 3.2.1 and Section 3.2.2, the set I of initial states of the NSPK example is a singleton set whose single element is given by the union of the facts (3.3), (3.4), and

(3.9), i.e.:

$$\begin{aligned}&\mathtt{state}(0, a, a, [a, i, ka, ka^{-1}, ki], 1)\\ &\mathtt{.state}(0, a, a, [a, b, ka, ka^{-1}, kb], 2)\,\mathtt{.state}(1, a, b, [b, a, kb, kb^{-1}, ka], 2)\\ &\mathtt{.ik}(i)\,\mathtt{.ik}(a)\,\mathtt{.ik}(b)\,\mathtt{.ik}(ki)\,\mathtt{.ik}(ki^{-1})\,\mathtt{.ik}(ka)\,\mathtt{.ik}(kb)\end{aligned} \tag{3.27}$$

Rewrite Rules. Figure 3.5 and Figure 3.6 depict the rewrite rules, $\mathsf{R}_H \cup \mathsf{R}_I$,[15] of the protocol insecurity problem Ξ^{NSPK}. These have been already explained in the previous Section 3.2.1 and Section 3.2.2.

Goals. A successful execution of the NSPK protocol should give evidence to A and B that they talked to each other. Such an authentication property is violated by any state containing a fact of the form $\mathtt{request}(R, S, O, T)$, but not containing a matching witness fact, i.e. $\mathtt{witness}(S, R, O, T)$. This set of "bad" states, B^{NSPK}, is represented symbolically by the propositional formula:

$$\begin{aligned}&\vee \quad \big(\mathtt{request}(A, B, nb, Nb) \wedge \neg\, \mathtt{witness}(B, A, nb, Nb)\big)\\ &\vee \quad \big(\mathtt{request}(B, A, na, Na) \wedge \neg\, \mathtt{witness}(A, B, na, Na)\big)\end{aligned} \tag{3.28}$$

where A and B are the initiator and the responder respectively, and Na and Nb are the authentication values used for the authentication identifiers na and nb respectively.

It is easy to see that the application of the sequence of rules shown in Figure 3.4 leads from the initial state specified by the facts in (3.27) to a bad state satisfying the goal predicate (3.28) and it is, therefore, a solution to the protocol insecurity problem Ξ^{NSPK}.

Let us describe each step of the attack trace depicted in Figure 3.4 in more details. The execution of rule (3.29) represents agent a that sends the first message of the NSPK to the intruder (her intended communication partner in session 1), i.e.:

$$(1.1)\;\; a \rightarrow I : \{a, na0\}_{ki}$$

The application of rules (3.30), (3.31), and (3.32) models the intruder that firstly receives the message from the network, then extends its knowledge with the pair $\langle a, na0\rangle$ by decrypting the message just received, and finally starts a new session of the NSPK with the agent b impersonating the identity of a, i.e.:

$$(2.1)\;\; i(a) \rightarrow b : \{a, na0\}_{kb}$$

[15] Notice that the set of rule labels $\mathsf{L}_H \cup \mathsf{L}_I$ readily follow from the set of rewrite rules and of facts.

1. $step_1(a, i, ka, ki, 1)$, (3.29)
2. $divert(a, i, 1, \{\langle a, na0\rangle\}_{ki})$, (3.30)
3. $decrypt(ki, \langle a, na0\rangle)$, (3.31)
4. $impersonate_{2,2}(a, b, ka, kb, na0, 2)$, (3.32)
5. $step_2(a, b, ka, kb, na0, 2)$, (3.33)
6. $divert(b, a, 2, \{\langle na0, nb0\rangle\}_{ka})$, (3.34)
7. $impersonate_{3,1}(a, i, ka, ki, na0, nb0, 1)$, (3.35)
8. $step_3(a, i, ka, ki, na0, nb0, 1)$, (3.36)
9. $divert(a, i, 3, \{nb0\}_{ki})$, (3.37)
10. $decrypt(ki, nb0)$, (3.38)
11. $impersonate_{4,2}(a, b, ka, kb, na0, nb0, 2)$ (3.39)
12. $step_4(a, b, ka, kb, na0, nb0, 2)$ (3.40)

Figure 3.4: NSPK protocol insecurity problem: attack trace

According to the protocol, agent b reacts to the above message and, believing it comes from a, he sends the following message to a:

$$(2.2)\ \ b \rightarrow a : \{na0, nb0\}_{ka}$$

This is represented by the execution of rule (3.33). The intruder diverts and learns such a message (see rule (3.34)) that is then forwarded without any change to a (see rule (3.35)), i.e.:

$$(1.2)\ \ i \rightarrow a : \{na0, nb0\}_{ka}$$

The application of rule (3.36) models agent a that, by providing to the intruder the nonce $nb0$ (i.e.,the challenge generated by b in session 2), acts unconsciously as an allied of the intruder, i.e.:

$$(1.3)\ \ a \rightarrow i : \{nb0\}_{ki}$$

For the intruder it is now trivial to violate the NSPK authentication requirement by simply receiving and decrypting the message $\{nb0\}_{ki}$ (see rules (3.37) and (3.38) respectively) and impersonating a in sending (see rule (3.39)) to b the following message:

$$(2.3)\ \ i(a) \rightarrow b : \{nb0\}_{kb}$$

This last step is modelled by the execution of rules (3.39) and (3.40) that leads in a state that contains the fact $\texttt{request}(b, a, na, na0)$ but not the

matching witness fact $\texttt{witness}(a, b, na, na0)$. This is due to the fact that a has not generated the nonce $na0$ to be used as authentication value for communicating with b in session 2, but with the purpose to be used as authentication value for communicating with i in session 1 (this is witnessed by the presence of the fact $\texttt{witness}(a, i, na, na0)$ in all the states reached after the application of the first rule (3.29)). It is immediate to see that the attack trace just presented exactly corresponds to the Lowe's attack on the NSPK described in Section 3.1.2.

It is worth pointing out that by applying the step compression optimisation (see Section 3.2.4) the number of rules to be executed in order to achieve the same attack falls from 12 to 7: in fact the four sequences of rules (3.29)-(3.30), (3.32)-(3.34), (3.35)-(3.37), and (3.39)-(3.40) can be collapsed into four correspondent step-compressed rules.

3.3 (Un-)Decidability of Security Protocols

A protocol insecurity problem models the question of whether a protocol indeed achieves its security requirements or not. In the sequel we will refer to this class of problem as $\mathcal{P}$. We have already mentioned that, in the general case, such class of problems $\mathcal{P}$ is undecidable [DLMS99, EG83, AL00]. This can be easily proved by showing that a well-known undecidable problem (e.g. the Post Correspondence Problem, the halting problem for two-counter Turing machines, etc) can be reduced to a protocol insecurity problem (see Section 3.3.1).

Despite this strong undecidability result, the problem of deciding whether a protocol is correct or not it is still worthwhile to be tackled. On the one hand, there exist semi-decision techniques which can be automated in various ways to deal with problems in $\mathcal{P}$. On the other hand, it is possible to classify problems in $\mathcal{P}$ through a set of main parameters (see Section 3.3.2) over which the introduction of some restrictions can lead to identify decidable subclasses of $\mathcal{P}$ (see Section 3.3.3).

3.3.1 Undecidability in the general case

In this section we describe how a well-known undecidable decision problem, namely the *Post Correspondence Problem*, can be expressed as a problem in $\mathcal{P}$ and specifically as a protocol insecurity problem.

Post Correspondence Problems (PCP)

The post correspondence problem was introduced by Emil Post in 1946 [Pos46]. Informally the problem can be described as follows: given a dictionary that consists of two arrays of words where the two arrays have the same length,

$$\texttt{state}(0, A, A, [A, B, Ka, Ka^{-1}, Kb], S)$$
$$\xrightarrow{step_1(A,B,Ka,Kb,S)}$$
$$\exists Na :$$
$$\texttt{state}(2, B, A, [Na, A, B, Ka, Ka^{-1}, Kb], S)$$
$$\texttt{.msg}(1, A, B, \{\langle A, Na\rangle\}_{Kb})$$
$$\texttt{.witness}(A, B, na, Na)$$

$$\texttt{msg}(1, A, B, \{\langle A, Na\rangle\}_{Kb})\ \texttt{.state}(1, A, B, [B, A, Kb, Kb^{-1}, Ka], S)$$
$$\xrightarrow{step_2(A,B,Ka,Kb,Na,S)}$$
$$\exists Nb :$$
$$\texttt{state}(3, A, B, [Na, Nb, B, A, Kb, Kb^{-1}, Ka], S)$$
$$\texttt{.msg}(2, B, A, \{\langle Na, Nb\rangle\}_{Ka})$$
$$\texttt{.witness}(B, A, nb, Nb)$$

$$\texttt{msg}(2, B, A, \{\langle Na, Nb\rangle\}_{Ka})\ \texttt{.state}(2, B, A, [Na, A, B, Ka, Ka^{-1}, Kb], S)$$
$$\xrightarrow{step_3(A,B,Ka,Kb,Na,Nb,S)}$$
$$\texttt{state}(4, B, A, [Nb, Na, A, B, Ka, Ka^{-1}, Kb], S)$$
$$\texttt{.msg}(3, A, B, \{Nb\}_{Kb})$$
$$\texttt{.request}(A, B, nb, Nb)$$

$$\texttt{msg}(3, A, B, \{Nb\}_{Kb})\ \texttt{.state}(3, A, B, [Na, Nb, B, A, Kb, Kb^{-1}, Ka], S)$$
$$\xrightarrow{step_4(A,B,Ka,Kb,Na,Nb,S)}$$
$$\texttt{state}(5, A, B, [Na, Nb, B, A, Kb, Kb^{-1}, Ka], S)$$
$$\texttt{.request}(B, A, na, Na)$$

Figure 3.5: NSPK protocol insecurity problem: honest agent rules

$$\texttt{state}(1, A, B, [B, A, Kb, Kb^{-1}, Ka], S) \,.\, \texttt{ik}(\{\langle A, Na\rangle\}_{Kb}) \xrightarrow{impersonate_{2,1}(A,B,Ka,Kb,Na,S)} \texttt{state}(1, A, B, [B, A, Kb, Kb^{-1}, Ka], S) \,.\, \texttt{msg}(1, A, B, \{\langle A, Na\rangle\}_{Kb}) \,.\, \texttt{ik}(\{\langle A, Na\rangle\}_{Kb})$$

$$\texttt{state}(1, A, B, [B, A, Kb, Kb^{-1}, Ka], S) \,.\, \texttt{ik}(\langle A, Na\rangle) \,.\, \texttt{ik}(Kb) \xrightarrow{impersonate_{2,2}(A,B,Ka,Kb,Na,S)} \texttt{state}(1, A, B, [B, A, Kb, Kb^{-1}, Ka], S) \,.\, \texttt{msg}(1, A, B, \{\langle A, Na\rangle\}_{Kb}) \,.\, \texttt{ik}(\langle A, Na\rangle) \,.\, \texttt{ik}(Kb)$$

$$\texttt{state}(1, A, B, [B, A, Kb, Kb^{-1}, Ka], S) \,.\, \texttt{ik}(A) \,.\, \texttt{ik}(Na) \,.\, \texttt{ik}(Kb) \xrightarrow{impersonate_{2,3}(A,B,Ka,Kb,Na,S)} \texttt{state}(1, A, B, [B, A, Kb, Kb^{-1}, Ka], S) \,.\, \texttt{msg}(1, A, B, \{\langle A, Na\rangle\}_{Kb}) \,.\, \texttt{ik}(A) \,.\, \texttt{ik}(Na) \,.\, \texttt{ik}(Kb)$$

$$\texttt{state}(2, B, A, [Na, A, B, Ka, Ka^{-1}, Kb], S) \,.\, \texttt{ik}(\{\langle Na, Nb\rangle\}_{Ka}) \xrightarrow{impersonate_{3,1}(A,B,Ka,Kb,Na,Nb,S)} \texttt{state}(2, B, A, [Na, A, B, Ka, Ka^{-1}, Kb], S) \,.\, \texttt{msg}(2, B, A, \{\langle Na, Nb\rangle\}_{Ka}) \,.\, \texttt{ik}(\{\langle Na, Nb\rangle\}_{Ka})$$

$$\vdots \qquad \vdots \qquad \vdots$$

$$\texttt{state}(3, A, B, [Na, Nb, B, A, Kb, Kb^{-1}, Ka], S) \,.\, \texttt{ik}(Nb) \,.\, \texttt{ik}(Kb) \xrightarrow{impersonate_{4,2}(A,B,Ka,Kb,Na,Nb,S)} \texttt{state}(3, A, B, [Na, Nb, B, A, Kb, Kb^{-1}, Ka], S) \,.\, \texttt{msg}(3, A, B, \{Nb\}_{Kb}) \,.\, \texttt{ik}(Nb) \,.\, \texttt{ik}(Kb)$$

$$\texttt{msg}(I, A, B, M) \xrightarrow{divert(A,B,I,M)} \texttt{ik}(M)$$

$$\texttt{ik}(\{M\}_K) \,.\, \texttt{ik}(K^{-1}) \xrightarrow{decrypt(K,M)} \texttt{ik}(\{M\}_K) \,.\, \texttt{ik}(K^{-1}) \,.\, \texttt{ik}(M)$$

$$\texttt{ik}(\langle M_1, M_2\rangle) \xrightarrow{decompose(M_1,M_2)} \texttt{ik}(M_1) \,.\, \texttt{ik}(M_2)$$

Figure 3.6: NSPK protocol insecurity problem: intruder rules

but the words in it can be of different length, decide if there is a *correspondence* between the arrays, i.e. there exists a list of indices which tell how to arrange the elements of the arrays such that both give the same sentence. Formally, a PCP is defined in terms of an indexed finite set of pairs of words (strings) $\{\langle U_1, V_1\rangle, \ldots, \langle U_m, V_m\rangle\}$ and concerns with the question of whether there exists a sequence of indices $i_1, \ldots, i_n$ ($1 \leq i_j \leq m$ for each $j = 1, \ldots, n$) so that the concatenation[16] of the strings $U_{i_1}, \ldots, U_{i_n}$ is equal to that of the strings $V_{i_1}, \ldots, V_{i_n}$ (see, for instance, Example 11).

Example 11. *Given a dictionary of four pairs of words:*

	1	2	3	4
U_j	aba	bbb	aab	bb
V_j	a	aaa	$abab$	$babba$

It is immediate to see that the sequence of indices $1, 4, 3, 1$ *is a solution to the PCP associated to the above dictionary. In fact:*

$$\begin{array}{lcccl} U_1, U_4, U_3, U_1 & = & aba \cdot bb \cdot aab \cdot aba & = & ababbaababa \\ V_1, V_4, V_3, V_1 & = & a \cdot babba \cdot abab \cdot a & = & ababbaababa \end{array}$$

Of course the sequence of indices $1, 2, 3$ *does not correspond to a solution of PCP as the following two lines show:*

$$\begin{array}{lcccl} U_1, U_2, U_3 & = & aba \cdot bbb \cdot aab & = & ababbbaab \\ V_1, V_2, V_3 & = & a \cdot aaa \cdot abab & = & aaaaabab \end{array}$$

PCP as a PIP

A generic PCP $\{\langle U_1, V_1\rangle, \ldots, \langle U_m, V_m\rangle\}$ can be easily reduced to the problem of deciding whether a secret message N can be learnt by the DY intruder when the security protocol depicted in Figure 3.7 is played. In such a proto-

Initiator:
  send $\{\epsilon, \epsilon\}_K$
Word Constructor j**-th**: (honest agent appends pair $\langle U_j, V_j\rangle$)
  receive $\{X, Y\}_K$
  send $\{X \cdot U_j, Y \cdot V_j\}_K$
Compromiser:
  receive $\{X, X\}_K$ if $X \neq \epsilon$
  send N

Figure 3.7: PCP as a Security Protocol

[16]The operator $\cdot$ indicates strings concatenation.

col $m+2$ honest agents—namely an *initiator*, a *compromiser*, and m *word constructors*—all sharing a secret key K (the intruder does not know such a key) simulate, by exchanging messages, the construction of the two sentences in the PCP. Intuitively, the initiator starts the protocol by sending over the network a message containing a pair of empty strings.[17] Then any of the m word constructors (let us say the j-th) can retrieve the current content of the pair of strings and append to them its pair of words ($\langle U_j, V_j \rangle$). Such a step can be repeated infinitely-many times. If the strings on the network are identical, the compromiser can send the secret message N in clear over the network allowing the intruder to violate the secrecy requirement. It must be noted that since the intruder does not know the shared key K, then it cannot never alterate the content of the pair of strings constructed by the honest agents. The intruder will just observe passively the protocol execution hoping in and waiting for the message N from the compromiser. It is simple to prove that any attack trace on the protocol, corresponds to a sequence of indices solution of the PCP and vice versa.

Let $\mathcal{C}$ the finite set of constants used to construct the words in the dictionary,[18] the above protocol and its security requirement can be recast into a protocol insecurity problem as follows:

Initial State. One initial state is sufficient to describe the PCP. Namely,

$$\begin{aligned}
&\mathtt{state}(0, \mathit{initiator}, \mathit{initiator}, [k, \mathit{empty}], 1)\\
&\mathtt{state}(1, \mathsf{W}_1, \mathsf{W}_1, [k, \mathsf{n}(\mathsf{U}_1), \mathsf{n}(\mathsf{V}_1)], 1)\\
&\quad\vdots\\
&\mathtt{state}(\mathsf{m}, \mathsf{W}_\mathsf{m}, \mathsf{W}_\mathsf{m}, [k, \mathsf{n}(\mathsf{U}_\mathsf{m}), \mathsf{n}(\mathsf{V}_\mathsf{m})], 1)\\
&\mathtt{state}(\mathsf{m}+1, c, c, [k, n], 1)
\end{aligned}$$

where m represents the number of pairs in the PCP dictionary, $\langle \mathsf{U}_j, \mathsf{V}_j \rangle$ represents an entry of the dictionary ($j = 1, \ldots, \mathsf{m}$), the agent W_j indicates a word constructor ($j = 1, \ldots, \mathsf{m}$), the agent c is the compromiser, k is the secret key shared between all the honest agents, and n is a normalisation function that transforms any string into its (canonical) normal form (see Example 12) so that the question of deciding whether two strings are identical or not can be reduced to a simple syntactic check.[19]

Example 12. *Let $\cdot$ and n be the non-interpreted concatenation function and the normalisation meta-function respectively. Given the string composed by*

[17] The empty string is denoted by the ϵ symbol.

[18] Notice that the empty string ϵ is not in $\mathcal{C}$.

[19] Notice that, the normalisation function is a meta-function since during the PCP to PIP reduction phase any term constructed via n is opportunely replaced by a term constructed by means of the non-interpreted concatenation function "$\cdot$".

the sequence of constants a, b, c, and d, the following holds:

$$\begin{aligned}\cdot(a,\cdot(b,\cdot(c,d))) &= \mathsf{n}(\cdot(\cdot(a,\cdot(b,c)),d))\\ &= \mathsf{n}(\cdot(a,\cdot(\cdot(b,c),d)))\\ &= \mathsf{n}(\cdot(\cdot(a,b),\cdot(c,d)))\\ &= \mathsf{n}(\cdot(\cdot(\cdot(a,b),c),d))\\ &= \mathsf{n}(\cdot(a,\cdot(b,\cdot(c,d))))\end{aligned}$$

Rewrite Rules. The rewrite rules that describe the behaviour of the honest agents—initiator, compromiser, and m word constructors—of Figure 3.7 are depicted in Figure 3.8. Notice that the protocol to be modelled exchanges

$$\begin{array}{l}\mathtt{state}(0, initiator, initiator, [k,\epsilon], 1)\\ \quad\xrightarrow{start}\\ \mathtt{state}(0, initiator, initiator, [k,\epsilon], 1)\\ \mathtt{.msg}(0, initiator, initiator, \{\langle\epsilon,\epsilon\rangle\}_k)\end{array}$$

$$\begin{array}{l}\mathtt{state}(J,W,W,[k,U,V],1)\,\mathtt{.msg}(0,T,T,\{\langle X,Y\rangle\}_k)\\ \quad\xrightarrow{step\_init_J(J,T,U,V,W,X,Y)}\\ \mathtt{state}(J,W,W,[tba,k,U,V],1)\\ \mathtt{.tba}(1,X,U,\epsilon)\,\mathtt{.tba}(2,Y,V,\epsilon)\end{array}$$

$$\begin{array}{l}\mathtt{state}(J,W,W,[tba,k,U,V],1)\\ \mathtt{.tba}(1,\epsilon,\epsilon,XU)\,\mathtt{.tba}(2,\epsilon,\epsilon,YV)\\ \quad\xrightarrow{step\_end_J(J,W,U,V,XU,YV)}\\ \mathtt{state}(J,W,W,[k,U,V],1)\\ \mathtt{.msg}(0,W,W,\{\langle XU,YV\rangle\}_k)\end{array}$$

$$\begin{array}{l}\mathtt{state}(\mathsf{m}+1,c,c,[k,n],1)\,\mathtt{.msg}(0,T,T,\{\langle X,X\rangle\}_k)\\ \quad\xrightarrow{compromise(T,X)}\\ \mathtt{state}(\mathsf{m}+1,c,c,[k,n],1)\\ \mathtt{.msg}(1,c,c,n)\\ \mathtt{.secret}(n,c)\end{array}$$

Figure 3.8: PCP as a PIP: honest agents rules

a set of messages where the intended receiver is not specified. The basic idea is that any message is sent in broadcast to all the participants. Any of them

can then get the message and reply to it. In terms of notation we represent this situation by imposing sender and receiver to be equal in each of the facts `msg` and `state` (e.g. the fact $\mathtt{msg}(0, initiator, initiator, \{\langle\epsilon, \epsilon\rangle\}_k)$ indicates the initiator sending in broadcast a pair of empty strings).

More important, it must be noted that even if strings U and X are in normal form, the string $\cdot(X, U)$ may not be in normal form and therefore a simple syntactic check would not be sufficient to establish whether $\cdot(X, U)$ and $\cdot(Y, V)$ are identical or not. This is why the action of the J-th word constructor, say W, is carried out in three phases. Firstly W gets from the network the current content of the pair of strings (X and Y) and prepares itself to append X and Y to U and V (its pair of words) respectively (see $step\_init_J$). The facts of the form $\mathtt{tba}(I, X, U, \epsilon)$ (acronym for "to be appended") are appositely introduced in the model to indicate that X is going to be appended to U in order to construct, in normal form, part I ($I = 1, 2$) of the pair of strings that W will send on the network. Such a construction represents the second phase of W and it is done by means of the rewrite rules pictured in Figure 3.9.[20] In there C represents any constant in $\mathcal{C}$ and Z indicates any string except ϵ, i.e. $\mathsf{Z} \in \mathcal{C} \cup \{\cdot(T_1, T_2)\}$. At the end of this second phase the two facts $\mathtt{tba}(1, \epsilon, \epsilon, \mathtt{n}(\cdot(X, U)))$ and $\mathtt{tba}(2, \epsilon, \epsilon, \mathtt{n}(\cdot(Y, V)))$ are in the state of the transition system. This enables the last phase of W, i.e. $step\_end_J$, to be be applied: the message containing the pair of strings just constructed is sent on the network and the state of W is updated (namely the term *tba* is removed from the knowledge of W to indicate that the construction phase has been completed) so that W is again ready to repeat its action.

For what concerns the DY intruder we can use the generic rewrite rules presented in Section 3.2.2.

Goal States. The goal states are given by all the states satisfying the predicate $\mathtt{secret}(n, c) \wedge \mathtt{ik}(n)$, i.e., the intruder learns the secret term n, while only c should be allowed to know n.

Example 13 shows the protocol insecurity problem corresponding to the PCP presented in Example 11.

Example 13. *Let consider the PCP described in Example 11. To simplify notation (when clear from the context) we denote a string in normal form* $\cdot(a_1, \cdot(a_2, \cdot(\ldots, a_j)))$ *simply as* $a_1a_2 \cdots a_j$*. The corresponding PIP is trivially*

[20] Notice that during the normalisation phase the input strings are inverted. This is why before to be returned they are re-inverted by means of the rules *invert0*, *invert1*, and *invert2*.

$$\mathtt{tba}(I, \cdot(X_1, X_2), U, \epsilon) \xrightarrow{clp01(I,X_1,X_2,U)} \mathtt{tba}(I, X_2, U, X_1)$$

$$\mathtt{tba}(I, \mathsf{C}, U, \epsilon) \xrightarrow{clp02(I,U)} \mathtt{tba}(I, \epsilon, U, \mathsf{C})$$

$$\mathtt{tba}(I, \cdot(X_1, X_2), U, \mathsf{Z}) \xrightarrow{clp11(I,U,X_1,X_2,\mathsf{Z})} \mathtt{tba}(I, X_2, U, \cdot(X_1, \mathsf{Z}))$$

$$\mathtt{tba}(I, \mathsf{C}, U, \mathsf{Z}) \xrightarrow{clp12(I,U,\mathsf{Z})} \mathtt{tba}(I, \epsilon, U, \cdot(\mathsf{C}, \mathsf{Z}))$$

$$\mathtt{tba}(I, \epsilon, \cdot(U_1, U_2), \epsilon) \xrightarrow{crp01(I,U_1,U_2)} \mathtt{tba}(I, \epsilon, U_2, U_1)$$

$$\mathtt{tba}(I, \epsilon, \mathsf{C}, \epsilon) \xrightarrow{crp02(I)} \mathtt{tba}(I, \epsilon, \epsilon, \mathsf{C})$$

$$\mathtt{tba}(I, \epsilon, \cdot(U_1, U_2), \mathsf{Z}) \xrightarrow{crp11(I,U_1,U_2,\mathsf{Z})} \mathtt{tba}(I, \epsilon, U_2, \cdot(U_1, \mathsf{Z}))$$

$$\mathtt{tba}(I, \epsilon, \mathsf{C}, \mathsf{Z}) \xrightarrow{crp12(I,\mathsf{Z})} \mathtt{tbi}(I, \cdot(\mathsf{C}, \mathsf{Z}), \epsilon)$$

$$\mathtt{tbi}(I, \cdot(X_1, X_2), \epsilon) \xrightarrow{invert0(I,X_1,X_2)} \mathtt{tbi}(I, X_2, X_1)$$

$$\mathtt{tbi}(I, \cdot(X_1, X_2), \mathsf{Z}) \xrightarrow{invert1(I,X_1,X_2,\mathsf{Z})} \mathtt{tbi}(I, X_2, \cdot(X_1, \mathsf{Z}))$$

$$\mathtt{tbi}(I, \mathsf{C}, \mathsf{Z}) \xrightarrow{invert2(I,\mathsf{Z})} \mathtt{tba}(I, \epsilon, \epsilon, \cdot(\mathsf{C}, \mathsf{Z}))$$

Figure 3.9: PCP as a PIP: concatenation rules

constructed as described above where the initial state is given by:

$$\begin{aligned}
&\texttt{state}(0, initiator, initiator, [k, \epsilon], 1)\\
&\texttt{state}(1, w1, w1, [k, aba, a], 1)\\
&\texttt{state}(2, w2, w2, [k, bbb, aaa], 1)\\
&\texttt{state}(3, w3, w3, [k, aab, abab], 1)\\
&\texttt{state}(4, w4, w4, [k, bb, babba], 1)\\
&\texttt{state}(5, c, c, [k, n], 1)
\end{aligned}$$

The application of the following sequence of rewrite rules leads to a secrecy violation of the message n:[21]

$$\begin{aligned}
0.\quad & start,\\
1.\quad & step_init_1(1, initiator, aba, a, w1, \epsilon, \epsilon),\\
& \ldots\\
& step_end_1(1, w1, aba, a, aba, a),\\
2.\quad & step_init_4(4, w1, bb, babba, w4, aba, a),\\
& \ldots\\
& step_end_4(4, w4, bb, babba, ababb, ababba),\\
3.\quad & step_init_3(3, w4, aab, abab, w3, ababb, ababba),\\
& \ldots\\
& step_end_3(3, w3, aab, abab, ababbaab, ababbaabab),\\
4.\quad & step_init_1(1, initiator, aba, a, w1, ababbaab, ababbaabab),\\
& \ldots\\
& step_end_1(1, w1, aba, a, ababbaababa, ababbaababa),\\
6.\quad & compromise(w1, ababbaababa),\\
7.\quad & divert(c, c, 1, n)
\end{aligned}$$

As previously showed, the Post Correspondence Problem that is well-known to be undecidable can be reduced to a protocol insecurity problem. Therefore we conclude that the protocol insecurity problem is in the general case undecidable too.

Despite this result, protocol insecurity problems are still worthwhile to be tackled. In fact it is possible to devise a semi-decision procedure which will always find an attack—under the Dolev-Yao model—in finite time if an attack exists and may not terminate if the protocol is correct. This can be done by simply enumerating and exploring all traces of the protocol's state transition system looking for a violation to some of the requirements.

The SAT-based Model-Checking approach proposed in this Thesis (see Chapter 4) falls in this class of semi-decision procedures for the analysis of security protocols.

[21] To simplify the presentation the rules applied to concatenate in normal form the strings are not showed.

3.3.2 High-level Parameters of Security Protocols

As a matter of fact the problem of establishing whether a security protocol is correct or not is undecidable in general, because in all existing models its analysis required the exploration of an infinite number of states. There are several reasons for this unboundedness:

Fresh Terms. A protocol insecurity problem can allow for the generation of infinitely-many fresh terms. For instance, this is the case when a rule that generates fresh terms can be applied an unbounded number of times (see Example 14).

Messages. There can be infinitely many different possible messages that can occur in the protocol insecurity problem. This is for instance the case of the protocol insecurity problem that expresses the Post Correspondence Problem (see Section 3.3.1).

Steps. The number of steps that an honest agent can perform to execute a session of the protocol can be unbounded. This is for instance the case when the protocol insecurity problem defines loops that the agent can repeat an unbounded number of times (see again Example 14 where the server does not change its state after having executed its step).

Sessions. The number of parallel sessions that the agents can execute may be unbounded; although each initial state is a finite set of ground terms, there can be rules that create new state-facts that correspond to new sessions of agents in their initial configuration (see Example 15).

Agents. The number of agents may be unbounded.

It is worth pointing out that the above sources of unboundedness are not totally independent. For instance, if the number of fresh terms is unbounded, then also the number of messages is not bounded. Similarly, an unbounded number of sessions of a protocol in which at least one nonce is generated implies that the number of fresh terms and of messages is unbounded.

Example 14. *Imagine a state-less server that simply provides a new symmetric key Kab at each incoming requests from A willing to communicate*

with B. This can be trivially modelled by the following rewrite rule:

$$\begin{array}{l}\texttt{state}(0, S, S, [Ks, inv(Ks)], 1) \,.\, \texttt{msg}(1, A, S, \{\langle A, \langle B, Ka\rangle\rangle\}_{Ks}) \\ \quad \xrightarrow{step(A,B,Ka,Ks,S)} \\ \exists Kab : \\ \quad \texttt{state}(0, S, S, [Ks, inv(Ks)], 1) \\ \quad .\,\texttt{msg}(2, S, A, \{\langle Kab, \langle A, B\rangle\rangle\}_{Ka})\end{array}$$

in which a server generates a fresh symmetric key, sends the key to the requester agent A, and stutters in its own state.

Example 15. *Imagine a group of agents that play a protocol in which besides other functionalities a group member can allow one friend to join the group. This can be modelled by means of the following rule:*

$$\begin{array}{l}\texttt{state}(0, M, M, [M, K], 1) \\ \quad \xrightarrow{add\_friend(F,K,M)} \\ \texttt{state}(0, M, M, [M, K], 1) \\ .\,\texttt{msg}(1, M, M, \{\langle join, friend(M)\rangle\}_K) \\ .\,\texttt{state}(0, friend(M), friend(M), [friend(M), K], 1)\end{array}$$

in which the group member M (i) *stutters in its state,* (ii) *sends a broadcast message where it announces to the other members that its friend $friend(M)$*[22] *is joining the group, and* (iii) *initialise a new state-fact for its friend so that also the friend can start playing the protocol (notice that in doing this M provides to its friend the group key K).*

3.3.3 (Un-)Decidability over the Parameter Space

In the last few years there has been a lot of work done towards (un-)decidability results (see [CS02] for a survey). The basic idea is to constrain some of the above protocol parameters to identify decidable subclasses of $\mathcal{P}$. The main results are summarised in Table 3.2. The two columns "With fresh terms" and "Without fresh terms" consider the case of whether the problem allows for the generation of some fresh term during execution (i.e. there is at least one path on which a rewrite rule having some existential variables in the RHS is executed) or not. The three rows of the table consider whether:

1. there is no bound over the parameter space, or
2. the problem allows for *(i)* an unbounded number of protocol sessions, and *(ii)* messages of bounded length (depth), or

[22] The function $friend$ is not interpreted and simply represents who is the friend of who.

Table 3.2: (Un-)Decidability Overview

	With fresh terms	Without fresh terms
No bounds	Undecidable	Undecidable
Unbounded #sessions + bounded depth messages	Undecidable	DEXPTIME-compl.
Bounded #sessions + bounded #steps + unbounded depth messages	NP-complete	NP-complete

3. the problem allows for *(i)* a bounded number of protocol sessions, *(ii)* a bounded number of steps performed by each role, and *(iii)* messages of unbounded depth.

The first condition to be respected for obtaining decidable subclasses of $\mathcal{P}$ consists in bounding the number of fresh terms that the protocol execution can generate. Otherwise, the ability to generate infinitely-many fresh terms could be used to simulate arbitrarily many memory locations and therefore encode machines with unbounded memory [DLMS99]. In order to bound the number of fresh terms either *(a)* the problem consists of an unbounded number of sessions such that each session execution does not generate any fresh term, or *(b)* the problem models a bounded number of sessions in which every role can generate fresh terms during its execution, but performs a bounded number of steps.

Bounding the number of fresh terms is not enough to ensure decidability for those problems in $\mathcal{P}$ that fall in the subclass defined by *(a)*. In fact depending on whether there is a bound on the message depth or not, such a subclass of problems is decidable or not, and when it is decidable, it belongs to DEXPTIME-complete[23] complexity class, see [DLMS02].

Even in presence of unbounded-depth messages, a decidability result can be proved for those problems in $\mathcal{P}$ that belong to the subclass defined by *(b)*. Namely such a subclass of problems belongs to the NP-complete[24] complexity class, as proved in [RT01]. The main result of [RT01] is a polynomial bound $p(n)$ on the number of rules that may be needed in order to realize the attack, where n is merely the number of subterms occurring in any possible message exchanged between the honest agents over the bounded scenario. Basically the authors exploit such a polynomial bound to prove the NP-membership of the problem, while the NP-hardness is proven by

[23]Notice that, the complexity class DEXPTIME is the set of all decision problems solvable by a deterministic Turing machine in $O(2^{p(n)})$ time, where $p(n)$ is a polynomial function of the problem size n.

[24]The complexity class NP (Non-deterministic Polynomial time) is the set of all decision problems solvable in polynomial time on a non-deterministic Turing machine.

means of a direct reduction from the well-known 3-SAT problem.[25] In the sequel we will refer to a protocol insecurity problem (PIP) belonging to this NP-complete class as NPC PIP.

3.4 NPC Protocol Insecurity Problems

Although the general verification problem is undecidable, for many protocols, verification can be reduced to verification of a bounded number of sessions. Moreover, even for those protocols that should be checked under a unbounded number of concurrent protocol executions, violations in their security requirements often exploit only a small number of sessions. Besides this, when the number of sessions is bounded, cycles (if any) in the execution of honest agents may be replaced by a fixed number of steps representing some iterations of the loop. For these reasons, in many cases of interest it is sufficient to consider a finite number of sessions in which each agent performs a fixed number of steps. In such situations, the correspondent protocol insecurity problem belongs to the NP-complete complexity class. For instance all the attacks on the well-know Clark-Jacob's library [CJ97] can be discovered by modelling each protocol as a NPC PIP with only two protocol sessions.

It is thus important to devise approaches for analysing NPC protocol insecurity problems. The SAT-based Model-Checking approach presented in this Thesis (see Chapter 4) provides a semi-decision procedure for the analysis of general protocol insecurity problems. However the core and the nature of the SAT-based Model-Checking technique—SAT is the traditional and first known NP-complete problem—candidates it as a possible successful approach for the analysis of NPC protocol insecurity problems.

Moreover, it is important to identify conditions that guarantee that a protocol insecurity problem is NP-complete. A simple method is based on the idea of defining a "light" version of the PIP where only `state`-facts and honest agents are considered, and then to require that following two statements are satisfied:

(NPC1) each rewrite rule in the "light" PIP is such that at most one `state`-fact occurs in the left-hand-side and at most one in the right-hand-side. Moreover if `state`-fact are on both sides of the rule, then these `state`-facts represent the state of the same honest agent involved in the same session. In other words, each rule models a single honest agent playing one of its steps in its session and such an execution is independent from the other agents' state.

[25] The 3-SAT problem is a version of the satisfiability problem (see Section 2.3) in conjunctive normal form with exactly three literals per clause.

(NPC2) any graph corresponding to the evolution of a single honest agent (each `state`-fact in one of the initial states represents the initial state of a single honest agent) in the "light" PIP is a DAG (directed acyclic graph) such that any path is characterised by at most one instance of each of the rules in the "light" PIP.

Experimental evidence on the effectiveness of the SAT-based Model-Checking approach against NPC Protocol Insecurity Problems is given in Chapter 7.

Chapter 4

SAT-based Model-Checking of Security Protocols

In this chapter we investigate how the problem of determining whether a protocol achieves its security requirements (e.g. secrecy, authentication) can be reduced to a sequence of propositional satisfiability problems (SAT) thereby enabling the use of state-of-the-art SAT solvers for discovering attacks on security protocols.

The basic points to be addressed are two: *(i)* security protocols and their requirements must be modelled as formal problems, *(ii)* these problems must be encoding into propositional formulae. The first point has been widely discussed in Chapter 3 where we showed how protocol insecurity problems can be suitably used to model the question of whether a protocol satisfies its desiderata or not. For what concerns the second point, we propose in the next two sections a reduction of protocol insecurity problems into propositional logic that is carried out in two steps. Protocol insecurity problems are first translated into planning problems (see Section 4.1) which are in turn encoded into propositional formulae (see Section 4.2).

The entire SAT-based Model-Checking approach is then discussed in Section 4.3 where additionally some termination considerations as well as an Abstraction/Refinement strategy are presented.

4.1 Protocol Insecurity Problems as Planning Problems

Protocol insecurity problems (see Section 3.2) and planning problems (see Section 2.2) are based on two similar formalisms that allow for the specification of reachability problems of state transition systems. They both share the idea of representing a state by means of a set of atomic formulae of a first-order language as well as the feature of using a rule-based language

for specifying the transition relation. In this section we describe how to translate a protocol insecurity problem into a planning problem.

The main difficulty in such an encoding relies in the reduction of the existential quantification used in the rewriting formalism.

Section 4.1.1 provides a solution to this point. Once a protocol insecurity problem has been translated into a quantifier-free protocol insecurity problem, the reduction towards planning is quite easy as Section 4.1.2 shows.

4.1.1 Freshness

As explained in Section 2.1 we model the generation of fresh terms by means of rewrite rules with existential quantification. As soon as a rewrite rule is applied, then its existential variables, if any, must be instantiated with fresh terms.[1] As a matter of fact the planning formalism (see Section 2.2) does not provide any similar mechanism. A simple way to bridge such a gap and pave the way to a direct reduction of protocol insecurity problems into planning problems consists in applying a preprocessing transformation on the original protocol insecurity problem to obtain a new protocol insecurity problem such that *(i)* freshness is guaranteed through some alternative method (see Definition 4), and *(ii)* each solution can be recast into a solution to the original protocol insecurity problem and vice versa (see Definition 5).

Definition 4 (Quantifier-free PIP). *Let $\Xi = \langle\langle \mathsf{F}, \mathsf{L}, \mathsf{I}, \mathsf{R}\rangle, \mathsf{B}\rangle$ be a protocol insecurity problem. We say that Ξ is a quantifier-free protocol insecurity problem if, and only if, every rewrite rule $\ell \in \mathsf{R}$ is such that $\mathsf{ex}(\ell) = \emptyset$, i.e. the rewrite rule is not required to generate any fresh term during its execution.*

Definition 5 (Attack-equivalent). *Let Ξ_1 and Ξ_2 be protocol insecurity problems. We say that Ξ_1 is attack-equivalent to Ξ_2 if, and only if, every solution s to Ξ_1 can be transformed into a solution $\tau(s)$ to Ξ_2 and vice versa, where τ is an invertible transformation function.*

In the sequel we present three alternative ways to construct an attack-equivalent quantifier-free protocol insecurity problem. A first simple way to construct an attack-equivalent quantifier-free protocol insecurity problem consists in introducing a counter that is increased any time a fresh term is generated. Any term required to be fresh is then a function that depends from the value of the counter.

Freshness via a counter. Given a protocol insecurity problem $\Xi = \langle\langle \mathsf{F}, \mathsf{L}, \mathsf{I}, \mathsf{R}\rangle, \mathsf{B}\rangle$ in which *freshness* is provided *via existential quantification* it is possible to construct an attack-equivalent quantifier-free protocol insecurity problem $\tilde{\Xi} = \langle\langle \tilde{\mathsf{F}}, \tilde{\mathsf{L}}, \tilde{\mathsf{I}}, \tilde{\mathsf{R}}\rangle, \mathsf{B}\rangle$ in which *freshness* is provided *via a counter*. Namely, $\tilde{\Xi}$ is obtained from Ξ as follows:

[1] We recall that a term is fresh if and only if it has not used before.

Facts. The set of facts $\tilde{\mathsf{F}}$ is obtained from F by adding facts of the form $\mathtt{c}(T)$ to indicate that the current value of the counter used to build fresh terms is T. Intuitively, the counter is increased any time a rewrite rule that generates a fresh term is executed. Fresh terms are thus represented as $ft(C, T)$ where T is one of the values assumed by the counter and C is a constant useful to distinguish between multiple fresh terms generated by a single rewrite rule. To conclude this means to slightly change (notice that the notion of individual symbol is relaxed to include fresh terms) and extend the BNF rules for facts of Figure 3.3 as follows:

$$\begin{aligned} \mathit{CounterFact} &::= \mathtt{c}(\mathit{Number}) \\ \mathit{IndSym} &::= \mathit{IndConst} \mid \mathit{IndVar} \mid \mathit{FTerm} \\ \mathit{FTerm} &::= ft(\mathit{IndConst}, \mathit{Number}) \\ \mathit{Number} &::= 0 \mid s(\mathit{Number}) \end{aligned}$$

Initial States. The set of initial states is given by $\tilde{\mathsf{I}} = \{\mathtt{c}(0) \centerdot S : S \in \mathsf{I}\}$. For instance, the initial state shown in the example of Section 3.2.5 becomes:

$$\begin{aligned} &\mathtt{c}(0) \\ &\centerdot \mathtt{state}(0, a, a, [a, i, ka, ka^{-1}, ki], 1) \\ &\centerdot \mathtt{state}(0, a, a, [a, b, ka, ka^{-1}, kb], 2) \centerdot \mathtt{state}(1, a, b, [b, a, kb, kb^{-1}, ka], 2) \\ &\centerdot \mathtt{ik}(i) \centerdot \mathtt{ik}(a) \centerdot \mathtt{ik}(b) \centerdot \mathtt{ik}(ki) \centerdot \mathtt{ik}(ki^{-1}) \centerdot \mathtt{ik}(ka) \centerdot \mathtt{ik}(kb) \end{aligned}$$

Rewrite Rules. Let $\mathsf{R}_F \subseteq \mathsf{R}$ be the set of rewrite rules that generate at least one fresh term. Moreover, let $\mathbf{y}$ be the set of variables $y_1, \ldots, y_n$ and $c_1, \ldots, c_n$ be a set of constants such that $c_i \neq c_j$ for each $i \neq j$ $(i, j = 1, \ldots, n)$. We define $\tilde{\mathsf{R}}_F$ to be the set such that the rewrite rule[2]

$$\mathtt{c}(z) \centerdot L(\mathbf{x}) \xrightarrow{\ell(\mathbf{x},z)} \mathtt{c}(s(z)) \centerdot R(\mathbf{x}, \mathbf{y})[y_1/ft(c_1, z), \ldots, y_n/ft(c_n, z)]$$

is in $\tilde{\mathsf{R}}_F$ if and only if $(L(\mathbf{x}) \xrightarrow{\ell(\mathbf{x})} \exists \mathbf{y} : R(\mathbf{x}, \mathbf{y}))$ is in R_F, where z is a variable such that $z \notin \mathbf{x} \cup \mathbf{y}$. Given this, $\tilde{\mathsf{R}} = (\mathsf{R} \setminus \mathsf{R}_F) \cup \tilde{\mathsf{R}}_F$.[3] For instance, the rewrite rule (3.5) shown in Section 3.2.1 is transformed into:

$$\begin{aligned} &\mathtt{c}(C) \centerdot \mathtt{state}(0, A, A, [A, B, Ka, Ka^{-1}, Kb], S) \\ &\quad \xrightarrow{step_1(A,B,C,Ka,Kb,S)} \\ &\mathtt{c}(s(C)) \\ &\centerdot \mathtt{state}(2, B, A, [ft(cna, C), A, B, Ka, Ka^{-1}, Kb], S) \\ &\centerdot \mathtt{msg}(1, A, B, \{\langle A, ft(cna, C)\rangle\}_{Kb}) \\ &\centerdot \mathtt{witness}(A, B, na, ft(cna, C)) \end{aligned}$$

[2] Let $g(x)$ a generic term, we denote with $g(x)[x/t]$ the term obtained from $g(x)$ by substituting all occurrences of x with t. Such a definition is easily extended to facts and set of facts.

[3] The set of rule labels $\tilde{\mathsf{L}}$ readily follow from the sets of rewrite rules and of facts.

When NPC protocol insecurity problems are considered (see Section 3.4), *freshness* can be simply provided *via skolemization*. The basic idea is that a NPC protocol insecurity problem ensures that any of its rewrite rules that generate fresh terms is applied at most one time and therefore any existential variable occurring in a rule can be simply replaced with an appropriate skolem function.

Freshness via skolemization. Given a NPC protocol insecurity problem $\Xi = \langle\langle \mathsf{F}, \mathsf{L}, \mathsf{I}, \mathsf{R}\rangle, \mathsf{B}\rangle$, it is possible to construct an attack-equivalent quantifier-free protocol insecurity problem $\tilde{\Xi} = \langle\langle \tilde{\mathsf{F}}, \mathsf{L}, \mathsf{I}, \tilde{\mathsf{R}}\rangle, \mathsf{B}\rangle$ in which *freshness* is provided *via skolemization*. Namely, $\tilde{\Xi}$ is obtained from Ξ as follows:

Facts. The set of facts $\tilde{\mathsf{F}}$ is obtained from F by relaxing the notion of individual symbol to accept skolem functions. This means to slightly change and extend the BNF rules for facts of Figure 3.3 as follows:

$$\begin{array}{rcl} \mathit{IndSym} & ::= & \mathit{IndConst} \mid \mathit{IndVar} \mid \mathit{SkolemFunction} \\ \mathit{SkolemFunction} & ::= & \mathit{IndConst}(\mathit{IndVar}(,\mathit{IndVar})^*) \end{array}$$

Rewrite Rules. Let $\mathsf{R}_F \subseteq \mathsf{R}$ be the set of rewrite rules that generate at least one fresh term. Moreover, let $\mathbf{y}$ be the set of variables $y_1, \ldots, y_n$. We define $\tilde{\mathsf{R}}_F$ to be the set such that the rewrite rule

$$L(\mathbf{x}) \xrightarrow{\ell(\mathbf{x})} R(\mathbf{x}, \mathbf{y})[y_1/\, sko_1(\mathbf{x}), \ldots, y_n/\, sko_n(\mathbf{x})]$$

is in $\tilde{\mathsf{R}}_F$ if and only if $(L(\mathbf{x}) \xrightarrow{\ell(\mathbf{x})} \exists \mathbf{y} :\ R(\mathbf{x}, \mathbf{y})) \in \mathsf{R}_F$, where sko_i is a skolem function that univocally identifies the existentially quantified variable y_i (for each $i = 1, \ldots, n$). Given this, $\tilde{\mathsf{R}} = (\mathsf{R} \setminus \mathsf{R}_F) \cup \tilde{\mathsf{R}}_F$. For instance, the rewrite rule (3.5) shown in Section 3.2.1 becomes:

$$\begin{array}{l} \mathtt{state}(0, A, A, [A, B, Ka, Ka^{-1}, Kb], S) \\ \quad \xrightarrow{step_1(A,B,Ka,Kb,S)} \\ \mathtt{.state}(2, B, A, [sko\,1(A, B, Ka, Kb, S), A, B, Ka, Ka^{-1}, Kb], S) \\ \mathtt{.msg}(1, A, B, \{\langle A, sko\,1(A, B, Ka, Kb, S)\rangle\}_{Kb}) \\ \mathtt{.witness}(A, B, na, sko\,1(A, B, Ka, Kb, S)) \end{array}$$

While freshness via skolemization is quite simple it could increase significantly the number of facts (i.e. $|\tilde{\mathsf{F}}| \gg |\mathsf{F}|$). To cope with such a difficulty it suffices to observe that if NPC protocol insecurity problems are considered, then *(i)* existential variables are only used to represent honest agents that generate fresh keys and nonces (e.g. no fresh agents are generated), and *(ii)*

any honest agent is involved in a bounded number of sessions in which it can execute only a bounded number of operations. As a consequence any honest agent can generate a bounded number of fresh terms per sessions and such a number can be computed in advance.

