POLICY COMPLIANCE, CONFIDENTIALITY AND COMPLEXITY IN COLLABORATIVE SYSTEMS

Paul D. Rowe

A Dissertation

in

Mathematics

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

2009

Andre Scedrov
Supervisor of Dissertation

Tony Pantev
Graduate Group Chairperson

Acknowledgments

There are many people who have contributed to my ability to complete this thesis, either through direct involvement with the work, or by helping to make my life a little more sane.

First and foremost I would like to thank my adviser Andre Scedrov. He support and guidance has been more than I could have hoped for. He has also introduced me to many people who have had an impact on my understanding of the ideas which appear in this thesis. Most importantly he connected me with Max Kanovich who not only helped introduce me to the topic of this thesis, but who helped guide me in its creation.

My