Freshness via pre-computation of fresh terms. Given a NPC protocol insecurity problem $\Xi = \langle\langle \mathsf{F}, \mathsf{L}, \mathsf{I}, \mathsf{R}\rangle, \mathsf{B}\rangle$, it is possible to construct an attack-equivalent quantifier-free protocol insecurity problem $\tilde{\Xi} = \langle\langle \tilde{\mathsf{F}}, \mathsf{L}, \mathsf{I}, \tilde{\mathsf{R}}\rangle, \mathsf{B}\rangle$ in which *freshness* is provided *via pre-computation of fresh terms* of the fresh terms. Let m and r be the number of sessions modelled by Ξ and the number of roles for session respectively. Moreover, let s_i $(i = 1, \ldots, r)$ and $n_{i,j}$ the number of steps played by the i-th role and the number of fresh terms that can be generated by the i-th role playing the j-th step of a single session respectively. The number of fresh terms needed to analyse in a complete manner the protocol insecurity problem is thus given by $m \sum_{i=1}^{r} \sum_{j=1}^{s_i} n_{i,j}$. Namely, $\tilde{\Xi}$ is obtained from Ξ as follows:

Facts. The set of facts $\tilde{\mathsf{F}}$ is obtained from F by adding facts of the form $\texttt{fresh}(ft(A, J, C, S))$ to indicate that the term $ft(A, J, C, S)$ has not been used yet by the agent A at step J of session S and therefore $ft(A, J, C, S)$ can be still considered a "fresh" term. Notice that C is just a constant useful to distinguish between multiple fresh terms generated by A at step J of session S. To conclude this means to slightly change and extend the BNF rules for facts of Figure 3.3 as follows:

$$
\begin{aligned}
\mathit{FFact} &::= \texttt{fresh}(\mathit{FTerm}) \\
\mathit{Key} &::= \mathit{IndSym} \mid \mathit{IndSym}^{-1} \mid \mathit{FTerm} \mid \mathit{FTerm}^{-1} \mid \mathit{Msg} \\
\mathit{NumSym} &::= \mathit{IndSym} \mid \mathit{FTerm} \\
\mathit{FTerm} &::= ft(\mathit{IndConst}, \mathit{Agent}, \mathit{StepId}, \mathit{SessId})
\end{aligned}
$$

Initial States. Let ag_i be the honest agent playing the i-th role, $\mathsf{st}_{i,j}$ be the protocol step at which the i-th role executes its that j-th step, and ses_k be the k-th session of the protocol. The set of pre-computed fresh facts, say Υ, is thus given as follow: $\texttt{fresh}(ft(\mathsf{ag}_i, \mathsf{st}_{i,j}, h, \mathsf{ses}_k)) \in \Upsilon$ for each i, j, h, and k such that $1 \leq i \leq r$, $1 \leq j \leq s_i$, $1 \leq h \leq n_{i,j}$, and $1 \leq k \leq m$.[4] From this the set of initial states is simply defined as $S \cup \Upsilon \in \tilde{\mathsf{I}}$ if and only if $S \in \mathsf{I}$. For instance, the initial state shown in the example of Section 3.2.5 (two

[4]Notice that, if the i-th role does not generate any fresh term at step j, then $n_{i,j} = 0$ and therefore no values can be assigned to h to construct a fresh fact.

parallel sessions of the NSPK protocol) becomes:[5]

$$\begin{aligned}
&\mathtt{fresh}(a,1,1,1)\\
&\mathtt{.fresh}(a,1,1,2)\\
&\mathtt{.fresh}(b,2,1,2)\\
&\mathtt{.state}(0,a,a,[a,i,ka,ka^{-1},ki],1)\\
&\mathtt{.state}(0,a,a,[a,b,ka,ka^{-1},kb],2)\,\mathtt{.state}(1,a,b,[b,a,kb,kb^{-1},ka],2)\\
&\mathtt{.ik}(i)\,\mathtt{.ik}(a)\,\mathtt{.ik}(b)\,\mathtt{.ik}(ki)\,\mathtt{.ik}(ki^{-1})\,\mathtt{.ik}(ka)\,\mathtt{.ik}(kb)
\end{aligned}$$

Rewrite Rules. Let $\mathsf{R}_F \subseteq \mathsf{R}$ be the set of rewrite rules that generate at least one fresh term. Moreover, let $\mathbf{y}$ be the set of variables $y_1, \ldots, y_n$. We define $\tilde{\mathsf{R}}_F$ to be the set such that the rewrite rule

$$\begin{aligned}
&L(\mathbf{x})\,\mathtt{.fresh}(ft(\mathsf{ag},\mathsf{st},1,\mathsf{ses}))\,\mathtt{.}\ldots\mathtt{.fresh}(ft(\mathsf{ag},\mathsf{st},n,\mathsf{ses}))\\
&\quad\xrightarrow{\ell(\mathbf{x})}\\
&R(\mathbf{x},\mathbf{y})[y_1/ft(\mathsf{ag},\mathsf{st},1,\mathsf{ses}),\ldots,y_n/ft(\mathsf{ag},\mathsf{st},n,\mathsf{ses})]
\end{aligned}$$

is in $\tilde{\mathsf{R}}_F$ if and only if $(L(\mathbf{x}) \xrightarrow{\ell(\mathbf{x})} \exists\mathbf{y}:\ R(\mathbf{x},\mathbf{y})) \in \mathsf{R}_F$, where ag is the agent that sends/receives messages in the rule, and st and ses are the protocol step and the session identifier associated to the rule respectively. Given this, $\tilde{\mathsf{R}} = (\mathsf{R} \setminus \mathsf{R}_F) \cup \tilde{\mathsf{R}}_F$.[6] For instance, the rewrite rule (3.5) shown in Section 3.2.1 becomes:

$$\begin{aligned}
&\mathtt{state}(0,A,A,[A,B,Ka,Ka^{-1},Kb],S)\,\mathtt{.fresh}(ft(A,1,1,S))\\
&\quad\xrightarrow{step_1(A,B,Ka,Kb,S)}\\
&\mathtt{.state}(2,B,A,[ft(A,1,1,S),A,B,Ka,Ka^{-1},Kb],S)\\
&\mathtt{.msg}(1,A,B,\{\langle A,ft(A,1,1,S)\rangle\}_{Kb})\\
&\mathtt{.witness}(A,B,na,ft(A,1,1,S))
\end{aligned}$$

It is immediate to see that once an instance of the above rule has been applied one time, the fresh fact is removed from the state and the same rule instance will not be applicable anymore.

In the sequel if $\Xi = \langle\Gamma,\mathsf{B}\rangle$ is a protocol insecurity problem, then $\tilde{\Xi} = \langle\tilde{\Gamma},\mathsf{B}\rangle$ is the attack-equivalent quantifier-free protocol insecurity problem where $\tilde{\Gamma} = \langle\tilde{\mathsf{F}},\tilde{\mathsf{L}},\tilde{\mathsf{I}},\tilde{\mathsf{R}}\rangle$ is obtained from $\Gamma = \langle\mathsf{F},\mathsf{L},\mathsf{I},\mathsf{R}\rangle$ by applying one of the aforementioned methods. It must be noted that since $\tilde{\Gamma}$ is a quantifier-free protocol insecurity problem, we are guaranteed that the transition system associated to $\tilde{\Gamma}$ has a finite branching factor.

[5] Notice that no fresh terms are provided for the intruder since in such a protocol the honest agents are not able to check that messages are different (i.e. they perform only equality checks). As a consequence the intruder can always send any message and the agent would not be able to establish if it is fresh or not.

[6] The set of rule labels $\tilde{\mathsf{L}}$ readily follow from the sets of rewrite rules and of facts.

4.1.2 Reduction to Planning

For the sake of simplicity in the rest of this section we abbreviate $L(\mathbf{x})$, $R(\mathbf{x})$, and $\ell(\mathbf{x})$ with L, R, and ℓ respectively. Given a quantifier-free protocol insecurity problem $\tilde{\Xi} = \langle \tilde{\Gamma}, \mathsf{B} \rangle = \langle \langle \tilde{\mathsf{F}}, \tilde{\mathsf{L}}, \tilde{\mathsf{I}}, \tilde{\mathsf{R}} \rangle, \mathsf{B} \rangle$, it is possible to build a planning problem $\{\![\tilde{\Xi}]\!\} = \langle \{\![\tilde{\Gamma}]\!\}, \mathcal{G} \rangle = \langle \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{O} \rangle, \mathcal{G} \rangle$ such that any plan of length k on $\{\![\tilde{\Xi}]\!\}$ corresponds to an attack of length k on $\tilde{\Xi}$ and vice versa. The reduction from $\tilde{\Xi}$ to $\{\![\tilde{\Xi}]\!\}$ is as follows:

(S1) $\mathcal{F} = \tilde{\mathsf{F}}$;

(S2) $\mathcal{A} = \tilde{\mathsf{L}}$;

(S3) $\mathcal{O} = \{(L\sigma \overset{\ell\sigma}{\Rightarrow} R\sigma \,;\, L\sigma \setminus R\sigma) : (L \overset{\ell}{\rightarrow} R) \in \tilde{\mathsf{R}}, \sigma \text{ is a ground substitution}\}$

(S4) $\mathcal{I} = \tilde{\mathsf{I}}$; and

(S5) $\mathcal{G} = \mathsf{B}$.

As showed in Appendix A the reduction $\{\![\tilde{\Xi}]\!\}$ is sound and complete.

4.2 Planning Problems as SAT

Given a planning system $\Theta = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{O} \rangle$ with a finite branching factor it is possible to build a propositional formula $[\![\Theta]\!]_k$ such that any model of $[\![\Theta]\!]_k$ corresponds to a (linearizable) partial-order planning path of Θ of length k and vice versa.

In this section if $\Theta = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{O} \rangle$, then $\hat{\Theta} = \langle \mathcal{F}, \hat{\mathcal{A}}, \mathcal{I}, \hat{\mathcal{O}} \rangle$, where $\hat{\Theta}$ is obtained from Θ by adding all linearizable compound actions and their corresponding operators. In Section 2.2 we have already proved that for each path of $\hat{\Theta}$ there exists a partial-order path of Θ and vice versa.

The encoding into SAT can be done in a variety of ways. The basic idea [KMS96, EMW97] is to add an additional time-index to the actions and fluents to indicate the state at which the actions begin or the fluents hold. Fluents are thus indexed by 0 through k and action by 0 through $k-1$. If p is a fluent or an action and i is an index, then p^i is the corresponding time-indexed propositional variable. If $\mathbf{p} = p_1, \ldots, p_n$ is a vector of fluents or actions and i is an index, then $\mathbf{p}^i = p_1^i, \ldots, p_n^i$ is the corresponding time-indexed vector of propositional variables. The propositional formula $[\![\Theta]\!]_k$ is of the form:

$$[\![\Theta]\!]_k \quad = \quad I(\mathbf{f}^0) \wedge \bigwedge_{i=0}^{k-1} T_i(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1}) \tag{4.1}$$

where $\mathbf{f}$ and $\mathbf{a}$ are vectors comprising the fluents and the actions in $\mathcal{F}$ and $\mathcal{A}$ respectively and $\bigwedge_{i=0}^{-1} T_i(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ stands for the propositional constant

TRUE. The formula $I(\mathbf{f}^0)$ encodes the set of initial states whereas the formula $T_i(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ encodes all the possible evolutions of the system from step i to step $i+1$. By this we mean that:

(R1) $I(\mathbf{f}^0)$ is such that if M is a model of $I(\mathbf{f}^0)$ then $\{f \in \mathcal{F} : f^0 \in M\} \in \mathcal{I}$ and, conversely, if $S \in \mathcal{I}$ then $\{f^0 : f \in S\}$ is a model of $I(\mathbf{f}^0)$.

(R2) $T_i(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ is such that for any $i = 0, \ldots, k-1$

(a) if M is a model of $T_i(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ then the actions in $\Lambda = \{a \in \mathcal{A} : a^i \in M\}$ are composable and their parallel composition $\|(\Lambda)$ is linearizable and such that $\mathrm{app}^{\Rightarrow}_{\|(\Lambda)}(S) = S'$ for $S = \{f \in \mathcal{F} : f^i \in M\}$ and $S' = \{f \in \mathcal{F} : f^{i+1} \in M\}$;

(b) if $S \subseteq \mathcal{F}$ is reachable in i steps in $\hat{\Theta}$, the actions in $\Lambda \subseteq \mathcal{A}$ are linearizable and their parallel composition $\|(\Lambda)$ is applicable in S, and $S' = \mathrm{app}^{\Rightarrow}_{\|(\Lambda)}(S)$, then $M = \{f^i : f \in S\} \cup \{f^{i+1} : f \in S'\} \cup \{a^i : a \in \Lambda\}$ is a model of $T_i(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$.

The following results state that any encoding enjoying the above requirements is sound and complete. To the best of our knowledge this is the first attempt to provide an abstract characterisation of AI planning SAT-reduction techniques that ensures the soundness and completeness of an encoding technique provided that it enjoys the above requirements.

Theorem 2 (Soundness). *If M is a model of $[\![\Theta]\!]_k$ for $k \geq 0$, then $S_0 a_0 S_1 \cdots S_{k-1} a_{k-1} S_k$ is a planning path of $\hat{\Theta}$, where $S_i = \{f \in \mathcal{F} : f^i \in M\}$ for $i = 0, \ldots, k$ and a_i is the parallel composition of the actions in $\{a \in \mathcal{A} : a^i \in M\}$ for $i = 0, \ldots, k-1$.*

Proof. The proof is by induction on k. The base case is easy as $[\![\Theta]\!]_0$ is logically equivalent to $I(\mathbf{f}^0)$ and by requirement R1 we know that $\{f \in \mathcal{F} : f^0 \in M\} = S_0 \in \mathcal{I}$. Therefore S_0 is a planning path of $\hat{\Theta}$. Let us now consider the step case. Let M be a model of $[\![\Theta]\!]_k$ for $k > 0$. Since $[\![\Theta]\!]_k = [\![\Theta]\!]_{k-1} \wedge T_{k-1}(\mathbf{f}^{k-1}, \mathbf{a}^{k-1}, \mathbf{f}^k)$, then M is also a model of $[\![\Theta]\!]_{k-1}$ and of $T_{k-1}(\mathbf{f}^{k-1}, \mathbf{a}^{k-1}, \mathbf{f}^k)$. By induction hypothesis we know that $S_0 a_0 S_1 \cdots S_{k-2} a_{k-2} S_{k-1}$ is a planning path of $\hat{\Theta}$, where $S_i = \{f \in \mathcal{F} : f^i \in M\}$ for $i = 0, \ldots, k-1$ and a_i is the parallel composition of the actions in $\{a \in \mathcal{A} : a^i \in M\}$ for $i = 0, \ldots, k-2$. By part (a) of requirement R2 we know that the parallel composition a_{k-1} of the actions in $\{a \in \mathcal{A} : a^{k-1} \in M\}$ is linearizable and that $\mathrm{app}^{\Rightarrow}_{a_{k-1}}(S_{k-1}) = \{f \in \mathcal{F} : f^k \in M\} = S_k$. We can therefore conclude that $S_0 a_0 S_1 \cdots S_{k-1} a_{k-1} S_k$ is a planning path of $\hat{\Theta}$. □

Let M be a model of $[\![\Theta]\!]_k$. We define $\pi(M) = S_0 a_0 S_1 \cdots S_{k-1} a_{k-1} S_k$, where $S_i = \{f : f^i \in M\}$ for $i = 0, \ldots, k$ and a_i is the parallel composition of the actions in $\{a : a^i \in M\}$ for $i = 0, \ldots, k-1$.

Theorem 3 (Completeness). *If $\pi_k = S_0 a_0 S_1 \cdots S_{k-1} a_{k-1} S_k$ is a planning path of $\hat{\Theta}$, then there exists a model M of $[\![\Theta]\!]_k$ such that $\pi(M) = \pi_k$.*

Proof. The proof is by induction on k. The base case is easy as $\pi_0 = S_0 \in \mathcal{I}$ and from R1 we know that $M = \{f^0 : f \in S_0\}$ is a model of $I(\mathbf{f}^0)$. But $[\![\Theta]\!]_0$ is logically equivalent to $I(\mathbf{f}^0)$ and therefore M is a model of $[\![\Theta]\!]_k$ for $k = 0$. Moreover $\pi(M) = \{f : f^0 \in M\} = S_0 = \pi_0$ is a planning path of $\hat{\Theta}$. Let us now consider the step case. Let $\pi_k = S_0 a_0 S_1 \cdots S_{k-1} a_{k-1} S_k$ be a planning path of $\hat{\Theta}$ where $a_i \in \hat{\mathcal{A}}$ is the parallel composition of the linearizable actions in $\Lambda_i \subseteq \mathcal{A}$ for $i = 0, \ldots, k-1$. Clearly also $\pi_{k-1} = S_0 a_0 S_1 \cdots S_{k-2} a_{k-2} S_{k-1}$ is a planning path of $\hat{\Theta}$. By induction hypothesis there exists a model M' of $[\![\Theta]\!]_{k-1}$ such that $\pi(M') = S_0 a_0 S_1 \cdots S_{k-2} a_{k-2} S_{k-1}$ is a planning path of $\hat{\Theta}$. Since S_{k-1} is reachable in $k-1$ steps in $\hat{\Theta}$, $\mathrm{pre}(a_{k-1}) \subseteq S_{k-1}$, and $\mathrm{app}^{\Rightarrow}_{a_{k-1}}(S_{k-1}) = S_k$, by part (b) of requirement R2 we know that $M'' = \{f^{k-1} : f \in S_{k-1}\} \cup \{a^{k-1} : a \in \Lambda_{k-1}\} \cup \{f^k : f \in S_k\}$ is a model of $T_{k-1}(\mathbf{f}^{k-1}, \mathbf{a}^{k-1}, \mathbf{f}^k)$. From this fact we can conclude that $M = M' \cup M''$ is a model of $[\![\Theta]\!]_k = [\![\Theta]\!]_{k-1} \wedge T(\mathbf{f}^{k-1}, \mathbf{a}^{k-1}, \mathbf{f}^k)$ and is such that $\pi(M) = S_0 a_0 S_1 \cdots S_{k-2} a_{k-2} S_{k-1} a_{k-1} S_k$ is a planning path of $\hat{\Theta}$. □

The above results about planning systems can be directly extended to planning problems. Given a planning problem $\Pi = \langle \Theta, \mathcal{G} \rangle$, it is thus possible to build a propositional formula

$$[\![\Pi]\!]_k = [\![\Theta]\!]_k \wedge B(\mathbf{f}^k) \tag{4.2}$$

such that any model of $[\![\Pi]\!]_k$ corresponds to a partial-order plan of Π of length k and vice versa where $[\![\Theta]\!]_k$ is defined as in (4.1) and describes the behaviour of a planning system Θ and $B(\mathbf{f}^k)$ encodes the set of bad states. Basically $B(\mathbf{f}^k)$ is such that if M is a model of $B(\mathbf{f}^k)$ then $\{f \in \mathcal{F} : f^k \in M\} \in \mathcal{G}$ and, conversely, if $S \in \mathcal{G}$ then $\{f^k : f \in S\}$ is a model of $B(\mathbf{f}^k)$. However for some encoding techniques (e.g. graphplan-based encodings) it is even possible to construct a simpler $B(\mathbf{f}^k)$ where the bad states that are known in advance to be unreachable are simply neglected. This means that if we know that $S' \in \mathcal{G}$ cannot be reached then it is perfectly fine to build a formula $B(\mathbf{f}^k)$ such that $\{f^k : f \in S'\}$ is not a model of $B(\mathbf{f}^k)$.

In the rest of this section we formally describe two of the best-known families of SAT-reductions for AI planning: the *linear* and the *graphplan-based* encoding techniques.

4.2.1 Linear Encodings

Given a planning system $\Theta = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{O} \rangle$ with finite and disjoint $\mathcal{F}$ and $\mathcal{A}$, a first simple approach to building $[\![\Theta]\!]_k$ is as follows.[7] The initial states are

[7] It must be noted that to reduce a planning problem into SAT by means of a linear encoding, it does not suffice a finite branching factor, but a more restrictive condition that bound the numbers of fluents and actions is required.

encoded as[8]

$$I(\mathbf{f}^0) = \bigvee_{S_0 \in \mathcal{I}} \Big\{ \bigwedge \{f^0 : f \in \mathcal{F}, f \in S_0\} \wedge \bigwedge \{\neg f^0 : f \in \mathcal{F}, f \notin S_0\} \Big\} \quad (4.3)$$

and the formula $T_i(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ encoding the i-th step of the transition relation is given by the conjunction of the following schemata of formulae:

Universal Axioms: for each $a \in \mathcal{A}$

$$a^i \supset \bigwedge \{f^i \mid f \in \mathrm{pre}(a)\} \quad (4.4)$$

$$a^i \supset \bigwedge \{f^{i+1} \mid f \in \mathrm{add}(a)\} \quad (4.5)$$

$$a^i \supset \bigwedge \{\neg f^{i+1} \mid f \in \mathrm{del}(a)\} \quad (4.6)$$

Explanatory Frame Axioms: for all $f \in \mathcal{F}$

$$(f^i \wedge \neg f^{i+1}) \supset \bigvee \{a^i \mid a \in \mathcal{A}, f \in \mathrm{del}(a)\} \quad (4.7)$$

$$(\neg f^i \wedge f^{i+1}) \supset \bigvee \{a^i \mid a \in \mathcal{A}, f \in \mathrm{add}(a)\} \quad (4.8)$$

Conflict Exclusion Axioms (CEA): for all pairs of actions $a_1, a_2 \in \mathcal{A}$ such that $a_1 \oplus a_2$

$$\neg(a_1^i \wedge a_2^i) \quad (4.9)$$

Intuitively, the universal axioms state that if an action a holds at the time-index i, then its preconditions must hold at the same time-index—see formula (4.4)—and its effects must hold at the next time-index—see formulae (4.5) and (4.6) for positive and negative effects respectively. The explanatory frame axioms express that if a fluent changes value from true to false (from false to true) between the time-indexes i and $i+1$, then one of the action that deletes (adds) the fluent must have been applied at time-index i—see formula (4.7) (see formula (4.8)). For what concerns the conflict exclusion axioms, they impose that two interfering actions (see Definition 3 in Section 2.2) cannot be executed at the same time-index—see formula (4.9).

Since the structure of the formula $T_i(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ is independent from i in the rest of this section we write $T(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ in place of $T_i(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$.

The soundness and completeness of linear encodings amount to showing that $I(\mathbf{f}^0)$ enjoys requirement R1 (see Lemma 4) and $T(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ enjoys part (a) and part (b) of requirement R2 (see Lemma 5 and Lemma 6 respectively).

[8]Since the number of initial states is in the worst case exponential in the number of fluents, the number of clauses in (4.3) is exponential in the number of fluents too. This difficulty can be simply overcome by specifying $\mathcal{I}$ as a formula instead of as an enumeration of states (e.g. $\mathcal{I} = (f_1 \wedge f_2)$ compactly represents the set of all the initial states in which the fluents f_1 and f_2 are true).

Lemma 4. *$I(\mathbf{f}^0)$ is such that if M is a model of $I(\mathbf{f}^0)$ then $\{f \in \mathcal{F} : f^0 \in M\} \in \mathcal{I}$ and, conversely, if $S \in \mathcal{I}$ then $\{f^0 : f \in S\}$ is a model of $I(\mathbf{f}^0)$.*

Proof. This proof readily follows from the definition of $I(\mathbf{f}^0)$ given in (4.3). □

Lemma 5. *If M is a model of $T(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ then the actions in $\Lambda = \{a \in \mathcal{A} : a^i \in M\}$ are composable and their parallel composition $\|(\Lambda)$ is linearizable and such that $\mathrm{app}^{\Rightarrow}_{\|(\Lambda)}(S) = S'$ for $S = \{f \in \mathcal{F} : f^i \in M\}$ and $S' = \{f \in \mathcal{F} : f^{i+1} \in M\}$ for $i = 0, \ldots, k-1$.*

Proof. If M is a model of $T(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$, then M is a model of (4.4)-(4.9). The truth of (4.5) and (4.6) in M implies that the effects of the actions in $\Lambda = \{a \in \mathcal{A} : a^i \in M\}$ are such that $\{f^{i+1} : f \in \mathrm{add}(a)\} \subset M$, $\{f^{i+1} : f \in \mathrm{del}(a)\} \cap M = \emptyset$, and therefore they do not contradict each other, i.e. $\bigcup_{a\in\Lambda} \mathrm{add}(a) \cap \bigcup_{a\in\Lambda} \mathrm{del}(a) = \emptyset$. Hence the actions in Λ are composable and because of (4.9) they are pairwise non-interfering and thus linearizable. Moreover for any $a \in \Lambda$, the formulae (4.4) state that $\{f^i : f \in \mathrm{pre}(a)\} \subset M$. As a consequence, the union of the preconditions of the actions in Λ are contained in $S = \{f \in \mathcal{F} : f^i \in M\}$ and therefore $\|(\Lambda)$ is applicable in S.

To conclude the proof we must thus show that $S' = \{f \in \mathcal{F} : f^{i+1} \in M\}$ is equal to $(S \setminus \bigcup_{a\in\Lambda} \mathrm{del}(a)) \cup \bigcup_{a\in\Lambda} \mathrm{add}(a)$. This boils down to proving that $p \in (S \setminus \bigcup_{a\in\Lambda} \mathrm{del}(a)) \cup \bigcup_{a\in\Lambda} \mathrm{add}(a)$ if, and only if, $p^{i+1} \in M$. The proof is articulated in the following two cases:

case 1 (if): Let $p \in (S \setminus \bigcup_{a\in\Lambda} \mathrm{del}(a)) \cup \bigcup_{a\in\Lambda} \mathrm{add}(a)$, i.e. either $p \in S \setminus \bigcup_{a\in\Lambda} \mathrm{del}(a)$ or $p \in \bigcup_{a\in\Lambda} \mathrm{add}(a)$, we must prove that $p^{i+1} \in M$.

- **case 1.1:** Let $p \in S \setminus \bigcup_{a\in\Lambda} \mathrm{del}(a)$, i.e. $p \in S$ and $p \notin \mathrm{del}(a)$ for all $a \in \Lambda$ (and thus $a^i \in M$). Thus every action $\tilde{a}$ that deletes p is not in Λ, i.e. $\tilde{a}^i \notin M$. As a consequence the conclusion of the first explanatory frame axiom (4.7) for fluent p is false in M. Moreover, since $p \in S$ from the definition of S it follows that $p^i \in M$. As a consequence the truth value of (4.7) in M requires $p^{i+1} \in M$.
- **case 1.2:** Let $p \in \bigcup_{a\in\Lambda} \mathrm{add}(a)$, i.e. there exists an action $a \in \Lambda$ (and thus $a^i \in M$) such that $p \in \mathrm{add}(a)$. Because of this, the truth of $(a^i \supset p^{i+1})$, i.e. one of the formulae (4.5), in M implies that $p^{i+1} \in M$.

case 2 (only if): Let $p^{i+1} \in M$, we must prove that $p \in (S \setminus \bigcup_{a\in\Lambda} \mathrm{del}(a)) \cup \bigcup_{a\in\Lambda} \mathrm{add}(a)$ i.e. either *(i)* $p \in S$ and $p \notin \bigcup_{a\in\Lambda} \mathrm{del}(a)$, or *(ii)* $p \in \bigcup_{a\in\Lambda} \mathrm{add}(a)$. Since $p^{i+1} \in M$, then for all $\tilde{a}$ such that $p \in \mathrm{del}(\tilde{a})$ the truth of $(\tilde{a}^i \supset \neg p^{i+1})$, i.e. one of the formulae (4.6), in M implies that $\tilde{a}^i \notin M$. Because of this and from the definition of Λ, we derive that $\tilde{a} \notin \Lambda$ and therefore that $p \notin \bigcup_{a\in\Lambda} \mathrm{del}(a)$. To conclude the

proof we must show that either $p \in S$ or $p \in \bigcup_{a\in\Lambda} \mathrm{add}(a)$. This is equivalent to proving that either $p^i \in M$ or there exists an action a such that $a^i \in M$ and $p \in \mathrm{add}(a)$. To prove this it suffices to observe that, for M to satisfy $(\neg p^i \supset \bigvee\{a^i \mid a \in \mathcal{A}, p \in \mathrm{add}(a)\})$—i.e. the formula constructed through schema (4.8) and simplified by using the assumption $p^{i+1} \in M$—it must hold that either $p^i \in M$ or there exists an action a such that $a^i \in M$ and $p \in \mathrm{add}(a)$.

□

Theorem 4 (Soundness of Linear Encodings). *If M is a model of $[\![\Theta]\!]_k$ for $k \geq 0$, then $S_0 a_0 S_1 \cdots S_{k-1} a_{k-1} S_k$ is a planning path of $\hat{\Theta}$, where $S_i = \{f \in \mathcal{F} : f^i \in M\}$ for $i = 0, \ldots, k$ and a_i is the parallel composition of the actions in $\{a \in \mathcal{A} : a^i \in M\}$ for $i = 0, \ldots, k-1$.*

Proof. Lemma 4 and Lemma 5 ensure that linear encodings enjoy requirement R1 and part (a) of requirement R2 respectively. Because of Theorem 2 we are thus guaranteed that linear encodings are sound. □

Lemma 6. *If $S \subseteq \mathcal{F}$ is reachable in i steps in $\hat{\Theta}$, the actions in $\Lambda = \{a_1, \ldots, a_n\} \subseteq \mathcal{A}$ are linearizable and their parallel composition $\|(\Lambda)$ is applicable in S, and $S' = \mathrm{app}^{\Rightarrow}_{\|(\Lambda)}(S)$, then $M = \{f^i : f \in S\} \cup \{f^{i+1} : f \in S'\} \cup \{a_1^i, \ldots, a_n^i\}$ is a model of $T(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ for $i = 0, \ldots, k-1$.*

Proof. Let $\tilde{a}$ and f be an action and a fluent respectively. By definition of M and Λ, it is immediate to see that *(i)* $\tilde{a} \in \Lambda$ if and only if $\tilde{a}^i \in M$, *(ii)* $f \in S$ if and only if $f^i \in M$, and *(iii)* $f \in S'$ if and only if $f^{i+1} \in M$.

To show that M is model of $T(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$, we must prove that M satisfies (4.4)-(4.9).

The formulae (4.4), (4.5), and (4.6) are true in M if and only if for every action $\tilde{a} \in \mathcal{A}$ if $\tilde{a}^i \in M$ (i.e. $\tilde{a} \in \Lambda$), then *(a)* any fluent $f \in \mathrm{pre}(\tilde{a})$ is such that $f^i \in M$ (i.e. $f \in S$), *(b)* any fluent $f \in \mathrm{add}(\tilde{a})$ is such that $f^{i+1} \in M$ (i.e. $f \in S'$), and *(c)* any fluent $f \in \mathrm{del}(\tilde{a})$ is such that $f^{i+1} \notin M$ (i.e. $f \notin S'$). This is trivially satisfied in M by any action $\tilde{a}$ that is not in Λ. Let us consider the case in which $\tilde{a} \in \Lambda$. By hypothesis the parallel composition $\|(\Lambda)$ is applicable in S (i.e. $\mathrm{pre}(\|(\Lambda)) = \bigcup_{a\in\Lambda} \mathrm{pre}(a) \subseteq S$), and the application of $\|(\Lambda)$ leads to $S' = (S \setminus \bigcup_{a\in\Lambda} \mathrm{del}(a)) \cup \bigcup_{a\in\Lambda} \mathrm{add}(a)$. Since $\tilde{a} \in \Lambda$, we have that $\mathrm{pre}(\tilde{a}) \subseteq \mathrm{pre}(\|(\Lambda)) \subseteq S$ and thus *(a)* is proved. Similarly $\mathrm{add}(\tilde{a}) \subseteq S'$ and therefore also *(b)* is demonstrated. Finally, to prove *(c)* it suffices to show that $\mathrm{del}(\tilde{a}) \cap S' = \emptyset$, i.e. both $\mathrm{del}(\tilde{a}) \cap (S \setminus \bigcup_{a\in\Lambda} \mathrm{del}(a)) = \emptyset$ and $\mathrm{del}(\tilde{a}) \cap \bigcup_{a\in\Lambda} \mathrm{add}(a) = \emptyset$. The former simply derives from the fact that $\tilde{a} \in \Lambda$ and thus $\mathrm{del}(\tilde{a}) \subseteq \bigcup_{a\in\Lambda} \mathrm{del}(a)$. By definition of parallel composition $\bigcup_{a\in\Lambda} \mathrm{del}(a) \cap \bigcup_{a\in\Lambda} \mathrm{add}(a) = \emptyset$ and since $\mathrm{del}(\tilde{a}) \subseteq \bigcup_{a\in\Lambda} \mathrm{del}(a)$ it follows immediately also the latter.

Formula (4.7) is true in M if and only if for every fluent $f \in \mathcal{F}$ if $f^i \in M$ (i.e. $f \in S$) and $f^{i+1} \notin M$ (i.e. $f \notin S'$), then at least one of the actions,

say $\tilde{a}$, that deletes f must be such that $\tilde{a}^i \in M$ (i.e. $\tilde{a} \in \Lambda$). It follows immediately that $M \models (4.7)$ holds for every fluent f such that either $f \notin S$ or $f \in S'$. By hypothesis $S' = (S \setminus \bigcup_{a\in\Lambda} \mathrm{del}(a)) \cup \bigcup_{a\in\Lambda} \mathrm{add}(a)$ and then

$$
\begin{aligned}
& f \notin S \vee f \in S' \\
&= f \notin S \vee (f \in S \wedge f \notin \textstyle\bigcup_{a\in\Lambda} \mathrm{del}(a)) \vee f \in \bigcup_{a\in\Lambda} \mathrm{add}(a) && \text{[by definition]} \\
&= f \notin S \vee f \notin \textstyle\bigcup_{a\in\Lambda} \mathrm{del}(a) \vee f \in \bigcup_{a\in\Lambda} \mathrm{add}(a) && \text{[by simplification]}
\end{aligned}
$$

As a consequence $M \models (4.7)$ holds for any fluent f such that either $f \notin S$, $f \notin \bigcup_{a\in\Lambda} \mathrm{del}(a)$, or $f \in \bigcup_{a\in\Lambda} \mathrm{add}(a)$. To conclude the proof of $M \models (4.7)$ it suffices to note that in the remaining cases—i.e. both $f \in S$, $f \in \bigcup_{a\in\Lambda} \mathrm{del}(a)$, and $f \notin \bigcup_{a\in\Lambda} \mathrm{add}(a)$— $f \in \bigcup_{a\in\Lambda} \mathrm{del}(a)$ i.e. there exists at least one action $\tilde{a} \in \Lambda$ such that $f \in \mathrm{del}(\tilde{a})$ and thus (4.7) is true in M.

To prove $M \models (4.8)$ we proceed as above but dually. The formula (4.8) is true in M if and only if for every fluent $f \in \mathcal{F}$ if $f^i \notin M$ (i.e. $f \notin S$) and $f^{i+1} \in M$ (i.e. $f \in S'$), then at least one of the actions, say $\tilde{a}$, that adds f must be such that $\tilde{a}^i \in M$ (i.e. $\tilde{a} \in \Lambda$). It follows immediately that $M \models (4.8)$ holds for every fluent f such that either $f \in S$ or $f \notin S'$. By hypothesis $S' = (S \setminus \bigcup_{a\in\Lambda} \mathrm{del}(a)) \cup \bigcup_{a\in\Lambda} \mathrm{add}(a)$ and then

$$
\begin{aligned}
& f \in S \vee f \notin S' \\
&= f \in S \vee ((f \notin S \vee f \in \textstyle\bigcup_{a\in\Lambda} \mathrm{del}(a)) \wedge f \notin \bigcup_{a\in\Lambda} \mathrm{add}(a)) && \text{[by definition]} \\
&= f \in S \vee f \notin \textstyle\bigcup_{a\in\Lambda} \mathrm{add}(a) && \text{[by simplification]}
\end{aligned}
$$

As a consequence $M \models (4.8)$ holds for any fluent f such that either $f \in S$ or $f \notin \bigcup_{a\in\Lambda} \mathrm{add}(a)$. To conclude the proof about $M \models (4.8)$ it suffices to observe that in the rest of the cases—i.e. both $f \notin S$ and $f \in \bigcup_{a\in\Lambda} \mathrm{add}(a)$— $f \in \bigcup_{a\in\Lambda} \mathrm{add}(a)$ i.e. there exists at least one action $\tilde{a} \in \Lambda$ such that $f \in \mathrm{add}(\tilde{a})$ and thus (4.8) is true in M.

To conclude the overall proof we must demonstrate that $M \models (4.9)$. This means to show that for all pair of actions $a_1, a_2 \in \mathcal{A}$ such that $a_1 \oplus a_2$ either $a_1^i \notin M$ (i.e. $a_1 \notin \Lambda$) or $a_2^i \notin M$ (i.e. $a_2 \notin \Lambda$). This is a simple consequence of the fact that the actions in Λ are linearizable, i.e. they are pairwise non-interfering.

□

Theorem 5 (Completeness of Linear Encodings). *If $\pi_k = S_0 a_0 S_1 \cdots S_{k-1} a_{k-1} S_k$ is a planning path of $\hat{\Theta}$, then there exists a model M of $[\![\Theta]\!]_k$ such that $\pi(M) = \pi_k$.*

Proof. Lemma 4 and Lemma 6 ensure that linear encodings enjoy requirement R1 and part (b) of requirement R2 respectively. Because of Theorem 3 we are thus guaranteed that linear encodings are complete. □

It is immediate to see that the number of atoms in $T(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ is in $O(|\mathcal{F}| + |\mathcal{A}|)$.[9] Moreover the number of clauses generated by the Universal Axioms is in $O(P_0|\mathcal{A}|)$ where P_0 is the maximal number of fluents mentioned in an action (usually a small number). The number of clauses generated by the Explanatory Frame Axioms is in $O(|\mathcal{F}|)$. Finally, the number of clauses generated by the CEA is in $O(|\mathcal{A}|^2)$.

The quadratic growth of the number of CEA in the number of actions can produce encodings of unmanageable size when the considered security protocol exchanges many complex messages. It is possible to avoid the addition of CEA altogether by using a bitwise representation of rule labels [EMW97]. However this comes to the cost of complicating the encoding and, more importantly, of ruling out compound actions from solutions thereby leading to longer solutions in most cases. The bitwise representation of actions is described in Section 4.2.1 and experimental results are given in Section 7.1.1. A further possibility is to check a posteriori the linearizability of solutions found by applying the abstraction/refinement strategy proposed in [AC04a] and explained in Section 4.3.2.

Bitwise Action Representation

The bitwise action representation does not require CEA. In order to represent univocally each action in $\mathcal{A} = \{a_1, \ldots, a_{|\mathcal{A}|}\}$, $\omega = \lceil log_2|\mathcal{A}| \rceil$ propositional variables $(b_1, \ldots, b_\omega)$ are used. Let $\|a_j\|$ the propositional formula whose model identifies the bitwise representation of the action a_j, then the combination of the aforementioned standard linear encoding with the Bitwise Action Representation (BAR) results in the formula $T(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ defined as in 4.2.1 where the conflict exclusion axioms are neglected and each occurrence of a^i is replaced by $\|a\|^i$.[10] The number of propositional atoms in $T(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ decreases, with respect to the standard linear encoding, to $O(|\mathcal{F}| + \omega)$. However, while the number of clauses generated by the Universal Axioms does not change and the CEA are not required, the number of clauses generated by the Explanatory Frame Axioms combined with the bitwise action representation becomes, in the worst case, exponential in the number of actions (if a naive clausification is used). Precisely, it is in $O(\omega^{|\mathcal{A}|}|\mathcal{F}|)$.

To avoid this exponential growth it is sufficient to restore in the Explanatory Frame Axioms the propositional variables associated with the actions in place of their bitwise representation and to extend $T(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ with the following formulae:

[9] Let S be a set of elements, then $|S|$ indicates its cardinality.

[10] Notice that $\|a\|^i$ is the formula $\|a\|$ in which each propositional variable occurring in it, is indexed by i.

Bitwise Axioms: for each $a \in \mathcal{A}$:

$$\|a\|^i \equiv a^i$$

With respect to the standard linear encoding, the number of propositional atoms in $T(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ increases to $O(|\mathcal{F}| + |\mathcal{A}| + \omega)$, and the number of clauses in $T(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ is neither quadratic nor exponential in the number of actions. In fact the number of clauses generated by the Bitwise Axioms is in $O(|\mathcal{A}| + \omega|\mathcal{A}|)$.

From here on we call *bitwise linear encoding* this variant of the standard linear encoding combined with the bitwise action representation.

It is worth noticing that an important difference between the standard linear encoding and the bitwise linear encoding is that the former allows for some actions to be executed in parallel (at the same step), while the latter imposes that one (and only one) action can be executed at each step. Hence solutions found at step $\overline{k}$ by applying the standard linear encoding will be found at step $k \geq \overline{k}$ if the bitwise linear encoding is applied. Section 7.1.1 provides detailed results on a comparison between the standard linear encoding and the bitwise linear encoding.

So far we have seen how to apply linear encodings to a planning system Θ. To apply one of them to a planning problem $\Pi = \langle \Theta, \mathcal{G} \rangle$, it suffices to define the formula encoding the bad states $\mathcal{G}$ simply as follows:[11]

$$B(\mathbf{f}^k) = \bigvee_{G \in \mathcal{G}} \left\{ \bigwedge \{f^k : f \in G\} \wedge \bigwedge \{\neg f^k : f \in \mathcal{F}, f \notin G\} \right\} \tag{4.10}$$

It is immediate to see that the formula $B(\mathbf{f}^k)$ is such that if M is a model of $B(\mathbf{f}^k)$ then $\{f \in \mathcal{F} : f^k \in M\} \in \mathcal{G}$ and, conversely, if $S \in \mathcal{G}$ then $\{f^k : f \in S\}$ is a model of $B(\mathbf{f}^k)$.

4.2.2 Graphplan-based Encodings

By using linear encodings, the structure of the formula encoding the transition relation, namely $T^{\mathsf{L}}(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$,[12] is independent from the time step i and this means that important simplifications are possible on the resulting formula. For instance, not all the actions are applicable at time step 0, yet the formula

$$T^{\mathsf{L}}(\mathbf{f}^0, \mathbf{a}^0, \mathbf{f}^1) \tag{4.11}$$

[11] Since the number of goal states is in the worst case exponential in the cardinality of $\mathcal{F}$, the number of clauses in (4.10) is exponential in the number of fluents too. This difficulty can be simply overcome by specifying $\mathcal{G}$ as a formula instead of as an enumeration of states (e.g. $\mathcal{G} = (f_1 \wedge \neg f_2)$ compactly represents the set of all the goal states in which the fluents f_1 and f_2 must be true and false respectively).

[12] Notice that we introduce the superscript $^{\mathsf{L}}$ to denote that the formula is encoded by means of linear encodings. Hence F^{L} denotes that F has been generated by means of linear encodings, while F simply indicates that graphplan-based encodings has been used to construct the formula.

encodes the effects of all possible actions, also those that are not applicable in any initial state. By looking at the initial states it is possible to build a simpler version of (4.11), say $T_0(\mathbf{f}^0, \mathbf{a}^0, \mathbf{f}^1)$, specifying the effect of only those actions that are applicable in an initial state. The same line of reasoning can be applied at the subsequent steps: by computing an over-approximation of the reachable states at time step i we can then determine a simplified encoding of the transition relation at time step i, say $T_i(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$, for $i = 0, \ldots, k-1$. Formally, the simplified encoding of such an over-approximation will be such that for every $j = 0, \ldots, k$ the formulae encoding the planning system by means of the linear encoding, i.e. $I^{\mathsf{L}}(\mathbf{f}^0) \wedge \bigwedge_{i=0}^{j-1} T^{\mathsf{L}}(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$, and by means of the graphplan-based encoding, i.e. $I(\mathbf{f}^0) \wedge \bigwedge_{i=0}^{j-1} T_i(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$, are logically equivalent.

The graphplan-based encoding is defined on the basis of the above ideas. Preliminary to the generation of the encoding is the construction of a data structure, called planning graph, used to determine for each time step an over-approximation of the reachable states, [BF95]. More in general, the planning graph concisely represents an over-approximation of the forward search tree and the encoding of the transition relation can be drastically simplified by exploiting this information. A nice feature of a planning graph is that it is immediate to conclude when none of the goal states is included in the over-approximation. However if the over-approximation contains a goal state, we cannot conclude that such a state is truly reachable. A variety of approaches have been proposed in the literature to solve this problem. In [BF95] a backward search procedure is used to determine the reachability of a goal state. We apply an approach, first proposed in [KMS96], that consists in building a propositional formula whose models (if any) correspond to the solutions of the original planning problem.

In the rest of this section we will first describe how to construct a planning graph representing the over-approximation of the reachable states for the time step k, and then we will present how to exploit such an over-approximation while building the propositional formula $I(\mathbf{f}^0) \wedge \bigwedge_{i=0}^{k-1} T_i(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ that encodes the unfolding of the transition relation up to k steps.

Planning Graphs

Let $\Theta = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{O} \rangle$ be a planning system and $k \geq 0$. The *k-planning graph* for Θ is a directed acyclic graph

$$[\![\Theta]\!]_k = \langle N_f, N_a, \underset{pre}{\longrightarrow}, \underset{add}{\longrightarrow}, \underset{del}{\longrightarrow}, \oplus_f, \oplus_a \rangle$$

where N_f is a time-indexed family of sets of *fluent nodes*, i.e. $N_f^i \subseteq \mathcal{F}$ is the set of fluent nodes of *layer* i; similarly $N_a \subseteq \mathcal{A}$ is a time-indexed family of sets of *action nodes*, i.e. N_a^i is the set of action nodes of *layer* i; $\underset{pre}{\longrightarrow}$ is a time-indexed relation between fluent nodes and action nodes, i.e.

$\xrightarrow[pre]{}^{i} \subseteq N_a^i \times N_f^i$, whose instances are called *preconditions* edges; $\xrightarrow[add]{}$ and $\xrightarrow[del]{}$ are time-indexed binary relations between action nodes and fluent nodes, i.e. $\xrightarrow[add]{}^{i} \subseteq N_a^i \times N_f^{i+1}$ and $\xrightarrow[del]{}^{i} \subseteq N_a^i \times N_f^{i+1}$, whose instances are called *add* and *delete edges* respectively; finally $\oplus_f$ and $\oplus_a$ are time-indexed (irreflexive and commutative) relations of mutual exclusion (*mutex* for short) between fluents (i.e. $\oplus_f^i \subseteq N_f^i \times N_f^i$) and actions (i.e. $\oplus_a^i \subseteq N_a^i \times N_a^i$) respectively.

As a preliminary step to the construction of the k-planning graph for $\Theta = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{O} \rangle$ the sets $\mathcal{A}$ and $\mathcal{O}$ are extended with the *no-operation actions* and operators of the form $\mathrm{nop}(f)$ and $(f \overset{\mathrm{nop}(f)}{\Rightarrow} f;)$ respectively, for all $f \in \mathcal{F}$. The k-planning graph for Θ is then inductively defined as follows:

(PG1) $f \in N_f^0$ if and only if $f \in S$ for some $S \in \mathcal{I}$; moreover, $f \in N_f^i$ if and only if there exists $a \in N_a^{i-1}$ such that either $a \xrightarrow[add]{}^{i-1} f$ or $a \xrightarrow[del]{}^{i-1} f$, for $i = 1, \ldots, k$;

(PG2) $a \in N_a^i$ if and only if $\mathrm{pre}(a) \subseteq N_f^i$ and for all fluents $f, f' \in \mathrm{pre}(a)$ not $f \oplus_f^i f'$, for $i = 0, \ldots, k-1$;

(PG3) for each $a \in N_a^i$ and for $i = 0, \ldots, k-1$:

$a \xrightarrow[pre]{}^{i} f$ if and only if $f \in \mathrm{pre}(a)$,

$a \xrightarrow[add]{}^{i} f$ if and only if $f \in \mathrm{add}(a)$, and

$a \xrightarrow[del]{}^{i} f$ if and only if $f \in \mathrm{del}(a)$;

(PG4) $a \oplus_a^i a'$ if and only if $a \neq a'$ and either

(a) there exists $f \in \mathcal{F}$ s.t. $a \xrightarrow[del]{}^{i} f$, and $a' \xrightarrow[add]{}^{i} f$ or $a' \xrightarrow[pre]{}^{i} f$, or

(b) there exist $f, f' \in \mathcal{F}$ s.t. $a \xrightarrow[pre]{}^{i} f$, $a' \xrightarrow[pre]{}^{i} f'$, and $f \oplus_f^i f'$,

for $i = 0, \ldots, k-1$;

(PG5) $f \oplus_f^0 f'$ if and only if $f, f' \in N_f^0$ and $\{f, f'\} \not\subseteq S$ for all $S \in \mathcal{I}$; moreover, $f \oplus_f^i f'$ iff $f \neq f'$ and for all $a, a' \in N_a^{i-1}$ such that $a \xrightarrow[add]{}^{i-1} f$ and $a' \xrightarrow[add]{}^{i-1} f'$ we have $a \oplus_a^{i-1} a'$, for $i = 1, \ldots, k$.

It can be shown that the time needed to build the k-planning graph for Θ is polynomial in the size of Θ.

As pointed out in [DSK00], it is possible to isolate an interesting class of mutexes on actions which is the class of *static mutexes* denoted with $\widehat{\oplus}$. Two actions are statically in mutex if one action deletes any precondition or effect of the other, i.e. static mutexes are constructed by considering only part (a) of PG4. It is immediate to see that the computation of static mutexes at

the layer i of the planning graph only depends on the actions present at the i-th layer. Vice versa, the rest of the mutexes—called *dynamic mutexes*, denoted with $\widetilde{\oplus} = \oplus \setminus \widehat{\oplus}$, and constructed through part (b) of PG4 and PG5—at a layer $i > 0$ of the planning graph, requires the computation of the mutexes at the previous layers.

An interesting point shown in [DSK00] is that, on average, the computation of the dynamic mutexes is time-expensive. Besides this, it is worth pointing out that all the dynamic mutexes, except those imposed at layer 0 to restrict the over-approximation on the possible initial states, are implied by static mutexes. These observations led us at investigating other variants of the graphplan-based encoding in which weaker version of the mutex relation are computed. It must be noted that by considering a weaker version of the mutex relation, we obtain a coarser over-approximation of the forward search tree since larger is the mutex relation, the more accurate is the over-approximation.

The construction of a planning graph correspondent to a simple planning problem, characterised by four operators and five fluents, is presented in Example 16.

Example 16. *Let us consider the planning system* $\Theta = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{O} \rangle$ *where* $\mathcal{F} = \{j, x, y, w, z\}$, $\mathcal{A} = \{a, b, c, d\}$, $\mathcal{I} = \{\{x, y\}\}$, *and* $\mathcal{O}$ *is given by the following four operators:*

$$x \overset{a}{\Rightarrow} y, z; \qquad w \overset{c}{\Rightarrow} ; \neg z \qquad x, z \overset{b}{\Rightarrow} w; \neg x \qquad j \overset{d}{\Rightarrow} x;$$

The forward search tree associated to Θ *up to depth 3 is depicted in Figure 4.1. In more details, by applying the action a the system moves from the initial state* $\{x, y\}$ *to the state* $\{x, y, z\}$ *(i.e., the fluent z is added by a); by executing any of the applicable no-operation actions—this is denoted by* $nop(\_)$*—the system remains in its current state* $\{x, y\}$ *and the same branch of the forward search tree can be repeated; no other actions can be applied from the initial state. From the state* $\{x, y, z\}$*, besides the actions* $nop(\_)$*, both the actions a, and b can be applied: by executing either a or any of the* $nop(\_)$ *the system remains in* $\{x, y, z\}$*; by applying b the system reaches the state* $\{y, z, w\}$ *(i.e., the fluent x is deleted and the fluent w is added). Finally, the only actions applicable from* $\{y, z, w\}$ *are c and* $nop(\_)$*: the former deletes the fluent z and makes the system to reach the state* $\{w, y\}$*; the latter does not cause any state transition.*

Figure 4.2 shows the 3-planning graph, $[\![\Theta]\!]_3$*, associated to* Θ*. First of all the set of fluent nodes at layer 0 is computed, i.e.* $N_f^0 = \{x, y\}$*. Secondly the mutex relation on the fluents belonging to this set is calculated, i.e.* $\oplus_f^0 = \{\}$*. Then the following sequence of operations is repeated for each time step* $i = 0, 1, 2$*:*

(i) the set of action nodes at layer i is computed, e.g. $N_a^0 = \{a, nop(x), nop(y)\}$*;*

(ii) the mutex relation on the actions at layer i is calculated, e.g. $a \oplus_a^1 b$ and $b \oplus_a^1 nop(x)$ at layer 1;

(iii) the set of fluent nodes at layer $i+1$ is computed, e.g. $N_f^1 = \{x, y, z\}$;

(iv) the mutex relation on the fluents at layer $i+1$ is calculated, e.g. $x \oplus_f^2 w$ at layer 2.

Concerning the different classes of mutex relations, it is immediate to see that while $(a \oplus_a^1 b) \in \widehat{\oplus}$ is a static mutex, $(x \oplus_f^2 w) \in \widetilde{\oplus}$ is a dynamic mutex.

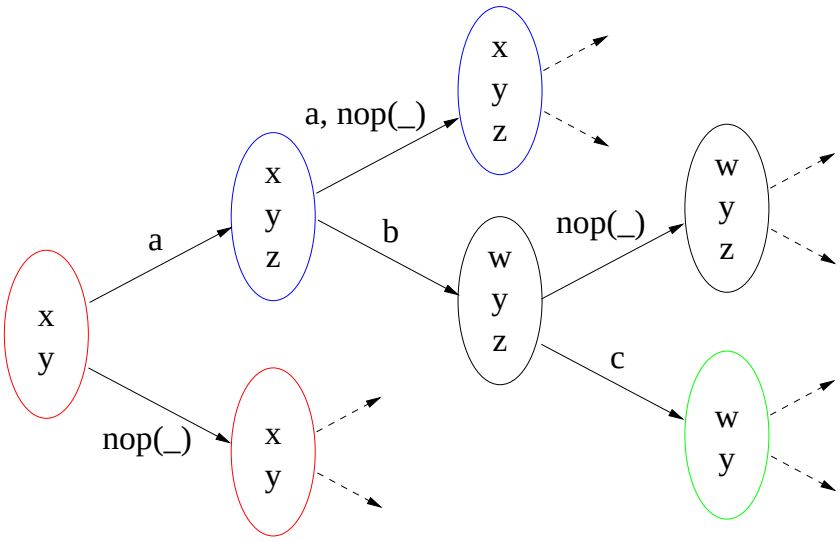

Figure 4.1: Forward search tree

Let us now formally define the over-approximation represented by a planning graph. We associate each fluent layer N_f^i with the set of states that may be reached at step $i \leq k$, i.e.

$$MBR(([\![\Theta]\!])_k, i) = \{S \subseteq N_f^i : \text{ not } f_1 \oplus_f^i f_2 \text{ for all } f_1, f_2 \in S\}$$

It is easy to show that $MBR(([\![\Theta]\!])_k, i)$ is an over-approximation of the states reachable in i steps, for $i = 0, \ldots, k$. That is, if S is reachable in i steps, then $S \in MBR(([\![\Theta]\!])_k, i)$, for $i = 0, \ldots, k$. The converse does not necessarily hold (see for instance Example 17).

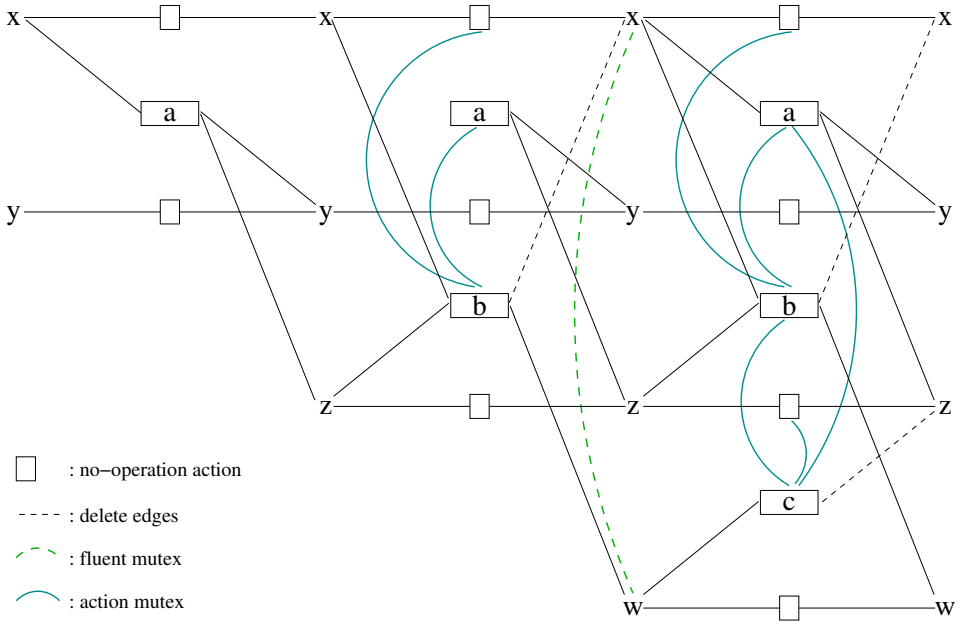

Figure 4.2: Planning Graph

Example 17. *With reference to the planning system Θ discussed in Example 16 and its associated planning graph shown in Figure 4.2, it is immediate to see that*

$$MBR((\!|\Theta|\!)_3, 1) = \{\{x\}, \{y\}, \{z\}, \{x, y\}, \{x, z\}, \{y, z\}, \{x, y, z\}\}$$

From this we are ensured that all the states in $2^{\mathcal{F}} \setminus MBR((\!|\Theta|\!)_3, 1)$, i.e. all the states containing the fluent w, are unreachable at depth 1. Yet we cannot guarantee that a state in $MBR((\!|\Theta|\!)_3, 1)$ is truly reachable. To show this it suffices to consider the state $\{x\}$ that is in $MBR((\!|\Theta|\!)_3, 1)$, but it cannot be reached at depth 1 as shown by the forward search tree of Figure 4.1.

Given a planning problem $\Pi = \langle \Theta, \mathcal{G} \rangle$, if $MBR((\!|\Theta|\!)_k, k) \cap \mathcal{G} = \emptyset$, then no state in $\mathcal{G}$ can be reached in k steps. However if there exists a goal state $S \in MBR((\!|\Theta|\!)_k, k) \cap \mathcal{G}$, we cannot conclude that S is truly reachable.

From Planning Graphs to SAT

Let $\Theta = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{O} \rangle$ be a planning system and let $(\!|\Theta|\!)_k = \langle N_f, N_a, \xrightarrow[pre]{}, \xrightarrow[add]{}, \xrightarrow[del]{}, \oplus_f, \oplus_a \rangle$ be its associated k-planning graph. The problem of establishing if a state S included in the over-approximation of the forward search tree for time step k (i.e., $S \in MBR((\!|\Theta|\!)_k, k)$) is reachable can be reduced to checking if $M = \{f^k : f \in S\}$ is a model of $[\![\Theta]\!]_k =$

$I(\mathbf{f}^0) \wedge \bigwedge_{i=0}^{k-1} T_i(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ where the initial states are encoded as

$$I(\mathbf{f}^0) = \bigvee_{S_0 \in \mathcal{I}} \left\{ \bigwedge \{ f^0 : f \in N_f^0, f \in S_0 \} \right\} \tag{4.12}$$

and the transition relation is built either through the parallelised encoding introduced in [KMS96] or by adapting the schema of formulae used for the linear encoding. These two variants mainly differ in the encoding of inertia law.[13] Namely, the former uses the Backward Chaining Axioms, while the latter exploits the already described Explanatory Frame Axioms. In the rest of this section we will describe in details the Explanatory Frame Axioms schema and we will provide soundness and completeness results for it. Moreover in order to have a term of comparison we will also present the Backward Chaining Axioms schema as defined in [KMS96].

Explanatory Frame Axioms schema (EFA). Such a schema exploits the axioms used for the linear encoding technique but instead of being applied on the whole planning system they are applied on the planning graph over-approximation only. With respect to the Backward Chaining Axioms schema (see below) it does not require no-operation actions that can thus be neglected in the construction of the planning graph. Given this the formula encoding the transition relation $T_i(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ for $i = 0, \ldots, k-1$ is the conjunction of the formulae defined by following schemata:

Universal Axioms: for each $a \in N_a^i$

$$a^i \supset \bigwedge \{ f^i \mid a \xrightarrow[pre]{}^i f \} \tag{4.13}$$

$$a^i \supset \bigwedge \{ f^{i+1} \mid a \xrightarrow[add]{}^i f \} \tag{4.14}$$

$$a^i \supset \bigwedge \{ \neg f^{i+1} \mid a \xrightarrow[del]{}^i f) \} \tag{4.15}$$

Explanatory Frame Axioms (EFA): for all $f \in N_f^i$

$$(f^i \wedge \neg f^{i+1}) \supset \bigvee \left\{ a^i \mid a \xrightarrow[del]{}^i f \right\} \tag{4.16}$$

$$(\neg f^i \wedge f^{i+1}) \supset \bigvee \left\{ a^i \mid a \xrightarrow[add]{}^i f \right\} \tag{4.17}$$

Extra EFA: for all $f \in (N_f^{i+1} \setminus N_f^i)$

$$f^{i+1} \supset \bigvee \left\{ a^i \mid a \xrightarrow[add]{}^i f \right\} \tag{4.18}$$

[13] The inertia law states that nothing changes unless there is a reason to.

Mutex Axioms: for all $p_1, p_2 \in (N_a^i \cup N_f^i)$ such that $p_1 \oplus^i p_2$:

$$\neg(p_1^i \wedge p_2^i) \tag{4.19}$$

The intuition behind the above axioms except (4.18) has been already explained in Section 4.2.1. The extra explanatory frame axioms simply state that if a fluent is such that it holds at time step $i+1$, and it occurs in the $(i+1)$-th layer N_f^{i+1} and not in the previous one N_f^i, then at least one of the actions occurring in the i-th action layer and adding f must hold at the time step i. Basically the extra explanatory frame axiom for the fluent f is just a simplification of the explanatory frame axiom for f where f is imposed to be false at time step i. Notice that this is allowed by the fact that if a fluent f is introduced only at the $(i+1)$-th fluent layer of the planning graph, then there does not exist a state reachable in i steps where f holds.

The soundness and completeness of the EFA schema amount to showing that $I(\mathbf{f}^0)$ enjoys requirement R1 (see Lemma 7) and $T_i(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ enjoys part (a) and part (b) of requirement R2 (see Lemma 8 and Lemma 9 respectively).

Lemma 7. *$I(\mathbf{f}^0)$ is such that if M is a model of $I(\mathbf{f}^0)$ then $\{f \in \mathcal{F} : f^0 \in M\} \in \mathcal{I}$ and, conversely, if $S \in \mathcal{I}$ then $\{f^0 : f \in S\}$ is a model of $I(\mathbf{f}^0)$.*

Proof. This proof readily follows from the definition of $I(\mathbf{f}^0)$ given in (4.12). □

Lemma 8. *If M is a model of $T_i(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ then the actions in $\Lambda = \{a \in \mathcal{A} : a^i \in M\}$ are composable and their parallel composition $\|(\Lambda)$ is linearizable and such that $\mathrm{app}^{\rightarrow}_{\|(\Lambda)}(S) = S'$ for $S = \{f \in \mathcal{F} : f^i \in M\}$ and $S' = \{f \in \mathcal{F} : f^{i+1} \in M\}$ for $i = 0, \ldots, k-1$.*

Proof. It must be noted that the formula $T_i(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ is defined over a subset of the atoms in $T^{\mathsf{L}}(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ and namely over those propositions occurring in $\{p^i : p \in N_f^i\} \cup \{p^i : p \in N_a^i\} \cup \{p^{i+1} : p \in N_f^{i+1}\}$. As a consequence Λ, S, and S', that by hypothesis are defined in terms of the model M, are such that $\Lambda \subseteq N_a^i$, $S \subseteq N_f^i$, and $S' \subseteq N_f^{i+1}$. Therefore for any action $a \in \Lambda$ we have that $\mathrm{pre}(a)$, $\mathrm{add}(a)$, and $\mathrm{del}(a)$ can be used interchangeably with $\{f : a \xrightarrow[pre]{}^i f\}$, $\{f : a \xrightarrow[add]{}^i f\}$, and $\{f : a \xrightarrow[del]{}^i f\}$ respectively. Given this the proof goes along the lines of Lemma 5 in which (4.4)-(4.6), (4.7), and (4.9) are replaced by (4.13)-(4.15), (4.16), and (4.19) respectively. The only significant difference concerns "case 2" of Lemma 5 that we describe entirely here.

case 2 (only if): Let $p^{i+1} \in M$, we must prove that $p \in (S \setminus \bigcup_{a\in\Lambda} \mathrm{del}(a)) \cup \bigcup_{a\in\Lambda} \mathrm{add}(a)$ i.e. either both $p \in S$ and $p \notin \bigcup_{a\in\Lambda} \mathrm{del}(a)$, or $p \in \bigcup_{a\in\Lambda} \mathrm{add}(a)$. Since $p^{i+1} \in M$, then for all $\tilde{a}$ such that $p \in \mathrm{del}(\tilde{a})$

the truth of $(\tilde{a}^i \supset \neg p^{i+1})$, i.e. the formula constructed through axiom (4.15), in M requires $\tilde{a}^i \notin M$. Because of this and from the definition of Λ, we derive that $\tilde{a} \notin \Lambda$ and therefore that $p \notin \bigcup_{a\in\Lambda} \mathrm{del}(a)$. To conclude the proof we must show that either $p \in S$ or $p \in \bigcup_{a\in\Lambda} \mathrm{add}(a)$, that is equivalent to prove that either $p^i \in M$ or there exists an action a such that $a^i \in M$ and $p \in \mathrm{add}(a)$. To prove this we must notice that $p^{i+1} \in M$ if and only if $p \in S' = \{f \in \mathcal{F} : f^{i+1} \in M\} \subseteq N_f^{i+1}$. Moreover by definition of planning graph we know that $N_f^i \subseteq N_f^{i+1}$ and therefore $N_f^{i+1} = N_f^i \cup N_f^{i+1} \setminus N_f^i$. We thus consider the following two cases.

case 2.1: Let $p \in N_f^i$, the truth of the formula constructed through axiom (4.17) and simplified by using the assumption $p^{i+1} \in M$ implies that either $p^i \in M$ or there exists an action a such that $a^i \in M$ and $p \in \mathrm{add}(a)$.

case 2.2: Let $p \in N_f^{i+1} \setminus N_f^i$. This implies trivially that $p \notin S$ (since by definition $S \subseteq N_f^i$) and thus we must prove that there exists an action a such that $a^i \in M$ and $p \in \mathrm{add}(a)$. This is simply derived from the truth value of the formula constructed through axiom (4.18) and simplified by using the assumption $p^{i+1} \in M$.

□

Theorem 6 (Soundness of the EFA schema). *If M is a model of $[\![\Theta]\!]_k$ for $k \geq 0$, then $S_0 a_0 S_1 \cdots S_{k-1} a_{k-1} S_k$ is a planning path of $\hat{\Theta}$, where $S_i = \{f \in \mathcal{F} : f^i \in M\}$ for $i = 0, \ldots, k$ and a_i is the parallel composition of the actions in $\{a \in \mathcal{A} : a^i \in M\}$ for $i = 0, \ldots, k-1$.*

Proof. Lemma 7 and Lemma 8 ensure that graphplan-based encodings using the EFA schema enjoy requirement R1 and part (a) of requirement R2 respectively. Because of Theorem 2 we are thus guaranteed that graphplan-based encodings using the EFA schema are sound. □

Lemma 9. *If $S \subseteq \mathcal{F}$ is reachable in i steps in $\hat{\Theta}$, the actions in $\Lambda = \{a_1, \ldots, a_n\} \subseteq \mathcal{A}$ are linearizable and their parallel composition $\|(\Lambda)$ is applicable in S, and $S' = \mathrm{app}^{\Rightarrow}_{\|(\Lambda)}(S)$, then $M = \{f^i : f \in S\} \cup \{f^{i+1} : f \in S'\} \cup \{a_1^i, \ldots, a_n^i\}$ is a model of $T_i(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ for $i = 0, \ldots, k-1$.*

Proof. The proof goes along the lines of Lemma 6. The differences are minor but many and therefore we prefer to present the overall proof here.

Let $\tilde{a}$ and f be an action and a fluent respectively. First of all notice that by definition of planning graph the reachability of S in i steps implies that $S \subseteq N_f^i$. Similarly the reachability of S' in one step from S by applying the actions in Λ implies that $S' \subseteq N_f^{i+1}$, $\Lambda \subseteq N_a^i$, and that for each $a \in N_a^i$

pre(a), add(a), and del(a) can be used interchangeably with $\{f : a \xrightarrow[pre]{}^{i} f\}$, $\{f : a \xrightarrow[add]{}^{i} f\}$, and $\{f : a \xrightarrow[del]{}^{i} f\}$ respectively. Because of this and by definition of M and Λ, it is immediate to see that *(i)* $\tilde{a} \in \Lambda$ if and only if $\tilde{a}^i \in M$, *(ii)* $f \in S$ if and only if $f^i \in M$, and *(iii)* $f \in S'$ if and only if $f^{i+1} \in M$.

To show that M is model of $T(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$, we must prove that M satisfies (4.13)-(4.19).

The formulae (4.13), (4.14), and (4.15) are true in M if and only if for every action $\tilde{a} \in N_a^i$ if $\tilde{a}^i \in M$ (i.e. $\tilde{a} \in \Lambda$), then *(a)* any fluent $f \in \text{pre}(\tilde{a})$ is such that $f^i \in M$ (i.e. $f \in S$), *(b)* any fluent $f \in \text{add}(\tilde{a})$ is such that $f^{i+1} \in M$ (i.e. $f \in S'$), and *(c)* any fluent $f \in \text{del}(\tilde{a})$ is such that $f^{i+1} \notin M$ (i.e. $f \notin S'$). This is trivially satisfied in M by any action $\tilde{a}$ that is in N_a^i but not in Λ. Let us now consider the case in which $\tilde{a} \in \Lambda$. By hypothesis the parallel composition $\|(\Lambda)$ is applicable in S (i.e. $\text{pre}(\|(\Lambda)) = \bigcup_{a\in\Lambda} \text{pre}(a) \subseteq S$), and the application of $\|(\Lambda)$ leads to $S' = (S \setminus \bigcup_{a\in\Lambda} \text{del}(a)) \cup \bigcup_{a\in\Lambda} \text{add}(a)$. Since $\tilde{a} \in \Lambda$, we have that $\text{pre}(\tilde{a}) \subseteq \text{pre}(\|(\Lambda)) \subseteq S$ and thus *(a)* is proved. Similarly $\text{add}(\tilde{a}) \subseteq S'$ and therefore also *(b)* is demonstrated. Finally, to prove *(c)* it suffices to show that $\text{del}(\tilde{a}) \cap S' = \emptyset$, i.e. both $\text{del}(\tilde{a}) \cap (S \setminus \bigcup_{a\in\Lambda} \text{del}(a)) = \emptyset$ and $\text{del}(\tilde{a}) \cap \bigcup_{a\in\Lambda} \text{add}(a) = \emptyset$. The former simply derives from the fact that $\tilde{a} \in \Lambda$ and thus $\text{del}(\tilde{a}) \subseteq \bigcup_{a\in\Lambda} \text{del}(a)$. By definition of parallel composition $\bigcup_{a\in\Lambda} \text{del}(a) \cap \bigcup_{a\in\Lambda} \text{add}(a) = \emptyset$ and since $\text{del}(\tilde{a}) \subseteq \bigcup_{a\in\Lambda} \text{del}(a)$ it follows immediately also the latter.

The formula (4.16) is true in M if and only if for every fluent $f \in N_f^i$ if $f^i \in M$ (i.e. $f \in S$) and $f^{i+1} \notin M$ (i.e. $f \notin S'$), then at least one of the actions, say $\tilde{a}$, that deletes f must be such that $\tilde{a}^i \in M$ (i.e. $\tilde{a} \in \Lambda$). It follows immediately that $M \models (4.16)$ holds for every fluent $f \in N_f^i$ such that either $f \notin S$ or $f \in S'$. By hypothesis $S' = (S \setminus \bigcup_{a\in\Lambda} \text{del}(a)) \cup \bigcup_{a\in\Lambda} \text{add}(a)$ and then

$$
\begin{aligned}
& f \notin S \vee f \in S' \\
&= f \notin S \vee (f \in S \wedge f \notin \textstyle\bigcup_{a\in\Lambda} \text{del}(a)) \vee f \in \bigcup_{a\in\Lambda} \text{add}(a) && \text{[by definition]} \\
&= f \notin S \vee f \notin \textstyle\bigcup_{a\in\Lambda} \text{del}(a) \vee f \in \bigcup_{a\in\Lambda} \text{add}(a) && \text{[by simplification]}
\end{aligned}
$$

As a consequence $M \models (4.16)$ holds for any fluent $f \in N_f^i$ such that either $f \notin S$, $f \notin \bigcup_{a\in\Lambda} \text{del}(a)$, or $f \in \bigcup_{a\in\Lambda} \text{add}(a)$. To conclude the proof about $M \models (4.16)$ it suffices to note that in the rest of the cases—i.e. $f \in N_f^i$ is such that both $f \in S$, $f \in \bigcup_{a\in\Lambda} \text{del}(a)$, and $f \notin \bigcup_{a\in\Lambda} \text{add}(a)$— $f \in \bigcup_{a\in\Lambda} \text{del}(a)$ i.e. there exists at least one action $\tilde{a} \in \Lambda$ such that $f \in \text{del}(\tilde{a})$ and thus (4.16) is true in M.

To prove $M \models (4.17)$ we proceed as above but dually. The formula (4.17) is true in M if and only if for every fluent $f \in N_f^i$ if $f^i \notin M$ (i.e. $f \notin S$) and $f^{i+1} \in M$ (i.e. $f \in S'$), then at least one of the actions, say $\tilde{a}$, that adds f must be such that $\tilde{a}^i \in M$ (i.e. $\tilde{a} \in \Lambda$). It follows immediately

that $M \models (4.17)$ holds for every fluent $f \in N_f^i$ such that either $f \in S$ or $f \notin S'$. By hypothesis $S' = (S \setminus \bigcup_{a\in\Lambda} \mathrm{del}(a)) \cup \bigcup_{a\in\Lambda} \mathrm{add}(a)$ and then

$$
\begin{aligned}
& f \in S \vee f \notin S' \\
&= f \in S \vee ((f \notin S \vee f \in \textstyle\bigcup_{a\in\Lambda} \mathrm{del}(a)) \wedge f \notin \bigcup_{a\in\Lambda} \mathrm{add}(a)) && \text{[by definition]} \\
&= f \in S \vee f \notin \textstyle\bigcup_{a\in\Lambda} \mathrm{add}(a) && \text{[by simplification]}
\end{aligned}
$$

As a consequence $M \models (4.17)$ holds for any fluent $f \in N_f^i$ such that either $f \in S$ or $f \notin \bigcup_{a\in\Lambda} \mathrm{add}(a)$. To conclude the proof about $M \models (4.17)$ it suffices to observe that in the rest of the cases—i.e. $f \in N_f^i$ is such that both $f \notin S$ and $f \in \bigcup_{a\in\Lambda} \mathrm{add}(a)$—$f \in \bigcup_{a\in\Lambda} \mathrm{add}(a)$ i.e. there exists at least one action $\tilde{a} \in \Lambda$ such that $f \in \mathrm{add}(\tilde{a})$ and thus (4.17) is true in M.

The formula (4.18) is true in M if and only if for every fluent $f \in N_f^{i+1} \setminus N_f^i$ if $f^{i+1} \in M$ (i.e. $f \in S'$), then at least one of the actions, say $\tilde{a}$, that adds f must be such that $\tilde{a}^i \in M$ (i.e. $\tilde{a} \in \Lambda$). By definition of S' and since $S \subseteq N_f^i$ it follows that for all fluent $f \in N_f^{i+1} \setminus N_f^i$, $f \in S'$ if and only if $f \in \bigcup_{a\in\Lambda} \mathrm{add}(a)$. Hence $M \models (4.18)$ holds for all $f \in N_f^{i+1} \setminus N_f^i$ such that $f \in S'$. In the other case, i.e. $f \in N_f^{i+1} \setminus N_f^i$ such that $f \notin S'$, again $M \models (4.18)$ trivially holds since the antecedent of the implication is false.

To conclude the overall proof we are left with proving that $M \models (4.19)$. As mentioned before mutexes are divided into the two disjoint classes of static mutexes over actions ($\widehat{\oplus}$) and of dynamic mutexes ($\widetilde{\oplus}$), i.e. $\oplus = \widetilde{\oplus} \cup \widehat{\oplus}$ and $\widetilde{\oplus} \cap \widehat{\oplus} = \emptyset$. Moreover dynamic mutexes are implied by static mutexes over actions and therefore if we prove that for all pair of actions $a_1, a_2 \in N_a^i$ such that $a_1 \widehat{\oplus}^i a_2$ $M \models \neg(a_1^i \wedge a_2^i)$ holds, then also $M \models (4.19)$ is proved. This amount showing that for all pair of actions $a_1, a_2 \in N_a^i$ such that $a_1 \widehat{\oplus}^i a_2$ either $a_1^i \notin M$ (i.e. $a_1 \notin \Lambda$) or $a_2^i \notin M$ (i.e. $a_2 \notin \Lambda$). This is a simple consequence of the facts that static mutexes are constructed by considering only part (a) of PG4 and that the actions in Λ are applicable and linearizable, i.e. their effects do not contradict each other and they are pairwise non-interfering. □

Theorem 7 (Completeness of the EFA schema). *If $\pi_k = S_0 a_0 S_1 \cdots S_{k-1} a_{k-1} S_k$ is a planning path of $\hat{\Theta}$, then there exists a model M of $[\![\Theta]\!]_k$ such that $\pi(M) = \pi_k$.*

Proof. Lemma 7 and Lemma 9 ensure that graphplan-based encodings using the EFA enjoy requirement R1 and part (b) of requirement R2 respectively. Because of Theorem 3 we are thus guaranteed that graphplan-based encodings using the EFA are complete. □

To apply the EFA schema to a planning problem $\Pi = \langle \Theta, \mathcal{G} \rangle$, it suffices to define the formula encoding the bad states $\mathcal{G}$ simply as follows:[14]

$$B(\mathbf{f}^k) = \bigvee_{G \in \mathcal{G}, G \subseteq N_f^k} \Big\{ \bigwedge \{ f^k : f \in G \} \wedge \bigwedge \{ \neg f^k : f \in N_f^k, f \notin G \} \Big\} \quad (4.20)$$

It is interesting to notice how the planning graph previously computed allows (4.20) to encode only the goal states that are in the over-approximation by simply neglecting those that are known to be unreachable. So the formula (4.20) is such that if M is a model of $B(\mathbf{f}^k)$ then $\{ f \in N_f^k : f^k \in M \} \in \mathcal{G} \cap MBR(([\![\Theta]\!]_k, k)$ and, conversely, if $S \in \mathcal{G} \cap MBR(([\![\Theta]\!]_k, k)$ then $\{ f^k : f \in S \}$ is a model of $B(\mathbf{f}^k)$.

Backward Chaining Axioms schema (BCA). By applying such a schema, devised in [KMS96], the formula encoding the transition relation $T_i(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ for $i = 0, \ldots, k-1$ is given by conjoining the following schemes of formulae:

Preconditions Add-effects Axioms (PAA): for all a in N_a^i

$$a^i \supset \bigwedge \{ f^i \mid a \xrightarrow[pre]{}^i f \} \quad (4.21)$$

Backward Chaining Axioms (BCA): for all f in N_f^{i+1}

$$f^{i+1} \supset \bigvee \{ a^i \mid a \xrightarrow[add]{}^i f \} \quad (4.22)$$

Mutex Axioms: for all $p_1, p_2 \in (N_a^i \cup N_f^i)$ such that $p_1 \oplus^i p_2$:

$$\neg (p_1^i \wedge p_2^i) \quad (4.23)$$

Intuitively, the PAA state that if an action a holds at the time-index i, then its preconditions must hold at the same time-index, see formula (4.21). The BCA—see formula (4.22)—express that if a fluent f is true at time step $i+1$ then at least one of the actions occurring in the i-th action layer and adding f must hold at the time step i. Finally, the mutex axioms—see formula (4.23)—state, accordingly to the mutex relation computed during the construction of the planning graph, what pair of actions cannot be executed simultaneously.

[14] Since the number of goal states is in the worst case exponential in the cardinality of N_f^k, the number of clauses in (4.20) is exponential in the number of fluents too. This difficulty can be simply overcome by specifying $\mathcal{G}$ as a formula instead of as an enumeration of states (e.g. $\mathcal{G} = (f_1 \wedge \neg f_2)$ compactly represents the set of all the goal states in which the fluents f_1 and f_2 must be true and false respectively).

To define the formula encoding the bad states $\mathcal{G}$ so to apply the BCA schema to a planning problem $\Pi = \langle \Theta, \mathcal{G} \rangle$ being sure that any model of $[\![\Pi]\!]_k$—defined as in (4.2)—would correspond to a partial-order plan of Π, it is not as easy as in the case of the EFA schema. Essentially the BCA schema is not able to handle negation in the goals so that a preprocessing phase is required on the original planning problem Π in order to obtain a new planning problem $\tilde{\Pi}$ for which $B(\mathbf{f}^k)$ can be expressed in the spirit of (4.20) but without any negation. The basic idea is to introduce into $\tilde{\Pi}$ a new fluent $\mathsf{un}(f)$ that simulates the dual behaviour of the fluent f whose k-indexed proposition f^k occurs negated into (4.20). This means that any time f is true, then $\mathsf{un}(f)$ is false and vice versa. Let $\mathcal{U}$ be the set of these new fluents, namely $\mathsf{un}(f) \in \mathcal{U}$ if and only if f^k occurs negated into (4.20) i.e. there exists $G \in \mathcal{G}$ such that $f \in N_f^k$ and $f \notin G$. The new planning problem $\tilde{\Pi} = \langle \langle \tilde{\mathcal{F}}, \mathcal{A}, \tilde{\mathcal{I}}, \tilde{\mathcal{O}} \rangle, \tilde{\mathcal{G}} \rangle$ is obtained from $\Pi = \langle \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{O} \rangle, \mathcal{G} \rangle$ as follows:

- $\tilde{\mathcal{F}} = \mathcal{F} \cup \mathcal{U}$;
- $(S \cup \{\mathsf{un}(f) \in \mathcal{U} : f \notin S\}) \in \tilde{\mathcal{I}}$ if and only if $S \in \mathcal{I}$;
- $(Pre \overset{a}{\Rightarrow} Add \cup \{\mathsf{un}(f) \in \mathcal{U} : f \in Del\}; Del \cup \{\mathsf{un}(f) \in \mathcal{U} : f \in Add\}) \in \tilde{\mathcal{O}}$ if and only if $(Pre \overset{a}{\Rightarrow} Add; Del) \in \mathcal{O}$
- $(G \cup \{\mathsf{un}(f) \in \mathcal{U} : f \notin G\}) \in \tilde{\mathcal{G}}$ if and only if $S \in \mathcal{G}$;

It must be noted that the behaviours of $\tilde{\Pi}$ and of Π are very similar whereas $\tilde{\Pi}$ provides redundant informations on those fluents f such that $\mathsf{un}(f) \in \mathcal{U}$. It is easy to show that $\tilde{S}_0 a_0 \tilde{S}_1 \cdots \tilde{S}_{k-1} a_{k-1} \tilde{S}_k$ is a planning path of $\tilde{\Theta}$ if and only if $S_0 a_0 S_1 \cdots S_{k-1} a_{k-1} S_k$ is a planning path of Θ where $\tilde{S}_i = S_i \cup \{\mathsf{un}(f) \in \mathcal{U} : f \notin S_i\}$ for each $i = 0, \ldots, k$. As a consequence $a_0 \cdots a_{k-1}$ is a plan of $\tilde{\Pi}$ if and only if $a_0 \cdots a_{k-1}$ is a plan of Π. These results allow us to construct and solve $[\![\tilde{\Pi}]\!]_k$ instead of $[\![\Pi]\!]_k$. In doing this, the formula encoding the bad states $\tilde{\mathcal{G}}$ is defined as follows:[15]

$$B(\mathbf{f}^k) = \bigvee_{G \in \tilde{\mathcal{G}}, G \subseteq N_f^k} \left\{ \bigwedge \{f^k : f \in G\} \right\} \tag{4.24}$$

Any model of $[\![\tilde{\Pi}]\!]_k = [\![\tilde{\Theta}]\!]_k \wedge (4.24)$ corresponds to a partial-order plan of $\tilde{\Pi}$ (and thus of Π), and vice versa.

[15] Since the number of goal states is in the worst case exponential in the cardinality of $\mathcal{F}$, the number of clauses in (4.24) is exponential in the number of fluents too. This difficulty can be simply overcome by specifying $\mathcal{G}$ (and thus $\tilde{\mathcal{G}}$) as a formula instead of as an enumeration of states For instance, $\mathcal{G} = (f_1 \wedge \neg f_2)$ (as well as $\tilde{\mathcal{G}} = f_1 \wedge \mathsf{un}(f_2)$) compactly represents the set of all the goal states in which the fluents f_1 and f_2 must be true and false respectively.

The main differences between the EFA and the BCA schemata are here summarised. On the one hand the BCA axioms (4.21)-(4.23) look simpler than the EFA axioms (4.13)-(4.19). On the other hand, contrary to the BCA schema, the EFA schema does not require *no-operation* actions to enforce the inertia of fluents that means that in the best case $|\mathcal{A}|$ atoms are saved. Moreover the EFA schema is able to handle negation in the goals and therefore, contrary to the BCA schema, it is not required to introduce new atoms for representing the fluents occurring negated in the goal and to execute the aforementioned preprocessing transformation on the planning problem. These advantages of the EFA schema versus the BCA schema are confirmed, at least on the domain of security protocols, by the results we have obtained. More details on this are given in Section 7.1.2.

For what concerns a comparison between graphplan-based encodings and linear encodings, it is immediate to see that in the worst case the complexity results on the number of atoms and clauses generated by the graphplan-based encoding are identical to those of linear encodings. However, the practical considerations that usually $|N_f^i| \ll |\mathcal{F}|$ (for $i = 0, \ldots, k$) and $|N_a^i| \ll |\mathcal{A}|$ (for $i = 0, \ldots, k-1$) lead to the conclusion that in most cases the formulae generated by graphplan-based encodings are significantly smaller than those built with linear encodings. In Section 7.1.3 we will provide experimental evidence that on the security protocol application-domain graphplan-based encodings outperform linear encodings in terms of both timing and size of the propositional formulae.

As for linear encodings, also the formulae produced by means of graphplan-based encodings are quadratic in the size of the problem. Even by restricting to the aforementioned weaker version of the mutex relation where only static mutexes are computed, the formulae generated are in the worst case still quadratic in the number of actions. As for linear encodings, it is possible to avoid the addition of mutexes altogether by applying an abstraction/refinement strategy (see Section 4.3.2) that consists in checking a posteriori the linearizability of solutions found by totally neglecting the mutexes. However, experimental results show that by applying such a strategy to the graphplan-based encoding, we have not the same gain as in the linear encoding case (see Section 7.1.2).

4.3 The overall Approach

By applying the notions introduced in Chapters 3 and in the previous two sections we are able to build a propositional formula Φ_k^{Ξ} parametric in a given integer k and in a protocol insecurity problem Ξ that models the question of whether a protocol truly achieves its security requirements or not up to k steps. The number k indicates the depth of the search space the proposi-

tional formula is currently representing. In general the unsatisfiability of Φ_k^{Ξ} does not permit to conclude that the modelled protocol satisfies its security requirements, but only that the security protocol does not suffer from any attack of length up to k. In order to provide a procedure able to discover any attack on security protocols, it suffices to apply iterative deepening on k. This results in executing bounded model-checking of security protocols by means of a reduction to propositional formulae with iterative deepening on the number of steps.

In more details, after having modelled a security protocol and its requirements as a protocol insecurity problem Ξ (see Chapter 3), Ξ is first compiled into a planning problem Π by applying the methods described in Section 4.1 and then Π is reduced to a SAT formula $[\![\Pi]\!]_k$ using one of the encoding techniques presented in Section 4.2 for increasing values of k. The propositional formula generated at each step is fed to a SAT solver and as soon as a satisfiable formula is found, the corresponding model is translated back into attacks on Ξ which are reported to the user. Figure 4.3 depicts the high-level steps underlying our SAT-based Model-Checking approach. It consists of four main phases:

Phase 1: Modelling security protocols. Firstly the question of whether a protocol indeed achieves its security requirements or not, is formalised into a protocol insecurity problem by following the methodology presented in Chapter 3.

Phase 2: Protocol Insecurity Problems into Planning Problems. Secondly the protocol insecurity problem is compiled into a planning problem by applying the method explained in Section 4.1.

Phase 3: Planning via SAT. Thirdly the planning problem is tackled by means of the Planning via SAT procedure that consists in interleaving the SAT-encoding and the SAT-solving sub-phases at each time step k until a satisfiable assignment—from which a corresponding partial-order plan is then extracted—is found (if any).

Phase 4: Retrieving Attack. Finally an attack is retrieved from the partial-order plan and it is returned to the user.

The first two phases are entirely explained in Chapter 3 and Section 4.1. The last phase is quite simple and therefore let us here focus on the third one and namely on two variants of it. In the sequel we start in Section 4.3.1 by presenting the traditional Planning via SAT procedure enriched with important termination concerns and we conclude in Section 4.3.2 by describing

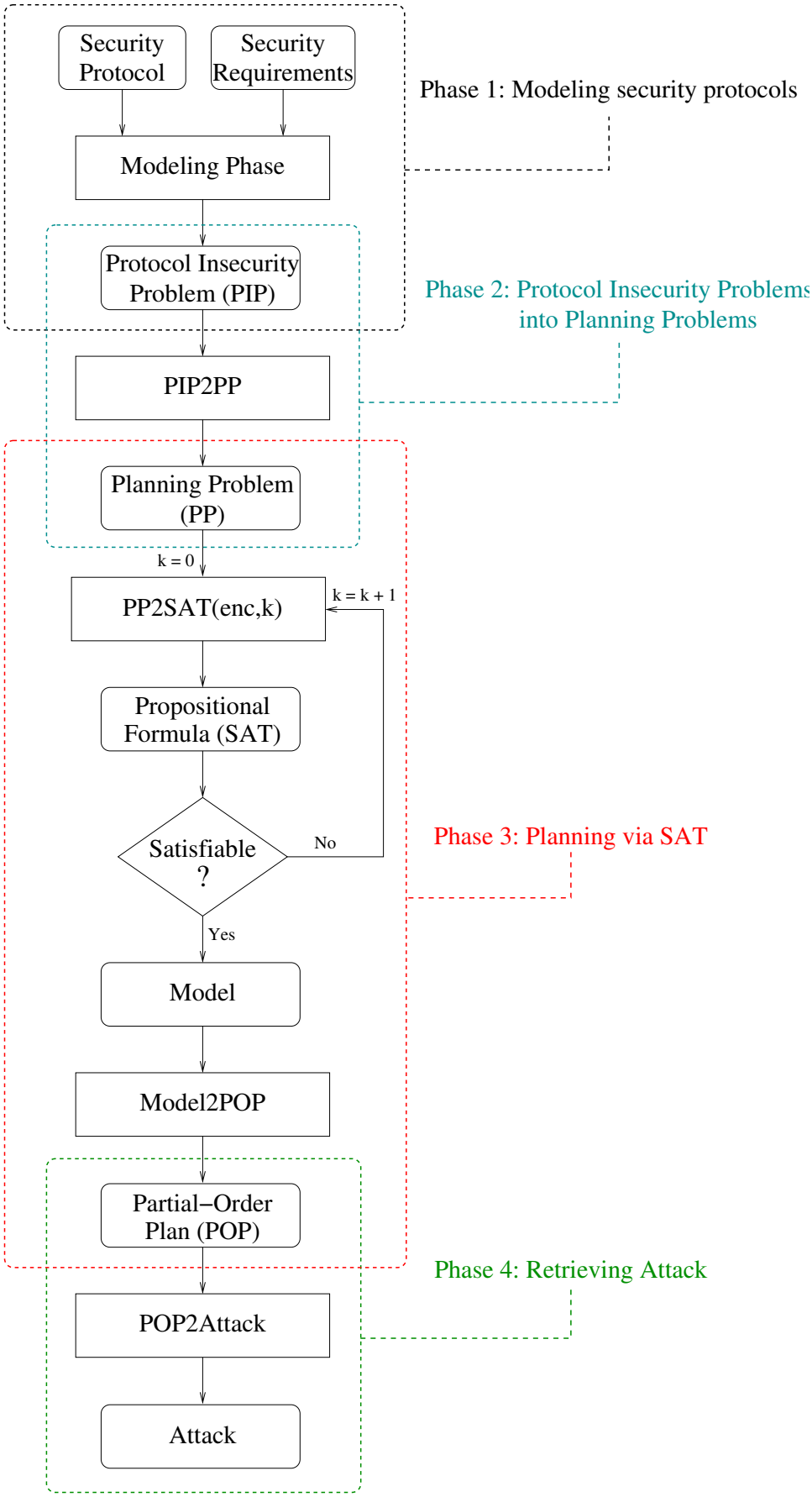

Figure 4.3: Approach

how to enhance the first one by means of an Abstraction/Refinement strategy which, by interleaving the encoding and solving phases, may lead to a significant improvement of the overall approach.

4.3.1 Planning via SAT

The Planning via SAT procedure, firstly introduced in [KMS96], performs a reduction of a planning problem into propositional formulae each one representing the entire search space up to a specified depth k. The basic idea is to apply iterative deepening on k by alternating at each step the phases of encoding to SAT—using either linear or graphplan-based encoding techniques—and of satisfiability checking. As soon as a satisfiable formula is found, the corresponding model is translated into a partial-order plan of the input planning problem.

As shown in Figure 4.3 the Planning via SAT phase starts by constructing the propositional formula for k equal to 0 that trivially results in

$$I(\mathbf{f}^0) \wedge B(\mathbf{f}^0) \tag{4.25}$$

that is the conjunction of the boolean representations of the initial states and of the goal states of the planning problem (see Section 4.2). If the formula (4.25) is satisfiable, then the empty sequence of actions ϵ is a trivial solution to the input planning problem. Otherwise k is increased to 1, and the SAT formula

$$I(\mathbf{f}^0) \wedge T_0(\mathbf{f}^0, \mathbf{a}^0, \mathbf{f}^1) \wedge B(\mathbf{f}^1) \tag{4.26}$$

is built and checked towards satisfiability. Again, if the formula (4.26) allows for a model M, then any possible permutation (without repetition) of the elements in $\Lambda = \{a : a^0 \in M\}$ represents a sequence of actions solution to the input planning problem. Notice that the singleton sequence composed by the only set of actions Λ represents a partial-order plan of the planning problem. Otherwise, if the formula (4.26) is unsatisfiable, then k is incremented to 2 and so on. The generic Planning via SAT procedure can be defined as in Figure 4.4. The fundamental feature of this algorithm is that it is truly neutral in the encoding technique applied to reduce the planning problem to a SAT problem. This means that new encoding techniques can be easily integrated by providing appropriate instances for those encoding-dependent methods required by the procedure. In the next two paragraphs we will see how linear and graphplan-based encodings are integrated in the Planning via SAT procedure. Afterthat interesting termination concerns about the Planning via SAT procedure are discussed.

Planning via SAT using Linear Encodings. When linear encodings are applied in the Planning via SAT procedure of Figure 4.4, the data structure $\mathsf{d}_\mathsf{k}^{linear}$ is simply given by the propositional formula generated by the

Procedure: Planning via SAT
input : A planning problem Π and an encoding technique enc
output : A partial-order plan of Π
begin
  $\mathsf{d}_\mathsf{k}^\mathsf{enc} \leftarrow \mathtt{Init}^\mathsf{enc}(\Pi)$
  $\mathsf{model} \leftarrow \mathtt{Solve}(\mathtt{GetFormula}^\mathsf{enc}(\mathsf{d}_\mathsf{k}^\mathsf{enc}))$
  while $\mathsf{model} = \mathit{unsat}$ **do**
    $\langle \mathsf{k}, \mathsf{d}_\mathsf{k}^\mathsf{enc} \rangle \leftarrow \mathtt{Extend}^\mathsf{enc}(\Pi, \mathsf{k}, \mathsf{d}_\mathsf{k}^\mathsf{enc})$
    $\mathsf{model} \leftarrow \mathtt{Solve}(\mathtt{GetFormula}^\mathsf{enc}(\mathsf{d}_\mathsf{k}^\mathsf{enc}))$
  end
  $\mathsf{pop} \leftarrow \mathtt{Model2pop}(\mathsf{k}, \mathsf{model})$
  return pop
end

Legend:

- $\mathsf{d}_\mathsf{k}^\mathsf{enc}$ is the data structure used by the selected encoding
- $\mathtt{Init}^\mathsf{enc}(\Pi)$ initializes the data structure $\mathsf{d}_\mathsf{k}^\mathsf{enc}$
- $\mathtt{Extend}^\mathsf{enc}(\Pi, \mathsf{k}, \mathsf{d}_\mathsf{k}^\mathsf{enc})$ increases k and extends the data structure
- $\mathtt{GetFormula}^\mathsf{enc}(\mathsf{d}_\mathsf{k}^\mathsf{enc})$ returns the formula stored in $\mathsf{d}_\mathsf{k}^\mathsf{enc}$
- $\mathtt{Solve}(\mathsf{d}_\mathsf{k}^\mathsf{enc})$ returns a model of $[\![\Pi]\!]_\mathsf{k}$ if $[\![\Pi]\!]_\mathsf{k}$ is a satisfiable propositional formula and returns *unsat* otherwise
- $\mathtt{Model2pop}(\mathsf{k}, \mathsf{model})$ returns the partial-order plan associated with the propositional model model

Figure 4.4: Planning via SAT

encoding (i.e. $\mathsf{d}_\mathsf{k}^{linear}$ is $\langle [\![\Pi]\!]_\mathsf{k} \rangle$). The method $\texttt{GetFormula}^{linear}$ simply takes in input the data structure $\mathsf{d}_\mathsf{k}^{linear}$ and returns the propositional formula $[\![\Pi]\!]_\mathsf{k}$. The methods $\texttt{Init}^{linear}$ and $\texttt{Extend}^{linear}$ are showed in Figure 4.5 and Figure 4.6 respectively. Both the methods use $\texttt{Encode}^{linear}$ to build the corresponding propositional formula by means of the linear encoding presented in Section 4.2.1. Namely, $\texttt{Encode}^{linear}$(FALSE,Π,0) constructs the formula $[\![\Pi]\!]_0 = I(\mathbf{f}^0) \wedge B(\mathbf{f}^0)$ and $\texttt{Encode}^{linear}([\![\Pi]\!]_\mathsf{k},\Pi,\mathsf{k}+1)$ generates the formula $[\![\Pi]\!]_{\mathsf{k}+1}$ by extending the formula $[\![\Pi]\!]_\mathsf{k}$ already computed at the previous step.

Method : $\texttt{Init}^{linear}(\Pi)$
begin
  $[\![\Pi]\!]_0 \leftarrow \texttt{Encode}^{linear}$(FALSE,$\Pi$,0)
  return $\langle [\![\Pi]\!]_0 \rangle$
end

Figure 4.5: Planning via SAT: Init^{linear}

Method : $\texttt{Extend}^{linear}(\Pi,\ \mathsf{k},\ \mathsf{d}_\mathsf{k}^{linear})$
begin
  $[\![\Pi]\!]_{\mathsf{k}+1} \leftarrow \texttt{Encode}^{linear}([\![\Pi]\!]_\mathsf{k},\Pi,\mathsf{k}+1)$
  return $\langle \mathsf{k}+1, \langle [\![\Pi]\!]_{\mathsf{k}+1} \rangle \rangle$
end

Figure 4.6: Planning via SAT: Extend^{linear}

Planning via SAT using Graphplan-Based Encodings. If graphplan-based encodings are applied in the Planning via SAT procedure of Figure 4.4, then the data structure $\mathsf{d}_\mathsf{k}^{gp}$ is given by a pair: the planning graph and the propositional formula generated by the encoding (i.e. $\mathsf{d}_\mathsf{k}^{gp}$ is $\langle (\![\Theta]\!)_\mathsf{k}, [\![\Pi]\!]_\mathsf{k} \rangle$). The method $\texttt{GetFormula}^{gp-\mathsf{schema}}$ takes in input the data structure $\mathsf{d}_\mathsf{k}^{gp}$ and returns the propositional formula $[\![\Pi]\!]_\mathsf{k}$. The methods $\texttt{Init}^{gp-\mathsf{schema}}$ and $\texttt{Extend}^{gp-\mathsf{schema}}$ are showed in Figure 4.7 and Figure 4.8 respectively. Let us briefly discuss each of them.

The method $\texttt{Init}^{gp-\mathsf{schema}}$ starts constructing $(\![\Theta]\!)_0$, i.e. the first layer of the planning graph. Afterthat it checks if the goals $\mathcal{G}$ are unreachable in $(\![\Theta]\!)_0$, i.e. to test whether $\mathcal{G} \cap MBR((\![\Theta]\!)_0, 0)$ gives the empty set or not. If this is the case, then the propositional formula is set to the falsehood value FALSE. Otherwise the formula $[\![\Pi]\!]_0 = I(\mathbf{f}^0) \wedge B(\mathbf{f}^0)$ is generated.

The method $\texttt{Extend}^{gp-\mathsf{schema}}$ starts extending the planning graph by computing the $(\mathsf{k}+1)$-th layer. Similar to $\texttt{Init}^{gp-\mathsf{schema}}$, if there exists at least one goal state that may be reachable in the computed over-approximation, then $\texttt{Extend}^{gp-\mathsf{schema}}$ generates the propositional formula $[\![\Pi]\!]_{\mathsf{k}+1}$ by extending the formula constructed at the previous step and by using the appropriate graphplan-based encoding schema (see Section 4.2.2). Otherwise FALSE is returned as current value of the formula.

```
Method : Init^{gp−schema}(Π)
begin
    let ⟨Θ, G⟩ = Π in
        (|Θ|)_0 ← InitGraph(Θ)
        if GoalsUnreachable((|Θ|)_0, G) then
            return ⟨(|Θ|)_0, FALSE⟩
        else
            [[Π]]_0 ← Encode^{gp−schema}(FALSE, (|Θ|)_0, G, 0)
            return ⟨(|Θ|)_0, [[Π]]_0⟩
    end
end
```

Figure 4.7: Planning via SAT: $\text{Init}^{gp-\mathsf{schema}}$

```
Method : Extend^{gp−schema}(Π, k, d_k^{gp})
begin
    let ⟨(|Θ|)_k, [[Π]]_k⟩ = d_k^{gp} and ⟨Θ, G⟩ = Π in
        (|Θ|)_{k+1} ← ExtendGraph((|Θ|)_k, Θ, k + 1)
        if GoalsUnreachable((|Θ|)_{k+1}, G) then
            return ⟨k + 1, ⟨(|Θ|)_{k+1}, FALSE⟩⟩
        else
            [[Π]]_{k+1} ← Encode^{gp−schema}([[Π]]_k, (|Θ|)_{k+1}, G, k + 1)
            return ⟨k + 1, ⟨(|Θ|)_{k+1}, [[Π]]_{k+1}⟩⟩
        end
    end
end
```

Figure 4.8: Planning via SAT: $\text{Extend}^{gp-\mathsf{schema}}$

Termination check. It is immediate to see that, in the general case, the Planning via SAT phase presented in Figure 4.4 corresponds to a semi-decision procedure for planning, i.e. the procedure halts with a plan when

it exists, but otherwise it may not terminate. However, when NPC protocol insecurity problems are considered, the corresponding planning problems have a finite number of fluents and actions and therefore the longest planning path would require the application of $2^{\mathcal{F}}$ actions in the worst case. This upper-bound (say k_{max}) can often be reduced considerably by exploiting the knowledge on the application domain. Let m, p, and j be the number of protocol sessions, the number of steps executed per session, and the maximum number of actions the intruder needs for decomposing the deepest message in the protocol respectively. Depending on whether the step compression optimisation technique is applied or not (cf. Section 3.2.4), $k_{max} = m * p * j$ or $k_{max} = 3 * m * p * j$. For instance, considering the NSPK protocol discussed in details in Chapter 3, $k_{max} = 2 * 3 * 2 = 12$ with step compression enabled.

It is worth pointing out that the above upper-bound refers to *linear plans*. Since Planning via SAT allows for *partial-order plans*, better upper-bounds can be computed. As proved in Section 4.2, any model of $[\![\Theta]\!]_k = I(\mathbf{f}^0) \wedge \bigwedge_{i=0}^{k-1} T_i(\mathbf{f}^i, \mathbf{a}^i, \mathbf{f}^{i+1})$ corresponds to a partially ordered set of actions of depth k that represents a set of planning paths of *length greater or equal to k*. This means that the formula $[\![\Theta]\!]_k$ may represent more than k levels of the search space.

It turns out that graphplan-based encodings are well suited to provide a more precise k_{max}. The basic idea consists in checking if the *leveled-off* has been reached during the construction of the planning graph [BF95]. We recall that the planning graph built at step i represents an over-approximation, in terms of reachable states and applicable actions, of the forward search tree up to depth i. To reach the leveled-off means to have computed an over-approximation of the whole forward search tree associated with the input planning problem Π. Let n be the step at which the leveled-off as been reached. It must be noted that in general the unsatisfiability of the SAT formula $[\![\Pi]\!]_n$ is not sufficient to conclude that no plan exists for the planning problem Π. In fact, a plan can require multiple executions of the same action at different steps as well as the sequential execution of actions in mutual exclusion. As a consequence, it could be that n steps are not enough to perform the attack. However, for any planning problem Π obtained from a NPC protocol insecurity problem, the unsatisfiability of $[\![\Pi]\!]_n$ guarantees that no plan exists for the planning problem Π. This means that n is the upper-bound for k, i.e. $k_{max} = n$.

To summarise, when NCP protocol insecurity problems are considered, it is possible to compute an upper-bound k_{max} over k such that the Planning via SAT procedure can be stopped after k_{max} iterations. Such an upper-bound mainly depends on the planning problem Π and on the kind of encoding technique selected.

In order to enhance the Planning via SAT procedure to enforce termination, it suffices *(i)* to define the function $\texttt{UpperBoundReached}^{\mathsf{enc}}(\Pi,\mathsf{k},\mathsf{d}_{\mathsf{k}}^{\mathsf{enc}})$

that returns true if and only if the upper-bound has been reached, *(ii)* to add to the while test the condition that checks whether the upper-bound has been reached, and *(iii)* to introduce an if-then-else test that, in case the procedure exits from the while loop, returns either the partial-order plan found or *no_pop_exists* to denote that no partial-order plan exists for the input planning problem. The whole procedure is depicted in Figure 4.9.

Procedure: Planning via SAT with termination
input : A planning problem Π and an encoding technique enc
output : A partial-order plan of Π or *no_pop_exists*
begin
  $\mathsf{d}_\mathsf{k}^{\mathsf{enc}} \leftarrow \mathtt{Init}^{\mathsf{enc}}(\Pi)$
  model $\leftarrow$ Solve(GetFormula$^{\mathsf{enc}}(\mathsf{d}_\mathsf{k}^{\mathsf{enc}})$)
  while model $=$ *unsat* **and not** UpperBoundReached$^{\mathsf{enc}}(\Pi, \mathsf{k}, \mathsf{d}_\mathsf{k}^{\mathsf{enc}})$ **do**
    $\langle \mathsf{k}, \mathsf{d}_\mathsf{k}^{\mathsf{enc}} \rangle \leftarrow \mathtt{Extend}^{\mathsf{enc}}(\Pi, \mathsf{k}, \mathsf{d}_\mathsf{k}^{\mathsf{enc}})$
    model $\leftarrow$ Solve(GetFormula$^{\mathsf{enc}}(\mathsf{d}_\mathsf{k}^{\mathsf{enc}})$)
  end
  if model $=$ *unsat* **then**
    return *no_pop_exists*
  else
    pop $\leftarrow$ Model2pop(k, model)
    return pop
  end
end

Legend:

- UpperBoundReached$^{\mathsf{enc}}(\Pi, \mathsf{k}, \mathsf{d}_\mathsf{k}^{\mathsf{enc}})$ is true if and only if the upper-bound has been reached

Figure 4.9: Planning via SAT with termination

4.3.2 Planning via SAT with Abstraction/Refinement

Abstraction [CGKS02] is a powerful technique for the analysis of large-state systems. The idea is to abstract the system under consideration into a simpler one that has all the behaviours of the original one, but may also exhibit some spurious behaviours. Thus—by construction—if a safety property holds for the abstract system, it will also hold for the concrete one. On the other hand, a counterexample for the abstract system does not always correspond to a counterexample for the concrete one. If the behaviour that violates the safety property in the abstract system cannot be reproduced in the concrete system, the counterexample is said to be *spurious*. When such

a counterexample is found, the abstraction must be refined in order to eliminate the spurious behaviour. The procedure is then iterated until either a non spurious counterexample is found, or the abstract system satisfies the safety property.

In Section 4.2 we described two families of SAT-reductions for AI planning, namely the linear and the graphplan-based encoding techniques. Since in all these SAT-reductions, except that using the bitwise representation, the number of clauses encoding the mutex relation (referred as "conflict exclusion axioms" and as "mutex axioms" in the linear and graphplan-based encodings respectively) is the main source of difficulty, it is tempting to avoid their generation altogether. It turns out that the resulting problem is an abstraction of the original problem. Basically the neglection of the mutex relation relaxes the transition relation to allow the parallel execution of any set of actions, even those including interfering actions. This means that spurious solutions where interfering actions are executed simultaneously can be returned. As a consequence any solution found has to be validated, i.e. check that each step of the counterexample does not involve interfering actions. When the validation procedure fails, the formula must be refined by adding to it the mutex constraints thereby avoiding the interfering actions discovered so far in spurious counterexamples. The whole procedure is repeated until either a counterexample without interfering actions is found, or the abstract system satisfies the safety property.

The entire Planning via SAT phase with the abstraction/refinement strategy described above is depicted in Figure 4.10. The module `PP2SAT(enc, k)` provide the abstract system by generating a propositional formula in which the mutex constraints are neglected. If the formula is found satisfiable, then the returned model M is translated into a pseudo partial-order plan $\eta = \Lambda_1, \ldots, \Lambda_k$ where $\Lambda_i = \{a : a^i \in M\}$ is the set of actions true at the time step i for each $i = 1, \ldots, k$. The validation procedure checks that each set of actions in the sequence of the pseudo partial-order plan η does not involve interfering actions. If the validation procedure fails, then the module `Refinement` refines the current propositional formula by adding one mutex constraints $\neg(a_1^i \wedge a_2^i)$ for each pairs of interfering actions $\langle a_1, a_2 \rangle$ that is found within the i-th set of actions of η, thereby avoiding the interfering actions discovered so far in spurious counterexamples. The whole procedure is repeated until either a pseudo partial-order plan without interfering actions is found, or the propositional formula is proved unsatisfiable. In the first case the pseudo partial-order plan corresponds to a partial-order plan that is returned by the procedure. In the second case k is increased and the entire procedure is repeated.

For what concerns termination, the same ideas presented in the previous section apply also on Planning via SAT with abstraction/refinement.

Finally, experimental evidence on the effectiveness of this abstraction/-refinement strategy is provided in Section 7.1.1.

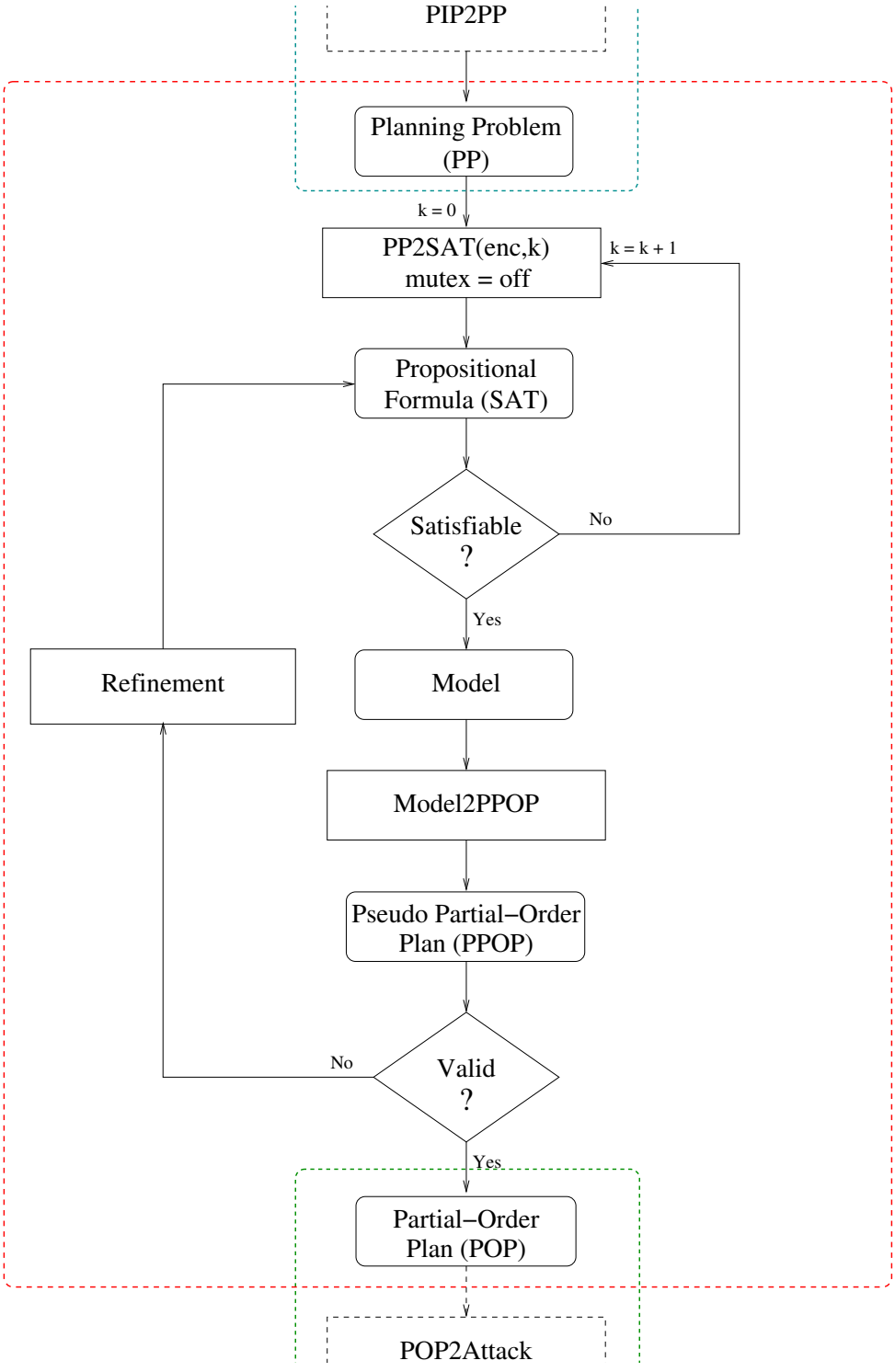

Figure 4.10: Planning via SAT with the abstraction/refinement strategy

Chapter 5

Specification Languages for Security Protocols

The specification of languages that support protocol designers in the activities of either finding flaws in a protocol or establishing their absence is indeed a critical and important point to be addressed in order to speed up the development of a next generation of secure network protocols and ultimately the acceptance of products based on them.

The design of protocol specification languages must consider a lot of conflicting requirements. On the one hand, protocol engineers of standardisation bodies desire a convenient, human readable, and easy to use language. On the other hand, computer machines need a low-level description of the protocol in order to execute on it an automated formal analysis. To cope with these two contrasting requirements, we provide a high level language accessible to protocol designers and easily translatable into a lower-level term-rewriting based language well-suited to model-checking tools. Such a framework, depicted in Figure 5.1, is one of the result of a joint work with others scientific institutions carried out in the context of the AVISS[1] [AVI01] and AVISPA[2] [AVI02a] projects. Security protocols are specified in the High-Level Protocol Specification Language (HLPSL) that is designed to be accessible to human users (protocol specifications should be easily read and written by protocol designers). To this end, HLPSL provides a high level of abstraction and has many features that are common to most protocol specifications – such as a variety of intruder models and of encryption primitives – built in. In contrast, the Intermediate Format (IF) – the language into which HLPSL specifications are automatically translated (see Figure 5.1) – is

[1]The AVISS project has involved the AI-Lab group at DIST of University of Genova (Genova, Italy), the CASSIS group at INRIA (Nancy, France), and the Institut fur Informatik Albert-Ludwigs-Universitat (Freiburg, Germany).

[2]The AVISPA project involves the AI-Lab group at DIST of University of Genova (Genova, Italy); the CASSIS group at INRIA (Nancy, France), the Information Security Group at ETHZ (Zurich, Switzerland), and Siemens AG (Munich, Germany).

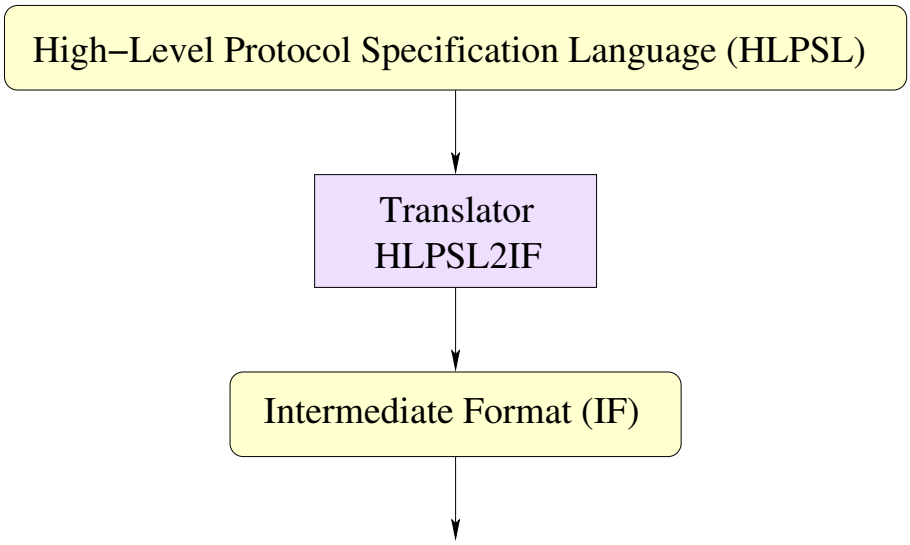

Figure 5.1: HLPSL to IF framework

a lower-level language at an accordingly lower abstraction level particularly amenable to automated formal analysis.

The above framework has been exploited in both AVISS and AVISPA leading to two different versions of both HLPSL and IF. Even if the first version of the languages is quite different with respect to the second one [CCC+04, AVI03a], for sake of brevity, in the rest of this chapter we will mainly focus on the last version of HLPSL and IF just mentioning the differences with respect to the first one.[3]

5.1 The High Level Protocol Specification Language (HLPSL)

HLPSL is the language through which end users and protocol modellers specify security protocols and the requirements they are expected to meet. As a matter of fact, protocol designers are not necessarily experts in the area of formal methods and therefore HLPSL should provide the appropriate level of abstraction to result easy to use. Besides this, HLPSL should be

[3]It must be noted that the AVISPA project has not been concluded since the writing of this Thesis. As a consequence, the official HLPSL and IF languages could have been changed in the meantime. We invite the interested reader to refer to [AVI02a] for consulting the up-to-date release of the languages. Besides this, it is worth pointing out that, even if HLPSL and IF allows for the declaration of algebraic equations to specify "cryptographic algebraic operators", this feature is not currently supported by the SAT-based approach and, therefore, it will not be discussed in the context of this Thesis.

expressive enough to be able to specify a wide range of security protocols.

The first version, HLPSL v.1, of this language supports the declaration of protocols using standard "Alice&Bob" style notation for indicating how messages are exchanged between roles [CJ97]. HLPSL v.1 has been thoroughly and successfully experimented in the activity of specifying the security protocols of the well-known Clark-Jacob's library (CJ, [CJ97]). It turns out that 46 of 51 (a coverage of 90%) of the protocols presented in the CJ are specifiable in such a specification language. However, constructs for modularity and flow control are missing in HLPSL v.1, making it difficult or even impossible to specify looping protocols (e.g. group protocols such as that proposed by Tanaka and Sato [TS01]) and other complex behaviour (e.g. an agent that should contact a key server when the agent itself does not yet possess the public key of his communication partner). Moreover, only little variants of the Dolev-Yao intruder can be specified, while a modern network infrastructures often employ heterogeneous technologies vulnerable to differing intruder threats. In addition to this last point, it is worth pointing out that, even worse than HLPSL v.1, others state-of-the-art protocol specification languages like CASPER [DNL99], CAPSL [DM00], and MuCAPSL [MD03] (an extension of CAPSL based on strand spaces and particularly suited for group protocols) leave the intruder model implicit.

On top of the experience accumulated in the development of HLPSL v.1, we have devised HLPSL v.2[4] with the main design objective of being able to specify and analyse modern Internet security protocols. In doing this, a lot of requirements, some of which have been already mentioned above, must to be considered.

Firstly, industrial-scale protocol suites are composed of smaller sub-protocols whose interaction should allow for the achievement of a variety of complex security goals. Depending on the current state of the protocol, one sub-protocol can be executed either in place of another or loop until a particular condition is satisfied. For instance, if two agents do not share a common key, a key establishment sub-protocol may be required before continuing the main protocol.

In addition, security protocols are used on heterogeneous communication channels characterised by different properties and thus requiring different intruder models. Consider, for instance, common cabled network communications compared with wireless communications: while the well-known Dolev-Yao (DY) intruder model [DY83] is particularly suited for the former, it is not suitable for the latter since the intruder cannot, in "location-limited" communication ([CGRZ03]), prevent messages from reaching their destinations.

Besides this, a high level language must still be accessible to engineers and protocol designers of standardisation bodies and it must be amenable

[4]In the sequel we will refer to HLPSL v.2 simply as HLPSL.

to automated formal analysis.

To accomplish the above requirements we have developed HLPSL. It has a formal semantics based on Lamport's Temporal Logic of Actions (TLA, [Lam94]) that makes it easily translatable into a declarative lower-level term rewriting based language (the IF v.2 described in Section 5.2 and in [AVI03b]) well-suited to automated analysis tools. HLPSL thus enjoys significant generality, as other tools can easily be made to employ HLPSL by simply adapting them to accept IF v.2 specifications as input. A pleasant side effect of this feature is the possibility of running different automated reasoning tools against the same input security problems by allowing, thus, a real comparison between the techniques underlying these tools.[5] HLPSL is modular and allows for the specification of complex control-flow patterns, data-structures, and different intruder models. Using a formal language with a temporal logic semantics to formalise security properties gives us great generality and expressiveness. Finally, HLPSL is not restricted to logicians, but it is particularly suited for engineers and protocols designers.

In the rest of this section, we start by introducing some TLA preliminaries. Section 5.1.2 is devoted to the description of the HLPSL language via a running example. We conclude in Section 5.1.3 with the formal description of the HLPSL semantics.

5.1.1 Preliminaries

The semantics of our High-Level Protocol Specification Language is based on Lamport's Temporal Logic of Actions (TLA, [Lam94]). TLA is an elegant and powerful language which lends itself well to specifying concurrent systems like Internet security protocols we want to model. In TLA, system behaviour is modelled by describing the *state* and then specifying the ways in which that state may change. The global state is defined by an assignment of values to all the system variables. Similarly, the state of a TLA module (or *role*, as we call them in HLPSL) is defined by an assignment of values to all the variables of the role, that is, those variables that are visible from within the role. The description of the transition relation is then given by predicates that relate the values of variables in one state (the current one) and another (the future, or next state). We refer to the variables in the next state as *primed* variables (e.g. let X be a variable, X refers to its value in the current state and $X^{'}$ to its value in the next). A *state predicate* is a first-order formula on a role's state variables and constants. Examples of valid state predicates thus include $X = 5$ and $State = done$. A *transition predicate* is similar but may include primed variables. For instance $X = 5 \wedge Y^{'} = 7$, while not being a state predicate, is a legal transition predicate.

[5] In this context we invite the interested reader to consult the results obtained in the AVISS [ABB+02] and AVISPA projects [ABB+05].

If V is a tuple of state variables and V' the correspondent tuple of primed state variables, then we define the set of *actions* as those transition predicates $p(V, V')$ with the property that $\forall V : \exists V' : p(V, V')$.[6] Actions may therefore include stuttering steps. A *basic event* is a conjunction of transition predicates, at least one of which is of the form $p(V') \neq p(V)$, where V is a tuple of variables and $p(V)$ is a state predicate. This definition ensures that events are *non-stuttering*, i.e. at least one state variable changes.[7]

5.1.2 The Language

Let us introduce HLPSL with the help of a running example, the complete Needham-Schroeder Public Key protocol including key-server (NSPK-KS, [NS78]) depicted in Figure 5.2.[8] This protocol involves an initiator A,

$$
\begin{array}{llcll}
(1) & A & \rightarrow & S & : \quad A, B \\
(2) & S & \rightarrow & A & : \quad \{B, Kb\}_{inv(Ks)} \\
(3) & A & \rightarrow & B & : \quad \{Na, A\}_{Kb} \\
(4) & B & \rightarrow & S & : \quad B, A \\
(5) & S & \rightarrow & B & : \quad \{A, Ka\}_{inv(Ks)} \\
(6) & B & \rightarrow & A & : \quad \{Na, Nb\}_{Ka} \\
(7) & A & \rightarrow & B & : \quad \{Nb\}_{Kb}
\end{array}
$$

Figure 5.2: The NSPK-KS protocol

a responder B, and a server S. It behaves similarly to the version without key-server described in Section 3.1.2, but NSPK-KS exploits a server for distributing keys between partners. While this is a relatively simple protocol, this example still allows us to illustrate some of the more advanced features of HLPSL. For instance, we add the requirement that each agent maintains a key-ring of public keys, i.e. a set of the agents and their respective public keys that the agent has already learned from the trusted key-server S. Moreover, an agent X should contact S *only* if X does not yet possess the public key of his communication partner in his key-ring. Such "if-then-else" style flow control is often missing from existing specification languages like for instance CASPER [DNL99], and the HLPSL v.1; indeed, such control patterns cannot even be adequately described in the simple Alice&Bob notation that is standard in the literature. Also, the modelling of such a key-ring requires the specification of a set of messages shared between all protocol sessions

[6] Notice that, the satisfaction of this property causes the transition relation to be *total*.

[7] Notice that, $p(V') \neq p(V) \Rightarrow V' \neq V$ can be simply derived from $V' = V \Rightarrow p(V') = p(V)$ by means of contrapposition.

[8] We use $inv(K)$ to indicate the inverse of public key K.

in which a given agent participates. This too is a non-trivial challenge that many existing specification languages cannot meet.

Protocol specifications in HLPSL are divided into roles that are parametrised with respect to a set of typed variables. Some roles (the so-called *basic* roles) serve to describe the actions of one single agent in a run of a protocol or sub-protocol. Others (*composed* roles) instantiate these basic roles to model a scenario the protocol designer intends to analyse with respect to some security goals. For instance, an entire protocol run (potentially consisting of the execution of multiple sub-protocols), and a session of the protocol between multiple agents are possible scenarios of interest. Given a set of roles describing a concrete scenario, the protocol designer then defines the security goals by means of safety temporal properties.

The HLPSL specification corresponding to the NSPK-KS is depicted in Figures 5.3, 5.5, and 5.6. Comments in HLPSL begin with the '`%`' symbol and continue to the end of the line.[9] Namely, Figure 5.3 describes the behaviour of the basic roles `Alice`[10] and `Server` that are, respectively, the counterpart of the roles A and S of Figure 5.2. Figures 5.5 and 5.6 describe a possible composition of basic roles instances and a security goal to be checked, respectively.

Let us present in details the main characteristics of HLPSL always by referring to the NSPK-KS example.

Types

In HLPSL, both variables and constants are typed. In terms of notation, variable names begin with a capital letter and constant names with a lowercase letter; moreover, we refer to the intruder by the constant `i`. HLPSL defines a set of basic types particularly suited to denote and capture the domain of security protocols. For instance, `agent`, and `public_key` denote the basic types representing agent names, and agents' public keys for asymmetric cryptography,[11] respectively. Types are specified using the '`:`' symbol. For instance, `A: agent` declares the variable `A` to be of type `agent`. Certain types (e.g. `public_key`, `text`, etc) may have attributes, enclosed within parentheses. These attributes specify additional properties that the elements of the type may take on. For instance, nonces are of type `text` but must have the additional property of uniqueness, which we call *freshness*.[12] Therefore, variables supposed to represent nonces, like for instance `Na` in

[9] As a notational convention, text intended to be read as coming directly from a HLPSL specification will appear in `typewriter` font.

[10] For sake of brevity we do not show the specification of the `Bob` role that is defined analogously to the `Alice` role.

[11] For a given public (respectively private) key `K`, its inverse private (respectively public) key is obtained by `inv(K)`.

[12] Nonces and in general any *fresh* term are intended to be both randomly generated by a principal and used *only once*.

```
role Alice(A, B, S  : agent,
           Ka, Ks   : public_key;
           KR       : (agent.public_key) set,
           SND, RCV : channel (dy)
          ) played_by A def=
 local State : nat,
       Na    : text (fresh),
       Nb    : text,
       Kb    : public_key
 const na, nb: protocol_id
 owns Ka
 knowledge(A) = {A, B, Ka, inv(Ka), Ks}
 init State = 0
 transition
  % Start of the protocol, if Alice already knows Kb.
  astep1a.  State =0 /\ START() /\ in((B.Kb'), KR)
        =|> State'=2 /\ SND({Na'.A}_Kb') /\ witness(A,B,na,Na')
  % Start of the protocol otherwise.
  astep1b1. State =0 /\ START() /\ not(in((B._), KR))
        =|> State'=1 /\ SND(A.B)
  % Receipt of response from server
  astep1b2. State =1 /\ RCV({B.Kb'}_inv(Ks))
        =|> State'=2 /\ KR' = cons((B.Kb'), KR)
                     /\ SND({Na'.A}_Kb') /\ witness(A,B,na,Na')
  % Receiving the second message and sending the third.
  astep2.   State =2 /\ RCV({Na.Nb'}_Ka)
        =|> State'=3 /\ SND({Nb'}_Kb) /\ request(A,B,nb,Nb')
end role

role Server(S        : agent,
            Ks       : public_key,
            KR       : (agent.public_key) set;
            SND, RCV : channel (dy)
           ) played_by S def=
 local A, B : agent,
       Kb   : public_key
 owns Ks
 knowledge(S) = {S, Ks, inv(Ks)}
 transition
  sstep0.  RCV(A'.B') /\ in((B'.Kb'),KR)
       =|> SND({B'.Kb'}_inv(Ks))
end role
```

Figure 5.3: NSPK-KS protocol: HLPSL spec. (basic roles)

role `Alice` of Figure 5.3, are of type `text (fresh)`. In the sequel, we will refer to variables associated to the attribute `fresh` as *fresh variables*.

The space of legal messages, represented by means of the most general HLPSL type `message`, is defined as the closure of the basic types under a set of standard operations such as concatenation (via the associative "`.`" operator) and encryption (which we write `{Msg}_Key` for a given message `Msg` and encryption key `Key`).[13] Thus, if we have an agent name `A` of type `agent`, a nonce `Na` of type `text (fresh)`, and a public key `Kb`, all of the following are valid messages:

```
A.Na          % A's name concatenated with the nonce
{A.Na}_Kb     % As above, but encrypted with Kb
```

HLPSL allows for the declaration of a variety of communication channels (see paragraph below on "Communication and Intruder Models") as well as aggregate types such as tuples, lists, and sets. For instance, `KR: (agent.public_key) set` allows us to declare `KR` as a key-ring of public keys, i.e. a set of pairs associating an agent with its public key. HLPSL also allows us to declare new function symbols. Such functions assume the properties of perfect cryptographic hash functions: they are injective and not invertible by the intruder. Moreover, function symbols are themselves messages like any other. They can therefore be either known or unknown to the intruder (modelling a publicly known or secret function, respectively) and can be transmitted within messages (modelling the negotiation of cryptographic algorithms). They can also be employed to model key-tables. For instance, if we assume that the key-server involved in NSPK-KS protocol knows the public keys of all the agents in the world (instead of a key-ring of some public keys), then we can re-define the role `Server` of Figure 5.3 as follows:

```
role Server(S        : agent,
            Keys     : function;
            SND, RCV : channel (dy)
           ) played_by S def=
  local A, B: agent
  transition
      sstep0. RCV(A'.B') =|> SND({B'.Keys(B')}_inv(Keys(S)))
end role
```

where we use a function `Keys` in place of the key-ring `KR` to model the association of agents to their public keys. Notice that, by introducing such a function we refer to single public keys like `Ks` and `Kb` simply by means of `Keys(S)` and `Keys(B)`, respectively.

[13] Notationally, we do not distinguish between symmetric and asymmetric encryption.

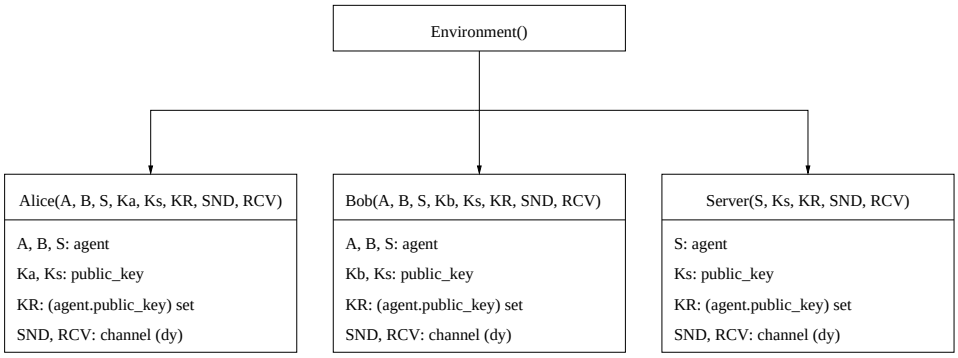

Figure 5.4: NSPK-KS protocol: hierarchy of roles

Typing is used to exclude implementation-dependent type-flaw attacks. Besides the specification of the above types HLPSL supports for the declaration of *compound types* which allow the protocol designer to declare variables of a type precisely restricted and specialised with respect to the items the variables represent. The example of the NSPK-KS protocol is not indicated to describe the idea of compound types and therefore we have dedicated an entire paragraph at the end of this section for discussing compound types in details.

Roles

We model the protocol in a modular fashion, focusing not on the exchange of messages that takes place, but rather specifying a hierarchy of roles. We distinguish between two different types of roles. At the lowest level of such a hierarchy there are basic roles that describe the actions of one agent involved in a single protocol or sub-protocol execution. On top of lowest level roles there are composed roles that instantiate and conjoin one or more other roles to specify a concrete scenario. For instance, the HLPSL specification presented for the NSPK-KS protocol can be associated to the hierarchy of Figure 5.4 where arrows from the role `Environment` to the basic roles `Alice`, `Bob`, and `Server` simply describe dependency relations stating that `Environment` is built on top of at least one instance of both `Alice`, `Bob`, and `Server`. Roles may be parametrised and may also declare local variables and constants. The specification of the basic roles `Alice` and `Server` is shown in Figure 5.3, while the composed role `Environment` is presented in Figure 5.5. Roles are defined over a set of parameters—which are variables that can be shared between roles (see paragraph below on "Instantiation" where is also discussed variables sharing)—and a set of local variables, not accessible from outside the role. With reference to `Alice`, while `A` belongs to her parameters and therefore its value can be accessed by other roles, `State`

```
role Environment() def=
 local Kr_A, Kr_B, Kr_S        : (agent.public_key) set,
       Snd_s,  Rcv_s,  Snd_ab,
       Rcv_ab, Snd_ba, Rcv_ba,
       Snd_ai, Rcv_ai          : channel (dy)

 const a, b, s, i            : agent,
       ka, kb, ks, ki        : public_key

 knowledge(i) = {i,a,b,s,ka,kb,ks,ki,inv(ki)}

 init  Kr_A = [a.ka] /\ Kr_B = [b.kb]
                     /\ Kr_S = [a.ka, b.kb, i.ki]
 composition
  Server(s,ks,Kr_S;Snd_s,Rcv_s) /\
  % Session 1, between agents a and b
  Alice(a,b,s,ka,ks;Kr_A,Snd_ab,Rcv_ab) /\
  Bob(a,b,s,kb,ks;Kr_B,Snd_ba,Rcv_ba) /\
  % Session 2: agent a talking to the intruder, posing as Bob
  Alice(a,i,s,ka,ks;Kr_A,Snd_ai,Rcv_ai)
end role
```

Figure 5.5: NSPK-KS protocol: HLPSL spec. (topmost-level role)

is a local variable whose scope[14] is thus restricted to `Alice` only.

When a basic role is declared also an identifier specifying the name of the agent playing the role, i.e. its *player*, must be defined. We use the `played_by` declaration to specify the player and we always assume that the name of the player is passed in via an argument and is not a local variable. For instance, `A` and `S` are the agents playing the roles `Alice` and `Server`, respectively (see Figure 5.3). It must be noted that the notion of player is useful to determine who is the sender/receiver of the messages sent/received over the channels used by the role and, therefore, to establish what agent knowledge is affected by such a message exchange. For instance, when transition `astep1b2` (see Figure 5.3) fires, then `A` receives the message `{B.Kb'}_inv(Ks)` and `A` extends her knowledge with such a message and everything can be derived from it (see Section 5.1.3).

Roles may declare *ownership* of parameters, asserting that the owned variables may change in *only* the way described by the owning role despite the fact that they are visible from outside. For instance, the public keys `Ka` and `Ks`, that are owned, respectively, by `Alice` and `Server`, are visible

[14]The scope of a variable is the context within which it is defined.

to everybody else, but their value can be changed only by the respective owners.

A role may also include one or more *knowledge* declarations describing the initial knowledge of agents involved in the execution of the role. For instance, the declaration `Knowledge(A) = {A, B, Ka, inv(Ka), Ks}` states that the agent `A` initially knows its identify, its public and private keys as well as the identity of `B` and the public key of the `Server`. Multiple knowledge declarations can be useful to model situations in which multiple parties are seen to "play" the role collectively, or, by parametrising a knowledge declaration by the intruder's name `i`, to experiment with scenarios in which certain information is compromised. It is worth pointing out that we assume every honest agent has always enough knowledge to compose messages occurring in his sending channels when a transition is executed (see Section 5.1.3). Such an assumption does not apply to the intruder whose knowledge grows up monotonically by interacting with honest agents. As a consequence the protocol designer should be very careful in declaring the knowledge of basic roles, since otherwise this could preclude the intruder to play some of the roles.

A role may be seen as the analogue to a module in TLA, describing an initial predicate and a next-state relation. The set of allowed initial states can be restricted by means of a `init` state predicate. For example, all the instances of the role `Alice` restrict their set of permissible initial states to only those in which the variable `State` is equal to 0 (the other variables may have arbitrary values initially). While the basic roles declare explicitly the next-state relation through transition rules (see paragraph below on "Transitions"), a composed role describes such a relation by means of parallel and sequential composition of the next-state relations of the roles on which it is built on (see paragraph below on "Instantiation" and Section 5.1.3).

Though not illustrated in the NSPK-KS example, a role may define *acceptance* state predicate to indicate the conditions for "successful" completion. This is needed for sequential composition and looping (see paragraph below on "Instantiation"): if we have two sequentially composed role instantiations, then the second role begins execution after the first one has reached one of its accepting states.

Communication and Intruder Models

Communication in HLPSL is synchronous (as in SCCS [Mil89]) and takes place over channels simply modelled as variables. Namely, each channel is implicitly paired with a flag which is toggled to indicate the sending/receipt of a new message: let *Ch* be a channel and *Msg* be a message, then the *channel macro* $Ch(Msg)$ stands for the conjunction

$$Ch' = Msg \wedge Ch\_flag' \neq Ch\_flag$$

Security protocols execute on heterogeneous communication channels characterised by different security properties and thus requiring different intruder models. To faithfully model such settings, each channel is associated with a particular intruder model, and our logic-based semantics allows us to describe alternative intruder models in a simple axiomatic manner. Since many protocols use identical intruder models, we find it convenient to separate their definitions into a so-called *prelude file* (see Section 5.2 for more details). Intruder capabilities are thus described in the prelude file, while, in HLPSL, the `channel` type takes an attribute which serves as a macro specifying which intruder model should be used. Each such macro corresponds to a set of intruder axioms in the prelude file, which includes, for instance, axioms describing channels controlled by the DY intruder; axioms describing channels modelling "location-limited" communication [CGRZ03] upon which the intruder cannot, in general, prevent messages from reaching their destinations; and axioms describing secure channels to which the intruder has no access whatsoever.

The specification of the NSPK-KS requires only DY channels. This is simply achieved by associating every channel with the macro attribute `(dy)`. When all the channels are controlled by the well-known DY intruder, then all communication is synchronous with the intruder, i.e. every message received by an honest agent comes from the intruder, and every message sent by an honest agent goes directly to the intruder and is added to his knowledge. This is not a restriction, as the DY intruder may intercept any message and replay it to any other agent. Thus, even though the model is synchronous, a message is not necessarily received by the intended recipient, but rather by the intruder. We may thus say that we identify the intruder with the network.

The user can also define new channel types. In HLPSL, the explicit specification of the intruder's capabilities allows us to model modern heterogeneous networks (by equipping roles with multiple channels of different types) and enables us to easily analyse protocols under alternative intruder models (by simply experimenting with different channel types). This is particularly important for formalising some of the protocols currently under discussion at the IETF [AVI03c] as well as protocols executing in settings like those of [CGRZ03].

Transitions

The next-state relation of basic roles is defined through a set of transitions rules within the `transition` section. HLPSL distinguishes between two different types of transitions, respectively called *spontaneous actions* (denoted by the `--|>` arrow) and *immediate reactions* (specified using the `=|>` arrow). Both kinds of transitions may be preceded by a label, which may be an alphanumeric string starting with a lower case letter and ending with a

period. It is worth pointing out that these labels carry no information about the order in which the transitions fire. They are merely names.

Spontaneous actions relate a state predicate on the left-hand-side (LHS) with an action on the right-hand-side (RHS). Intuitively, a spontaneous action transition *sp* `--|>` *act* expresses that, whenever we are in a state that fulfils state predicate *sp*, we *may* make a transition labelled by the action *act* into a new state. Note, however, that we do not require that this transition fire right away. When and if it does fire, we obtain the new state by taking the old state and applying any side effects that *act* might produce; that is, any state variables not affected by *act* remain the same. We call such transitions "spontaneous" because, although they are enabled whenever state predicate *sp* is true, they convey nothing about when they must fire.

Immediate reaction transitions, on the other hand, have the form *event* `=|>` *act* and relate a trigger event, *event*, on the LHS with an associated action, *act*, on the RHS. This expresses that, whenever we take a transition that satisfies the non-stutter event predicate *event*, then we *must* immediately (more precisely, simultaneously) execute action *act*. Hence as soon as *event* holds, we obtain the new state by taking the old state and applying any side effects that *act* might produce; that is, any state variables not affected by *act* remain the same.

At this stage, it is important to pointing out that every spontaneous action *sp* `--|>` *act* can be expressed as an immediate reaction *sp* `/\` *sgn* `=|>` *act* where *sgn* is a signal that has been introduced specifically for such an immediate reaction. A *signal* is a kind of channel that does not pass any value and that represents, thus, an independent event. In the sequel we will use the signal `START()` as an independent event for starting any specified protocol sessions.[15]

In Section 5.1.1 we have already mentioned that a primed variable X', i.e. the representation of the variable X in the next state, can occur in an action and that at least one non-stuttering predicate of the form $p(V') \neq V$ (where V is a tuple of variables) must occur in an event. Here we informally explain primed variables with reference to HLPSL transition rules[16] and freshness. Intuitively, a transition predicate occurring in the LHS of an immediate reaction[17] and containing the primed variable X', imposes a constraint on the application of the action in the RHS. As a consequence the action can be executed only if such a constraint on the next value of X is satisfied. Usually, a primed variable in the LHS occurs inside a receiving channel to assign the message (or part of it) on the channel to the value of this variable

[15] By HLPSL convention, all protocol sessions begin with the occurrence of the `START()` signal.

[16] Without any loss of generality, here and in the sequel, we restrict state and transition predicates to first-order formula using only conjunction and negation operators.

[17] By definition, primed variables cannot occur in the LHS of spontaneous actions.

in the next state.

When a primed variable X' occurs in the RHS of a transition rule, we simply refer to the value of X in the next state. However when X is a fresh variable (i.e. it is tagged as `fresh`) that occurs primed only in the RHS of a transition, then a new random value never used before as fresh value is assigned to X' when such a transition fires.

It is worth pointing out that if a protocol exchanges messages only on DY channels, then we can safely group the receipt of a message and the sending of the corresponding response into a single immediate reaction as proposed in [DMGF00]. The reason is that, in the DY model, we have synchronous communication between honest agents and the intruder (see paragraph above on "Communication and Intruder Models"). The intruder himself, however, need not act synchronously. He lies between two communicating agents and introduces a delay between receiving a message and sending the response, because he himself will not react immediately as the honest agents do in the NSPK-KS example. Together, the identification of intruder and network combined with this immediate reaction comprise a technique known as *step compression* that has already been discussed discussed in Chapter 3: as we do not need to consider all interleavings of intermediate transitions of the honest agents.

While the use of only immediate reactions is fine for the Dolev-Yao intruder model, it is not for location-limited channels. In such channels the intruder may not prevent messages from arriving at their intended destinations. The agents may communicate directly with one another, and we will therefore not have this delay introduced by the intruder. Without it, however, the whole protocol run between honest agents will chain together a series of immediate reactions and will appear to be executed in a single step. The intruder will have no chance to perform any actions to influence the protocol run, he will only see the state before the run begins and then the state after the run is finished. Clearly, we cannot allow this in location-limited channels. Instead, we must enforce that the agents themselves introduce the required delay. For this, we need the spontaneous action arrow (`--|>`). In an location-limited model, we would model the receipt of a message (`Request`) and the sending of the corresponding response (`Reply`) through the following two transitions:

```
state = 0 /\ RCV(Request) =|>  state' = 1
state = 1                 --|>  state' = 2 /\ SND(Reply)
```

This more general model works for both DY and location-limited channels.

The basic roles specified for the NSPK-KS example of Figure 5.3 make use of only DY channels and therefore the correspondent next-state relations can be modelled as sets of immediate reactions. For instance, `Alice` role is defined on top of four immediate reactions. The first three model the aforementioned "if-then-else" control-flow whereby `Alice` should contact the

`Server` *only* if she does not yet possess `Bob`'s public key in her key-ring `KR`. Namely, when `Alice` is in `State` equal to 0, and the `START()` signal occurs, then the action on the RHS of either `astep1a` or `astep1b1` can be immediately applied depending on whether `Alice` already knows `Bob`'s public key or not (this is modelled by means of the membership check on the key-ring `KR`). If `astep1a` fires, then `State` is updated to 2, a fresh value for the nonce `Na` is generated,[18] a message encrypted with `Bob`'s public key and containing the above nonce and `Alice`'s identity is sent, and a goal fact is asserted (see paragraph below on "Goals"). Vice versa, when there is no entry for the agent `B` inside the key-ring, then the action on the RHS of `astep1b1` fires. Immediately, `State` is updated to 1 and a message containing the identities of `Alice` and `Bob` is sent as a request for the `Server` to receive back the public key of `Bob` (see the `Server`'s immediate reaction `sstep0` in Figure 5.3). When `State` is set to 1, the only transition that `Alice` can execute is `astep1b2` which also requires the reception of a message signed by the `Server` and containing `Bob`'s identity and his public key. By executing `astep1b2`, `State` is updated to 2, an entry for `Bob`'s public key is added to `Alice`'s key-ring, a fresh value for the nonce `Na` is generated, a message encrypted with `Bob`'s public key and containing such a nonce and `Alice`'s identity is sent, and a goal fact is asserted. Finally, `Alice`'s last transition (`astep2`) expresses that when she is in `State` equal to 2 and she receives a message encrypted with her public key and containing the nonce `Na` she has previously sent and a new fresh value of nonce `Nb`, then `State` is updated, a message encrypted with `Bob`'s public key is immediately sent, and a goal fact is asserted.[19]

Instantiation

Given the basic roles that make up our protocol specification, we can now instantiate them via composed roles. Composition can be sequential (using the `;` operator) or parallel (using the `/\` operator). Using these composition operators, we can model situations such as an agent who executes several sub-protocols of a particular protocol suite in order, or an agent who is involved in several protocol sessions at once.

One possible instantiation of the NSPK-KS example protocol is shown in Figure 5.5. Here, we instantiate a single trusted server and two sessions of the NSPK protocol without key-server in parallel: one between agents `a` and `b` and one in which agent `a` initiates a protocol run with the intruder `i`. Note that in this latter session, the intruder need not be explicitly modelled

[18] Since `Na` is a fresh variable (i.e. it is tagged as `fresh`), then it is intended to be freshly generated every time `Alice` updates the value of `Na`.

[19] Notice that, if the variable `Nb` would not be primed in such a transition, then also the value of `Nb` would be simply compared with the correspondent part of the message on the receiving channel.

by a call to role `Bob`, as his DY capabilities subsume this. Although we give one particular ground instantiation in the example, we can also specify an instantiation that contains variables for the agent names and thereby symbolically represents a set of possible instantiations. The given scenario is sufficient to detect, for instance, the man-in-the middle attack on NSPK-KS described by Lowe in [Low96].

The instantiation also illustrates another feature of HLPSL: shared knowledge. We can easily associate a given variable, in this case the key-rings, to a particular agent, and share that variable across all protocol sessions in which the agent participates. Here, agent `a` plays in two parallel instances of `Alice`, and any public keys she learns in one will thus be available to her in the other. Such variable sharing is important for the modelling of group protocols and greatly enhances the expressiveness of HLPSL.

Goals

HLPSL focuses on safety temporal properties [AVI03c]. To specify goals, we can compose temporal formulae using operators `[]` ("always"), and `<->` ("sometime in the past"), and `(-)` ("one time instant in the past"), as well as standard logical connectives such as conjunction and negation. Goal formulae are defined over so-called *goal facts*: intuitively, predicates that are true at the moment they appear on the RHS of a transition. Hence a goal fact corresponds to the assignment of the truth value to an appositely devised HLPSL boolean variable. Such a boolean variable is always false except in those states reached by transitions that active it. For instance, a goal fact Gf asserted in the RHS of an HLPSL transition is just a macro for $Gf' = \text{TRUE}$. Note that a rich set of security goals are expressible in this way, including secrecy, authentication, and a host of others described in [AVI03c].

Assume that, in the NSPK-KS protocol, `Bob` should weakly authenticate `Alice` on her nonce `Na`. According to Lowe's notion of *non-injective agreement* in [Low97], this can be informally expressed as:

> Whenever `B` (acting as `Bob`) completes a run of the protocol, apparently with `A` (acting as `Alice`), then *(i)* `A` has previously been running the protocol apparently with `B`, and *(ii)* the two agents agreed on `Na`.

To express such a property the user is free to define arbitrary goal facts and interpret them as is desired. We simply use two goal facts, `witness` and `request`, for specifying authentication properties. The basic idea is the following: when the authenticator `Bob` completes its run of the protocol, he arises a `request` that, in order to guarantee the authentication property, should be faithfully satisfied by a correspondent `witness` fact actived by `Alice`. Since the above property relies on an agreement between the

```
goal
 % Weak authentication.
 [](request(B,A,na,Na) --> <-> witness(A,B,na,Na))
end goal
```

Figure 5.6: NSPK-KS protocol: HLPSL spec. (goals)

authentication partners on the value assigned to the nonce generated by `Alice`, then `witness` and `request` must specify, respectively, what is this value from the point of view of `Alice` and `Bob`. In order to achieve this we devise an identifier (e.g. `na`), declared in HLPSL of type `protocol_id`, for each object (e.g. `Na`) on which the authentication partners should agree. Specifically, we define our goal authentication facts as follows:

- `witness(A,B,na,Na)` should be read "agent `A` playing the role `Alice` wants to execute the protocol with agent `B` by using `Na` as value for the authentication identifier `na`."
- `request(B,A,na,Na)` should be read "agent `B` accepts the value `Na` and now relies on the guarantee that agent `A` exists and agrees with him on this value for the authentication identifier `na`."

We thus express our desired security property in a `goal` declaration as shown in Figure 5.6.[20] Intuitively, in order to satisfy weak authentication (also called non-injective agreement), it must be the case that a `request` fact has been preceded by a correspondent `witness` fact. This satisfies the above Lowe's conditions *(i)* and *(ii)*.

Compound Types

HLPSL supports for the declaration of compound types which allow the protocol designer to declare HLPSL variables of a type restricted and specialised with respect to the items the variables represent. Let us describe compound types by means of the Kerberos protocol depicted in Figure 5.7. In such a protocol, a client C obtains credentials from the trusted authentication server A (steps (1) and (2)) and from the ticket granting server G (steps (3) and (4)) in order to request access to a service provided by the server S (steps (5) and (6)). A detailed explanation of the Kerberos protocol goes beyond the scope of this paragraph and the interested reader should consult [Ker, NT94].

Let us concentrate our attention on the term $Ticket_1$ (the same ideas apply also to $Ticket_2$). One can immediately see that since C does not know

[20] Even if it is not depicted explicitly by the Figure 5.6, HLPSL allows for the declaration of multiple goal formulae intended conjunctively.

$$
\begin{array}{llcll}
(1) & C & \rightarrow & A & : & C, G, Lifetime_1, N_1 \\
(2) & A & \rightarrow & C & : & C, Ticket_1, \{G, K_{CG}, Tstart_1, Texpire_1, N_1\}_{K_{CA}} \\
(3) & C & \rightarrow & G & : & S, Lifetime_2, N_2, Ticket_1, \{C, T_1\}_{K_{CG}} \\
(4) & G & \rightarrow & C & : & C, Ticket_2, \{S, K_{CS}, Tstart_2, Texpire_2, N_2\}_{K_{CG}} \\
(5) & C & \rightarrow & S & : & Ticket_2, \{C, T_2\}_{K_{CS}} \\
(6) & S & \rightarrow & C & : & \{T_2\}_{K_{CS}}
\end{array}
$$

where the terms $Ticket_1$ and $Ticket_2$ are defined as:

$$
\begin{array}{lcl}
Ticket_1 & := & \{C, G, K_{CG}, Tstart_1, Texpire_1\}K_{AG} \\
Ticket_2 & := & \{C, S, K_{CS}, Tstart_2, Texpire_2\}K_{GS}
\end{array}
$$

Figure 5.7: The Kerberos protocol

the key K_{AG} shared between A and G, then C cannot decrypt $Ticket_1$ sent by A at protocol step (2). The client C in the protocol step (3) just forwards such a term to the intended receiver that will be able to decrypt it and to check its content.

Of course, the protocol designer could declare $Ticket_1$ as belonging to the most general HLPSL type `message`, but this allows the intruder to send whatever he wants to the agents playing role C in place of $Ticket_1$. However, even if the first principal who receives $Ticket_1$ cannot decrypt it, $Ticket_1$ is forwarded and later on it will be decrypted and checked for well-formedness. In terms of the search tree, this corresponds to the exploration of a huge number of useless branches that may dramatically affect the performance of a model-checking tool. Such an explosion of the search space can be avoided by using compound types that allow the protocol designer to restrict "a priori" the patterns of messages like for instance those of messages that a receiver cannot decrypt.

For instance, concerning the Kerberos protocol, the HLPSL declaration

```
Ticket_1: {agent.agent.symmetric_key.text.text}_(symmetric_key)
```

specifies the type of the HLPSL variable `Ticket_1` representing the object $Ticket_1$ to be restricted to the domain of all the terms of the form $\{A_1.A_2.K_1.Num_1.Num_2\}_{K_2}$ such that A_1 and A_2 are of type `agent`, K_1 and K_2 are of type `symmetric_key`, and Num_1 and Num_2 are of type `text`.

The correctness of compound typing is obvious as it excludes computations and therefore any attack that exists in the model with compound typing also exists in the model without it. As far as completeness is concerned, preliminary work in this direction indicates that compound typing is indeed attack-preserving (that is, if there is an attack on the protocol in the model without compound typing, then there is also an attack in the model

with compound typing) for protocols satisfying the following two conditions: *(i)* they work under the free-algebra assumption, and *(ii)* every time a principal receives a message of a compound type that he cannot decrypt, then such a message is simply forwarded in clear-text to a third party and it has no further relevance to that receiver. One can immediately see that the Kerberos protocol belongs to this class.

5.1.3 The Semantics

The semantics of HLPSL is based on TLA[Lam94], a powerful logic which is well-suited to the specification of concurrent systems like security protocols. TLA itself has an intuitive and easily understandable semantics, making it a formalism that protocol designers and engineers can find accessible.

TLA specifies a transition system by describing its allowed behaviours by means of a single formula of the form:

$$System(\Psi) \triangleq Init(\Psi) \wedge \Box Next(\Psi, \Psi')$$

where Ψ is the set of state variables ranging on a domain $\mathcal{D}$[21] (Ψ' refers to this set of variables in the next state), *Init* and *Next* are first-order formulae representing the initial states and the next-state relation, respectively. The above formula corresponds to a transition system $\mathcal{T} = \langle \Sigma, \mathcal{S}_0, \rightarrow \rangle$, where Σ, the set of states, is a set of total assignments $\sigma : \Psi \mapsto \mathcal{D}$; $\mathcal{S}_0 \subseteq \Sigma$, such that $\models_{\sigma_0} Init(\Psi)$ for each $\sigma_0 \in \mathcal{S}_0$, is the set of initial states; and $\rightarrow \subseteq \Sigma \times \Sigma$ is the transition relation such that $\sigma_1 \rightarrow \sigma_2$ iff $\models_{\sigma_1 \cup \sigma_2'} Next(\Psi, \Psi')$ where $\sigma' = \{\langle x', d\rangle \mid \langle x, d\rangle \in \sigma\}$.[22]

For a sequence of states $\pi = \sigma_0, \sigma_1, \ldots$, we define $\pi(i) = \sigma_i$, $\pi^{\geq i} = \sigma_i, \sigma_{i+1}, \ldots$, and $\pi_{\leq i} = \sigma_0, \sigma_1, \ldots, \sigma_i$ for $i = 0, 1, \ldots$, and we say that π is a *behaviour* in $\mathcal{T}$ iff $\sigma_i \rightarrow \sigma_{i+1}$ for each $i = 0, 1, \ldots$.

In the rest of this section we will show how to express a security protocol specified in HLPSL as a TLA formula. However, besides a TLA formula corresponding to a security protocol, an HLPSL specification declares also channels associated to intruder models and the properties that the protocol is required to satisfy. The formal definition of what it means to check that a security protocol achieves its security properties under an hostile environment will be presented.

Messages

We begin by specifying the structure of messages and the properties of the operations on the set *Msg* of all messages. For brevity, we focus on

[21] For simplicity we consider a single domain for all the variables. However what follows can be easily extended to multi-domain.

[22] Notice that $\models$ is the entailment relation in classical first-order logic.

those operations needed for the NSPK-KS running example: namely, pairing $pair(M_1, M_2) = M_1.M_2$ and asymmetric encryption, $acrypt(K, M) = \{M\}_K$.[23]

In general, HLPSL allows for the declaration of algebraic equations that specify an equational theory $\approx$. Let Ω be the signature from which Msg is constructed, the interpretation of HLPSL is required to be a quotient interpretation of a free term algebra $\mathcal{T}_\Omega$ modulo the equational theory $\approx$ (see [VG91]). For instance, $\approx$ could include an equation stating that pairing is associative $Pair(Pair(m1, m2), m3) \approx Pair(m1, Pair(m2, m3))$, equations expressing properties of cryptographic algebraic operators (e.g. the associativity and commutativity of the XOR operator), etc. However, in the context of this Thesis we do not assume properties neither for cryptographic operators nor for pairing (this means that, for instance, $pair(m1, m2)$ is equal to $pair(m2, m1)$ iff $m1$ is equal to $m2$).

Translating HLPSL Roles to TLA

In this subsection, we show how HLPSL roles are mapped into TLA. For simplicity, we do not distinguish between variables that are local to a role (or module) and variables that are owned by the role. More precisely, we rename local variables to avoid name clashes with the environment and later we replace them by *owned* shared variables. They are in principle visible to the environment, but due to the renaming have absolutely no effect on it. Thus this transformation preserves the semantics. Moreover, we rename each occurrence of the starting signal `START()` to a different signal $\texttt{START}_i\texttt{()}$, where $i = 1, 2, \ldots$ enumerates all the possible occurrences of `START()` overall the specified role instances. $\texttt{START}_i\texttt{()}$ is then an independent event, $\texttt{START}_i{}'$ $\neq \texttt{START}_i$, that can happen at any time.

Figure 5.8 shows the structure of a basic role (denoted with B) and of a composed role (denoted with P) in HLPSL where Ψ and Θ are, respectively, the set of state variables and the set of owned variables, pl is the agent playing B and knowing κ at the beginning, and R_1 and R_2 are roles (for brevity, sequential composition, the loop construct and the acceptance conditions are not discussed). Moreover, even if it is not depicted explicitly by the Figure 5.8, let Υ be the set of fresh variables updated by the role. By definition it holds that $\Theta \cup \Upsilon \subseteq \Phi$. Since immediate reactions can be also used to express spontaneous actions (see paragraph above on "Transitions"), we focus here only on them.

We will proceed inductively translating to TLA, starting with basic roles B and then giving the translation of the composed role P in terms of the translation of its components, R_1 and R_2.

[23] Notice that, we can easily extend this model of messages to include other operators like symmetric encryption, exponentiation, XOR, and their associated properties.

```
role B(Ψ) played_by pl def=
  owns Θ
  init Init
  knowledge(pl) = κ
  transition
    lb1.  ev1 =|> act1
    lb2.  ev2 =|> act2
    ...
end role
```

```
role P(Ψ) def=
  owns Θ
  init Init
  composition
    R1 ∧ R2
end role
```

Figure 5.8: HLPSL roles: generic structure.

In order to describe the TLA translation of a basic role, let us first define tr_i be the i-th transition rule of the basic role (i.e., $tr_i \triangleq ev_i$ =|> act_i), $RMsg(tr_i, B)$ and $SMsg(tr_i, B)$ be the sets of messages received and sent by pl in transition tr_i of B (i.e., $RMsg(tr_i, B) \triangleq \{m \mid ch \textit{ is a channel}, ch(m) \textit{ occurs in } ev_i\}$, and $SMsg(tr_i, B) \triangleq \{m \mid ch \textit{ is a channel}, ch(m) \textit{ occurs in } act_i\}$), $\Upsilon(tr, B)$ be the set of variables that are freshly generated by the transition tr in B (i.e., $\Upsilon(tr, B) \triangleq \{v \mid v \textit{ in } \Upsilon,\ v' \textit{ occurs in } tr\}$), and $Used_{\langle B,tr_i\rangle}$ (with $i = 1, 2, \ldots$) be TLA variables each one devised to model a set that keeps track of those fresh values that have already been generated in executing the transition tr_i (with $i = 1, 2, \ldots$) of B. Moreover, let $Used_B$ be the TLA variable representing the set that keeps track of those fresh values that have already been generated by the role B, and let $Kw_{\langle B,pl\rangle}$ be the TLA variable representing the knowledge of B's player pl. The TLA translation of a basic role is then given by the following:

$$TLA(B) \triangleq Init(B) \wedge \Box\, Next(B)$$

where $Init(B) \triangleq Init \wedge Kw_{\langle B,pl\rangle} = \kappa$, and $Next(B)$ is defined as:

$$\begin{aligned}
&\wedge \bigwedge_i ev_i \Rightarrow \wedge\ act_i && (5.1)\\
&\qquad\qquad \wedge \bigwedge_{(v_1, v_2 \textit{ in } \Upsilon(tr_i,B))} {v_1}' \neq {v_2}'\\
&\qquad\qquad \wedge \bigwedge_{(v \textit{ in } \Upsilon(tr_i,B))} v' \notin Used_{\langle B,tr_i\rangle} \wedge v' \in {Used_{\langle B,tr_i\rangle}}'\\
&\qquad\qquad \wedge \bigwedge_{(m \textit{ in } RMsg(tr_i,B))} An(m, {Kw_{\langle B,pl\rangle}}')\\
&\qquad\qquad \wedge \bigwedge_{(m \textit{ in } SMsg(tr_i,B))} Cmp(m, {Kw_{\langle B,pl\rangle}}')\\
&\wedge \bigwedge_{(i \neq j)} Used_{\langle B,tr_i\rangle} \cap Used_{\langle B,tr_j\rangle} = \emptyset && (5.2)\\
&\wedge \bigwedge_{(\theta \textit{ in } \Theta)} \theta' \neq \theta \Rightarrow Mod(\theta, B) && (5.3)\\
&\wedge Kw_{\langle B,pl\rangle} \subseteq {Kw_{\langle B,pl\rangle}}' && (5.4)\\
&\wedge Used_B = \bigcup_i Used_{\langle B,tr_i\rangle} && (5.5)\\
&\wedge Used_B \subseteq {Used_B}' && (5.6)
\end{aligned}$$

with:[24]

$$Mod(x, B) \triangleq \bigvee_i \{ev_i \mid x' \text{ occurs in } tr_i\} \tag{5.7}$$

$$\begin{aligned} An(m, Kw) \triangleq\ & \wedge\ m \in Kw \\ & \wedge\ \exists_{m1,m2 \in Msg}\ m = pair(m1, m2) \Rightarrow \wedge\ An(m1, Kw) \\ & \qquad\qquad\qquad\qquad\qquad\qquad \wedge\ An(m2, Kw) \\ & \wedge\ \exists_{m1,k1 \in Msg} \begin{pmatrix} \wedge\ m = acrypt(k1, m1) \\ \wedge\ Cmp(inv(k1), Kw) \end{pmatrix} \Rightarrow An(m1, Kw) \end{aligned} \tag{5.8}$$

$$\begin{aligned} Cmp(m, Kw) \triangleq\ & \vee\ m \in Kw \\ & \vee\ \exists_{m1,m2 \in Msg}\ m = pair(m1, m2) \wedge\ Cmp(m1, Kw) \\ & \qquad\qquad\qquad\qquad\qquad\qquad \wedge\ Cmp(m2, Kw) \\ & \vee\ \exists_{m1,k1 \in Msg}\ m = acrypt(k1, m1) \wedge\ Cmp(k1, Kw) \\ & \qquad\qquad\qquad\qquad\qquad\qquad \wedge\ Cmp(m1, Kw) \end{aligned} \tag{5.9}$$

Initially, *Init* holds, and in every step the above conjunction of formulae denoted with $Next(B)$ must be satisfied. Namely, (5.1) states that if an event is triggered, then the changes specified by the corresponding action take place, the fresh variables updated by the transition are assigned to different values that have never been used as fresh value by this transition, the player's knowledge is extended with all the terms that can be derived by analysing (An) the messages received, and the player is able to compose (Cmp) the messages he is going to send. Moreover, (5.2) imposes that the sets of fresh values issued by different transitions are disjoint. Besides this, (5.3) states that if one of the variables owned by the role changes, then the variable is actually modified by this role. It is our convention that if a role owns a variable then this variable is never modified by any role "outside" the current one. Formula (5.4) express the monotonicity of the knowledge of the player. Finally, (5.5) defines a TLA variable representing all the fresh values used by the role B and (5.6) imposes that such a set grows up monotonically.

An agent may simultaneously participate both in different roles and in different sessions of the protocol. In this case, the two role instances could share some internal variables of the agent. This variable sharing is not done via some channel, but it is a straightforward consequence of using the same TLA variable in both the formulae representing the roles that share the variable.

Note that a transition that has to refer to the already known value of a variable will use the name of this variable, without prime, in any side of the transition. Vice versa, when a transition has to assign a value to a variable

[24]Notice that, also a channel *Ch* can be owned by a role. In such a case, $Mod(Ch, B)$ is intended to be applied on transitions in which the channel macro has been replaced with its appropriate conjunction of predicates (see paragraph above on "Communication and Intruder Models"). However, it is recommended to avoid ownership of channels since it can be cause of clashes with the TLA formulae declared for the intruder.

will use the name of this variable with prime. For what concerns a generic role *Role*, let $\Theta(Role)$, and $\Upsilon(Role)$ be the set of its owned variables, and the set of its fresh variables, respectively. It is immediate to see that for a basic role B, $\Theta(B) = \Theta$ and $\Upsilon(B) = \Upsilon$.

The TLA translation of the parallel composition of R_1 and R_2 is as follows:

$$
\begin{aligned}
TLA(P) \quad \triangleq \quad & \wedge\ TLA(R_1) \wedge TLA(R_2) && (5.10)\\
& \wedge\ Init && (5.11)\\
& \wedge \Box\ \wedge \bigwedge\nolimits_{(\theta\ in\ \Theta)} \theta' \neq \theta \Rightarrow Mod(\theta, P) && (5.12)\\
& \qquad \wedge\ Used_{R_1} \cap Used_{R_2} = 0 && (5.13)\\
& \qquad \wedge\ Used_P = Used_{R_1} \cup Used_{R_2} && (5.14)
\end{aligned}
$$

where:[25]

$$Mod(x, P) \quad \triangleq \quad Mod(x, R_1) \vee Mod(x, R_2)$$

It is thus defined as the conjunction of $TLA(R_1)$, $TLA(R_2)$ (see (5.10)) and some extra terms accounting for extra initial constraints (see (5.11)), taking ownership of variables (see (5.12)) and freshness. Namely (5.12) states that if a variable is owned by P then the value of such variable can be modified only by applying transitions of R_1 or R_2. For what concerns freshness, (5.13) imposes that the sets of fresh values issued by R_1 and R_2 are disjoint, and (5.14) defines a TLA variable that keeps track of those fresh values that have already been used by P, i.e. the union of the fresh values used by R_1 and R_2.[26] Moreover, it is immediate to see that $\Theta(P) = \Theta(R_1) \cup \Theta(R_2) \cup \Theta$ and $\Upsilon(P) = \Upsilon(R_1) \cup \Upsilon(R_2)$.

The TLA translation of sequentially composed roles, which we omit here for brevity, is analogous. One must augment the translation with an auxiliary flag recording which of the two roles, R_1 or R_2, is executing and take into account the acceptance conditions. Once R_1 has reached an accepting state, we toggle the flag to indicate that R_2 should begin execution.

Intruder Model

We formalise the capabilities of the intruder as a set of rules the intruder may execute. We focus here on the well-known Dolev-Yao (DY) intruder model of [DY83] but note that the definition of alternate intruder models is a simple matter of axiomatically describing their capabilities. In this way, we can easily model a system in which the intruder has full DY capabilities

[25] Notice that, the sets of basic and composed roles are disjoint. Therefore it is immediate to select the appropriate *Mod* predicate to be applied on a given role.

[26] Notice that, the monotonicity of $Used_P$ simply follow by the monotonicity of $Used_{R_1}$ and $Used_{R_2}$.

$$
\begin{array}{lcl}
Read(SND) & \triangleq & \exists_{m \in Msg} \ \wedge\ SND(m) \ \wedge\ IK' = IK \cup \{m\} \\
ASplit & \triangleq & \exists_{m1,m2 \in Msg} \ \wedge\ pair(m1,m2) \in IK \ \wedge\ IK' = IK \cup \{m1,m2\} \\
AAdec & \triangleq & \exists_{k,m \in Msg} \ \wedge\ acrypt(k,m) \in IK \ \wedge\ inv(k) \in IK \ \wedge\ IK' = IK \cup \{m\} \\
GPair & \triangleq & \exists_{m1,m2 \in Msg} \ \wedge\ m1 \in IK \ \wedge\ m2 \in IK \ \wedge\ IK' = IK \cup \{pair(m1,m2)\} \\
GAcrypt & \triangleq & \exists_{k,m \in Msg} \ \wedge\ k \in IK \ \wedge\ m \in IK \ \wedge\ IK' = IK \cup \{acrypt(k,m)\} \\
GFresh & \triangleq & \exists_{x \in Msg} \ \wedge\ x \notin IUsed \ \wedge\ x \in IUsed' \ \wedge\ IK' = IK \cup \{x\}
\end{array}
$$

Figure 5.9: Dolev-Yao intruder knowledge formulae

over certain communication channels, can only listen on others, and has no access to a third set of channels.

The DY intruder controls any channel tagged with the `(dy)` attribute. In the sequel, *SND* and *RCV* refer to sending and receiving DY channels, respectively. The DY intruder reads every sending channel *SND* (namely, it reads every message that the agents write on these channels), analyses the messages, (i.e. generates terms and messages based on them), and inserts the composed messages into any receiving channel *RCV*. The "knowledge of the intruder", *IK*, is the set of all terms that the intruder may create. The initial value of this variable is explicitly set in HLPSL (and is augmented by the initial knowledge of any agent roles played by the intruder) and changes according to the formulae of Figure 5.9: when the intruder reads a new message from a channel *SND*, *Read*(*SND*), when he analyses his knowledge (decomposing a pair into its components, *ASplit*, or decrypting encrypted terms, if he possesses the appropriate key, *AAdec*), or when he composes new terms (generating pairs, *GPair*, encrypting a message using a known key, *GAcrypt*, or generating fresh terms,[27] *GFresh*).

IK may only change when one of these actions happens, allowing the intruder to introduce his knowledge into the network. The intruder behaviour is thus formalised by as follows:

$$
\begin{array}{rcl}
Intruder_{DY} & \triangleq & \Box \ \wedge\ IK' \neq IK \Rightarrow \vee \bigvee_{SND} Read(SND) \\
 & & \qquad\qquad \vee\ ASplit \vee AAdec \\
 & & \qquad\qquad \vee\ GPair \vee GAcrypt \vee GFresh \\
 & & \wedge \bigwedge_{RCV} (RCV(msg) \Rightarrow msg \in IK')
\end{array}
$$

Freshness

In the previous paragraphs we showed that each role *Role* keeps track in the set $Used_{Role}$ of those fresh values that have already been generated by

[27] Notice that, the TLA variable *IUsed* keeps track of those fresh values that have already been generated by the intruder.

itself. In doing this *Role* also guarantees the freshness of these values for what concerns its execution. The intruder makes the same for itself too and therefore we can guarantee the freshness of the whole system simply by enforcing the following:

$$Nonce\_Prop \triangleq \Box \ \ Used(TR) \cap IUsed = \emptyset$$

where TR is the role at the topmost-level of the HLPSL hierarchy.[28]

Goals

As already presented in Section 5.1.2, goals in HLPSL are specified as temporal formulae built on top of goal facts that are explicitly asserted by basic roles in executing their transitions. To assert a goal fact corresponds to assign the truth value to a HLPSL boolean variable representing the goal fact.

Let Γ be the collection of such boolean variables, TR be the role at the topmost-level of the HLPSL hierarchy, and $\mathcal{T} = \langle \Sigma, \mathcal{S}_0, \rightarrow \rangle$ the transition system represented by the following TLA formula

$$\begin{array}{lll} \wedge & TLA(TR) & \\ \wedge & \bigwedge_{(\gamma \ in \ \Gamma)} \wedge \gamma = \text{FALSE} & \\ & \quad \wedge \Box \ (\gamma' = \text{TRUE} \Leftrightarrow Mod(\gamma, TR)) & (5.15) \end{array}$$

where (5.15) states that a goal fact *(i)* is initially false and *(ii)* holds only in those states reached by those transitions that assert it.

Besides this, let π be a behaviour in $\mathcal{T}$ and ϕ a generic safety temporal formula, then ϕ holds in π at time i, denoted with $(\pi, i) \models \phi$, is inductively defined as in Figure 5.10.[29] A safety temporal formula ϕ is valid on a

$$\begin{array}{lll} (\pi, i) \models p & \text{iff} & p \text{ holds in } \pi(i) \\ (\pi, i) \models \neg\phi_1 & \text{iff} & (\pi, i) \not\models \phi_1 \\ (\pi, i) \models \phi_1 \vee \phi_2 & \text{iff} & (\pi, i) \models \phi_1 \text{ or } (\pi, i) \models \phi_2 \\ (\pi, i) \models \phi_1 \wedge \phi_2 & \text{iff} & (\pi, i) \models \phi_1 \text{ and } (\pi, i) \models \phi_2 \\ & & \\ (\pi, i) \models \odot\phi_1 & \text{iff} & i > 0 \text{ and } (\pi, i-1) \models \phi_1 \\ (\pi, i) \models \diamondsuit\phi_1 & \text{iff} & \text{exists } 0 \leq j \leq i \text{ s.t. } (\pi, j) \models \phi_1 \end{array}$$

Figure 5.10: The semantics of safety temporal formulae

behaviour π in $\mathcal{T}$, denoted with $\pi \models \phi$, iff $(\pi, 0) \models \phi$. A safety temporal formula ϕ is universally valid in $\mathcal{T}$, written $\mathcal{T} \models \Box\phi$, iff $\pi \models \phi$ for every behaviour π in $\mathcal{T}$.

[28] Alternatively, we could enforce that all the set of used fresh values are disjoint.

[29] In Figure 5.10, ϕ_1 and ϕ_2 are safety temporal formulae and p is a goal fact.

Given this and let $\mathcal{G}$ be the HLPSL safety formula (see for instance Figure 5.6) the security protocol is required to satisfy, then we say that the security protocol (specified in HLPSL) achieves its security properties (expressed in HLPSL too) iff $\mathcal{T} \models \bigwedge_{(g\ in\ \mathcal{G})} \forall(g)$.[30]

5.2 The Intermediate Format (IF)

In order to provide the appropriate abstraction level amenable to automated formal analysis, security protocol specified in HLPSL are then automatically translated (see Figure 5.1) into equivalent ones expressed in a lower-level language. Such a language is based on first-order multiset rewriting and it is called the Intermediate Format (IF).[31]

The IF specifies both a *state transition system* (states, initial states, and transition relation) describing the security protocol as well as a set of *goal* (i.e. *attack*) *states*. A state is represented through a set of facts (e.g. the fact that the intruder knows a particular message), while the transition relation is given by means of a set of (labelled) *conditional rewrite rules* on states.[32] An *attack trace* is thus a path that leads from an initial state to an attack state. It is immediate to see that IF allows for the specification of protocol insecurity problems already described in Section 3.2.

We defined a preliminary version of the IF, denoted with IF v.1, as part of the AVISS project. Since then, in order to be able to analyse industrial-scale Internet security protocols, we have redesigned the language still preserving its underlying basic idea of being based on the first-order multiset rewriting formalism. The most important new concept is the extension of the left-hand side (LHS) of rules with conditions and negative facts, in order to allow for the explicit modelling of a wider class of protocols and properties in a natural way. The introduction of negation also requires us to extend the model of the intruder that in IF v.1 did not have the ability to generate fresh data himself. Since without negation the honest agents could not check that messages are different (i.e. they perform only equality checks), the intruder could always send the same messages again (as is done in approaches based on *data independence*, such as [RB99]). With inequalities allowed in rules, we must in general allow the intruder to generate an unbounded number of fresh messages.

The first-order multiset rewriting formalism gives to the IF a great flexibility for further extensions. For instance, to integrate a new intruder model requires only introducing new facts and new intruder rules, while the rest of the IF syntax and semantics remains unchanged. To better exploit the

[30] Let ϕ be a safety temporal formula, then $\forall(\phi)$ is its universal closure.

[31] The idea of regarding security protocols as multiset-rewriting systems is not new and it has been already asserted and validated in [DMT98, CDL+99, JRV99].

[32] Rewrite rules allow transforming a set of facts into another one by making localised changes to the elements that appear in it [CDL+00].

IF flexibility we use a *prelude* file that defines all protocol-independent aspects of the model, such as intruder models. The prelude file is thus a fixed file that is shared between the HLPSL and the IF, and that contains all declarations that are protocol-independent (while each IF-file contains only declarations for a particular protocol).

The IF language is well-suited to specify a spread spectrum of security protocols as well as to be formally analysed by means of a variety of automated reasoning techniques. However in the sequel we will focus only on the IF features that are exploited by our SAT-based approach presented in Chapter 4. For instance, even if the IF allows for the specification of algebraic properties of cryptographic operators (e.g. the exponentiation operator used in the Diffie-Hellman protocol [DH76]), we will not describe such a feature since our SAT-based approach does not support for it.[33]

In the rest of this section, we start presenting the IF language via a running example, the NSPK-KS protocol (see Figure 5.2), and we conclude with the formal description of the IF semantics.

5.2.1 The Language

Let us introduce IF with the help of a running example, the complete Needham-Schroeder Public Key protocol including key-server (NSPK-KS, [NS78]) presented in Section 5.1. As already mentioned before, a security protocol problem consists of an IF-file and a prelude file. While the latter contains all declarations that are protocol-independent, the former presents those declarations that are specific for the considered protocol. Figure 5.11 depicts a fragment of the prelude file, namely the type identifiers, some signature declarations, and the rules modelling the DY intruder. The IF specification of the NSPK-KS protocol is shown in Figure 5.12, Figure 5.13, and Figure 5.14.

In there, we model the server as a "persistent" agent, whose `state`-fact never changes and that uniformly reacts to every incoming key-request with the appropriate certificate if there is such a certificate in his database. The key-table the server possesses is modelled by a finite set of pairs of agents and their public keys. Alternatively, we could use a function to this end, which would simplify matters when the designer desires to model a situation in which the server knows the public key of each agent in the world.

IF specifications and prelude are divided into sections. Let us see each of them in more details.

In the sections `signature` and `types`, the types of variables, constants, function symbols and fact symbols are declared.

Section Signature. This section contains declarations of the used function and fact symbols, and, more specifically, their types. Also it

[33] As explained in Chapter 9, this will be subject of future work.

```
section typeSymbols:
  agent, text, symmetric_key, public_key, function,
  message, fact, nat, protocol_id, bool, set

section signature:
  message > agent
  % ..other declarations
  pair : message * message -> message
  % ..other declarations

section intruder (dy):
  % generate rules
  step gen_pair (PreludeM1,PreludeM2) :=
    iknows(PreludeM1).iknows(PreludeM2)
     => iknows(pair(PreludeM1,PreludeM2))
  step gen_crypt (PreludeM1,PreludeM2) :=
    iknows(PreludeM1).iknows(PreludeM2)
     => iknows(crypt(PreludeM1,PreludeM2))
  step gen_scrypt (PreludeM1,PreludeM2) :=
    iknows(PreludeM1).iknows(PreludeM2)
     => iknows(scrypt(PreludeM1,PreludeM2))
  step gen_exp (PreludeM1,PreludeM2) :=
    iknows(PreludeM1).iknows(PreludeM2)
     => iknows(exp(PreludeM1,PreludeM2))
  step gen_xor (PreludeM1,PreludeM2) :=
    iknows(PreludeM1).iknows(PreludeM2)
     => iknows(xor(PreludeM1,PreludeM2))
  step gen_apply (PreludeM1,PreludeM2) :=
    iknows(PreludeM1).iknows(PreludeM2)
     => iknows(apply(PreludeM1,PreludeM2))

  % analysis rules
  step ana_pair (PreludeM1,PreludeM2) :=
    iknows(pair(PreludeM1,PreludeM2))
     => iknows(PreludeM1).iknows(PreludeM2)
  step ana_crypt (PreludeK,PreludeM) :=
    iknows(crypt(PreludeK,PreludeM)).iknows(inv(PreludeK))
     => iknows(PreludeM)
  step ana_scrypt (PreludeK,PreludeM) :=
    iknows(scrypt(PreludeK,PreludeM)).iknows(PreludeK)
     => iknows(PreludeM)

  % Generating new constants of any type:
  step generate (PreludeM) :=
    =[exists PreludeM]=> iknows(PreludeM)
```

Figure 5.11: Prelude file

```
section signature:
 state_Alice  : agent * agent * public_key * public_key * set
                * nat * text * text * public_key * nat -> fact

 state_Bob    : agent * agent * public_key * public_key * set
                * nat * text * text * public_key * nat -> fact

 state_Server : agent * public_key * set * agent * agent
                * public_key * nat -> fact

section types:
 A, B, S, a, b, s, i               : agent
 Ka, Kb, Ks, ka, kb, ks, ki        : public_key
 Na, Nb                            : text
 na, nb                            : protocol_id
 KR, keyset_s, keyset_b, keyset_a  : set

 % dummy, session identifiers, etc
 Dummy_Na, Dummy_Nb, d_tx          : text
 Dummy_Ka, Dummy_Kb, d_pk          : public_key
 Dummy_B,  Dummy_A,  d_ag, AGoal   : agent
 SID, SID1, SID2, 0, 1, 2, 3, 4, 5 : nat
 MGoal                             : message

section inits:
 initial_state init1 :=
  % intruder initial knowledge
  iknows(i) .iknows(a) .iknows(b).
  iknows(ka).iknows(kb).iknows(ks).
  iknows(ki).iknows(inv(ki)).
  % server initial state
  state_Server(s,ks,keyset_s,d_ag,d_ag,d_pk,1).
  % session 1 [A:a, B:b]
  state_Alice(a,b,ka,ks,keyset_a,0,d_tx,d_tx,d_pk,2).
  state_Bob(b,a,kb,ks,keyset_b,0,d_tx,d_tx,d_pk,3).
  % session 2 [A:a, B:i]
  state_Alice(a,i,ka,ks,keyset_a,0,d_tx,d_tx,d_pk,4).

  % initial a keyset={(a,ka)}
  contains(pair(a,ka),keyset_a).
  % initial b keyset={(b,kb)}
  contains(pair(b,kb),keyset_b)
  % initial server keyset={(a,ka),(b,kb),(i,ki)}
  contains(pair(a,ka),keyset_s).
  contains(pair(b,kb),keyset_s).
  contains(pair(i,ki),keyset_s).
```

Figure 5.12: NSPK-KS protocol: IF spec. (signature, types, and inits)

```
section rules:

 step step1 (A,B,Ka,Ks,KR,Dummy_Na,Nb,Dummy_Kb,Na,SID) :=
  state_Alice(A,B,Ka,Ks,KR,0,Dummy_Na,Nb,Dummy_Kb,SID).
  contains(pair(B,Kb),KR).
 =[exists Na]=>
  state_Alice(A,B,Ka,Ks,KR,2,Na,Nb,Dummy_Kb,SID).
  iknows(crypt(Kb,pair(Na,A))).
  witness(A,B,na,Na).
  contains(pair(B,Kb),KR)

 step step_2 (A,B,Ka,Ks,KR,Na,Nb,Dummy_Kb,SID) :=
  state_Alice(A,B,Ka,Ks,KR,0,Na,Nb,Dummy_Kb,SID) &
  not(contains(pair(B,Kb),KR))
  =>
  state_Alice(A,B,Ka,Ks,KR,1,Na,Nb,Dummy_Kb,SID).
  iknows(pair(A,B))

 step step_3 (A,B,Ka,Ks,KR,Dummy_Na,Nb,Dummy_Kb,Na,Kb,SID) :=
  state_Alice(A,B,Ka,Ks,KR,1,Dummy_Na,Nb,Dummy_Kb,SID).
  iknows(crypt(inv(Ks),pair(B,Kb)))
 =[exists Na]=>
  state_Alice(A,B,Ka,Ks,KR,2,Na,Nb,Kb,SID).
  iknows(crypt(Kb,pair(Na,A))).
  witness(A,B,na,Na).
  contains(pair(B,Kb),KR)

 step step_4 (A,B,Ka,Ks,KR,Na,Dummy_Nb,Kb,Nb,SID) :=
  state_Alice(A,B,Ka,Ks,KR,2,Na,Dummy_Nb,Kb,SID).
  iknows(crypt(Ka,pair(Na,Nb)))
  =>
  state_Alice(A,B,Ka,Ks,KR,3,Na,Nb,Kb,SID).
  iknows(crypt(Kb,Nb)).
  request(A,B,nb,Nb,SID)

 % similar rules for role bob

 step step_0 (S,Ks,KR,Dummy_A,Dummy_B,Dummy_Kb,A,B,SID) :=
  state_Server(S,Ks,KR,Dummy_A,Dummy_B,Dummy_Kb,SID).
  iknows(pair(A,B)).
  contains(pair(B,Kb),KR)
  =>
  state_Server(S,Ks,KR,A,B,Dummy_Kb,SID).
  iknows(crypt(inv(Ks),pair(B,Kb))).
  contains(pair(B,Kb),KR)
```

Figure 5.13: NSPK-KS protocol: IF spec. (rules)

```
section goals:
 goal weak_authenticate_Na (A,B,Na,SID) :=
  request(B,A,na,Na,SID) &
  not(witness(A,B,na,Na)) &
  not(equal(A,i))
```

Figure 5.14: NSPK-KS protocol: IF spec. (goals)

contains subtype declarations (e.g. the declaration `message > agent` states that an agent is also a message). Almost all the information in this section is protocol independent (and hence part of the prelude, not of the concrete IF file), however there is one exception: the signature of a state-fact describing the state of an honest agent is protocol-dependent. For instance, the declaration of Figure 5.12,

```
state_Alice: agent * agent * public_key * public_key * set
           * nat * text * text * public_key * nat -> fact
```

defines the signature of `state_Alice` as a 9-ary fact where the first two arguments are agents, the third a public key and so on.

Section Types. In this section, the types for all constants and variables can be specified. This implies that throughout an IF file an identifier cannot be used with two different types (while the scope of each variable is limited to the rule it appears in). For instance the declaration `A,B,S,a,b,s,i:agent` (see Figure 5.12) states that the variables `A`, `B`, and `S` as well as the constants `a`, `b`, `s`, and `i` are of type `agent`.

The sections `inits`, `rules`, and `intruder` describe a transition system, and section `goals` describes a goal (or attack) predicate on states.

Section Inits. In this section, we specify one or more initial states of the protocol, and thus consider several parallel runs of the protocol. For instance, section `inits` in Figure 5.12 defines a single initial state (`init1`) modelling *(i)* an intruder that initially knows itself, its public and private key, as well as the identities and the public keys of the others agents; *(ii)* a trusted server and two sessions in parallel: one between agents `a` and `b` and one in which agent `a` initiates a protocol run with the intruder `i`; and *(iii)* what each agent's key set contains at the beginning.

Section Rules. In this section, we specify the transition rules of the honest agents executing the protocol.

For the declaration of rules, we also use the following syntactic sugar. We assume that the `iknows` fact (`iknows(M)` means that the intruder

knows the message `M`) is persistent, in the sense that if an `iknows` fact holds in a state, then it holds in all successor states (i.e. the intruder never forgets messages). Therefore, if an `iknows` fact appears in the left-hand side (LHS) of a rule, it should also be contained in the rule's right-hand side (RHS). To simplify the rules, however, we do not write the `iknows` facts that already appeared in the LHS. In other words, in our rules `iknows` is *implicitly persistent*, and we interpret the rules as if every LHS `iknows` also appears in the RHS.[34]

Also, a rule allows for a list of existentially quantified variables. Their purpose is to introduce new constants like fresh nonces. Note that in IF v.1 we instead used a method of creating unique terms; this is, of course, still possible, but the new specification language does no longer prescribe a particular method to create new constants. In this way each analysis technique applied on the IF can decide how to deal with freshness.

It is worth pointing out that the rules of Figure 5.13 are generated by taking into account that all the communication channels are controlled by a DY intruder. In such a situation it is perfectly fine to assume that all the messages sent and received by honest agents are intercepted and sent by the intruder, respectively. For instance, the first rule (`step1`) of Figure 5.13 states that if `Alice` is in her initial state and she already possesses `Bob`'s public key in her key set, then the rewrite rule can be executed and in such a case, `Alice` generates a fresh value for the nonce `Na`, she update her state, she sends a message encrypted with `Bob`'s public key and containing the above nonce and her identity, and a witness fact is asserted (see paragraph on "Goals" in Section 5.1.2).

Section Goals. The goals are defined in terms of predicates on states and are conceptually not different from the LHS of rules (and we will define their semantics similarly); consequently, we allow also conditions and negative facts in the goals. For instance, the goal declared in Figure 5.12 expresses that an attack state is reached any time a request fact belongs to the state, while the correspondent witness fact does not.

Section Intruder. The rules in this section (see Figure 5.11) describe the abilities of the intruder, namely composition and decomposition of messages he knows (according to the standard Dolev-Yao intruder). As these abilities are again independent of the protocol, they are included in the prelude. Moreover, it is immediate to see that in order to change the intruder model, it is sufficient to change these rules.

[34] Such a syntactic assumption is perfectly fine unless a protocol designer would like to model an intruder that forgets information.

Context-sensitive Properties

All used identifiers must be different from the IF keywords (`step`, `section`, `intruder`, `equal`, `leq`, `not`, `state`). The identifiers for types used in declarations can only be those identifiers that have been introduced as type identifiers in the prelude (see section `typeSymbols` of Figure 5.11). Identifiers for operators are only those that have been declared in the signature section of the prelude as having range type message. Similarly, fact symbols are only the ones declared in the signature section of the prelude or the IF file as having range type fact. The identifiers that name initial states, rules, or goals must be unique and distinct from all constants and variables and declared identifiers.

For a rule declaration, the variables in the parameter list must contain exactly those variables that occur in the LHS of the rule and in the existential quantification. The variables of the RHS must be a subset of the variables in the positive facts of the LHS (excluding those variables that occur only in the conditions or the negative facts of the rule) and the existentially quantified variables. Analogous restrictions apply for initial states. More precisely, variables cannot occur in an initial state as it can be seen as the RHS of a rewrite rule that can be applied only once with an empty LHS.

Negation

We have extended the standard rewriting approach to consider rules that contain also negative facts and conditions. The conditions are conjunctions of equalities and inequalities on terms. Note that the truth value of a condition depends only on the substitution for the variables of the contained terms, while the truth value for positive and negative facts depends also on the current state (hence the conceptual distinction between facts and conditions).

The conditions can also contain comparisons between natural numbers, which are necessary for instance for protocols where agents loop according to a counter, but again their truth value depends only on the substitution for the variables of the contained terms and not on the current state.

Shared Variables and Sets

The analysis of security protocols and properties often requires us to model that an agent can remember a set of objects shared over all protocol sessions he participates in, e.g. the set of known public-key certificates of other agents, or the set of nonces the agent has seen. This can be readily modelled in the IF. For example, if `keyset_a` denotes the set of key certificates known to agent `a`, then the situation in which `keyset_a` contains a message `pair(b,kb)` can be represented by a state containing the fact `contains(keyset_a,pair(b,kb))`. This allows us to easily check if a certain

element is or is not contained in a set, as well as to add and to remove elements from a set. In Figure 5.12, Figure 5.13, and Figure 5.14, we describe how such a set can be used to give a precise model of the Needham-Schroeder public-key protocol with key-server, where the agents ask the key-server for a key iff they do not know it yet.

5.2.2 The Semantics

In this section, we formally describe the semantics of the IF. Recall that, as we remarked above, the translation performed by the HLPSL2IF translator defines a semantics for the HLPSL in terms of the IF, which provides an alternative to the semantics of HLPSL based on TLA (see Section 5.1.3).

The basis of the semantics are terms, which are built from the constants and function symbols of the prelude and the IF files. To smoothly integrate the existential quantifier, we assume a set of fresh constants that is disjoint from all constants in the prelude and if file. For these constants, we assume a function `fresh` that maps a state and a set of variables to a substitution that replaces the variables with constants that do not appear in the given state.

Types

Let *type* be a partial function that yields for every constant and variable the respective type that has been declared.

Note that our syntax allows also *compound* types (see Section 5.1.2), e.g.

$$\texttt{M} : \texttt{scrypt}(\texttt{symmetric\_key}, \texttt{pair}(\texttt{nonce}, \texttt{agent}))$$

Such a variable declaration is used when the receiver is not supposed to analyse a certain message-part according to the protocol. For instance, in the case of the Otway-Rees protocol, A should send to B a message M that is encrypted with a key K_{AS} that is shared between A and a trusted server S. B has to forward this message M to S and cannot read it himself. Hence an intruder, impersonating A, can send any message in the place of M since B will not try to analyse it. For a *typed* model, however, we want B to accept M only if it is of the proper format (according to the protocol), i.e. if it is an encryption with a symmetric key and the contents after decryption are also of the proper format. In other words, even though B cannot decrypt the message, we assume that he can check whether the received message is of the correct type and reject it if not.

Semantically, let *op* be an n-ary IF operator, `M` a variable and $t_1, \ldots, t_n$ types (atomic or themselves composed). Then the declaration

$$\texttt{M} : op(t_1, ..., t_n)$$

is equivalent to the declarations

$$\mathtt{M_1} : t_1, \ldots, \mathtt{M_n} : t_n$$

if $\mathtt{M}_i$ (with $i = 1, \ldots, n$) are fresh variables (that do not appear in the IF file) and every occurrence of `M` in the IF file is replaced with the term $op(\mathtt{M_1}, \ldots, \mathtt{M_n})$.

One may hence see composed types as syntactic sugar, but they allow us to write the rules for an IF file independent of the question of typing, so that the same IF specification can be analysed with respect to both the typed and the untyped model simply considering or not the signature and types sections.

Unification

We define unification on IF terms in the standard way, only that types have to be respected.[35] Formally, a unifier of two terms is a substitution, such that the type of every substituted variable agrees with the type of the term it is replaced with. (In an untyped model, the types are not considered and hence do not constrain the unification.) As we adopt the standard notion of sorted unification, we will not go into further details here but refer the reader to [BN98].

We use the "." as an associative, commutative, and idempotent operator, i.e. we have:

$$t_1.(t_2.t_3) = (t_1.t_2).t_3$$
$$t_1.t_2 = t_2.t_1$$
$$t.t = t$$

Note, however, that these three properties work only on facts and not on messages. With these properties, the operator "." works as a set constructor for facts, and in the following we will consequently talk about sets, union, and set difference for facts as a shorthand.

Rule Application

Let us denote an IF rewrite rule by means of the triple $\langle l, exVar, r \rangle$, where l is the LHS of the rule, r is the RHS of the rule, and $exVar$ is the list of existentially quantified variables.

A LHS of a rule contains a set of positive and negative facts as well as a set of conditions, i.e. a set of equalities and inequalities as follows: the IF condition $\mathtt{equal}(t_1, t_2)$ represents equality of the terms t_1 and t_2, `not`

[35] In the sequel we assume that all states are ground terms (i.e. terms that do not contain variables) and hence we employ a special case of unification, *matching*, i.e. unification of two terms one of which is ground.

represents negation of a condition, and $\leq (t_1, t_2)$ represents $t_1 \leq t_2$. For a substitution σ, we define $\sigma \models \mathit{Cond}$ on conditions as expected:

$$\begin{array}{lll}
\sigma \models t_1 = t_2 & \text{iff} & t_1\sigma = t_2\sigma \text{ (where } t_1 \text{ and } t_2 \text{ are arbitrary terms)} \\
\sigma \models t_1 \leq t_2 & \text{iff} & t_1\sigma \leq t_2\sigma \text{ (where } t_1 \text{ and } t_2 \text{ are natural numbers)} \\
\sigma \models \phi \wedge \psi & \text{iff} & \sigma \models \phi \text{ and } \sigma \models \psi \text{ (where } \phi \text{ and } \psi \text{ are conditions)} \\
\sigma \models \neg\phi & \text{iff} & \text{not } \sigma \models \phi \text{ (where } \phi \text{ is a condition)}
\end{array}$$

For the LHS l of a rule, we define the functions $PF(l)$ for the positive facts, $NF(l)$ for the negative facts, $PC(l)$ for the positive conditions (i.e. without not), and $NC(l)$ for the negative conditions.

Figure 5.15 defines when a rule is applicable to a (ground) state by the function *matches* that takes as argument the LHS of the rule l and yields a function that maps a state s to the set of substitutions σ such that $l\sigma$ can be applied to s (this set is empty if the rule is not applicable).[36] In there, the predicate *ground* checks that a given substitution is ground, the function *dom* returns the domain of a given substitution, and the function v returns the variables occurring in a given term. The intuition behind this definition

$$\begin{aligned}
\mathit{matches} \;&:\; \mathit{Rule_LHS} \to (\mathit{State} \to 2^{\mathit{Substitution}}) \\
\mathit{matches}\ l\ s \;&=\; \{\sigma \mid \mathit{ground}(\sigma), && (5.16)\\
&\qquad \mathit{dom}(\sigma) = v(PF(l)) \cup v(PC(l)), \\
&\qquad PF(l)\sigma \subseteq s,\ \ \sigma \models PC(l), && (5.17)\\
&\qquad \forall \rho.\, \mathit{dom}(\rho) = (v(NF(l)) \cup v(NC(l))) \setminus \mathit{dom}(\sigma), && (5.18)\\
&\qquad\quad NF(l)\sigma\rho \cap s = \emptyset,\ \sigma\rho \models NC(l)\} && (5.19)
\end{aligned}$$

Figure 5.15: Applicability of a IF rewrite rule

is as follows: we consider every (ground) substitution σ (see (5.16)) such that under σ the positive facts can be unified with a subset of the current state (hence $PF(l)\sigma$ is necessarily ground) and the positive conditions are satisfied (see (5.17)). Furthermore, for all ground substitutions ρ for the remaining variables, i.e. those variables that appear only in negative facts and in negative conditions (see (5.18)), we postulate that none of the negative facts under $\sigma\rho$ is contained in the state and none of the conditions is satisfied for $\sigma\rho$ (see (5.19)). We recall that the RHS of the rule can only contain variables from the positive facts of the LHS and the existentially quantified variables (which will be replaced by fresh constants below), therefore all substitutions that result from *matches* are ground and so are all successor states. Note also that *matches* is applied in the same way for goal states (which are syntactically the same as a rule's LHS).

[36] Note that 2^S denotes the power-set of a set S.

Figure 5.16 describes the semantics of a rule as a state-transition function. In there we use the applicability check *matches*. Besides for this check, the conditions and the negative facts of the rule do not play any role: the transition itself is concerned only with the positive *facts* of the LHS of the rule, the existentially quantified variables, and the RHS. Intuitively, if the

$$
\begin{aligned}
[\![\cdot]\!] &: Rule \rightarrow (State \rightarrow 2^{State}) \\
[\![\langle l, exVar, r\rangle]\!](s) &= \{s' \mid \exists \sigma, \rho.\ \sigma \in matches\ l\ s, && (5.20)\\
&\quad \rho = \mathtt{fresh}(s, exVar), && (5.21)\\
&\quad s' = (s \setminus PF(l)\sigma) \cup r\rho\sigma\} && (5.22)
\end{aligned}
$$

Figure 5.16: Semantics of a IF rewrite rule

rewrite rule is applicable to state s (see (5.20)), then its application leads to a state s' obtained from s by removing the facts in the LHS of the rule and by adding those in the RHS to the result (see (5.21)), where a different fresh constant is generated for any existentially quantified variable of the rule (see (5.22)). Note that here the semantics of a rule is defined as a state-transition function operating only on ground terms, i.e. s cannot contain variables (otherwise the definition of the transition relation may not behave as one would expect); the resulting s' is then also ground, as the rules cannot introduce any new variables.

The rest of the semantics is straightforward: we have one or more ground initial states and a transition relation; this defines an state transition system. A protocol, described by an IF file, is secure iff there are no reachable state s and goal g such that *matches* g s holds.

5.3 Experimental Modelling Results

Both the versions of HLPSL and IF have been thoroughly and successfully experimented. Namely, the first version of the languages has been tested against the well-known Clark-Jacob's library (CJ, [CJ97]). It turns out that 46 of 51 (a coverage of 90%) of the protocols presented in the CJ are specifiable in HLPSL v. 1 and automatically translatable into IF v. 1 to be promptly analysed by means of automated reasoning tools [ABB+02, AVI02b].

As far as the second version of HLPSL and IF is concerned, we have selected, in the context of the AVISPA project, a substantial set of security problems[37] associated with practically relevant, industrial protocols that

[37] A security problem is given by both a protocol and a security property the protocol should satisfy.

have recently been or are currently being standardised by organisations like the Internet Engineering Task Force IETF. (We presented these protocols at an Open Security Area Directorate meeting of the IETF, `www.ietf.org/proceedings/04mar/198.htm`, where the relevance and representativeness of this collection was confirmed.) We have then formalised in HLPSL v. 2 a large subset of these protocols, and the result of this specification effort is the *AVISPA Library* (publicly available at the AVISPA web-page, `www.avispa-project.org`), which at present comprises 215 security problems derived from 48 protocols. Table 5.1 shows an excerpt of the AVISPA library, namely all those security protocols that work under the free-algebra assumption. For each protocol, we indicate the group it belongs to (according to the classification described in [AVI03c]), references to the relevant literature where a description of the protocol can be found, and the number of secrecy, weak and strong authentication properties that we have formalised for the protocol. When the number in the last three columns is different from 1, then we refer to the various authentication and secrecy problems that arise by distinguishing several authentication and secrecy properties, namely on different data or between different roles, where we split mutual authentication into unilateral authentication properties.

Finally, it must be noted that all the protocol in the AVISPA library have been automatically translated into IF in order to be analysed by means of automated reasoning tools [ABB$^+$05]. The reported results confirm the suitability of the aforementioned languages in supporting protocol designers in the activity of specifying industrial-scale security protocols.

Table 5.1: HLPSL coverage

Protocol			Property		
Name	**Group**	**Reference**	**S**	**WA**	**SA**
UMTS-AKA	3GPP	[AH03]	2		2
ISO-PK1	ISO	[ISO97]			1
ISO-PK2	ISO	[ISO97]			1
ISO-PK3	ISO	[ISO97]		2	
ISO-PK4	ISO	[ISO97]			2
CHAPv2	ppp-wg	[Zor00]	2		2
EKE	PAKE	[BM92]	2		2
TLS	TLS	[DA99]	2		2
LPD-MSR	LPD	[BM98]	1	1	
LPD-IMSR	LPD	[BM98]	1	1	
Kerb-basic	krb-wg	[NYHR04]	6	7	
Kerb-Cross-Realm	krb-wg	[NYHR04]	11	2	5
Kerb-Ticket-Cache	krb-wg	[NYHR04]	6		5
Kerb-Forwardable	krb-wg	[NYHR04]	7		5
Kerb-PreAuth	krb-wg	[Har04]	6		6
Kerb-PKINIT	krb-wg	[TLHM04]	6		6
CRAM-MD5	challenge-response	[KCK97]	1		1
PBK	ipv6	[BMS03]			1
PBK-fixed	ipv6	[BMS03]			1
PBK-fix-weak-auth	ipv6	[BMS03]		1	
DHCP-delayed-auth	DHC	[DA01]	1		1
TSIG	DNSext	[Eas00, Wel00]		2	
ASW	Payment	[ASW98, HDM04]	1		2
ASW-abort	Payment	[ASW98, HDM04]	2		2
FairZG	Payment	[ZG96]			5
SET-purchase	E-Commerce	[SET, BMP01]	2	1	1
SET-p.-hon.-payment-gw	E-Commerce	[SET, BMP01]	2	1	1
AAAMobileIP	mobileip-wg	[CLG+03]	3	6	
Simple	impp and simple	[JPW02, RSC+02]	1	2	
CTP-non_predictive-fix	seamoby	[BLMTM04]	1		2
geopriv	Geopriv	[CMM+04]	3	1	1
pervasive	Geopriv	[CMM+04]	1		1
two_pseudonyms	Geopriv	[CMM+04]	4		1
QoS-NSLP	NSIS	[McD03]		2	
sip	SIP	[GMBPL+05, FHBH+99]			1
		Total	74	29	60
		Grand total		163	

Legenda:

S : secrecy property
WA : weak authentication property
SA : strong authentication property

Chapter 6

Implementing a SAT-based Model-Checker: SATMC

The ideas described in this Thesis have been implemented in SATMC [AC04b], a SAT-based Model Checker for automatic analysis of security protocols.[1]

Given a positive integer k and a protocol insecurity problem specified in the IF language,[2] SATMC automatically generates a propositional formula by using sophisticated encoding techniques developed for planning. State-of-the-art SAT-solvers taken off-the-shelf are then used to check the propositional formula for satisfiability and any model found by the solver is turned into a partially ordered set of transitions of depth k whose linearizations correspond to attacks on the protocol.[3] If the formula is found to be unsatisfiable, then k is incremented and the whole procedure is iterated until either a termination condition is met or a given upper bound for k is reached.[4]

SATMC has been developed in SICStus Prolog, a declarative and expressive programming language easy to use and particularly suited for rapid prototyping though offering a reasonable degree of efficiency. The use of Prolog unification and non-determinism results in a concise program, that is easy to read and simply to debug.

In implementing SATMC, we have given a great deal of care on design issues related to flexibility, modularity, and efficiency. The result of such an effort is an open and flexible platform for SAT-based bounded model checking of security protocols. Improvements of SAT technology can be readily exploited by integrating in a plug and play manner state-of-the-art SAT-solvers (e.g. the best performers in the SAT competition, [SAT]). Similarly,

[1] The tool is available at `http://www.ai.dist.unige.it/satmc`

[2] Notice that both IF v.1 and IF v.2 are supported by SATMC.

[3] We recall that, a partially ordered set is a directed acyclic graph. The depth of a partially ordered set is the largest number of backward nodes on any path of the graph.

[4] Notice that the user may specify an infinite upper bound. In that case the procedure is not guaranteed to terminate.

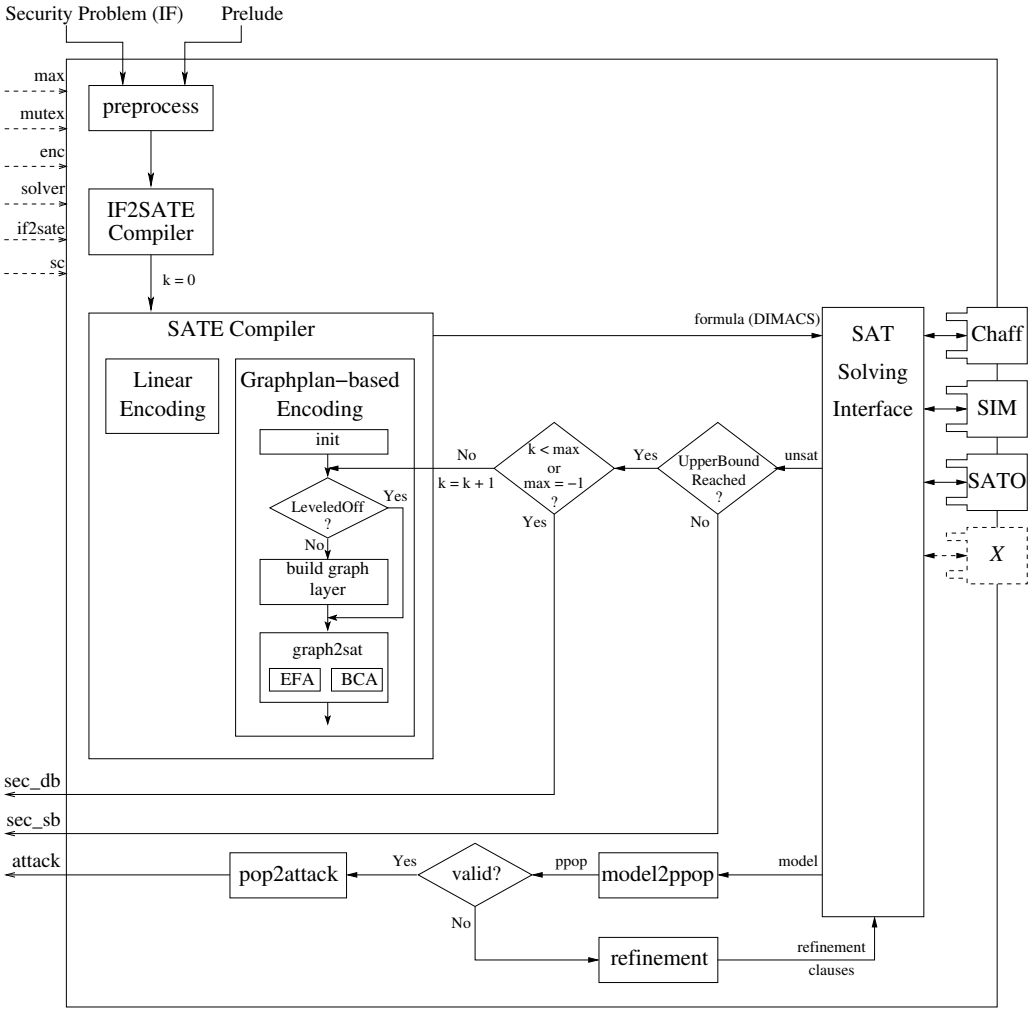

Figure 6.1: Architecture of SATMC

advancements and novelties in AI planning SAT-reduction techniques can be promptly implemented in SATMC.

In the rest of this chapter we start presenting in Section 6.1 a detailed view of the SATMC architecture and of its interface. Section 6.2 is devoted to the description of the SATE language for the specification of planning problems. We conclude in Section 6.3 by describing how to invoke SATMC.

6.1 The Architecture

The architecture of SATMC is shown in Figure 6.1. SATMC takes as input an IF specification,[5] and the following parameters:

[5]Notice that both IF v.1 and IF v.2 are supported by SATMC.

- `max`: maximum depth of the search space up to which SATMC will explore (the parameter `max` can be set to `-1` meaning *infinite*, but in this case the procedure is not guaranteed to terminate); usually an upper bound of ten steps suffices to discover attacks on protocols.
- `mutex`: level of the mutexes to be used during the SAT-reduction; if set to `0`, then the abstraction/refinement strategy provided by SATMC is enabled; otherwise the abstraction/refinement strategy is disabled and the following hold:
  - if set to `1`, then only the static mutexes are taken into account;
  - if set to `2`, then both static and dynamic mutexes are computed.
- `enc`: the selected SAT reduction encoding technique (see Section 4.2); it can be set to either:
  - `linear`: the linear encoding is applied;
  - `gp-bca`: the graphplan-based encoding using the backward chaining (BCA) schema is selected; or
  - `gp-efa`: the graphplan-based encoding using the explanatory frame (EFA) schema is applied.
- `solver`: the selected state-of-the-art SAT solver (Chaff [MMZ+01], SIM[GMTZ01], and SATO [Zha97] are currently supported); it ranges over the values `chaff`, `sim`, and `sato`.
- `if2sate`: the version of the IF specification given in input so to determine what version of the IF2SATE compiler will be used; it can be set either to 1 or 2.
- `sc`: a boolean parameter stating whether the step-compression optimisation (cf. Section 3.2.4) is enabled or not.

SATMC then returns either:

- `attack`: stating that an attack has been discovered and is returned to the user in a user-friendly format that recast the well-known Alice&Bob notation;
- `sec_db`: standing for "`sec`ure up to the `d`epth `b`ound" and stating that no attack has been found on the input protocol insecurity problem after having explored up to *max* depth levels of the search space; or
- `sec_sb`: standing for "`sec`ure up to the `s`ession `b`ound" and meaning that the input protocol insecurity problem does not suffer from any attack in the scenario analysed (i.e., a bounded number of sessions in which each role exchanges a finite sequence of messages) and under

the assumptions of perfect cryptography, strong typing, and intruder based on the Dolev-Yao model.

In more detail, SATMC starts preprocessing the IF specification in order to check, besides others, whether the decidability of the input protocol insecurity problem can be established or not. This means to test if the conditions introduced in Section 3.4 are satisfied by the protocol insecurity problem. In case they are, then the protocol insecurity problem is guaranteed to be decidable and the SATMC procedure is ensured to terminate independently from the encoding technique selected.

After that, SATMC first compiles the IF specification into a SATE specification representing a planning problem and then applies the selected encoding technique to this one for increasing values of k. The propositional formula generated at each step, say Φ_k, is fed to the selected SAT solver in the standard DIMACS format.[6]

As soon as the formula is found satisfiable, the corresponding model is translated back into a "pseudo" partial-order plan (*ppop*).[7] If the abstraction/refinement strategy is disabled (i.e. `mutex` > 0), then we are guaranteed that any pseudo partial-order plan corresponds to a partial-order plan. Otherwise, as explained in Section 4.3.2, the pseudo partial-order plan has to be checked and if it corresponds to a spurious counterexample, a refinement phase is applied on the current propositional formula that is fed back to the SAT-solver. The whole procedure is iterated until a non-spurious counterexample is met or the formula becomes unsatisfiable. As soon as a partial-order plan is found, it is translated into attacks which are reported to the user in a user-friendly format in the Alice&Bob notation (see Figure 6.3).

Let us now consider the situation in which the procedure has not discovered a partial-order plan at the step k.[8] If the upper-bound that guarantees that enough levels of the search space has been explored is reached (cf. termination paragraph in Section 4.3.1),[9] then SATMC returns `sec_sb` stating that in the bounded scenario analysed the input protocol insecurity problem does not suffer from any attack. Otherwise, it is checked whether the maximum number of steps (`max`) imposed by the user has been already explored. If yes, then SATMC returns `sec_db` meaning that even if the input protocol insecurity problem has not been verified with respect to its entire scenario,

[6]Notice that SAT Solvers are currently used as black-boxes. This allows for a plug and play integration of state-of-the-art SAT-solvers provided that two predicates, one for invoking the SAT solver and another one for parsing its output, are implemented.

[7]We recall that a partial-order plan is a partially ordered set of planning actions that concisely represents a set of plans.

[8]This means that either the propositional formula generated at step k was unsatisfiable or the abstraction/refinement strategy concluded without discovering any non-spurious counterexample.

[9]It must be noted that if the preprocess module fails in establishing the decidability of the input protocol insecurity problem, then it is not possible to compute this upper-bound and this test will be simply neglected.

the protocol does not suffer from any flaw after having explored `max` levels of the search space. Otherwise k is incremented and the SATE compiler firstly invokes the selected encoding technique for computing both $B(\mathbf{f}^{k+1})$ and $T_k(\mathbf{f}^k, \mathbf{a}^k, \mathbf{f}^{k+1})$, and secondly it assembles the propositional formula Φ_{k+1} as Φ_k where the sub-formula $B(\mathbf{f}^k)$ is removed and $B(\mathbf{f}^{k+1}) \wedge T_k(\mathbf{f}^k, \mathbf{a}^k, \mathbf{f}^{k+1})$ is appended. The new formula is then fed to the SAT-solver and the whole procedure is repeated.

6.2 The SATE Language

To express planning problems we have designed a STRIPS-like specification language for planning problems called SATE. The motivation for the design of a new language is that the existing planning languages (most notably PDDL, [GHK+98]) are unable to handle complex term structure (only individual symbols allowed) and this makes them unsuited to our task (see Example 18 in Section 8.1).

The syntax of the SATE language is given by the BNF grammar of Figure 6.2. Here is a description of the main constructs of the language:

- **Sort declaration** (*sort_decl*). For example, the declarations `sort(fluent)`, `sort(action)`, and `sort(time)` declare `fluent`, `action`, and `time` to be sorts.

- **Super-Sort declaration** (*super_sort_decl*). For example, the declarations `super_sort(atom,mr)` and `super_sort(atom,pk)` declare `atom` to be super-sort of `mr` and `pk`.

- **Constant declaration** (*constant_decl*). For example, the constant declarations `constant(0,time)` and `constant(s(time),time)` declare `0` to be an individual constant of sort `time` and `s` to be a function symbol of arity `time` $\rightarrow$ `time`.

- **Static** (*static*). A declaration of the form `static(`P`)` states that all ground instances of P are static fluents i.e. their initial truth value cannot possibly change. Formally, if p is a ground fluent instance of P, then $p^0 \equiv p^i$ for $i = 1, \ldots, k$.

- **Monotone** (*monotone*). A declaration of the form `monotone(`P`)` states that all ground instances of P can change their truth value from false to true, but not vice versa. Formally, if p is a ground fluent instance of P, then $p^i \supset p^{i+1}$ for $i = 0, \ldots, n-1$.

- **Initial State** (*initial_state*). This statement specifies the initial states. For example, the declaration

sate_spec	::=	*(assertion .)**
assertion	::=	*sort_decl* | *super_sort_decl* | *constant_decl* |
		monotone | *static*
		initial_state | *goal_state* | *operator*
sort_decl	::=	`sort(`*sort_id*`)`
super_sort_decl	::=	`super_sort(`*sort_id*`,`*sort_id*`)`
constant_decl	::=	`constant(`*arity*`,`*sort_id*`)`
monotone	::=	`monotone(`*fluent*`)`
static	::=	`static(`*fluent*`)`
initial_state	::=	`facts(`*init*`)`
goal_state	::=	`goal([`*cond**`],`*goal*`)` | `goal(`*goal_id*`,[`*cond**`],`*goal*`)`
operator	::=	`action(`*action*`,[`*cond**`],`*pre*`,`*add*`,`*del*`)`
arity	::=	*const_id* | *const_id*`(`*sort_id**`)`
pre, add, del	::=	`[`*fluent**`]`
init, goal	::=	`[`*wff**`]`
wff	::=	*fluent* | `~` *wff* | *wff* `&` *wff* | *wff* `v` *wff*
fluent, action, goal_id	::=	*const_id* | *const_id*`(`*term**`)`
cond	::=	*term* `=` *term* | *term* `\=` *term*
term	::=	*var* | *const_id* | *const_id*`(`*term**`)`
sort_id, const_id	::=	`[a-z,A-Z][_,a-z,A-Z,0-9]`*
var	::=	`[_,A-Z][_,a-z,A-Z,0-9]`*

Proviso:

- in the production rule for *operator*: $Vars([cond^\star])$, $Vars(prec)$, $Vars(add)$, $Vars(del) \subseteq Vars(action)$
- in the production rule for *goal*: $Vars([cond^\star])$, $Vars(goal) \subseteq Vars(goal\_id)$

Legend:

- $X^\star$ abbreviates $(X(,X)^*)$, and
- $X_1, \ldots, X_n ::= E$ abbreviates $X_1 ::= E \ldots X_n ::= E$.

Figure 6.2: Grammar for the SATE specification language

```
facts([state(0,a,a,[a,i,ka,ka^-1,ki],1),
       state(0,a,a,[a,b,ka,ka^-1,kb],2),
       state(1,a,b,[b,a,kb,kb^-1,ka],2),
       ik(i),ik(a),ik(b),ik(ki),ik(inv(ki)),ik(ka),ik(kb)
```

asserts that the listed fluents hold at time 0 and all the other fluents does not hold at time 0 (*close-world-assumption*).

- **Goal** (*goal_state*). For example, the assertion

```
goal(auth_flaw(X,Y),true,
     [~(witness(a,b,X,Y)),request(b,a,X,Y)]).
```

declares that every state S, s.t. `request(b,a,X`σ`,Y`σ`)` is in S and `witness(mr(a),mr(b),X`σ`,Y`σ`)` is not in S for some grounding substitution σ, is a goal state.

- **Operator** (*operator*). For example, the assertion

```
action(step_1(A,B,Ka,Kb,S),
        true,
        [state(0,A,A,[A,B,Ka,inv(Ka),Kb],S),
         fresh(ft(A,1,1,S))],
        [state(2,B,A,[ft(A,1,1,S),A,B,Ka,inv(Ka),Kb],S),
         msg(1,A,B,crypt(Kb,pair(A,ft(A,1,1,S)))),
         witness(A,B,na,ft(A,1,1,S))
        [state(0,A,A,[A,B,Ka,inv(Ka),Kb],S),
         fresh(ft(A,1,1,S))]).
```

states that if S is a state and σ a grounding substitution such that `state(0,A,A,[A,B,Ka,inv(Ka),Kb],S)`σ and `fresh(ft(A,1,1,S))`σ are in S, then the state obtained from S by removing `state(0,A,A,[A,B,Ka,inv(Ka),Kb],S)`σ and `fresh(ft(A,1,1,S))`σ and by adding `state(2,B,A,[ft(A,1,1,S),A,B,Ka,inv(Ka),Kb],S)`σ, `msg(1,A,B,crypt(Kb,pair(A,ft(A,1,1,S))))`σ, and `witness(A,B,na,ft(A,1,1,S))`σ is a successor state of S reachable by applying the action `step_1(A,B,Ka,Kb,S)`σ.

6.3 Running SATMC

SATMC can be invoked by typing at the command-line

```
satmc <filename> [--max=<number>] [--mutex=<number>]
                 [--enc=<encoding>] [--solver=<solver>]
                 [--if2sate=<if2sate>] [--sc=<sc>]
                 [--prelude=<fileprelude>]
```

where `<filename>` is the IF problem to be analysed, and each option is as described in Section 6.1.[10]

Figure 6.3 displays the output returned by SATMC when run against the NSPK protocol (see Section 3.1.2) through the following command:

```
satmc NSPK.if --if2sate=2 --enc=gp-efa --mutex=1 --solver=chaff
```

The output indicates that an authentication flaw has been discovered on the NSPK protocol and the corresponding attack trace is reported to the user in the Alice&Bob notation. Moreover some interesting statistics are given. For instance, the attack has been found by the `chaff` solver in a negligible amount time at step 8 on a propositional formula with 670 atoms and 1936 clauses whose generation has required 0.41 seconds.

It is worth pointing out that SATMC is also one of the back-ends of the AVISPA tool [AVI02a]. This platform offers two user interaction modes: a web interface available on-line at `www.avispa-project.org` and a package that can be downloaded and easily installed on the user machine. The web interface is a graphical front-end to get the feeling of AVISPA and thus of SATMC. A nice feature of the web interface is that it displays attacks in a graphical format, using Message Sequence Charts. The package is suited for advanced use and it features an xemacs mode for an effective interaction with AVISPA and with SATMC as well. Figure 6.4 depicts a screen-shot of the user interaction modes: namely it shows the web interface (background), part of the specification of the NSPK protocol in the xemacs mode (top-right window), and the message sequence chart corresponding to the attack that SATMC has found (bottom window).

[10] Notice that the prelude file may be needed only when a IF v.2 input problem is considered.

```
SUMMARY
  UNSAFE

DETAILS
  ATTACK_FOUND
  STRONGLY_TYPED_MODEL
  BOUNDED_NUMBER_OF_SESSIONS
  BOUNDED_SEARCH_DEPTH
  BOUNDED_MESSAGE_DEPTH

PROTOCOL
  NSPK.if

GOAL
  authentication_on_bob_alice_na(b,a,na0(a,6),4)

BACKEND
  SATMC

COMMENTS

STATISTICS
  attackFound               true      boolean
  stopConditionReached      false     boolean
  graphLeveledOff           no        boolean
  satSolver                 chaff     solver
  maxStepsNumber            30        steps
  stepsNumber               8         steps
  atomsNumber               670       atoms
  clausesNumber             1936      clauses
  encodingTime              0.41      seconds
  solvingTime               0.01      seconds
  if2sateCompilationTime    0.09      seconds

ATTACK TRACE
  i       -> (a,6) : start
  (a,6)   -> i : {na0(a,6).a}_ki
  i       -> (b,4) : {na0(a,6).a}_kb
  i       -> (b,4) : {na0(a,6).a}_kb
  (b,4)   -> i : {na0(a,6).nb0(b,4)}_ka
  i       -> (a,6) : {na0(a,6).nb0(b,4)}_ka
  (a,6)   -> i : {nb0(b,4)}_ki
  i       -> (b,4) : {nb0(b,4)}_kb
  i       -> (b,4) : {nb0(b,4)}_kb
```

Figure 6.3: Output generated by SATMC

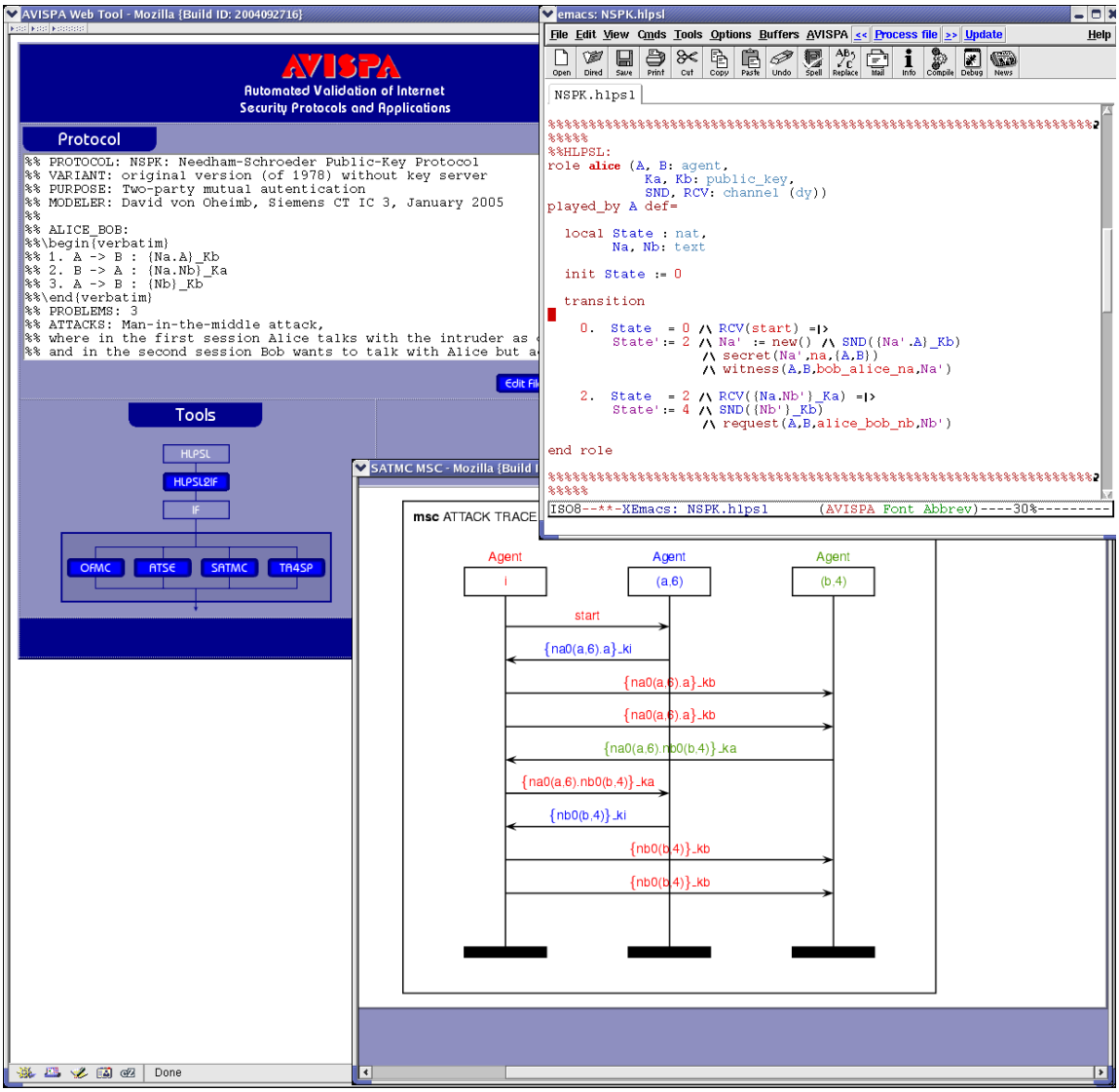

Figure 6.4: A screen-shot of SATMC within the AVISPA Tool

Chapter 7

Experimental Results

In this chapter we present a large amount of experimental results that on the one hand provide a clear comparison between the various encoding techniques implemented in SATMC, and on the other hand demonstrate the effectiveness of our SAT-based Model-Checking approach with respect to other automated reasoning techniques.

This experimental analysis has been carried out by considering *(i)* a large number of security protocols drawn from the well-known Clark-Jacob's library [CJ97], and *(ii)* a selection of practically relevant, industrial protocols borrowed from the AVISPA Library (see Section 5.3).

7.1 Comparison of the encodings in SATMC

We have thoroughly experimented the encoding techniques available in our tool against a selection of (flawed) security protocols drawn from the Clark-Jacob library [CJ97]. For each protocol we have built a corresponding protocol insecurity problem modelling a scenario with a bounded number of sessions (usually two sessions) in which the involved principals exchange a finite sequence of messages on a channel controlled by the most general intruder based on the Dolev-Yao model. More concretely, protocols and their security requirements have been specified in HLPSL v.1 and automatically translated into IF v.1 by means of the HLPSL2IF v.1 translator (see Chapter 5). We have then analysed all the IF v.1 specifications with SATMC for each one of the encoding techniques implemented in it (see Chapter 6).

The results obtained with the linear and graphplan-based encoding techniques are reported in Section 7.1.1 and Section 7.1.2 respectively, while Section 7.1.3 is dedicated to their comparison.

7.1.1 Linear Encodings

In this section we report on the results obtained by running SATMC with each one of the linear encoding techniques described in Section 4.2.1 against a selection of security protocols drawn from the Clark-Jacob's library.

The results of our experiments are reported in Table 7.1. Each protocol has been analysed by applying one of the following encoding techniques:[1]

CEA: standard linear encoding with generation of the CEA enabled,

BITWISE: bitwise linear encoding, and

NoCEA: standard linear encoding with generation of the CEA disabled and therefore combined with the Abstraction/Refinement strategy.

For each protocol we give the number of fluents (**F**) and of actions (**Acts**) in the correspondent planning problem,[2] and for each encoding technique we give the smallest value of k at which the attack is found (**K**), the number of propositional variables (**A**) and clauses (**CL**) in the SAT formula (in thousands), the time spent to generate the SAT formula (**EncT**), the time spent by SAT solver to solve the last SAT formula (**Last**), and the total time spent by the SAT solver to solve all the SAT formulae generated for that protocol (**Tot**).[3] In the context of **NoCEA** a comparison between the SAT solvers Chaff and SIM has been performed and the solving times are shown together with the number of iterations of the Abstraction/Refinement strategy (#).

When the **CEA** encoding technique is applied, encoding time largely dominates solving time and this is strictly related to the size of the SAT instances generated. Both the size and the time spent to generate the SAT formulae drop significantly if **BITWISE** is used. This enabled us to encode and find attacks on protocols (e.g. *Andrew*, *KaoChow 2*, *KaoChow 3*, and *Woo-Lam M*) that could not be solved by using the **CEA** encoding technique. However, finding an attack using **BITWISE** can require more iterative-deepening steps (cf. the **K** columns) than those required by applying the standard linear encoding. As a consequence on some instances, such as the *NSCK* and the *SPLICE* protocols, **CEA** performance better than **BITWISE** both in terms of the size the SAT formulae and in encoding time. Moreover, the addition of the bitwise axioms can significantly increase the solving time (see, e.g., *KaoChow 2* and *KaoChow 3* protocols).

To cope with the above issues it is convenient to apply the **NoCEA** encoding technique that is based on the *Abstraction/Refinement strategy* described in Section 4.3.2. Of course, by disabling the generation of the

[1] Experiments have been carried out on a PC with a 1.4 GHz CPU and 1 GB of RAM.

[2] It must be noted that the numbers of fluents and actions provide a good estimation of the numbers of facts and rewrite rules characterising the protocol insecurity problem.

[3] Times are measured in seconds.

Table 7.1: Experimental data on protocols from the Clark/Jacob library using Linear encodings

			CEA						**BITWISE**						**NoCEA**									
Protocol	**F**	**Acts**	**K**	**A** (K)	**CL** (K)	**EncT**	**Chaff** **Last**	**Chaff** **Tot**	**K**	**A** (K)	**CL** (K)	**EncT**	**Chaff** **Last**	**Chaff** **Tot**	**K**	**A** (K)	**CL** (K)	**EncT**	**Chaff** **Last**	**Chaff** **Tot**	**Chaff** #	**SIM** **Last**	**SIM** **Tot**	**SIM** #
Andrew	465	15,650	9	145	MO	MO	MO	MO	11	178	5,348	220.8	4.6	21.6	9	145	2,256	111.4	2.0	12.1	1	1.7	6.2	0
EKE	1,148	10,924	5	62	13,949	7,100	7.6	19.7	5	62	1,630	113.1	1.0	3.1	5	62	783	74.1	0.7	3.7	2	0.0	0.9	0
ISO-CCF-1 U	39	22	4	< 1	< 1	0.1	0.0	0.0	4	< 1	1	0.1	0.0	0.0	4	< 1	< 1	0.1	0.0	0.0	0	0.0	0.0	0
ISO-CCF-2 M	81	132	4	< 1	6	0.3	0.0	0.1	4	< 1	10	0.5	0.0	0.0	4	< 1	4	0.2	0.0	0.1	0	0.0	0.0	0
ISO-PK-1 U	61	49	4	< 1	2	0.1	0.0	0.0	4	< 1	3	0.2	0.0	0.0	4	< 1	2	0.1	0.0	0.0	0	0.0	0.0	0
ISO-PK-2 M	125	279	4	2	17	0.9	0.0	0.0	4	2	22	1.1	0.0	0.0	4	2	10	0.6	0.1	0.1	0	0.0	0.1	0
ISO-SK-1 U	39	25	4	< 1	< 1	0.1	0.0	0.0	4	< 1	1	0.1	0.0	0.0	4	< 1	< 1	0.1	0.0	0.0	1	0.1	0.1	0
ISO-SK-2 M	100	87	4	< 1	3	0.4	0.0	0.0	4	< 1	6	0.5	0.0	0.0	4	< 1	3	0.4	0.0	0.0	0	0.0	0.0	0
KaoChow 1	2,753	2,042	7	36	355	18.4	0.1	0.7	8	41	345	15.8	0.9	1.6	7	36	131	8.0	0.1	0.7	4	0.2	0.6	0
KaoChow 2	32,963	22,344	9	531	35,178	1,494	MO	MO	11	642	6,244	309.7	180.9	473.1	9	531	1,804	140.4	1.6	15.6	5	9.2	26.4	0
KaoChow 3	49,378	55,727	9	995	MO	MO	MO	MO	11	1,206	17,528	1,009.9	303.4	1,063.5	9	995	5,737	585.5	7.5	41.6	1	28.5	127.6	3
NSCK	6,755	5,220	9	115	787	41.3	0.4	1.6	10	127	1,131	46.3	1.3	4.6	9	115	334	17.1	0.3	1.3	0	0.4	1.5	0
NSPK	429	662	7	7	51	2.3	0.1	0.1	7	7	83	3.4	0.1	0.2	7	7	33	1.5	0.1	0.1	0	0.1	0.1	0
NSPK-server	298	604	8	9	212	8.0	0.1	0.2	8	9	115	5.1	0.1	0.2	8	9	54	2.8	0.1	0.1	0	0.1	0.2	0
SPLICE	822	658	9	14	91	4.6	0.1	0.2	11	17	160	6.9	0.2	1.5	9	14	62	3.6	0.1	0.2	0	0.1	0.3	1
Swick 1	496	258	5	4	17	1.0	0.1	0.1	7	6	38	1.7	0.0	0.1	5	4	13	0.8	0.0	0.1	1	0.1	0.1	0
Swick 2	693	450	6	8	59	3.2	0.1	0.1	7	9	65	2.9	0.0	0.1	6	8	29	1.7	0.1	0.1	0	0.1	0.1	0
Swick 3	526	482	4	5	12	0.8	0.1	0.1	6	7	46	2.0	0.0	0.1	4	5	11	0.7	0.0	0.1	1	0.1	0.1	0
Swick 4	1,357	1,283	5	15	64	11.0	0.1	0.2	6	17	162	15.9	0.2	0.7	5	15	57	10.2	0.1	0.3	1	0.3	0.5	0
Stubblebine rep	1,409	2,408	3	13	2,048	82.9	0.3	0.6	3	13	194	10.8	0.1	0.2	3	13	95	6.3	0.1	0.1	1	0.1	0.1	0
Woo-Lam M	45,584	27,051	6	481	MO	MO	MO	MO	7	554	5,977	433.6	4.0	12.4	6	481	2,498	304.4	1.8	7.7	1	2.3	5.6	0

(K) indicates that the value in this column are given in thousands;
MO means that a memory out has been reached; and
< 1 means that the number of atoms or clauses is less than 1 thousand;

conflict exclusion axioms, we are no longer guaranteed that the solutions found are linearizable and hence executable.[4] SATMC therefore looks for conflicting (interfering) actions in the partial order plan found and extends the previously generated encoding with clauses negating the conflicts (if any). The resulting formula is then fed back to the SAT-solver and the whole procedure is iterated until a solution without conflicts is met or the formula becomes unsatisfiable.

The **NoCEA** encoding technique performs uniformly better on all the analysed protocols. Another interesting point is that while the time spent by Chaff and SIM to solve the propositional formulae is comparable in most cases (see the columns **Last** in the table **NoCEA**),[5] the number of iterations of the Abstraction/Refinement strategy is in most of the cases smaller when SIM is used. This is due to the different strategies implemented by the two solvers. Both solvers incrementally build a satisfying assignment, but while SIM terminates (possibly returning a partial assignment) as soon as all the clauses are subsumed by the current satisfying assignment, Chaff avoids the subsumption check for efficiency reasons and thus halts only when the assignment is total. As a consequence the partial-order plans found using Chaff usually contain more actions and thus are likely to have more conflicts (pairwise interfering actions). This explains why SIM is in many cases more effective than Chaff.

7.1.2 Graphplan-Based Encodings

In this section we report on the results obtained by running SATMC with each one of the graphplan-based encoding techniques described in Section 4.2.2 against a selection of security protocols drawn from the Clark-Jacob's library. We recall that we have implemented in SATMC two variants of the graphplan-based encoding: the first one—widely discussed in [KMS96]—uses the Backward Chaining Axioms (BCA schema), while the second exploits the Explanatory Frame Axioms (EFA schema). The results of our experiments are reported in Table 7.2. Each protocol has been analysed by applying one of the following encoding techniques:[6]

BCA with $\oplus = \oplus_a \cup \oplus_f$: graphplan-based encoding that uses the BCA schema and generates all the mutex relations (on fluents and actions),

BCA with $\oplus = \widehat{\oplus}$: graphplan-based encoding that uses the BCA schema and generates only the static mutexes over the actions,

EFA with $\oplus = \widehat{\oplus}$: graphplan-based encoding that uses the EFA schema and generates only the static mutexes over the actions.

[4] A counterexample in the model checking problem corresponds to a model of the propositional formula generated and, therefore, to solutions to the planning problem.

[5] Notice that Chaff performs better on big instances.

[6] Experiments have been carried out on a PC with a 1.4 GHz CPU and 1 GB of RAM.

Table 7.2: Experimental data on protocols from the Clark/Jacob library using Graphplan-based encodings

		BCA with $\oplus = \oplus_a \cup \oplus_f$				**BCA with $\oplus = \widehat{\oplus}$**				**EFA with $\oplus = \widehat{\oplus}$**			
Protocol	**K**	**A**	**CL**	**EncT**	**SolT**	**A**	**CL**	**EncT**	**SolT**	**A**	**CL**	**EncT**	**SolT**
Andrew	9	699	4,459	17.0	0.00	701	1,811	1.5	0.00	447	1,368	0.17	0.00
EKE	5	545	3,584	10.9	0.00	553	1,648	1.5	0.00	396	1,340	0.20	0.00
ISO-CCF-1 U	4	154	621	0.2	0.00	154	398	0.1	0.00	107	300	0.03	0.00
ISO-CCF-2 M	4	169	600	0.3	0.00	169	402	0.2	0.00	118	314	0.04	0.00
ISO-PK-1 U	4	202	794	0.5	0.00	202	494	0.2	0.00	154	423	0.06	0.00
ISO-PK-2 M	4	196	707	0.6	0.00	196	468	0.2	0.00	132	366	0.06	0.00
ISO-SK-1 U	4	142	561	0.2	0.00	142	367	0.1	0.00	98	270	0.02	0.00
ISO-SK-2 M	4	234	747	0.5	0.00	234	511	0.2	0.00	122	319	0.06	0.00
KaoChow 1	7	526	4,055	14.6	0.00	583	2,230	1.5	0.01	436	1,778	0.21	0.00
KaoChow 2	9	1,020	10,325	108.1	0.02	1,136	4,507	4.2	0.03	736	3,397	0.38	0.01
KaoChow 3	9	1,229	18,532	419.1	0.01	1,467	7,927	8.8	0.05	1,000	6,122	0.80	0.01
KLS rep.	7	2,018	62,836	3,557.4	0.03	2,092	31,510	23.5	0.07	1,658	23,219	5.86	0.03
NSCK	9	622	3,823	9.5	0.01	627	1,710	1.3	0.01	448	1,403	0.17	0.00
NSPK	7	559	3,129	6.7	0.00	572	1,555	1.0	0.00	423	1,276	0.14	0.00
NSPK-server	8	1,077	6,043	24.5	0.00	1,079	2,951	3.9	0.00	848	2,704	0.32	0.00
SPLICE	9	972	7,545	52.2	0.01	1,008	2,953	3.8	0.00	963	3,166	0.50	0.00
Swick 1	5	329	1,300	1.1	0.00	329	783	0.4	0.01	196	559	0.08	0.00
Swick 2	6	438	1,937	2.6	0.00	438	1,143	0.1	0.00	261	843	0.14	0.00
Swick 3	4	231	889	0.7	0.00	231	594	0.3	0.00	175	503	0.05	0.00
Swick 4	5	405	1,469	2.7	0.00	405	939	0.6	0.00	223	657	0.08	0.00
Stubblebine rep	3	220	791	0.7	0.00	222	608	0.3	0.00	147	481	0.07	0.01
Woo-Lam M	6	665	3,806	17.3	0.00	687	1,813	2.9	0.00	485	1,600	0.27	0.00

For each protocol we give the smallest value of k at which the attack is found (**K**) and for each encoding technique we give the number of propositional variables (**A**) and clauses (**CL**) in the SAT formula, the time spent to generate the SAT formulae (**EncT**) and the total time spent by the SAT solver to solve the SAT formulae (**SolT**).

The results obtained using the setting **BCA with** $\oplus = \oplus_a \cup \oplus_f$ based on the backward chaining axioms, show that the graphplan-based encoding performs well on a large set of examples, however, for bigger ones (see, e.g., *KLS rep.* protocol) the time spent at generating the formula (**EncT**) gets high. Moreover, as pointed out in Section 4.2.2, all the dynamic mutexes, except those imposed at layer 0 to restrict the over-approximation on the possible initial states, are logical consequence of static mutexes. This means that even if we restrict the computation of the mutex relation over action nodes only, the graphplan-based encoding is still sound and complete.

These observations led us at investigating a new direction based on the computation of a weaker version of the mutex relation. We recall that over the set $\oplus = \oplus_a \cup \oplus_f$, that is computed as specified by statements PG4 and PG5, we identify the static mutexes, $\widehat{\oplus}$, as those mutex relations obtained by part (a) of statement PG4. Moreover, we call dynamic mutexes the set $\widetilde{\oplus} = \oplus \setminus \widehat{\oplus}$.

Table 7.2 shows that when we restrict the computation of $\oplus$ to $\widehat{\oplus}$ (columns headed by **BCA with** $\oplus = \widehat{\oplus}$) we have a gain up to two orders of magnitude in the time of the planning graph analysis. See for instance the *KLS rep.* protocol. It must be noted that by considering a weaker version of the mutex relation, we obtain a rougher over-approximation of the forward search tree since the more mutexes we have, the more accurate is the over-approximation. On the other side, when we compute all mutexes, we get either a less or equal number of atoms and a greater or equal number of clauses. This means that the over-approximation is more accurate. However experimental results show that this gain in accuracy on the over-approximation does not pay off (see, e.g., the *KaoChow 3* protocol).

Under the BCA schema, we considered also a third variant of the graphplan-based encoding in which the mutex relation was neglected and hence we did not generate any "Mutex Axioms" during the "Planning Graphs to SAT" phase. Similarly to the linear encoding, we applied an *Abstraction/Refinement strategy* that checked if the pseudo partial-order plan found could be linearised (and hence executed). SATMC therefore looked for conflicting actions[7] in the pseudo partial-order plan found and extended the previously generated formula with clauses negating the conflicts (if any). The resulting formula was then fed back to the SAT-solver and the whole procedure was iterated until a solution without conflicts was met or the formula

[7]Differently from the linear encoding, two distinct actions α_1, α_2 are said to be in conflict iff $\alpha_1 \widehat{\oplus} \alpha_2$.

became unsatisfiable. However, by applying the Abstraction/Refinement strategy to the graphplan-based encoding, we have not had the same gain as in the linear encoding case. Also, there were no significant improvement in performance when compared to the version of the encoding denoted by **BCA with** $\oplus = \widehat{\oplus}$ and this is why such results are not shown in the table.

Once established that the best results are obtained by computing only the static mutexes, we have experimented over the schema dimension. It turns out that the EFA schema performs uniformly better than the BCA schema on all the analysed protocols. Namely, by using the setting **EFA with** $\oplus = \widehat{\oplus}$ based on the explanatory frame schema, we obtain *(i)* SAT instances whose sizes are up to 48% smaller, and *(ii)* up to 92% better encoding times with respect to the setting **BCA with** $\oplus = \widehat{\oplus}$ based on the backward chaining schema. This is mainly due to the following two reasons: first of all, contrary to the BCA schema, the EFA schema does not require the introduction of *no-operation actions* to enforce the inertia of fluents; and secondly the EFA schema is not required to introduce new fluents for representing the fluent occurring negated in the goal.

7.1.3 Linear versus Graphplan-Based Encodings

The comparison between the linear and the graphplan-based encoding techniques is summarised in Figures 7.1 and 7.2. For each encoding techniques and for each protocol those figures show (in logarithmic scale) the size of the SAT formulae (number of atoms and clauses), and the encoding time, respectively. Notice that, concerning the linear encoding technique we plot the results obtained by applying the Abstraction/Refinement strategy (see Section 4.3.2), while concerning the graphplan-based encoding technique we report the results obtained by using the EFA schema (see Section 4.2.2) and by generating only the static mutex relation. The comparison shows that, by using the graphplan-based encoding technique, we obtain (i) SAT instances whose size is up to 2 orders of magnitude smaller and (ii) better encoding and solving times. This enables SATMC to analyse protocols (e.g., the *KLS rep.* protocol), that were not feasible using the linear approach.

7.2 Comparison with other Automated Reasoning Tools

In order to assess the effectiveness of our SAT-based Model-Checking approach we have experimented it versus other automated reasoning techniques. On the one hand we have implemented prototype translators from our SATE language (see Section **??**) to PDDL (the Planning Domain Definition Language, [GHK$^+$98]) and LPARSE (the language taken in input by

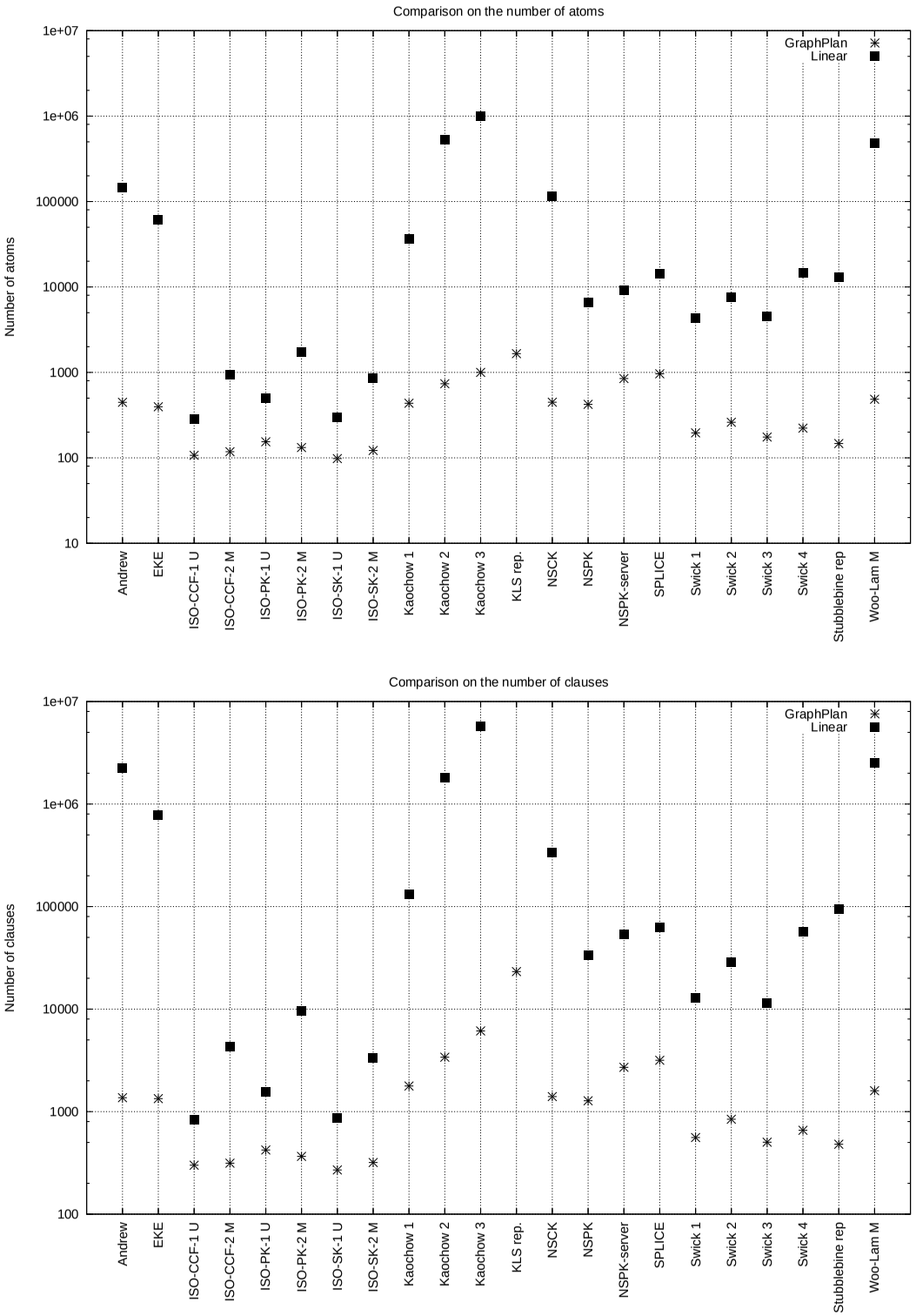

Figure 7.1: Comparison between graphplan-based and linear encodings on the number of atoms and clauses

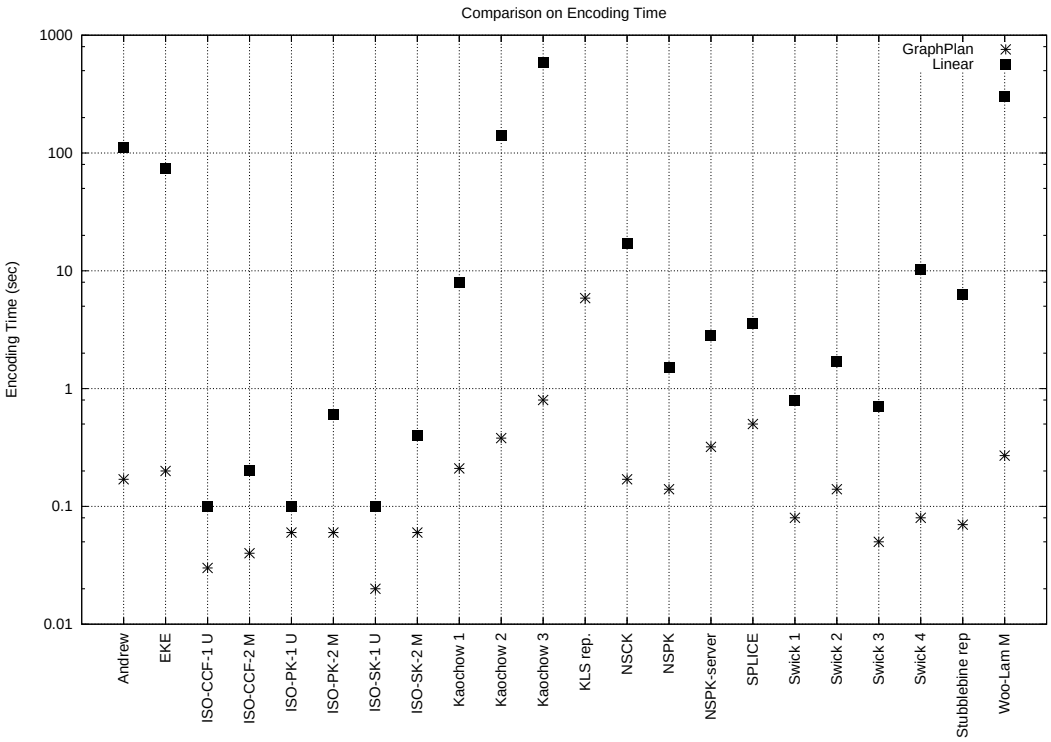

Figure 7.2: Comparison between graphplan-based and linear encodings on the encoding time

LPARSE,[8] i.e. a general purpose grounder for logic programming), to compare SATMC against the blackbox planning system (see Section 7.2.1) and against the logic programming solvers CMODELS and SMODELS (see Section 7.2.2). On the other hand, SATMC is one of the back-ends of the AVISPA tool [ABB+05] and of its predecessor, the AVISS tool [ABB+02]. In such a context SATMC has been thoroughly compared against other domain-specific back-ends, a Constraint-Logic-based Attack Searcher (CL-AtSe) and an On-the-fly Model-Checker (OFMC), for the analysis of security protocols. The results of this comparison are reported in Section **??** and Section 7.2.3.

7.2.1 Planning Systems: blackbox

Preliminary results obtained by running the SAT technology planning system blackbox v.4.1 ([KS99]) against our benchmark of protocols are reported in Table 7.3 in comparison the SATMC's performance.[9] For each protocol

[8] `http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz`.

[9] Experiments have been carried out on a PC with a 1.4 GHz CPU and 1 GB of RAM.

Table 7.3: Experiments on the Clark/Jacob library: SATMC versus blackbox

		SATMC			blackbox			
Protocol	**K**	**A**	**CL**	**E/S Time**	**A**	**CL**	**S2P**	**Time**
Andrew	9	447	1,368	0.17/0.00	633	2,381	98.53	2.30
EKE	5	396	1,340	0.20/0.00	1,056	5,855	50.96	1.46
ISO-CCF-1 U	4	107	300	0.03/0.00	413	1,059	0.07	0.03
ISO-CCF-2 M	4	118	314	0.04/0.00	522	1,455	0.36	0.05
ISO-PK-1 U	4	154	423	0.06/0.00	576	1,614	0.12	0.04
ISO-PK-2 M	4	132	366	0.06/0.00	667	1,924	0.95	0.17
ISO-SK-1 U	4	98	270	0.02/0.00	396	1,001	0.06	0.02
ISO-SK-2 M	4	122	319	0.06/0.00	494	1,349	0.29	0.09
KaoChow 1	7	436	1,778	0.21/0.00	891	5,416	6.46	0.96
KaoChow 2	9	736	3,397	0.38/0.01	904	5,382	74.03	9.52
KaoChow 3	9	1,000	6,122	0.80/0.01	1,015	7,209	277.65	32.17
KLS rep.	7	1,658	23,219	5.86/0.03			813.12	MO
NSCK	9	448	1,403	0.17/0.00	767	2,418	12.50	1.83
NSPK	7	423	1,276	0.14/0.00	815	2,682	1.71	0.15
NSPK-server	8	848	2,704	0.32/0.00	1,486	5,298	2.69	0.63
SPLICE	9	963	3,166	0.50/0.00	1,023	3,841	2.81	0.35
Swick 1	5	196	559	0.08/0.00	603	2,001	0.75	0.08
Swick 2	6	261	843	0.14/0.00	695	2,568	1.32	0.11
Swick 3	4	175	503	0.05/0.00	662	2,543	0.70	0.08
Swick 4	5	223	657	0.08/0.00	751	2,963	2.57	0.45
Stubblebine rep	3	147	481	0.07/0.01	1,086	8,857	10.11	2.14
Woo-Lam M	6	485	1,600	0.27/0.00	1,058	5,271	143.48	14.27

MO indicates that a memory out has been reached

we give the smallest value of k at which the attack is found (**K**), and for each tool the number of atoms (**ATs**) and clauses (**CLs**) in the SAT formula generated by them. For SATMC, both the time spent to generate the SAT formula (**E**) and that spent by the SAT-solver—we used the Chaff solver [MMZ+01] for these experiments—to solve the formula (**S**) are reported. Moreover we provide the time required to compile the SATE specification into a PDDL one (**S2P**) and the time spent by blackbox to solve it.

The results indicate that *(i)* SATMC performs uniformly better than blackbox on all the analysed protocols, and that *(ii)* the PDDL language used by blackbox (and by state-of-the-art planners) to specify planning problems is not appropriate to express in a compact way security protocols. This is due to the fact that PDDL is unable to specify complex term structure (only individual symbols are allowed). Hence, in order to translate our compact protocol specifications into PDDL, time consuming operations (see column **S2P** in Table 7.3) such as flattening and grounding of fluents and operators must be executed.

7.2.2 Logic Programming: CMODELS and SMODELS

By applying the ideas described in [ACL04] we have enhanced SATMC with a translator able to reduce the protocol insecurity problem Ξ specified in the IF language into a logic program Λ^k_Ξ (for increasing values of k). The logic program Λ^k_Ξ is then fed to the grounder LPARSE[10] and then to a state-of-the-art logic programming solver. (Currently both SMODELS and CMODELS can be used to this end.) Once an answer set is found, SATMC transforms it into an attack that is reported to the user.

We ran this extended version of SATMC against a selection of flawed security protocols drawn from the Clark & Jacob library [CJ97]. Table 7.4 reports the experimental results obtained by compiling to logic programs (**LP** sub-table) and by compiling to SAT through the linear encoding technique (**SAT** sub-table).[11] We used CMODELS as solver for logic programs and Chaff [MMZ+01] as SAT solver. Experiments were carried out on a PC with a 1.4 GHz CPU and 1 GB of RAM. For each protocol we give the smallest value of k at which the attack is found (**K**), the number of facts (**F**) and of rewrite rules (**Rules**) of the protocol insecurity problem. We also provide the time spent for generating the logic program starting from a SATE specification (SATE2LP) and the time spent by LPARSE for grounding it (LPARSE),[12] the number of lp-atoms (**LP ATs**) and lp-rules (**LP Rules**) in the logic program (in thousands), the time spent by CMODELS for solving

[10] `http://www.tcs.hut.fi/Software/smodels/lparse.ps.gz`.

[11] The reduction to SAT has been done by using the linear encoding technique.

[12] The time spent for generating the ground logic program (**LP EncT**) is the sum of SATE2LP and LPARSE.

Table 7.4: Experiments on the Clark/Jacob library: SAT versus Logic Programming

				Reduction to Logic programming (**LP**)						Reduction to SAT (**SAT**)			
Pb	**K**	**F**	**Rules**	**LP EncT** (SATE2LP / LPARSE)	**LP ATs**	**LP Rules**	CMODELS **ATs**	**CLs**	**LP SolT**	**EncT**	**ATs**	**CLs**	**SolT**
Andrew	9	465	15,650	0.41 / MO						111.4	145	2,256	12.1
EKE	5	1,148	10,924	1.84 / MO						74.1	62	783	3.7
ISO-CCF-1 U	4	39	22	0.10 / 0.28	< 1	2	< 1	1	0.02	0.1	< 1	< 1	0.0
ISO-CCF-2 M	4	81	132	0.18 / 1.12	2	7	2	3	0.05	0.2	< 1	4	0.1
ISO-PK-1 U	4	61	49	0.19 / 0.95	1	4	1	2	0.03	0.1	< 1	2	0.0
ISO-PK-2 M	4	125	279	0.50 / 3.87	4	17	4	4	0.14	0.6	2	10	0.1
ISO-SK-1 U	4	39	25	0.09 / 0.29	< 1	2	< 1	1	0.02	0.1	< 1	< 1	0.0
ISO-SK-2 M	4	100	87	0.25 / 1.65	4	11	4	4	0.11	0.4	< 1	3	0.0
KaoChow 1	7	2,753	2,042	3.58 / MO						8.0	36	131	0.7
KaoChow 2	9	32,963	22,344	45.71 / MO						140.4	531	1,804	15.6
NSCK	9	6,755	5,220	7.09 / MO						17.1	115	334	1.3
NSPK	7	429	662	0.46 / 7.54	12	80	12	14	0.65	1.5	7	33	0.1
NSPK-server	8	298	604	2.74 / 26.54	17	198	17	19	1.47	2.8	9	54	0.1
SPLICE	9	822	658	1.79 / 72.44	27	333	27	32	2.47	3.6	14	62	0.2
Swick 1	5	496	258	0.97 / 14.81	11	36	11	11	0.32	0.8	4	13	0.1
Swick 2	6	693	450	1.63 / 51.81	16	148	16	18	1.11	1.7	8	29	0.1
Swick 3	4	526	482	0.38 / 24.36	6	12	6	7	0.13	0.7	5	11	0.1
Swick 4	5	1,357	1,283	1.85 / 137.98	20	62	20	21	0.59	10.2	15	57	0.3
Stubblebine rep	3	1,409	2,408	1.40 / 371.00	29	2,010	29	30	14.37	6.3	13	95	0.1
Woo-Lam M	6	45,584	27,051	65.04 / MO						304.4	481	2,498	7.7

MO means that a memory out has been reached in running LPARSE; and

< 1 means that the number of lp atoms, lp rules, atoms, or clauses is less than 1 thousand;

the logic program (**LP SolT**)[13] together with the number of atoms (**ATs**) and clauses (**CLs**) in the SAT formula generated by CMODELS in this solving phase (in thousands). Moreover, we give the time spent by SATMC for generating the SAT formula (**EncT**), the number of propositional variables (**ATs**) and clauses (**CLs**) in the SAT formula (in thousands), and the time spent by Chaff to solve the SAT formula (**SolT**).

The results indicate that even using the quite inefficient linear encoding technique the **SAT** approach performs uniformly better than the **LP** approach. This is particular evident on medium size protocols where the **LP** approach cannot conclude its analysis since LPARSE is not able to allocate the necessary memory.

Since CMODELS [LM04] reduces the problem of finding answer sets of logic programs by reduction to SAT, it is interesting to compare the number of atoms and clauses directly generated by SATMC with that generated by CMODELS by compiling the logic program obtained by applying the reduction technique proposed in [ACL04]. SATMC uses a noticeably smaller number of atoms, while the number of clauses generated often speaks in favour of CMODELS. The difference in the number of atoms can be explained by the advantage of the grounder of SATMC tuned to the protocol insecurity problems over the general purpose grounder LPARSE. As far as the number of clauses is concerned, the difference is due to the fact that CMODELS performs some reductions on the logic program before translating it into a propositional formula.[14]

As far are the timing are concerned, the experimental results indicate that the **SAT** approach outperforms the **LP** approach. But this is mainly due to the time spent by the grounder LPARSE that largely dominates the other times.

It is worth pointing out that even if the reduction to SAT exhibits better performance, the reduction to logic programming results to be normally simpler to adapt in response to changes or extensions to the underlying model. For instance it can readily take into account specific algebraic properties of cryptographic operators. This is widely explained in [ACL04] where by adopting the **LP** approach we have been able to generate with SATMC a logic program whose answer set corresponds to a (known) attack on the Diffie-Hellman protocol [DH76].

7.2.3 AVISPA/AVISS back-ends: OFMC and CL-AtSe

AVISPA (*A*utomated *V*alidation of *I*nternet *S*ecurity-sensitive *P*rotocols and *A*pplications, [ABB+05]) is a push-button tool for the automated valida-

[13] The results obtained by running SMODELS on this application domain are really comparable with those of CMODELS.

[14] The details on the reduction can be found at `http://www.cs.utexas.edu/users/tag/cmodels/cmodels-1.ps`.

tion of Internet security-sensitive protocols and applications. The AVISPA

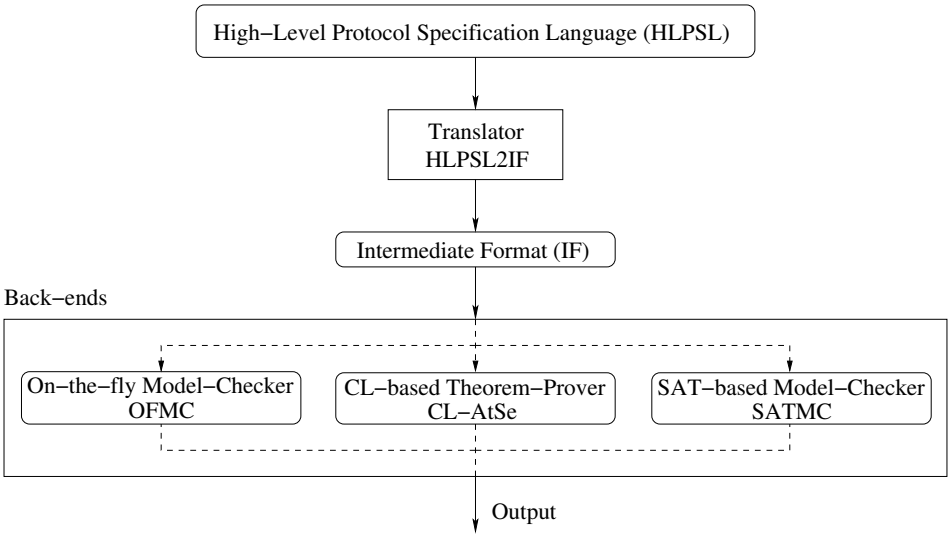

Figure 7.3: AVISPA/AVISS Architecture

Tool is a successor to the AVISS Tool (see [ABB+02]) from which inherit the successful framework depicted in Figure 7.3 to support the integration of back-ends implementing different search techniques, allowing their systematic and quantitative comparison and paving the way to their effective interaction. At different stages of their evolution, SATMC and the following two back-ends:

On-the-fly Model-Checker (OFMC): performs protocol falsification and bounded verification by exploring the transition system described by an IF specification in a demand-driven way (see [BMV04]); and

Constraint-Logic-based Attack Searcher (CL-AtSe): applies constraint solving as in [CV02], with some powerful simplification heuristics and redundancy elimination techniques;

have been integrated within AVISS and AVISPA.

AVISS. The AVISS Tool has been really successful in analysing small and medium scale protocols such as those in the Clark-Jacob's library [CJ97] and the results obtained under the strong typing assumption (i.e. type-flaw attacks are not reported) are shown in Table 7.5. For each protocol we give the kind of attack found and two main columns: the former, labelled by "Version 1", shows the times obtained in May 2001 by running the first version of AVISS back-ends on the protocols, while the latter denoted with "Version 2" reports on the results obtained nowadays by running the second

release of OFMC and SATMC.[15] For SATMC, both the time spent to generate the SAT formula with the linear encoding (**Enc**) and that spent by the SAT-solver—we used the Chaff solver [MMZ+01] for these experiments—to solve the formula (**Sol**) are reported.

With respect to "Version 1", the experiments show that OFMC v.1 outperforms the other two. However, for what concerns SATMC v.1 this is mainly due to the time required to generate the SAT formulae with a naive implementation of the linear encoding technique. It is interesting to observe that the time spent by SATMC v.1 to solve the SAT formulae is comparable to that of OFMC v.1. For what the constraint logic model checker is concerned, SATMC v.1 performs better than CL-Atse v.1 on the majority of the protocols. However the poorer timing of CL-Atse v.1 are balanced by the fact that it is based on an off-the-shelf prover (daTac) and it offers other advantages such as the simple integration of algebraic relations on message constructors (e.g. commutativity of encryptions in RSA). Even if such results are out-of-date, they are indeed significant to outline the impressive improvements and advancements in the back-ends evolutions. To show this, it suffices to observe Figure 7.4 that compares the SATMC timing between "Version 1" and "Version 2": SATMC v.2 gains up to 3 order of magnitude in the encoding time.

AVISPA. The AVISPA Tool significantly extends the scope, effectiveness, and performance of the AVISS Tool. In fact, the AVISPA Tool is able to scaling up to large scale Internet security protocols that are definitely more complex and challenging than those within the Clark/Jacob library. We have thoroughly compared SATMC, CL-AtSe, and OFMC by running them against the protocols in the AVISPA Library (see Table 5.1). The experimental results obtained under the assumptions of typed model (i.e. type-flaw attacks are not reported) and free-algebra[16] are summarised in Table 7.6.[17] For each of the protocols, the table gives the number of security problems ("#Pb") analysed and the number of problems for which attacks are detected ("#Att"),[18] and for each back-end, the average time spent by the back-end to analyse (i.e. either to find the attacks or to report that no attack exists in the given bounded scenario) a single problem of the protocol. For SATMC we report only the time spent by the back-end to generate the SAT formula, since in this case the time spent by the SAT-solver—we used

[15] Notice that, CL-AtSe has been completely redesigned in its evolution from AVISS to AVISPA and in such a process backward compatibility with IF v.1 has not been maintained.

[16] It must be noted that OFMC and CL-AtSe can relax the free-algebra assumption to deal with protocols requiring some special properties of cryptographic operators such as exponentiation.

[17] Appendix B provides more details on this experimental analysis.

[18] Notice that a problem is given by both a protocol and a security property the protocol should satisfy.

Table 7.5: Experiments on the Clark/Jacob library: AVISS back-ends

		Version 1			Version 2	
Protocol	**Attack**	**CL**	**OF**	**SATMC (Enc / Sol)**	**OF**	**SATMC (Enc / Sol)**
Andrew	Replay	32.74	0.05	80.57 / 2.65	0.02	0.17 / 0.00
EKE	Parallel sess.	240.77	0.10	75.39 / 1.78	0.05	0.20 / 0.00
ISO-CCF-1 U	Replay	2.23	0.00	0.17 / 0.00	0.01	0.03 / 0.00
ISO-CCF-2 M	Replay	4.55	0.01	0.46 / 0.01	0.00	0.04 / 0.00
ISO-PK-1 U	Replay	4.23	0.02	0.32 / 0.00	0.00	0.06 / 0.00
ISO-PK-2 M	Replay	11.06	0.01	1.18 / 0.01	0.01	0.06 / 0.00
ISO-SK-1 U	Replay	1.98	0.01	0.18 / 0.00	0.01	0.02 / 0.00
ISO-SK-2 M	Replay	3.86	0.01	0.43 / 0.01	0.01	0.06 / 0.00
KaoChow 1	Replay STS	76.82	0.45	16.34 / 0.17	0.06	0.21 / 0.00
KaoChow 2	Replay STS	45.25	0.48	339.70 / 2.11	0.03	0.38 / 0.01
KaoChow 3	Replay STS	50.09	0.49	1,288 / MO	0.03	0.80 / 0.01
KLS rep.	Parallel sess.	199.43	0.20	MO / -	0.14	5.86 / 0.03
NSCK	Replay STS	63.43	0.31	29.25 / 0.39	0.05	0.17 / 0.00
NSPK	MIM	12.91	0.04	1.77 / 0.05	0.02	0.14 / 0.00
NSPK-server	MIM	TO	1.12	4.29 / 0.04	0.20	0.32 / 0.00
SPLICE	Replay	352.42	4.02	5.48 / 0.05	0.09	0.08 / 0.00
Swick 1	Replay	TO	1.19	1.37 / 0.02	0.30	0.14 / 0.00
Swick 2	Replay	TO	0.86	2.68 / 0.03	0.55	0.05 / 0.00
Swick 3	Replay	2.68	0.03	1.50 / 0.02	0.01	0.08 / 0.00
Swick 4	Replay	35.97	0.04	8.18 / 0.13	0.02	0.07 / 0.01
Stubblebine rep	Replay STS	3.54	0.03	15.17 / 0.21	0.00	0.50 / 0.00
Woo-Lam M	Parallel sess.	245.56	0.27	1,024.08 / 7.95	0.08	0.27 / 0.00

TO: Time Out MO: Memory Out NS: Not Supported

Replay STS: Replay attack based on a Short-Term Secret
MIM: Man-in-the-middle

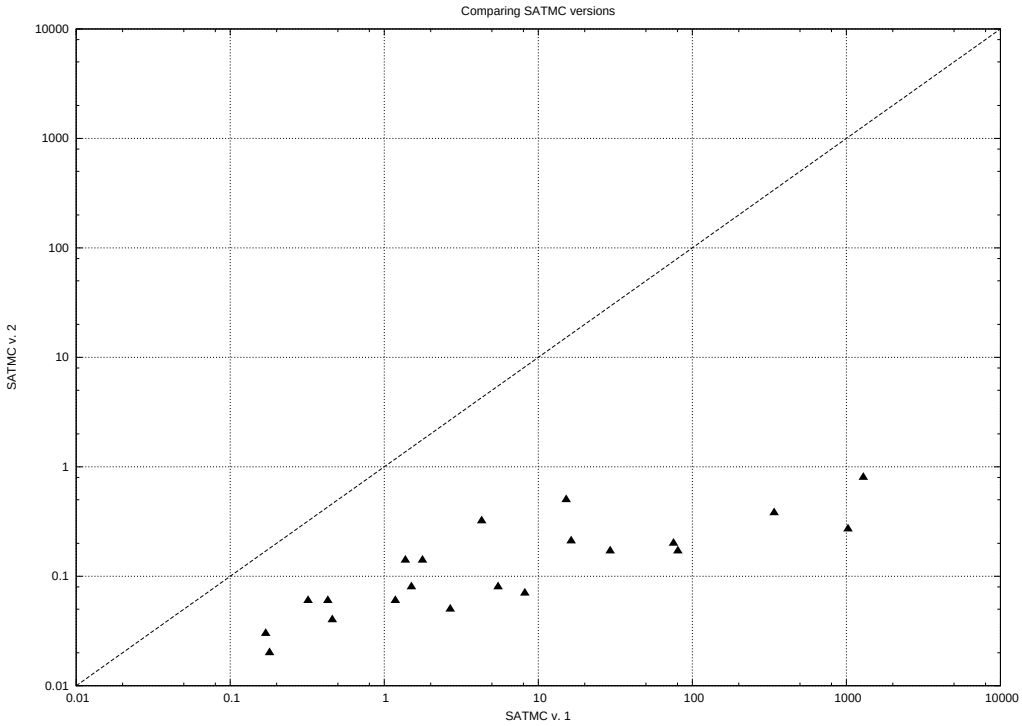

Figure 7.4: Comparing SATMC versions

the Chaff solver [MMZ+01] for these experiments—to solve the formula is really negligible.[19]

The boxed number in the ISO-PK-3 (also known as "ISO Public Key Two-Pass Mutual Authentication", [ISO97]) row denotes that the back-ends have found a new (previously unknown in literature) attack on such a protocol. It was already known that ISO-PK3 is vulnerable to replay attacks and hence it does not provide strong authentication [DNL99]: nothing in the messages ensures the freshness of the messages for the responder role. Our analysis, however, shows that the ISO-PK3 protocol does not even guarantee weak authentication, i.e. after successfully executing the protocol, neither the initiator nor the responder can be sure about the authenticity of the exchanged messages.

The experiments show that the time required by SATMC v.2 and by the other back-ends for analysing most of the problems is very low and thus acceptable for a modeller involved in security protocol design. For what concerns a back-end comparison, OFMC v.2 and CL-AtSe v.2 are very fast and performs better than SATMC v.2 in most of the cases. However on a

[19]Results are obtained by each single back-end with a resource limit of 1 hour CPU time and 1GB memory, on a Pentium IV 2.4GHz under Linux.

Table 7.6: Experiments on the AVISPA library

			Version 2		
Protocol	**#Pb**	**#Att**	**CL**	**OF**	**SATMC**
UMTS-AKA	4	0	0.01	0.03	0.01
ISO1	1	1	0.02	0.02	0.04
ISO2	1	0	0.02	0.07	0.63
ISO3	2	2	0.03	0.03	0.39
ISO4	2	0	0.03	0.38	208.31
ChapV2	4	0	0.02	0.18	0.10
EKE	4	2	0.03	0.10	0.09
TLS	4	0	0.05	0.29	1018.28
LPD-MSR	2	2	0.02	0.02	0.06
LPD-IMSR	2	0	0.04	0.04	0.10
Kerb-basic	10	0	0.07	0.61	6.24
Kerb-Cross-Realm	18	0	0.52	2.22	5.70
Kerb-Ticket-Cache	11	0	0.08	0.60	28.90
Kerb-PKINIT	12	0	0.06	0.47	27.21
Kerb-Forwardable	12	0	0.16	7.12	TO
Kerb-PreAuth	12	0	0.12	0.39	20.54
CRAM-MD5	2	0	0.04	0.23	0.17
PBK	1	1	0.01	0.34	0.25
PBK-fix	1	1	0.03	0.14	0.09
PBK-fix-weak-auth	1	0	0.49	3.47	0.33
DHCP-delayed-auth	2	0	0.02	0.06	0.12
TSIG	2	0	0.05	0.19	0.38
ASW	3	0	0.10	0.35	TO
ASW-abort	4	1	0.20	1.98	65.75
FairZG	5	0	0.37	8.76	0.28
SET-purchase	4	2	23.66	1.24	TO
SET-p.-hon.-payment-gw	4	0	0.61	0.83	TO
AAAMobileIP	9	0	0.03	0.14	0.11
Simple	3	0	100.17	78.50	0.50
CTP-non_predictive-fix	3	0	0.06	0.23	TO
geopriv	5	0	0.04	0.25	0.08
pervasive	2	0	47.67	30.52	4.00
two_pseudonyms	5	0	0.06	0.30	0.07
QoS-NSLP	2	0	32.16	16.01	0.21
sip	1	0	0.05	1.86	810.01

Legenda:

n : n new attacks has been found

TO : time-out

few instances like CRAM-MD5 and PBK-fix-weak, SATMC v.2 is winning. Besides this, it is interesting to observe that the time spent by SATMC v.2 to generate the SAT formula largely dominates that spent by the SAT-solver, and that the latter is negligible in most cases and comparable to that of the other back-ends. Moreover, it must be noted that small changes in the protocol (e.g. modify its security requirements) may require only the re-computation of a small part of the SAT encoding. Therefore the encoding time may have a smaller impact in practical situations.

The results obtained also give evidence of the following. On the one hand the techniques underlying OFMC and CL-AtSe are much more insensitive to the size of messages than the SAT-based Model-Checking approach. This is due to the fact that OFMC and CL-AtSe handle messages in a symbolic fashion, while SATMC is required to work with ground messages.[20]

On the other hand SATMC is much more insensitive than OFMC and CL-AtSe to the number of concurrent sessions analysed. The combination of graphplan-based encodings with efficient SAT solvers seems to be very suited in such a context. Basically the over-approximation of the search tree constructed by the graphplan-based encoding is then reduced to a propositional formula where the parallel execution of any pair of non-interfering actions is allowed and efficiently exploited by state-of-the-art SAT solvers. To further investigate on this point we have compared SATMC, OFMC, and CL-AtSe against both the flawed and the fixed variants of the NSPK-KS protocol (see Section 5.1.2) for increasing number of sessions. In both the protocols we have declared a key-server able to serve two keys requests for each session specified. The results obtained are shown in Table 7.7. It is immediate to see that contrary to OFMC and CL-AtSe that are not able to analyse the protocol when more than three sessions are declared, SATMC successfully analyse the protocol with respect to all the number of sessions considered.[21]

[20] More details about OFMC and CL-AtSe are given in Section 8.4.

[21] Notice that while all the sessions share the same server, the roles *Alice* and *Bob* are instantiated with appropriate combinations of the agents i, a, and b. Let $\langle A, B, S\rangle$ be one session where A, B, and S are playing *Alice*, *Bob*, and *Server* respectively, the sessions that can be specified are given by any subset of $\{\langle a, b, s\rangle, \langle a, b, s\rangle, \langle a, b, s\rangle, \langle a, b, s\rangle, \langle a, b, s\rangle, \langle a, b, s\rangle\}$.

Table 7.7: Experiments on the NSPK-KS with increasing number of sessions

Protocol	**#S**	**Attack**	**CL**	**OF**	**SATMC (Enc / Sol)**
NSPK-KS	2	YES	0.06	3.56	2.05 / 0.03
	3	YES	185.90	1038.40	3.89 / 0.04
	4	YES	TO	TO	6.54 / 0.03
	5	YES	TO	TO	8.55 / 0.04
	6	YES	MO	TO	12.95 / 0.05
NSPK-KS-fix	2	NO	2616.40	10.74	2.07 / 0.02
	3	NO	TO	1035.29	4.66 / 0.03
	4	NO	TO	TO	7.19 / 0.06
	5	NO	TO	TO	10.72 / 0.07
	6	NO	MO	TO	14.69 / 0.06

Legenda:

- #S : number of sessions
- YES : an attack has been found
- NO : the protocol is safe under the analysed scenario
- MO : memory-out
- TO : time-out

Chapter 8

Related Work

The area of security protocols has received a growing interest in the last decade from the formal methods community. This has lead to devise and develop a variety of techniques and rival tools. In this chapter we survey the most relevant works in this area. Moreover, we present general-purpose automated reasoning approaches more related to ours and we show how they compare with our SAT-based Model-Checking approach.

Besides the development of a SAT-based Model-Checker for security protocol analysis, this Thesis has contributed to the request for assistance arisen from protocol designers that ask for specification languages to support their activities. In the last section of this chapter we discuss how the High-Level Protocol Specification Language presented in Chapter 5 compares with respect to the most relevant specification languages for security protocols.

8.1 Planning as Satisfiability.

Planning aims to find a sequence of actions to apply to an input problem, in order to achieve a set of predefined goals, starting from a given initial state (see Section 2.2). The SAT-based Model-Checking approach proposed in this Thesis borrows from a specific area of planning called "Planning as Satisfiability" whose basis have been established in the seminal work of Kautz et al., [KMS96]. Their ideas have been implemented in blackbox, a planner that aims to combine the best features of Graphplan [BF95] and SATPLAN [KMS96] to solve planning problems.

However, preliminary results obtained by running blackbox v.4.1 [1] against the Clark-Jacob library [CJ97] show that our SAT-based Model-Checker per-

[1] Recently a new version of blackbox called SATPLAN04 [Kau04] has been released. The authors state that "the single most important difference between blackbox and SATPLAN04 is the SAT solvers used". However, since in our comparison the time for solving SAT formulae is usually negligible, we thus expect that a comparison between SATMC and SATPLAN04 will produce results similar to those already obtained.

forms uniformly better than blackbox (see Table 7.3).

Moreover this preliminary comparison clearly show that the PDDL language (the Planning Domain Definition Language, [GHK+98]) used by state-of-the-art planners (and by blackbox as well) to specify planning problems is not appropriate to express in a compact way security protocols. This is due to the fact that PDDL is unable to specify complex term structure, since only individual symbols are allowed as terms. Hence, in order to translate our compact protocol specifications into PDDL, time consuming operations such as flattening and grounding of fluents and operators must be executed (see, for instance, Example 18).

Example 18. *Let $X, Y \in \{1, 2\}$, then the planning operator*

$$\{p(X, f(X, g(Y))), q(a, g(X))\} \overset{foo(X,Y)}{\Rightarrow} \{q(Y, g(X))\} \ ; \ \{p(X, f(X, g(Y)))\}$$

is compactly specified in the SATE language (see Section 6.2) as follows:

```
action(foo(X,Y),
        true,
        [p(X,f(X,g(Y))), q(X)],
        [q(Y)],
        [p(X,f(X,g(Y)))]).
```

The PDDL language does not allow for the specification of complex term structure like $p(X, f(X, g(Y)))$ since $f(X, g(Y))$ is not an individual symbol but a more complex term. As a consequence, the above operator cannot be declared as compact as in the SATE language, but only by means of multiple[2] *PDDL ground operators:*

```
(:action  |foo_1_1|
   :parameters (  )
   :precondition   (and (p |1| |f(1,g(1))|)
                        (q |1|))
   :effect         (and (q |1|)
                        (not (p |1| |f(1,g(1))|))))

(:action  |foo_1_2|
   :parameters (  )
   :precondition   (and (p |1| |f(1,g(2))|)
                        (q |1|))
   :effect         (and (q |2|)
                        (not (p |1| |f(1,g(2))|))))
```

[2]Notice that the precise number of PDDL ground operators that are required is given by the cardinality of the Cartesian product of all the variable domains.

```
(:action  |foo_2_1|
   :parameters (  )
   :precondition   (and (p |2| |f(2,g(1))|)
                        (q |2|))
   :effect         (and (q |1|)
                        (not (p |2| |f(2,g(1))|))))

(:action  |foo_2_2|
   :parameters (  )
   :precondition   (and (p |2| |f(2,g(2))|)
                        (q |2|))
   :effect         (and (q |2|)
                        (not (p |2| |f(2,g(2))|))))
```

where if `t` *is a ground term, then* `|t|` *is an appropriate individual constant that represents* `t`.

Finally it is worth mentioning [RHN04], an interesting work in the area of Planning as Satisfiability where the authors promote the encoding into SAT of *alternative execution models* that still preserve the interleaving semantics. We are currently applying the standard *step semantics* execution model enhanced with an abstraction/refinement strategy. Though the other two execution models—i.e. process semantics and 1-linearization semantics—proposed in [RHN04] may lead to shorter attacks, they require the construction of the data structures (e.g. disabling graphs) and the enforcing of a number of additional propositional constraints that may affect significantly the performance of the overall approach. However, this topic will be subject of future work.

8.2 Answer Set Programming.

Answer set programming (ASP) is a form of declarative programming oriented towards difficult combinatorial search problems. The idea of answer set programming is to represent a computational problem as a logic program whose answer sets correspond to solutions to the given program.

It is well-known that logic programming with answer sets semantics is closely related to propositional logic [Nie99, LZ04] and AI planning [DNK97]. Hence it is surely interesting to experiment the logic programming approach (let us refer to it as "LP approach") against security protocols and to compare it with respect to our SAT-based Model-Checking approach.

A first work in such a direction is presented in [AM01]. In there the authors put forward the idea of formalising protocol insecurity problems (modelled according to Paulson's inductive model [Pau98]) in logic programming and of using the SMODELS solver for automating the analysis of

security protocols. Though the underlying models for security protocols are different—while they consider the Paulson's inductive model, we rely on the multi-set rewriting formalism—the approach presented in [AM01] has been also experimented against the Clark-Jacob' library. Unfortunately only few results are made available by the authors and these results only indicate that our SAT-based Model-Checking approach performs uniformly better.

In order to further investigate the impact of answer set programming against security protocols, we have then recast the ideas proposed in [DNK97] to devise a procedure for the automatic compilation of protocol insecurity problems into logic programming [ACL04]. This reduction, combined with the optimising transformations introduced in [AC02], paves the way to the construction of model-checkers for security protocols based on state-of-the-art solvers for logic programs. We have also thoroughly assessed the effectiveness of the proposed reduction by running our prototype implementation against a selection of flawed security protocols drawn from the Clark & Jacob library [CJ97] and using CMODELS to solve the resulting logic programs. The comparison with the SAT-based Model-Checking approach has been already shown in Table 7.4 and discussed in Section 7.2.2. Here we summarise the main two results:

- even using the quite inefficient linear encoding technique the SATMC approach performs uniformly better than the LP approach; this is mainly due to the LPARSE grounder that *(i)* requires for its activity an amount of time which largely dominates the other times, and *(ii)* on medium size protocols reaches a memory-out making the LP approach inconclusive on such instances;

- the reduction to logic programming results to be normally simpler to adapt in response to changes or extensions to the underlying model (e.g. it can readily take into account specific algebraic properties of cryptographic operators, see [ACL04]).

For what concerns this last point it is worth pointing out that we are currently working on extending our SAT-based Model-Checking approach to partially deal with algebraic properties of cryptographic operators. More details on this are given in Section 9.2.1.

8.3 Model Checking.

Model Checking [CGP99], already introduced in Chapter 1, is a wide research field in computer science. It embraces a variety of different methods for solving the general model-checking problem $M \models \phi$, i.e. to verify if a given *finite-state* concurrent system M satisfies a given (temporal logic) formula ϕ expressing some desired requirement.

In the rest of this section we will describe the model-checking methods more related to our work (either in terms of application-domain or in terms of similarities in the approach). However before to proceed further in their detailed description, it is worth pointing out some significant and general observations that relate our approach based on the "Model-Checking as Planning" paradigm with respect to standard model-checking methods.

On the one hand model-checking approaches, especially those performing an explicit enumeration and analysis of the set of states, are several limited by the *state explosion problem*, i.e. the size of the search space grows exponentially with respect to the size of the specification. On the other hand the success of our SAT-based Model-Checking approach is strongly dependent from the *size of the propositional formulae* to be generated.

On the standard model-checking side, two of the most successful techniques for dealing with state explosion problem are the construction of the *cone of influence* and the use of *partial-order reductions*. The cone of influence technique is a form of abstraction that attempts to decrease the size of the state transition graph by focusing on the variables of the system that are referred to in the specification. The reduction is obtained by eliminating variables that do not influence the variables in the specification. In this way, the properties checked are preserved, but the size of the model that needs to be verified is smaller. The partial-order reduction technique decreases the number of possible interleaving sequences that must be considered by exploiting the commutativity of concurrently executed transitions. As a result, the size of the state space that needs to be searched by model checking algorithms is reduced.

On the side of our SAT-based Model-Checking approach, the use of the planning graph over-approximation (see Section 4.2.2) significantly reduce—up to three order of magnitude, see Section 7.1.3—the size of the propositional formulae generated. Moreover the SAT-reduction techniques (except the bitwise variant) illustrated in Section 4.2 encode the input protocol insecurity problem Ξ into SAT formulae modelling an extended variant of Ξ in which the partial-order execution of (non-interfering) rules is allowed. In other words, the model represented by a SAT formula built through such encoding techniques allows for the parallel execution of any pair of actions that is not prevented by the relation of mutual exclusion. As a result we obtain a twofold benefit: the size of the propositional formulae generated may be tremendously smaller and the number of transition steps to be applied for finding attacks may decrease significantly.

It would be interesting to investigate further the relationship between the cone of influence and the planning graph over-approximation as well as the relationship between partial-order reduction techniques for model-checking and the partial-order execution of actions allowed by SAT-reductions for AI planning. We plan to do that in the future.

Another interesting connection between planning and model-checking

concerns the study of *alternative execution models* that provide different ways to allow partial-order execution of transitions still guaranteeing the interleaving semantics of the overall model. Some work in such a direction has been done by Niemelä et al. both in the area of bounded model checking [JHN03] and in the area of planning [RHN04]. If on the one hand the application of these alternative execution models may lead to shorter attacks/solutions, on the other hand the computational effort required by the algorithms that implement these alternative execution models may be overkilling for the whole approach and must be thus considered carefully. Nevertheless it would be interesting to investigate further in such a direction.

8.3.1 Bounded Model Checking.

Since its introduction in 1999 by Biere, Cimatti, Clarke and Zhu [BCCZ99], bounded model checking has received a growing interest as a complementary technique to the more traditional BDD based symbolic model checking.

NuSMV [CCG+02] is a state-of-the-art model checker that provides a SAT-based model checking component for bounded model checking of input specifications. Our preliminary attempt to tackle the question of deciding whether a security protocol achieves its desiderata or not by means of NuSMV has soon revealed that the NuSMV input language is not suited to specify such a question, as instead *action-based languages* are. This is mainly due to two reasons. First of all, while state-based languages, like the NuSMV input language, require the explicit declaration of variable domains as *enumerations*, action-based languages, like SATE, allow for the specification of a *signature* for each domain. Though this may be not a problem for expressing control-intensive application-domains usually tackled by model-checking tools, it may be problematic when data-intensive domains like those of security protocols are considered. For instance to specify in the NuSMV language the domain of the protocol messages, requires to enumerate all their possible instances (this is usually a "huge" enumeration). Secondly, state-based languages require to explicitly declare and enforce the frame axioms in the input specification, while action-based languages assume that a certain fact does not change its value unless this is explicitly caused by an action (i.e. the inertia law). As a consequence protocol specifications written through state-based languages may be significantly bigger than those specified by means of an action-based language.

8.3.2 Process Calculi.

Gavin Lowe and his group at the University of Leicester (UK) have applied the FDR model checker to analyse security protocols specified in CSP (Communicating Sequential Processes, [Hoa85]), i.e. an abstract language for the

description of concurrent systems. Basically, the various protocol roles are modelled as processes that exchange messages through channels. Similarly the intruder and the network are modelled as CSP processes. Channels are used to model both the DY intruder abilities (e.g., interception, faking) and to keep track of important events in the protocol (e.g., commit points useful to express authentication requirements).

In [DNL99] the authors have analysed problems from the Clark-Jacob library [CJ97] using Casper/FDR2. This approach has been very successful for discovering new flaws in protocols in the last decade. However, first experiments on the search time indicate that SATMC is more effective than Casper/FDR2. Besides that Casper/FDR2 is not able to handle non-atomic keys which is a real drawback for analysing real-world protocols, as it is fairly common in protocol design to construct symmetric keys from shared secrets and other data exchanged as part of the protocol. Our approach does not suffer from this limitation. Moreover, in our model we do not bound any fixed public key infrastructures (e.g. every agent has a pair of keys and knows the public key of every other agent), rather we allow for the specification of security protocols where keys are generated, distributed, and revoked. Finally, function application provides us with a direct and powerful mechanism to model, for example, cryptographic hash-functions and key-tables.

8.3.3 Explicit State Model Checking.

Mitchell and others [MMS97] have used the general purpose state enumeration based model-checker Murphi for analysing some small cryptographic protocol such as the Needham-Schroeder Public-Key. In Murphi the state of the transition system is given by the values of a set of global state variables, including the shared variables that are used to model communication. Experimental results indicate that Murphi suffers from state-space explosion. To cope with this problem the user must supply several restrictions on the model. For instance, the size of the network channel as well as the size of the intruder knowledge are fixed a priori. Moreover the intruder model is not provided by the tool, but the burden is on the user to come out with an appropriate finite state machine describing the intruder behaviour. As a result, an attack to a protocol may be missed because of either a bound imposed for a parameter that is too low (e.g. the attack may require an intruder with more memory) or an ability of the intruder that is not specified.

Another interesting work in the area of explicit state model checking of security protocol has been done by Clarke, Jha, and Marrero [CJM00b]. They have developed BRUTUS, a model checker that combines two orthogonal components: one that actually performs a depth-first search of the state graph and the other implementing a message derivation mechanism that models the intruder's capabilities in the spirit of the Paulson's synthesise

and analyse operators [Pau98]. A powerful specification logic is provided to describe a variety of security properties including secrecy, authentication, non-repudiation and a weak form of anonymity. Also BRUTUS, as other explicit state model checkers, suffers from the state explosion problem. To deal with it, the authors in [CJM00a] have proposed some partial-order reduction techniques specific for security protocols. However the approach has been experimented only on few protocols and thus its scalability needs more investigation. Moreover BRUTUS does not allow for non-atomic keys so to prevent its application to Internet protocols like for instance the Transport Layer Security (TLS) protocol and the UMTS Authentication and Key Agreement protocol.

8.4 The AVISPA approaches.

Our SAT-based Model-Checker is one of the back-ends of the AVISPA system [ABB+05]. In the context of this work we have performed a thorough comparison between the back-ends integrated in AVISPA (see Section 7.2.3).[3]

The On-the-fly Model-Checker (OFMC, [BMV04]) performs protocol falsification and bounded verification by exploring the search space system described by a protocol insecurity problem expressed in the IF specification language (see Section 5.2) in a demand-driven way. At the core of OFMC are two main features: *lazy data-types* that allow for the formalisation of protocol models as infinite trees, with tree only being computed on-the-fly as the states are explored; and the *lazy intruder model* that allow for representing the knowledge of the DY intruder in a symbolic manner by means of variables and symbolic constraints over them. In such a way OFMC avoids explicitly enumerating the possible messages the intruder can generate. These two methods are efficiently combined to search for protocol flaws (secrecy and authentication) in the state space associated to the protocol insecurity problem. When NPC protocol insecurity problems (see Section 3.4) are considered, then OFMC is guaranteed to terminate. In the general case, OFMC is a semi-decision procedure that halts with the correct attack trace when the protocol is flawed, but otherwise may not terminate.

Similarly to OFMC, the Constraint-Logic-based Attack Searcher (CL-AtSe, [CV02, Tur03]) approach also combines a symbolic technique for the lazy evaluation of the search space with a deduction system for determining

[3]The automatic verifier TA4SP developed on the ideas introduced in [GK00] has been recently integrated in the AVISPA tool. The TA4SP approach is merely complementary with respect to the other back-ends of the AVISPA tool. While TA4SP is particularly suited in trying to prove the correctness of protocols in an unbounded scenario, the other back-ends of the AVISPA tool are particularly suited in trying to prove the incorrectness of protocols searching for flaws in a bounded scenario. For such a reason the TA4SP back-end is not considered in such a comparison.

whether a particular term can be derived from a given set of terms. This is merely a lazy intruder model whereby instead of generating all the terms the intruder knows at a certain point, the system just checks, by evaluating unification constraints over variables, to see if the intruder can generate a particular message that an honest player expects. The protocol analysis problem is then reduced to establishing whether there exists an instantiation of variables computable by the intruder that satisfies the logic constraints implied by the trace. Contrary to OFMC and SATMC, CL-AtSe is currently limited to analyse only bounded scenarios. However it should not be difficult to extend the approach in order to come out with a semi-decision procedure for finding error in faulty protocols.

Both OFMC and CL-AtSe suffers from the *concurrency* issue i.e. the underlying approaches are really sensitive to the number of parallel sessions executed by honest agents. To overcome with such a difficulty OFMC has been extended with a partial order reduction technique, called *constraint differentiation* [BMV03b], that exploits redundancies in the symbolic states (i.e. the reduction exploits overlapping of the sets of ground states).

The comparison of OFMC and CL-AtSe with the SAT-based Model-Checking approach has been discussed in Section 7.2.3. In brief we have thoroughly compared the aforementioned tools by running them against the protocols in the AVISPA Library (see Table 5.1) under the assumptions of typed model and free-algebra.[4] Here we summarise the main points:

- the time required by all the three tools for analysing most of the problems is very low and thus acceptable for a modeller involved in security protocol design (see Table 7.6);
- OFMC and CL-AtSe are very fast and performs better than SATMC in most of the cases. However on a few instances like CRAM-MD5 and PBK-fix-weak, SATMC is winning (see Table 7.6);
- the time spent by SATMC to generate the SAT formula dominates that spent by the SAT-solver, and the latter is negligible in most cases and comparable to that of OFMC and CL-AtSe (see Table 7.6); and
- more interesting the fact that while OFMC and CL-AtSe are quite insensitive to the size of messages, SATMC is much more insensitive (even than OFMC extended with constraint differentiation) to the number of concurrent sessions analysed (see Table 7.7).

A last interesting difference between the SATMC approach and the other two concerns with the handling of the DY intruder knowledge. While both

[4]It must be noted that OFMC and CL-AtSe can relax the free-algebra assumption to deal with protocols requiring, for instance, some special properties of cryptographic operators such as exponentiation.

CL-AtSe and OFMC use a deduction system to determine whether a message can be constructed by the intruder, SATMC handles the intruder knowledge by means of standard rewrite rules, one for each of the intruder capability (e.g. decomposition of a pairing message, decryption of an encrypted message provided that the appropriate key belongs to the intruder knowledge, etc).

As a consequence, on the one hand SATMC provides a more operational view of the intruder than OFMC and CL-AtSe, in the sense that in order to derive a specific term from the intruder knowledge the system will try to apply step by step the intruder rules, rather than trying to solve in one single step a set of constraints (see for instance Example 19. On the other hand the method applied by OFMC and CL-AtSe to handle the intruder knowledge is more efficient since it does not require any unfolding of the transition relation, i.e. the intruder is somehow able to compute instantaneously its knowledge.

Example 19. *Suppose the intruder overhears from the network a message* $m = \{\langle a, s\rangle\}_{ki}$ *encrypted with its public key* ki *and assume the intruder needs to derive the term* s *for executing a particular rule, say* ℓ*. In order to retrieve* s*, SATMC will apply sequentially the rules* $decrypt(ki, \langle a, s\rangle)$ *and* $decompose(a, s)$ *(see rules* (3.13) *and* (3.14) *in Section 3.2.2) for decrypting and decomposing respectively the overheard message, whereas CL-AtSe and OFMC will add the constraint* $from(s, \{m\} \cup IK)$ *to the set of constraints and then they will check immediately whether the resulting constraints system is satisfiable.*

In [AC05] we have proposed an alternative intruder model for SATMC where the intruder knowledge is derived instantaneously. This is however part of a more general extension of the overall SAT-based Model-Checking approach that is currently a work-in-progress (see Section 9.2.1).

8.5 Strand Spaces approaches.

A lot of recent interest has been received by the strand spaces [FHG98] that provide an alternative approach to state-based analysis of completed protocol runs by emphasising causal interactions among protocol participants. Before to describe the main strand spaces-based approaches for security protocol analysis, let us introduce the basic concepts of this formalism.

A *strand space* is a collection of strands, with a graph structure which expresses causal relations. A *strand* is a sequence of events that a single agent may engage in, where an event is the sending or reception of a message, represented by a node in the graph. Nodes have a sign: a positive sign indicates a message was sent, and a negative sign indicates it has been received. Nodes in separate strands are adjacent when they represent the sending and reception of the same message. A *bundle* is a finite acyclic

subgraph of the strand space that is in a certain sense backwards-closed: all received messages occurring in strands in the bundle must have come from nodes also in the bundle, and if an event on a strand is in the bundle, then all preceding events on that strand must also be in the bundle. Intuitively, a bundle models one possible execution of a protocol. This means that the problem of deciding whether a security requirement is achieved by the protocol or not can be reduced to that of proving whether the requirement holds for all the bundles or not.

In the original strand space model [FHG98], messages are ground terms, but subsequent development and applications of the approach [SBP01, CDL+03] allow messages to contain variables (parameters) so that a single schema can represent all possible strands for a particular role in a protocol. In such a way generic protocol roles can be represented as strands with variables, also called *parametric strands*.

Figure 8.1 depicts the parametric strands describing the roles of the Needham-Schroeder Public Key (NSPK) protocol. The protocol has been already introduced and presented in Section 3.1.2.

$$
\begin{array}{ccc}
\underline{Alice[A, B, Ka, Kb, Na, Nb]} & & \underline{Bob[A, B, Ka, Kb, Na, Nb]} \\
\\
1 : +\{\langle A, Na\rangle\}_{Kb} & \rightarrow & 1 : -\{\langle A, Na\rangle\}_{Kb} \\
\Downarrow & & \Downarrow \\
2 : -\{\langle Na, Nb\rangle\}_{Ka} & \leftarrow & 2 : +\{\langle Na, Nb\rangle\}_{Ka} \\
\Downarrow & & \Downarrow \\
3 : +\{Nb\}_{Kb} & \rightarrow & 3 : -\{Nb\}_{Kb}
\end{array}
$$

Figure 8.1: NSPK protocol as parametric strands

The variables A, B, Ka, Kb, Na, and Nb can be bound to specific values (specific principal names for A and B, specific public keys for Ka and Kb, and specific nonces for Na and Nb) to instantiate the roles. Such an instantiation would correspond to a session of the protocol. Note that nodes belonging to the same role are connected sequentially via a double arrow ($\Rightarrow$). In general, for any two nodes n_1 and n_2, $n_1 \Rightarrow n_2$ means that both nodes occur in the same strand and that the node n_2 represents the action immediately following the action n_1. In other words, $\Rightarrow$ represents the ordering of actions in individual strands. This graph can be augmented with single arrows ($\rightarrow$) as is done Figure 8.1. This notation represents the send action and the receive action corresponding to a particular message. More formally, for two nodes n_1 and n_2, $n_1 \rightarrow n_2$ means that $n_1 = +m$ and $n_2 = -m$ for some message term m.

In this formalism, the two relations, $\rightarrow$ and $\Rightarrow$ together, define a causal precedence. In other words, it is possible to define a new relation $\preceq$ to be

the reflexive transitive closure of the union of the two arrow relations. This relation has the property that $n_1 \preceq n_2$ if and only if n_1 must occur before n_2. It captures which events must precede which other events. In [FHG98], Thayer, Herzog, and Guttman prove that this relation is a partial order. Because the only ordering imposed on nodes is this causal precedence relation, strands-based tools avoids the full interleaving semantics and so eliminates the need to perform a partial order reduction in the first place. In a sense, the partial order reduction is already built into the model itself. It is interesting to observe that similar features are provided by graphplan-based encodings. The main differences between a graphplan-based approach and a strand-based approach are the following: while the former works forward against a ground search space, the latter proceeds backward towards a symbolic search space. We plan to investigate more on this research direction in the future.

The first security protocol analyser based on the strand spaces formalism is Athena [SBP01]. It combines model checking and theorem proving techniques with the (parametric) strand space model to reduce the search space and automatically prove the correctness of or find an attack on security protocols (when it terminates). Besides the strands for honest roles, Athena uses strands for modelling the behaviour of the DY intruder. As a matter of facts the DY intruder is very "prolific": he can generate and send infinitely many messages. To cope with this issue, Athena additionally uses the notions of *semi-bundles* and *goal-bindings*. A semi-bundle is like a bundle but closed only under backwards tracing of strands (i.e. the relation $\Rightarrow$), not under sending of received messages. Goal-bindings are used to keep track of the search for ways a message might have originated, from another honest agent or from an intruder. In other words, goal-bindings record ways in which a semi-bundle could be expanded towards becoming a bundle. These concepts are used to remove some redundancy from the protocol trace representation. The idea is to eliminate search space explosion caused by infinite forwarding of a message, where arbitrarily many intruder strands may, for instance, receive and pass on the same message. The problem of deciding whether a security requirement is achieved by the protocol is thus expressed in terms of reachability of semi-bundles. This means to determine whether a semi-bundle can be completed into a bundle or not, and, if so, what substitution (instantiating the semi-bundle variables) is necessary to make it possible. A real limitation of Athena is the fact that it does not support non-atomic keys so that industrial-scale protocols requiring the construction of composed freshly keys cannot be taken into account.

Millen and Shmatikov enhanced the Athena approach in two directions [MS01]: they uses a constraint satisfaction procedure that overcomes the Athena limitation to support only atomic-keys and they introduce a term closure operator—defined in the spirit of Paulson's synthesise and analyse operators—that avoids the construction of costly intruder strands. The approach has

been further improved in [CE02]: flaws associated to partial runs of the protocol can be discovered; and a more expressive syntax is provided in order to be able to specify explicit tests executed by honest agents involved in the protocol. Notice that all the aforementioned improvements of the strand spaces model are easily accomplished by our underlying model based on multiset rewriting.[5] A detailed comparison between strand spaces and multiset rewriting for security protocol analysis goes beyond the scope of this paragraph and the interested reader should consult [CDL+03].

While using bundles as a model of computation seems promising (partial order reduction is already built into the model itself), the language used to specify requirements in strands-based tools is a little cumbersome. In this language, the atomic propositions have the form $s \in C$ where s is a strand constant and C is a bundle variable that is typically universally quantified. Specifications then have the form $\forall C\big(\bigwedge \Phi \supset \bigvee \Psi\big)$ where Φ and Ψ are sets of atomic propositions. Intuitively, the specification states that for any bundle C (for any causally closed collection of strands), if certain strands are in the bundle C (as specified by Ψ), then some other strand (as specified by Φ) must also be in the bundle C. While this is sufficient to specify typical authentication properties, it is not clear how to extend this to a more general logic.

Strands-based tools has been successfully experimented against the Clark-Jacob library [CJ97]. Since SATMC and these tools have a different input language, a direct comparison is not possible, but according to the published material (see [SBP01, MS01, CE02]) the results obtained with these tools are very good and promising. However it is not clear whether these tools are able to scale-up against protocols of growing complexity and of practical relevance in industrial environments like those included in the AVISPA library where SATMC has demonstrated to be successful (see Section 7.2.3).

8.6 Protocol Specification Languages.

In this section we compare the High-Level Protocol Specification Language presented in Chapter 5 with respect to a selection of the most closely related work.

Several specification languages have been used for protocol analysis. They include domain-specific languages specifically designed for protocols analysis (like HLPSL itself) as well as more generic modelling languages (e.g. LOTOS [LG00] or PROMELA, the input language to the SPIN [Hol97] model checker).

A powerful protocol specification language must fulfil several require-

[5] The SAT-based Model-Checking approach discussed in this Thesis does not use any term closure operator, but other tools (e.g. OFMC [BMV04] and CL-AtSe [CV02, Tur03]) based on the same model of SATMC make use of it.

ments. Among them we count the ability to explicitly specify different channel types (i.e. different intruder models). CASPER [DNL99], CAPSL [DM00], and MuCAPSL [MD03] (an extension of CAPSL based on strand spaces and particularly suited for group protocols) leave the intruder model implicit, while in HLPSL the intruder model can be specified as a parameter of the `channel` type (where each intruder model corresponds to a set of axioms that define its behaviour), allowing the modeller to describe heterogeneous network settings.

Another important issue in protocol design is the capability to model a wide range of goals. The aforementioned languages restrict themselves to secrecy and authentication goals, whereas HLPSL's underlying logic enables us to model very general security goals in a flexible way using temporal logic formulae.

Moreover, the possibility to specify compound keys allows the protocol designer to model a wide range of protocols, for example those based on the Diffie-Hellman exponentiation. Like CASPER, Winskel's SPL [CW01], a language based on Petri nets semantics, provides neither this feature nor the possibility to employ control-flow patterns.

Chapter 9

Conclusions and Perspectives

9.1 Wrapping up

In this thesis we have contributed to the research area of security protocol analysis by providing added values to the following points:

Specification Languages for Security Protocols. We have contributed to the design of a framework for the easy specification of security protocols (see Chapter 5).[1] This framework is depicted in Figure 5.1: security protocols are specified in the High-Level Protocol Specification Language (HLPSL, see Section 5.1) that is designed to be accessible to protocol designers—that are not necessarily experts in the area of formal methods—and that is automatically translatable into a lower-level term-rewriting based language, called Intermediate Format (IF, see Section 5.2), well-suited to model-checking tools. The strength of our framework is confirmed by the large collection of security protocols of varying levels of complexity—from the NSPK example presented in Section 3.1.2 to more complex industrial-scale protocols such as SET and TLS—we have been able to formalise with it. This set of security protocols, which we call the *AVISPA Library*, currently contains a total of 215 problems build from 48 protocols. Table 5.1 shows an excerpt of them. As far as we know no other rivals languages provide the same level of scope and coverage.

Automatic Tools for Security Protocol Analyses. We have developed an approach for analysing security protocols (see Section 4.3) that results from the combination of a reduction of security problems to planning problems (see Section 4.1) and SAT-encoding techniques tailored for planning (see Section 4.2). In doing this we have devised an abstract characterisation of AI planning SAT-reduction techniques that ensures the soundness

[1] We recall that this framework is one of the result of a joint work with other researchers carried out in the context of the EU projects AVISS [AVI01] and AVISPA [AVI02a].

and completeness of an encoding technique provided that it enjoys three well-defined requirements. Moreover we have investigated interesting aspects of cross-fertilisation between planning and model-checking. In fact, while the application of model-checking techniques to planning—usually referred as "Planning as Model-Checking", [FP99, AFEP97]—have been explored quite widely, the opposite direction, i.e. "Model-Checking as Planning", has not received the same attention. The SAT-based approach described in this thesis applies with success AI planning SAT-reduction techniques for bounded model checking [BCCZ99] of security protocols and thus contributes to show that model-checking problems—like those in the domain of security protocols that are particularly suited to be specified by means of expressive action-based formalisms—can be efficiently tackled via planning strategies.

Scaling-up towards Industrial-Scale Security Protocols. We have developed SATMC, a SAT-based Model-Checker, based on our ideas (see Chapter 6). SATMC is an open and flexible platform for bounded model-checking of security protocols that can readily exploit both improvements of SAT technology and advancements in AI planning SAT-reduction. We have thoroughly assessed (see Chapter 7) our approach by running SATMC against security protocols of growing complexity, from those in the Clark-Jacob's library [CJ97] to those, practically relevant in industrial environments, included in the AVISPA library. The results obtained confirm the effectiveness of the approach and show that SAT-based Model-Checking technologies usually applied to finite-state transition systems and control-intensive application-domains, can be successfully applied also on data and control-intensive domains like those of security protocols. For instance, the tight combination of graphplan-based encodings and efficient SAT solvers results in a very successful technique for analysing security protocols that is much more insensitive to the number of concurrent sessions considered than other state-of-the-art security protocol analysers. To the best of our knowledge no other tools, except those integrated in the AVISPA tool [AVI02a], exhibit the same level of scope and robustness while enjoying the same performance and scalability.

9.2 Future development(s)

There are several avenues of research that we hope are eventually explored. Many have to do with improving our SAT-based Model-Checking approach so that it can increase its scope and effectiveness. Two of them are discussed in Section 9.2.1 and Section 9.2.2. Others apply to security protocol analysis in general. For instance, a long-term goal that could be accomplished by future PhD students consists in extending the planning graph discussed in Section 4.2.2 to take into account variables and constraints. Basically

instead of computing a *ground* over-approximation of the forward search tree, we would like to compute a *symbolic* over-approximation of it and then to apply state-of-the-art planners (or maybe develop a new specific planner) to efficiently explore this symbolic over-approximation. This approach may be furthermore enhanced by combining it with a lazy intruder that has already proven itself to be an effective strategy for handling the prolific DY intruder.

9.2.1 Axioms

An interesting advancement of our SAT-based Model-Checking approach concerns with the extension of protocol insecurity problems to allow the specification of *axioms*. An axiom is a formula that states a relation between the facts of the transition system and that holds in each state of the transition system.[2] It turns out that axioms could be applied in two directions: *(i)* to represent relations between intruder knowledge facts, and *(ii)* to express algebraic equations that specify special properties of cryptographic operators such as exponentiation in the Diffie-Hellman protocol.

In [AC05] we have already experimented the first direction by using axioms to specify an alternative intruder model for our SAT-based Model-Checking approach where the intruder knowledge is derived instantaneously. The basic idea is to model the decomposition of the intruder knowledge by means of axioms instead of rules. For instance, the axiom

$$ik(\{M\}_K) \wedge ik(K^{-1}) \supset ik(M)$$

states that, every time the intruder knows both a message encrypted with the key K and the decryption key K^{-1}, then it knows *instantaneously* also the content M of the message. The main difference between an axiom and a rule is in the fact that the former must hold in every state of the transition system, while the latter is not forced to be executed even if all its preconditions are satisfied and it spends a transition for being executed. As a consequence by replacing decomposing rules with appropriate decomposing axioms we obtain a twofold benefit: the size of the propositional formulae generated can decrease as well as the number of transition steps to be applied for finding attacks. Notice that, given the set of intruder decomposing rules $\mathcal{DR} \subset \mathsf{R}$, it is straightforward to build the set of decomposing axioms $\mathcal{DA}$:

$$\mathcal{DA} = \{p_1 \wedge \cdots \wedge p_j \supset c \mid (p_1 \centerdot \ldots \centerdot p_j \xrightarrow{\ell} C) \in \mathcal{DR}, c \in C\}$$

It can be proved both that this optimisation is correct as it preserves the existing attacks and does not introduce new ones, and that it leads to equal or shorter attacks. We have extended our SAT-based Model-Checking for

[2] Notice that, if an axiom contains variables, then they are intended universally quantified.

supporting this kind of axioms and we have run our tool against a selection of (flawed) security protocols drawn from the Clark/Jacob library [CJ97]. SATMC is able to discover up to 40% shorter attacks[3] by generating up to 50% smaller propositional formulae.

Axioms are also useful to model specific algebraic properties of cryptographic operators. For instance, the Diffie-Hellman protocol relies on the following property of exponentiation:

$$(g^X)^Y = (g^Y)^X$$

Such a property can be modelled as a set of axioms representing equivalence classes over facts. For instance, the axioms

$$ik((g^X)^Y) \supset ik((g^Y)^X) \qquad \text{and} \qquad ik((g^Y)^X) \supset ik((g^X)^Y)$$

state that $ik((g^X)^Y)$ and $ik((g^Y)^X)$ are in the same equivalence class. However, this kind of axioms are not supported yet by our SAT-based Model-Checking approach that is currently limited to take into account set of axioms that does not entail any propositional equivalence between facts. We are currently working to overcome such difficulty.

9.2.2 Abstraction for verification

The SAT-based Model-Checking approach described in this thesis support the verification of security protocols whenever the number of sessions is bounded and each agent executes a finite number of steps. To establish the correctness of the protocol with respect to an unbounded scenario is in general an undecidable problem [DLMS99, EG83]. A classical way to circumvent this undecidability barrier is to employ abstraction techniques so to reduce a protocol insecurity problem into a simpler and finite problem such that every attack (of the original problem) is contained in the abstract problem. The converse usually does not hold since, due to the simplification, we may have false positives in the abstract problem even when the original problem is flawless.

A promising abstraction exploits the result of Comon and Cortier [CLC03] to verify security protocols by considering an unbounded number of protocol sessions. In [CLC03] the authors show that it is always sufficient to analyse the protocol with respect to a bounded number of agents b. Namely they prove that if there is an attack involving n agents, then there is an attack involving at most b agents. (In most cases b is equal to 2.) Let r the number of roles involved in the protocol, we have that at most b^r different sessions

[3]Notice that for each security protocol analysed, the attacks found on the corresponding protocol insecurity problem with and without the optimised intruder are identical except for the fact that in the case of the former we save all those rules executed for decomposing the knowledge of the intruder.

can be constructed, but each of them may be played infinitely many times. It turns out that our model can be simply transformed to represent a finite number of sessions executed arbitrarily many times. This is achieved by making state facts persistent: in every transition the agent goes to a new local state while "another incarnation" of the agent remains in the previous local state. Practically this consists to preserve `state`-facts in the application of IF rewrite rules. The protocol insecurity problem obtained after this transformation is still undecidable since, even if the model is typed, an infinite number of fresh terms may be generated by executing arbitrarily many times the same session. Here is where abstraction comes to the rescue. The idea is to bound in advance the number of fresh terms by assuming that two nonces are equivalent if they are generated by the same (honest) agent for the same communication partner(s) for the same protocol step. For instance, in this abstraction the agent *alice* playing arbitrarily many times the NSPK protocol with the agent *bob* would always use the same nonce *na* for communicating with *bob*.

Of course this abstraction may lead to false attacks caused by the neglection of the freshness property, but if no attacks are discovered, then the protocol is thus established to be correct with respect to an unbounded scenario. In the future we plan to incorporate this abstraction technique in our SAT-based Model-Checking approach.

Appendix A

Reduction to planning: soundness and completeness

Given a quantifier-free protocol insecurity problem $\tilde{\Xi} = \langle \tilde{\Gamma}, \mathsf{B} \rangle = \langle \langle \tilde{\mathsf{F}}, \tilde{\mathsf{L}}, \tilde{\mathsf{I}}, \tilde{\mathsf{R}} \rangle, \mathsf{B} \rangle$, it is possible to build a planning problem $\{\!|\tilde{\Xi}|\!\} = \langle \{\!|\tilde{\Gamma}|\!\}, \mathcal{G} \rangle = \langle \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{O} \rangle, \mathcal{G} \rangle$ such that any plan of length k on $\{\!|\tilde{\Xi}|\!\}$ corresponds to an attack of length k on $\tilde{\Xi}$ and vice versa.[1] The following simple results state that the reduction $\{\!|\tilde{\Xi}|\!\}$ presented in Section 4.1.2 is sound and complete.

Lemma 10. *If $S' = \mathrm{app}^{\Rightarrow}_{a}(S)$ with $(Pre \overset{a}{\Rightarrow} Add; Del) \in \mathcal{O}$, then there exist $(L \xrightarrow{\ell} R) \in \tilde{\mathsf{R}}$ and a ground substitution σ such that $a = \ell\sigma$, $Pre = L\sigma$, $Add = R\sigma$, $Del = L\sigma \setminus R\sigma$, and $S' = \mathrm{app}^{\rightarrow}_{\ell\sigma}(S)$.*

Proof. By S3 we know that for each $(Pre \overset{a}{\Rightarrow} Add; Del) \in \mathcal{O}$ there exist a rule $(L \xrightarrow{\ell} R) \in \tilde{\mathsf{R}}$ and a ground substitution σ such that $Pre = L\sigma$, $Add = R\sigma$, $Del = L\sigma \setminus R\sigma$. Moreover, $\mathrm{app}^{\Rightarrow}_{a}(S) = S'$ implies that $Pre \in S$ and $S' = (S \setminus Del) \cup Add = S \setminus (L\sigma \setminus R\sigma)) \cup R\sigma$. Since $(A \setminus (B \setminus C)) \cup C = (A \setminus B) \cup C$ for all sets A, B, and C, we can conclude that $S' = (S \setminus L\sigma) \cup R\sigma$. Therefore, it is immediate to conclude that $\ell\sigma$ is executable in S (i.e. $L\sigma \in S$) and $\mathrm{app}^{\rightarrow}_{\ell\sigma}(S) = (S \setminus L\sigma) \cup R\sigma = S'$. □

Lemma 11. *If $S_0 a_1 S_1 \cdots S_{k-1} a_k S_k$ is a planning path of $\{\!|\tilde{\Gamma}|\!\}$, then there exist ground substitutions $\sigma_1, \ldots, \sigma_k$ such that $S'_0 \ell_1 \sigma_1 S'_1 \cdots S'_{k-1} \ell_k \sigma_k S'_k$ is a rewriting path of $\tilde{\Gamma}$, $S_i = S'_i$ for each $i = 0, \ldots, k$, and $a_i = \ell_i \sigma_i$, $\mathrm{pre}(a_i) = lhs(\ell_i\sigma_i)$, $\mathrm{add}(a_i) = rhs(\ell_i\sigma_i)$, and $\mathrm{del}(a_i) = lhs(\ell_i\sigma_i) \setminus rhs(\ell_i\sigma_i)$ for each $i = 1, \ldots, k$.*

Proof. The proof is by induction on k. The base case is easy as by S4 we know that $\mathcal{I} = \tilde{\mathsf{I}}$ and therefore if $S_0 \in \mathcal{I}$ is a planning path of $\{\!|\tilde{\Gamma}|\!\}$ then $S_0 \in \tilde{\mathsf{I}}$ is also a rewriting path of $\tilde{\Gamma}$. The step case readily follow by the induction hypothesis and Lemma 10. □

[1] For the sake of simplicity in the rest of this section we abbreviate $L(\mathbf{x})$, $R(\mathbf{x})$, and $\ell(\mathbf{x})$ with L, R, and ℓ respectively.

Theorem 8 (Soundness). *Let $\tilde{\Xi}$ be a quantifier-free protocol insecurity problem. If $a_1, \ldots, a_k$ is a plan of $\{\![\tilde{\Xi}]\!\}$, then $a_1, \ldots, a_k$ is also a solution to $\tilde{\Gamma}$.*

Proof. The proof readily follows from Lemma 11 and S5. □

Lemma 12. *Let $(L \xrightarrow{\ell} R) \in \tilde{\mathsf{R}}$ and σ be a ground substitution. If $S' = \mathrm{app}^{\rightarrow}_{\ell\sigma}(S)$, then there exists $(Pre \overset{a}{\Rightarrow} Add; Del) \in \mathcal{O}$ such that $\ell\sigma = a$, $L\sigma = Pre$, $R\sigma = Add$, $L\sigma \setminus R\sigma = Del$, and $S' = \mathrm{app}^{\Rightarrow}_{a}(S)$.*

Proof. By S3 we know that for each ground substitution σ and $(L \xrightarrow{\ell} R) \in \tilde{\mathsf{R}}$ such that $\ell\sigma \in \mathsf{L}$ there exists $(Pre \overset{a}{\Rightarrow} Add; Del) \in \mathcal{O}$ such that $L\sigma = Pre$, $R\sigma = Add$, $L\sigma \setminus R\sigma = Del$, and $\ell\sigma = a$. Moreover, $\mathrm{app}^{\rightarrow}_{\ell\sigma}(S) = S'$ states that $L\sigma \in S$ and $S' = (S \setminus L\sigma) \cup R\sigma$. By set theory we know that $(A \setminus (B \setminus C)) \cup C = (A \setminus B) \cup C$ for all sets A, B, and C. As a consequence $S' = (S \setminus (L\sigma \setminus R\sigma)) \cup R\sigma$. Since $(S \setminus Del) \cup Add = S \setminus (L\sigma \setminus R\sigma)) \cup R\sigma = S'$, it is immediate to conclude that a is applicable in S (i.e. $Pre \in S$) and that $\mathrm{app}^{\Rightarrow}_{a}(S) = (S \setminus Del) \cup Add = S'$. □

Lemma 13. *If $S_0 \ell_1 \sigma_1 S_1 \cdots S_{k-1} \ell_k \sigma_k S_k$ is a rewriting path of $\tilde{\Gamma}$, then there exists a planning path $S'_0 a_1 S'_1 \cdots S'_{k-1} a_k S'_k$ of $\{\![\tilde{\Gamma}]\!\}$ such that $S_i = S'_i$ for each $i = 0, \ldots, k$, and $\ell_i\sigma_i = a_i$, $lhs(\ell_i\sigma_i) = \mathrm{pre}(a_i)$, $rhs(\ell_i\sigma_i) = \mathrm{add}(a_i)$, and $lhs(\ell_i\sigma_i) \setminus rhs(\ell_i\sigma_i) = \mathrm{del}(a_i)$ for each $i = 1, \ldots, k$.*

Proof. The proof is by induction on k. The base case is easy as by S4 we know that $\tilde{\mathsf{I}} = \mathcal{I}$ and therefore if $S_0 \in \tilde{\mathsf{I}}$ is a rewriting path of $\tilde{\Gamma}$ then $S_0 \in \mathcal{I}$ is also a planning path of $\{\![\tilde{\Gamma}]\!\}$. The step case readily follow by the induction hypothesis and by Lemma 12. □

Theorem 9 (Completeness). *Let $\tilde{\Xi}$ be a quantifier-free protocol insecurity problem. If $\ell_1\sigma_1, \ldots, \ell_k\sigma_k$ is a solution to $\tilde{\Gamma}$, then $\ell_1\sigma_1, \ldots, \ell_k\sigma_k$ is also a plan of $\{\![\tilde{\Xi}]\!\}$.*

Proof. The proof readily follows from Lemma 13 and S5. □

Appendix B

Experimental results: AVISPA details

The AVISPA Tool is able to scaling up to large scale Internet security protocols that are definitely more complex and challenging than those within the Clark/Jacob library. We have thoroughly compared SATMC, CL-AtSe, and OFMC by running them against the protocols in the AVISPA Library (see Table 5.1). The experimental results obtained under the assumptions of typed model and free-algebra[1] are summarised in Table B.1.[2] For each problem, that is identified by the protocol (see the column "Protocol"), the kind of property (see the column "property"), and the item on which the property is checked (see the column "on"), we report whether an attack is found (YES) or not (NO) (see the column "Attack")[3] and the time in seconds spent by each back-end to analyse the problem.[4] The boxed "YES" denotes that the back-ends have found a new (previously unknown in literature) attack on such a protocol. This is the case for the ISO-PK-3 (also known as "ISO Public Key Two-Pass Mutual Authentication", [ISO97]), the ASW protocol (Asokan, Shoup, and Waidner protocol, [ASW98]), and the SET protocol (Secure Electronic Transactions, [SET]). For instance, it was already known that ISO-PK3 is vulnerable to replay attacks and hence it does not provide strong authentication [DNL99]: nothing in the messages ensures the freshness of the messages for the responder role. Our analysis, however, shows that the ISO-PK3 protocol does not even guarantee weak

[1] Notice that OFMC and CL-AtSe can relax the free-algebra assumption to deal with protocols using special properties of cryptographic operators such as exponentiation.

[2] Results are obtained by each single back-end with a resource limit of 1 hour CPU time and 1GB memory, on a Pentium IV 2.4GHz under Linux.

[3] It must be noted that a NO indicates that the back-ends have been able to establish that the protocol satisfies the security property under the analysed scenario.

[4] For SATMC we report only the time spent by the back-end to generate the SAT formula, since in this case the time spent by the SAT-solver—we used the Chaff solver [MMZ+01] for these experiments—to solve the formula is really negligible.

authentication, i.e. after successfully executing the protocol, neither the initiator nor the responder can be sure about the authenticity of the exchanged messages.

Table B.1: Experiments on the AVISPA library

Problem			Atk	Backends		
Protocol	**property**	**on**		**OF**	**CL**	**SATMC**
UMTS-AKA	auth	r1	NO	0.03	0.02	0.02
	auth	r2	NO	0.01	0.01	0.00
	secrecy	sseq1	NO	0.04	0.00	0.02
	secrecy	sseq2	NO	0.03	0.01	0.01
ISO1	auth	na	YES	0.02	0.02	0.04
ISO2	auth	ra	NO	0.07	0.02	0.63
ISO3	wauth	na	YES	0.02	0.02	0.61
	wauth	nb	YES	0.03	0.03	0.16
ISO4	auth	na	NO	0.39	0.02	191.13
	auth	nb	NO	0.36	0.03	225.49
ChapV2	auth	na	NO	0.17	0.01	0.12
	auth	nb	NO	0.19	0.02	0.14
	secrecy	sec_kab1	NO	0.18	0.03	0.13
	secrecy	sec_kab2	NO	0.16	0.01	0.02
EKE	auth	na	YES	0.09	0.03	0.16
	auth	nb	YES	0.06	0.03	0.09
	secrecy	sec_k1	NO	0.14	0.02	0.06
	secrecy	sec_k2	NO	0.12	0.04	0.03
TLS	auth	na_nb1	NO	0.29	0.04	1019.56
	auth	na_nb2	NO	0.29	0.07	1026.48
	secrecy	sec_clientk	NO	0.29	0.04	1013.58
	secrecy	sec_serverk	NO	0.29	0.05	1013.49
LPD-MSR	secrecy	secx	YES	0.03	0.04	0.03
	wauth	x	YES	0.01	0.00	0.08
LPD-IMSR	secrecy	secx	NO	0.04	0.03	0.09
	wauth	x	NO	0.04	0.04	0.10
Kerb-basic	secrecy	sec_a_K_CG	NO	0.61	0.07	7.22
	secrecy	sec_c_K_CG	NO	0.57	0.06	1.93
	secrecy	sec_c_K_CS	NO	0.60	0.08	1.93
	secrecy	sec_g_K_CG	NO	0.63	0.07	7.31

continued on next page

Legend :

YES	a known attack has been found
YES	a new attack has been found
NO	the protocol is safe under the analysed scenario
MO	memory out
TO	time-out

continued from previous page

Problem			Atk	Backends		
Protocol	**property**	**on**		**OF**	**CL**	**SATMC**
	secrecy	sec_g_K_CS	NO	0.57	0.07	7.28
	secrecy	sec_s_K_CS	NO	0.59	0.07	7.15
	wauth	k_cg	NO	0.70	0.08	7.41
	wauth	k_cs	NO	0.64	0.08	7.52
	wauth	t1	NO	0.61	0.07	7.42
	wauth	t2a	NO	0.57	0.07	7.28
Kerb-Cross-Realm	auth	n1	NO	2.32	0.50	6.70
	auth	n1r	NO	2.29	0.52	6.85
	auth	n2	NO	2.24	0.53	6.98
	auth	t2a	NO	2.21	0.62	7.09
	auth	t2b	NO	2.30	0.52	6.88
	secrecy	sec_a_KC_TGSlocal	NO	2.26	0.52	6.74
	secrecy	sec_c_KC_Sremote	NO	2.19	0.53	1.82
	secrecy	sec_c_KC_TGSlocal	NO	2.17	0.52	1.83
	secrecy	sec_c_KC_TGSremote	NO	2.15	0.51	1.84
	secrecy	sec_c_T3	NO	2.17	0.50	1.83
	secrecy	sec_s_KC_Sremote	NO	2.21	0.52	6.75
	secrecy	sec_s_T3	NO	2.15	0.52	6.87
	secrecy	sec_tl_KC_TGSlocal	NO	2.20	0.52	6.74
	secrecy	sec_tl_KC_TGSremote	NO	2.23	0.53	6.77
	secrecy	sec_tr_KC_Sremote	NO	2.20	0.50	6.69
	secrecy	sec_tr_KC_TGSremote	NO	2.21	0.51	6.75
	wauth	t1	NO	2.24	0.51	6.83
	wauth	t1r	NO	2.23	0.50	6.67
Kerb-Ticket-Cache	auth	n1	NO	0.61	0.09	31.25
	auth	n2	NO	0.60	0.08	40.07
	auth	t1	NO	0.59	0.08	7.37
	auth	t2a	NO	0.58	0.09	39.07
	auth	t2b	NO	0.62	0.10	32.44
	secrecy	sec_c_Kcg	NO	0.57	0.06	7.31
	secrecy	sec_c_Kcs	NO	0.57	0.08	7.35
	secrecy	sec_k_Kcg	NO	0.64	0.07	37.50
	secrecy	sec_s_Kcs	NO	0.61	0.08	37.18

continued on next page

Legend :

YES	a known attack has been found
YES	a new attack has been found
NO	the protocol is safe under the analysed scenario
MO	memory out
TO	time-out

continued from previous page

Problem			Atk	Backends		
Protocol	**property**	**on**		**OF**	**CL**	**SATMC**
	secrecy	sec_t_Kcg	NO	0.60	0.07	42.84
	secrecy	sec_t_Kcs	NO	0.58	0.08	35.52
Kerb-Forwardable	auth	n1	NO	7.36	0.14	TO
	auth	n2	NO	7.39	0.18	TO
	auth	t1	NO	7.02	0.16	TO
	auth	t2a	NO	6.96	0.17	TO
	auth	t2b	NO	7.53	0.16	TO
	secrecy	sec_a_Kcg	NO	7.24	0.17	TO
	secrecy	sec_c_Kcg1	NO	6.98	0.15	TO
	secrecy	sec_c_Kcg2	NO	6.71	0.15	TO
	secrecy	sec_c_Kcs	NO	6.82	0.20	TO
	secrecy	sec_s_Kcs	NO	7.05	0.16	TO
	secrecy	sec_t_Kcg	NO	7.20	0.16	TO
	secrecy	sec_t_Kcs	NO	7.15	0.16	TO
Kerb-PreAuth	auth	n1	NO	0.41	0.12	27.52
	auth	n2	NO	0.41	0.11	28.64
	auth	t0	NO	0.39	0.14	4.68
	auth	t1	NO	0.37	0.12	4.69
	auth	t2a	NO	0.40	0.11	28.64
	auth	t2b	NO	0.37	0.13	34.25
	secrecy	sec_a_Kcg	NO	0.38	0.11	26.33
	secrecy	sec_c_Kcg	NO	0.37	0.09	4.70
	secrecy	sec_c_Kcs	NO	0.38	0.10	4.77
	secrecy	sec_s_Kcs	NO	0.37	0.11	28.28
	secrecy	sec_t_Kcg	NO	0.38	0.14	26.01
	secrecy	sec_t_Kcs	NO	0.39	0.11	28.02
Kerb-PKINIT	auth	n1	NO	0.47	0.05	34.76
	auth	n2	NO	0.46	0.05	34.41
	auth	t0	NO	0.46	0.05	14.07
	auth	t1	NO	0.47	0.06	14.18
	auth	t2a	NO	0.47	0.08	33.44
	auth	t2b	NO	0.48	0.06	33.01
	secrecy	sec_a_Kcg	NO	0.51	0.08	33.39

continued on next page

Legend :

YES	a known attack has been found
YES	a new attack has been found
NO	the protocol is safe under the analysed scenario
MO	memory out
TO	time-out

continued from previous page

Problem			Atk	Backends		
Protocol	**property**	**on**		**OF**	**CL**	**SATMC**
	secrecy	sec_c_Kcg	NO	0.47	0.05	14.25
	secrecy	sec_c_Kcs	NO	0.44	0.06	14.20
	secrecy	sec_s_Kcs	NO	0.45	0.05	33.60
	secrecy	sec_t_Kcg	NO	0.48	0.08	33.66
	secrecy	sec_t_Kcs	NO	0.46	0.06	33.51
CRAM-MD5	auth	auth	NO	0.21	0.06	0.28
	secrecy	sec_SK	NO	0.24	0.02	0.06
PBK	auth	msg	YES	0.34	0.01	0.25
PBK-fix	auth	msg	YES	0.14	0.03	0.09
PBK-fix-weak-auth	wauth	msg	NO	3.47	0.49	0.33
DHCP-delayed-auth	auth	sig	NO	0.06	0.02	0.13
	secrecy	sec_k	NO	0.05	0.02	0.10
TSIG	wauth	client_server_k_ba	NO	0.19	0.08	0.37
	wauth	server_client_k_ab	NO	0.18	0.02	0.39
ASW	auth	no	NO	0.36	0.11	TO
	auth	nr	NO	0.36	0.09	TO
	secrecy	no_secret	NO	0.34	0.10	TO
ASW-abort	auth	no	NO	2.29	0.22	75.83
	auth	nr	NO	2.17	0.19	75.38
	secrecy	no_secret	NO	2.05	0.21	74.99
	secrecy	secret_ref	YES	1.39	0.19	36.79
FairZG	wauth	alice_bob_nrr	NO	8.85	0.16	0.29
	wauth	alice_server_con	NO	8.71	0.13	0.27
	wauth	bob_alice_nro	NO	8.88	0.97	0.29
	wauth	bob_alice_sub	NO	8.68	0.33	0.28
	wauth	bob_server_con	NO	8.68	0.25	0.26
SET-purchase	auth	deal	MO	MO	TO	TO
	secrecy	order	MO	MO	69.90	TO
	secrecy	payment	YES	0.94	0.13	TO
	wauth	deal	YES	1.53	0.96	TO
SET-purchase-hon.-	auth	deal	NO	0.87	1.85	TO
payment-gateway	secrecy	order	NO	0.80	0.09	TO
	secrecy	payment	NO	0.75	0.08	TO

continued on next page

Legend :

YES	a known attack has been found
YES	a new attack has been found
NO	the protocol is safe under the analysed scenario
MO	memory out
TO	time-out

continued from previous page

Problem			**Atk**	**Backends**		
Protocol	**property**	**on**		**OF**	**CL**	**SATMC**
	wauth	deal	NO	0.88	0.43	TO
AAAMobileIP	secrecy	secFAHA	NO	0.14	0.02	0.04
	secrecy	secFAMN	NO	0.15	0.03	0.05
	secrecy	secMNHA	NO	0.13	0.02	0.06
	wauth	k_faha1	NO	0.13	0.03	0.14
	wauth	k_faha2	NO	0.13	0.03	0.14
	wauth	k_mnfa1	NO	0.14	0.03	0.14
	wauth	k_mnfa2	NO	0.15	0.03	0.14
	wauth	k_mnha1	NO	0.14	0.03	0.14
	wauth	k_mnha2	NO	0.13	0.01	0.15
Simple	secrecy	presenceinfo	NO	78.87	12.86	0.48
	wauth	ps_wr_user	NO	80.68	12.59	0.48
	wauth	wr_ps_presenceinfo	NO	75.96	275.05	0.53
CTP-non_predictive-fix	auth	npaa_pac_nnpaa	NO	0.21	0.05	TO
	auth	ppaa_pac_ip_pac	NO	0.26	0.05	TO
	secrecy	mac_key	NO	0.21	0.07	TO
geopriv	auth	lr_mu_n_lr	NO	0.28	0.06	0.16
	secrecy	filtered_loc	NO	0.24	0.03	0.05
	secrecy	k_psi	NO	0.23	0.05	0.05
	secrecy	psi	NO	0.25	0.02	0.02
	wauth	ls_mu_psi	NO	0.26	0.04	0.13
pervasive	auth	lbs_t_n_lbs	NO	30.94	67.71	4.77
	secrecy	loc	NO	30.10	27.63	3.22
two_pseudonyms	auth	lr_t_n_lr	NO	0.35	0.06	0.16
	secrecy	filtered_loc	NO	0.27	0.04	0.05
	secrecy	loc	NO	0.30	0.07	0.04
	secrecy	psi_lr	NO	0.30	0.06	0.05
	secrecy	psi_t	NO	0.29	0.06	0.04
QoS-NSLP	wauth	router_server_clientid	NO	15.76	42.87	0.20
	wauth	server_client_service	NO	16.26	21.44	0.23
sip	auth	y	NO	1.86	0.05	810.01

Bibliography

[ABB+02] Alessandro Armando, David Basin, Mehdi Bouallagui, Yannick Chevalier, Luca Compagna, Sebastian Mödersheim, Micha"el Rusinowitch, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. The AVISS Security Protocol Analysis Tool. In *Proceedings of CAV'02*, LNCS 2404, pages 349–354. Springer-Verlag, 2002. URL of the AVISS and AVISPA projects: `www.avispa-project.org`.

[ABB+05] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, P. Hankes Drielsma J. Cuellar, P.C. Heam, O. Kouchnarenko, S. Moedersheim J. Mantovani, D. von Oheimb, M. Rusinowitch, M. Turuani J. Santiago, L. Viganò, and L. Vigneron. The AVISPA Tool for the Automated Validation of Internet Security Protocols and Applications. In *In Proceedings of the 17th International Conference on Computer-Aided Verification (CAV'05)*. 2005.

[AC02] A. Armando and L. Compagna. Automatic SAT-Compilation of Protocol Insecurity Problems via Reduction to Planning. In D.A. Peled and M.Y. Vardi, editors, *Proceedings of 22nd IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE)*, LNCS 2529, pages 210–225. Springer-Verlag, Houston, Texas, November 2002. Also presented at the FCS & Verify Workshops, Copenhagen, Denmark, July 2002. Available at `www.avispa-project.org`.

[AC04a] A. Armando and L. Compagna. Abstraction-driven SAT-based Analysis of Security Protocols. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing*, LNCS 2919, pages 257–271. Springer-Verlag, 2004. Selected Revised Papers. Presented to SAT 2003, S. Margherita Ligure, Italy. Available at `www.avispa-project.org`.

[AC04b] Alessandro Armando and Luca Compagna. SATMC: a SAT-based Model Checker for security protocols. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA 2004)*, LNAI 3229, Lisbon, Portugal, September 2004. Springer-Verlag.

[AC05] Alessandro Armando and Luca Compagna. An Optimized Intruder Model for SAT-based Model-Checking of Security Protocols. In A. Armando and L. Viganò, editors, *Electronic Notes in Theoretical Computer Science*, volume 125, pages 91–108. Elsevier Science Publishers, March 2005. Presented to the IJCAR04 Workshop ARSPA, available at `http://www.avispa-project.org`.

[ACG03] A. Armando, L. Compagna, and P. Ganty. SAT-based Model-Checking of Security Protocols using Planning Graph Analysis. In K. Araki, S. Gnesi, and D. Mandrioli, editors, *Proceedings of the 12th International Symposium of Formal Methods Europe (FME)*, LNCS 2805, pages 875–893. Springer-Verlag, 2003. Available at `www.avispa-project.org`.

[ACL04] Alessandro Armando, Luca Compagna, and Yuliya Lierler. Automatic compilation of protocol insecurity problems into logic programming. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA 2004)*, LNAI 3229, Lisbon, Portugal, September 2004. Springer-Verlag.

[AFEP97] Cimatti A., Giunchiglia F., Giunchiglia E., and Traverso P. Planning via model checking: a decision procedure for ar. In Steel S. and Alami R., editors, *4th European conference on planning (ECP'97)*, Toulouse, 24-26 September 1997.

[AH03] Jari Arkko and Henry Haverinen. EAP AKA Authentication, October 2003. Work in Progress.

[AL00] R.M. Amadio and D. Lugiez. On the reachability problem in cryptographic protocols. In Catuscia Palamidessi, editor, *Proceedings of Concur'00*, LNCS 1877, pages 380–394. Springer-Verlag, 2000.

[AM01] L. Carlucci Aiello and F. Massacci. Verifying security protocols as planning in logic programming. *ACM Trans. on Computational Logic*, 2(4):542–580, 2001.

[ASW98] N. Asokan, Victor Shoup, and Michael Waidner. Asynchronous protocols for optimistic fair exchange. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 86–99, 1998.

[AVI01] AVISS. FET Open Assessment project IST-2000-26410, Automated Validation of Infinite-State Systems. `http://www.avispa-project.org/aviss/`, 2001.

[AVI02a] AVISPA. Shared-cost RTD (FET open) project IST-2001-39252, Automated Validation of Internet Security Protocols and Applications. `http://www.avispa-project.org`, 2002.

[AVI02b] AVISS. Deliverable 1.3: Final project report. For more information on the AVISS project see `http://www.avispa-project.org/aviss/`, 2002.

[AVI03a] AVISPA. Deliverable 2.1: The High-Level Protocol Specification Language. Available at `http://www.avispa-project.org`, 2003.

[AVI03b] AVISPA. Deliverable 2.3: The Intermediate Format. Available at `http://www.avispa-project.org`, 2003.

[AVI03c] AVISPA. Deliverable 6.1: List of selected problems. Available at `http://www.avispa-project.org`, 2003.

[BAN90] M. Burrows, M. Abadi, and R. Needham. A Logic of Authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, 1990.

[BCCZ99] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic Model Checking without BDDs. In *Proceedings of TACAS'99*, LNCS 1579, pages 193–207. Springer-Verlag, 1999.

[BF95] Avrim Blum and Merrick Furst. Fast planning through planning graph analysis. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI 95)*, pages 1636–1642, 1995.

[BLMTM04] Julien Bournelle, Maryline Laurent-Maknavicius, Hannes Tschofenig, and Yacine El Mghazli. Handover-aware access control mechanism: Ctp for pana. In *ECUMN*, pages 430–439, 2004.

[BM92] S. Bellovin and M. Merritt. Encrypted Key Exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, May 1992.

[BM98] Colin Boyd and Anish Mathuria. Key establishment protocols for secure mobile communications: A selective survey. *Lecture Notes in Computer Science*, 1438:344ff, 1998.

[BMP01] Giampaolo Bella, Fabio Massacci, and Lawrence C. Paulson. Verifying the SET Purchase Protocols. Technical Report 524, University of Cambridge, November 2001. URL: `http://www.cl.cam.ac.uk/Research/Reports/TR524-lcp-purchase.pdf`.

[BMS03] Scott Bradner, Allison Mankin, and Jeffrey Schiller. A Framework for Purpose-Built Keys (PBK), June 2003. Work in Progress.

[BMV03a] D. Basin, S. Mödersheim, and L. Viganò. An On-The-Fly Model-Checker for Security Protocol Analysis. In Einar Snekkenes and Dieter Gollmann, editors, *Proceedings of ESORICS'03*, LNCS 2808, pages 253–270. Springer-Verlag, 2003. Available at `http://www.avispa-project.org`.

[BMV03b] D. Basin, S. Mödersheim, and L. Viganò. Constraint Differentiation: A New Reduction Technique for Constraint-Based Analysis of Security Protocols. In Vijay Atluri and Peng Liu, editors, *Proceedings of CCS'03*, pages 335–344. ACM Press, 2003. Available at `http://www.avispa-project.org`.

[BMV04] D. Basin, S. Mödersheim, and L. Viganò. OFMC: A Symbolic Model-Checker for Security Protocols. *International Journal of Information Security*, 2004.

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[CCC+04] Yannick Chevalier, Luca Compagna, Jorge Cuellar, Paul Hankes Drielsma, Jacopo Mantovani, Sebastian Mödersheim, and Laurent Vigneron. A High Level Protocol Specification Language for Industrial Security-Sensitive Protocols. In *In Proceedings of SAPS'2004*. 2004.

[CCG+02] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, volume 2404 of *LNCS*, Copenhagen, Denmark, July 2002. Springer.

[CDL+99] I. Cervesato, N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. A meta-notation for protocol analysis. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop: CSFW'99*, pages 55–69. IEEE Computer Society Press, Mordano, Italy, June 1999.

[CDL+00] Iliano Cervesato, Nancy Durgin, Patrick D. Lincoln, John C. Mitchell, and Andre Scedrov. Relating strands and multiset rewriting for security protocol analysis. In Paul Syverson, editor, *Proceedings of the 13th IEEE Computer Security Foundations Workshop: CSFW'00*, pages 35–51. IEEE Computer Society Press, 2000.

[CDL+03] Iliano Cervesato, Nancy Durgin, Patrick D. Lincoln, John C. Mitchell, and Andre Scedrov. A Comparison between Strand Spaces and Multiset Rewriting for Security Protocol Analysis. In *Proceedings of ISSS 2002*, LNCS 2609, pages 356–383. Springer-Verlag, 2003.

[CE02] Ricardo Corin and Sandro Etalle. An Improved Constraint-Based System for the Verification of Security Protocols. In *Proceedings of SAS 2002*, LNCS 2477, pages 326–341. Springer-Verlag, 2002.

[CER03] CERT/CC Overview, 2003. Available at `http://www.cert.org`.

[CGKS02] E. Clarke, A. Gupta, J. Kukula, and O. Strichman. SAT based abstraction-refinement using ILP and machine learning techniques. In *Proc. of Conference on Computer-Aided Verification (CAV'02)*, LNCS, 2002.

[CGP99] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[CGRZ03] S. Creese, M. Goldsmith, B. Roscoe, and I. Zakiuddin. The attacker in ubiquitous computing environments: Formalising the threat model. In *Proceedings of the First International Workshop on Formal Aspects in Security and Trust*, pages 83–97, Italy, 2003.

[CJ97] J. Clark and J. Jacob. A Survey of Authentication Protocol Literature: Version 1.0, 17. Nov. 1997. URL: `www.cs.york.ac.uk/~jac/papers/drareview.ps.gz`, 1997.

[CJM00a] Edmund M. Clarke, Somesh Jha, and Wilfredo R. Marrero. Partial order reductions for security protocol verification. In *TACAS*, pages 503–518, 2000.

[CJM00b] Edmund M. Clarke, Somesh Jha, and Wilfredo R. Marrero. Verifying security protocols with brutus. *ACM Trans. Softw. Eng. Methodol.*, 9(4):443–487, 2000.

[CLC03] H. Comon-Lundh and V. Cortier. Security properties: two agents are sufficient. In *Proceedings of ESOP'2003*, LNCS 2618, pages 99–113. Springer-Verlag, 2003.

[CLG+03] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, and J. Arkko. RFC 3588: Diameter Base Protocol, September 2003. Status: Proposed Standard.

[CMM+04] J. Cuellar, J. Morris, D. Mulligan, J. Peterson, and J.Polk. RFC 3693: Geopriv requirements, 2004. `http://www.faqs.org/rfcs/rfc3693.html`.

[Coo71] S. A. Cook. The complexity of theorem proving procedures. In *3rd Annual ACM Symposium on the Theory of Computation*, pages 151–158, 1971.

[CS02] H. Comon and V. Shmatikov. Is it possible to decide whether a cryptographic protocol is secure or not? *Journal of Telecommunications and Information Technology. Special issue on Cryptographic protocol verification*, 4:5–15, 2002.

[CSI04] Computer Crime and Security Survey, 2004. Available at `http://www.gocsi.com`.

[CV02] Y. Chevalier and L. Vigneron. Automated Unbounded Verification of Security Protocols. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Proceedings of CAV'02*, LNCS 2404, pages 324–337. Springer-Verlag, 2002.

[CW01] F. Crazzolara and G. Winskel. Events in security protocols. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 96–105. ACM Press, 2001.

[DA99] T. Dierks and C. Allen. RFC 2246: The TLS Protocol Version 1.0, January 1999. Status: Proposed Standard.

[DA01] R. Droms and W. Arbaugh. RFC 3118: Authentication for DHCP Messages, June 2001. Status: Proposed Standard.

[DH76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[DLL62] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Journal of the ACM*, 5(7):394–397, 1962.

[DLMS99] N. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. Undecidability of Bounded Security Protocols. In *Proceedings of the FLOC'99 Workshop on Formal Methods and Security Protocols (FMSP'99)*, 1999.

[DLMS02] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Multiset rewriting and the complexity of bounded security protocols, 2002.

[DM00] G. Denker and J. Millen. CAPSL integrated protocol environment. In *DARPA Information Survivability Conference (DISCEX 2000)*, pages 207–221. IEEE Computer Society, 2000.

[DMGF00] G. Denker, J. Millen, A. Grau, and J. Filipe. Optimizing protocol rewrite rules of CIL specifications. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW '00)*, pages 52–63. IEEE, July 2000.

[DMT98] G. Denker, J. Meseguer, and C. Talcott. Protocol specification and analysis in Maude. In N. Heintze and J. Wing, editors, *Proceedings of the Workshop on Formal Methods and Security Protocols*. 1998.

[DNK97] Yannis Dimopoulos, Bernhard Nebel, and Jana Koehler. Encoding planning problems in nonmonotonic logic programs. In *ECP*, pages 169–181, 1997.

[DNL99] B. Donovan, P. Norris, and G. Lowe. Analyzing a Library of Security Protocols using Casper and FDR. In *Proceedings of the Workshop on Formal Methods and Security Protocols*, 1999.

[DSK00] Minh Binh Do, Biplav Srivastava, and Subbarao Kambhampati. Investigating the effect of relevance and reachability constraints on SAT encodings of planning. In *Artificial Intelligence Planning Systems*, pages 308–314, 2000.

[DY83] D. Dolev and A. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.

[Eas00] D. Eastlake 3rd. RFC 2930: Secret Key Establishment for DNS (TKEY RR), September 2000. Status: Proposed Standard.

[EG83] Shimon Even and Oded Goldreich. On the security of multi-party ping pong protocols. In *Proceedings of 24th IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society, 1983.

[EMW97] M. D. Ernst, T. D. Millstein, and D. S. Weld. Automatic SAT-compilation of Planning Problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 1169–1177. Morgan Kaufmann, 1997.

[ENS95] Kutluhan Erol, Dana S. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence*, 76(1-2):75–88, 1995.

[FHBH+99] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. RFC 2617: HTTP Authentication: Basic and Digest Access Authentication, June 1999. Status: Draft Standard.

[FHG98] F. Javier Thayer Fábrega, Jonathan C. Herzog, and Joshua D. Guttman. Strand spaces: Why a security protocol is correct? In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, pages 160–171. IEEE Computer Society Press, New York, May 1998.

[FIP77] FIPS. Data Encryption Standard. 1977. Federal Information Processing Standards Publication. Reaffirmed 1988. Superseded by FIPS 46-2 (1993), FIPS 46-3 (1999).

[FN71] R. E. Fikes and N. J. Nilsson. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence*, 2(3–4):189–208, 1971.

[FP99] Giunchiglia F. and Traverso P. Planning as model checking. In Biundo S. and Fox M., editors, *5th European conference on planning (ECP'99)*, Durham, 8-10 September 1999.

[GHK+98] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. The PDDL Planning Domain Definition Language, 1998. The AIPS-98 Planning Competition Comitee.

[GK00] Thomas Genet and Francis Klay. Rewriting for cryptographic protocol verification. In *Proceedings of CADE'00*, LNCS 1831, pages 271–290. Springer-Verlag, 2000.

[GMBPL+05] M. Garcia-Martin, M. Belinchon, M. Pallares-Lopez, C. Canales, and K. Tammi. Diameter Session Initiation Protocol (SIP) Application, March 2005. Work in Progress.

[GMTZ01] E. Giunchiglia, M. Maratea, A. Tacchella, and D. Zambonin. Evaluating Search Heuristics and Optimization Techniques in Propositional Satisfiability. In R. Goré, A. Leitsch, and T. Nipkow, editors, *Proceedings of IJCAR'2001*, LNAI 2083, pages 347–363. Springer-Verlag, 2001.

[GMW00] Ian P. Gent, Hans Van Maaren, and Toby Walsh, editors. *SAT2000. Highlights of Satisfiability Research in the Year 2000.* IOS Press, 2000.

[GPFW97] Jun Gu, Paul W. Purdom, John Franco, and Benjamin W. Wah. Algorithms for the satisfiability (sat) problem: A survey. *Satisfiability Problem: Theory and Applications*, pages 19–153, 1997.

[Har04] Sam Hartman. A Generalized Framework for Kerberos Pre-Authentication, October 2004. `http://www.ietf.org/internet-drafts/draft-ietf-krb-wg-preauth-framework%-02.txt`, Work in Progress.

[HDM04] Paul Hankes Drielsma and Sebastian Mödersheim. The asw protocol revisited: A unified view. In *Proceedings of the IJCAR04 Workshop ARSPA*, 2004. In ENTCS, available at `http://www.avispa-project.org`.

[HIR68] Peter L. Hammer (Ivănescu) and Sergiu Rudeanu. *Boolean Methods in Operations Research and Related Areas.* Number VII in Ökonometrie und Unternehmensforschung, Econometrics and Operations Research. Springer, Berlin, 1968.

[HLS00] James Heather, Gavin Lowe, and Steve Schneider. How to prevent type flaw attacks on security protocols. In *Proceedings of The 13th Computer Security Foundations Workshop (CSFW'00).* IEEE Computer Society Press, 2000.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[Hol97] G. J. Holzmann. The Spin Model Checker. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

[ISO97] ISO/IEC. ISO/IEC 9798-3: Information technology - Security techniques - Entity authentication - Part 3: Mechanisms using digital signature techniques, 1997.

[JHN03] Toni Jussila, Keijo Heljanko, and Ilkka Niemelä. Bmc via on-the-fly determinization. *Electr. Notes Theor. Comput. Sci.*, 89(4), 2003.

[JPW02] C. Jennings, J. Peterson, and M. Watson. RFC 3325: Private Extensions to the Session Initiation Protocol (SIP) for Asserted Identity within Trusted Networks, November 2002. Status: Informational.

[JRV99] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Narrowing Cryptographic Protocols. Technical Report 99-R-303, LORIA, Vandoeuvre les Nancy, December 1999. URL: `www.loria.fr/equipes/protheo/SOFTWARES/CASRUL`.

[JRV00] F. Jacquemard, M. Rusinowitch, and L. Vigneron. Compiling and Verifying Security Protocols. In M. Parigot and A. Voronkov, editors, *Proceedings of LPAR 2000*, LNCS 1955, pages 131–160. Springer-Verlag, 2000.

[Kau04] Henry A. Kautz. SATPLAN04: Planning as Satisfiability. Informal note of the International Planning Competition, 2004.

[KCK97] J. Klensin, R. Catoe, and P. Krumviede. RFC 2195: IMAP/POP AUTHorize Extension for Simple Challenge/Response, September 1997. Status: Proposed Standard.

[KCZ03] M. P. Loeb K. Campbell, L. A. Gordon and L. Zhou. The Economic Cost of Publicly Announced Information Security Breaches: Empirical Evidence from the Stock Market. *Journal of Computer Security*, 11:431–448, 2003.

[Ker] Kerberos: The Network Authentication Protocol. URL: `http://web.mit.edu/kerberos/www/`.

[KMS96] H. Kautz, H. McAllester, and B. Selman. Encoding Plans in Propositional Logic. In L. C. Aiello, J. Doyle, and S. Shapiro, editors, *KR'96: Principles of Knowledge Representation and Reasoning*, pages 374–384. Morgan Kaufmann, 1996.

[KS99] Henry A. Kautz and Bart Selman. Unifying SAT-based and graph-based planning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI'99)*, pages 318–325. Morgan Kaufmann, 1999.

[Lam94] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.

[LG00] G. Leduc and F. Germeau. Verification of Security Protocols using LOTOS – Method and Application. *Computer Communications, special issue on "Formal Description Techniques in Practice*, 23(12):1089–1103, 2000.

[LM04] Yuliya Lierler and Marco Maratea. Cmodels-2: SAT-based answer set solver enhanced to non-tight programs. In *Proc. LPNMR-04*, pages 346–350, 2004.

[Low96] G. Lowe. Breaking and Fixing the Needham-Shroeder Public-Key Protocol Using FDR. In T. Margaria and B. Steffen, editors, *Proceedings of TACAS'96*, LNCS 1055, pages 147–166. Springer-Verlag, 1996.

[Low97] Gavin Lowe. A hierarchy of authentication specifications. In *Proceedings of the 10th IEEE Computer Security Foundations Workshop (CSFW'97)*, pages 31–43. IEEE Computer Society Press, 1997.

[Low98] G. Lowe. Casper: a Compiler for the Analysis of Security Protocols. *Journal of Computer Security*, 6(1):53–84, 1998. See `http://web.comlab.ox.ac.uk/oucl/work/gavin.lowe/Security/Casper/`.

[LZ04] Fangzhen Lin and Yuting Zhao. ASSAT: computing answer sets of a logic program by SAT solvers. *Artif. Intell.*, 157(1-2):115–137, 2004.

[McD03] Andrew McDonald. A Quality of Service NSLP for NSIS, June 2003. Work in Progress.

[McM93] K. McMillan. *Symbolic model checking: An approach to the state explosion problem.* Kluwer Academic Publishers, 1993.

[MD03] J. Millen and G. Denker. MuCAPSL. In *DISCEX III, DARPA Information Survivability Conference and Exposition*, pages 238–249. IEEE Computer Society, 2003.

[Mea96] C. Meadows. The NRL Protocol Analyzer: An Overview. *Journal of Logic Programming*, 26(2):113–131, 1996. See `http://chacs.nrl.navy.mil/projects/crypto.html`.

[MH69] John McCarthy and Patrick J. Hayes. Some philosophical problems from the staindpoint of artificial intelligence. In Bernhard Meltzer and Donald Michie, editors, *Machine Intelligence. Volume IV*. Edinburgh University Press, Edinburgh, UK, 1969.

[Mil89] R. Milner. *Communication and Concurrency*. International Series in Computer Science. Prentice Hall, 1989. SU Fisher Research 511/24.

[MMS97] J. C. Mitchell, M. Mitchell, and U. Stern. Automated Analysis of Cryptographic Protocols Using Murphi. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 141–153, 1997.

[MMZ+01] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, 2001.

[MS01] Jonathan K. Millen and Vitaly Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of the ACM Conference on Computer and Communications Security CCS'01*, pages 166–175, 2001.

[Neb00] Bernhard Nebel. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research. (JAIR)*, 12:271–315, 2000.

[Nie99] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Ann. Math. Artif. Intell.*, 25(3-4):241–273, 1999.

[NS78] R. M. Needham and M. D. Schroeder. Using Encryption for Authentication in Large Networks of Computers. Technical Report CSL-78-4, Xerox Palo Alto Research Center, Palo Alto, CA, USA, 1978. Reprinted June 1982.

[NT94] B. Clifford Neuman and Theodore Ts'o. Kerberos : An authentication service for computer networks. Technical Report ISI/RS-94-399, USC/ISI, 1994.

[NYHR04] Clifford Neuman, Tom Yu, Sam Hartman, and Ken Raeburn. The Kerberos Network Authentication Service (V5), September 2004. `http://www.ietf.org/internet-drafts/draft-ietf-krb-wg-kerberos-clarific%ations-07.txt`, Work in Progress.

[Pau98] L. C. Paulson. The Inductive Approach to Verifying Cryptographic Protocols. *Journal of Computer Security*, 6(1):85–128, 1998.

[PCK99] Mukul R. Prasad, Philip Chong, and Kurt Keutzer. Why is atpg easy? In *DAC '99: Proceedings of the 36th ACM/IEEE conference on Design automation*, pages 22–28. ACM Press, 1999.

[PG86] D.A. Plaisted and S. Greenbaum. A Structure-preserving Clause Form Translation. *Journal of Symbolic Computation*, 2:293–304, 1986.

[Pos46] E.L. Post. A Variant of a Recursively Unsolvable Problem. *Bulletin of the American Mathematical Society*, 52:264–268, 1946.

[RB99] A.W. Roscoe and P.J. Broadfoot. Proving security protocols with model checkers by data independence techniques. *Journal of Computer Security*, 7:147–190, 1999.

[RHN04] Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä. Parallel encodings of classical planning as satisfiability. In *JELIA*, pages 307–319, 2004.

[RSA78] R. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[RSC+02] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. RFC 3261: SIP: Session Initiation Protocol, June 2002. Status: Proposed Standard.

[RT01] M. Rusinowitch and M. Turuani. Protocol Insecurity with Finite Number of Sessions is NP-complete. In *Proceedings of the 14th IEEE Computer Security Foundations Workshop*. IEEE Computer Society Press, 2001.

[SAT] SAT. The SAT Live Web Site. `http://www.satlive.org`.

[SBP01] Dawn Song, Sergey Berezin, and Adrian Perrig. Athena: a novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9:47–74, 2001.

[SET] SET – Secure Electronic Transaction LLC. `http://www.setco.org/`.

[Sym04] Symantec Internet Security Threat Report, 2004. Volume 5, available at `http://ses.symantec.com/ITR`.

[TLHM04] B. Tung, C. Neuman L., Zhu M. Hur, and S. Medvinsky. Public Key Cryptography for Initial Authentication in Kerberos, December 2004. `http://www.ietf.org/internet-drafts/draft-ietf-cat-kerberos-pk-init-22.%txt`, Work in Progress.

[TS01] S. Tanaka and F. Sato. A key distribution and rekeying framework with totally ordered multicast protocols. In *15th International Conference on Information Networking (ICOIN'01)*, pages 831–838. IEEE Computer Society Press, 2001.

[Tur02] Hudson Turner. Polynomial-length planning spans the polynomial hierarchy. In *JELIA '02: Proceedings of the European Conference on Logics in Artificial Intelligence*, pages 111–124, London, UK, 2002. Springer-Verlag.

[Tur03] M. Turuani. *Sécurité des Protocoles Cryptographiques: Décidabilité et Complexité*. PhD thesis, Université Henri Poincaré, Nancy, December 2003.

[VG91] Sperschneider V. and Antoniou G. *Logic, A Foundation for Computer Science*. Addison-Wesley, 1991.

[Wel00] B. Wellington. RFC 3007: Secure Domain Name System (DNS) Dynamic Update, November 2000. Status: Proposed Standard.

[ZG96] J. Zhou and D. Gollmann. A fair non-repudiation protocol. In *Proc. of the 15th IEEE Symposium on Security and Privacy*, pages 55–61. IEEE Computer Society Press, 1996.

[Zha97] H. Zhang. SATO: An Efficient Propositional Prover. In W. McCune, editor, *Proceedings of CADE 14*, LNAI 1249, pages 272–275. Springer-Verlag, 1997.

[Zor00] G. Zorn. RFC 2759: Microsoft PPP CHAP Extensions, Version 2, January 2000. Status: Informational.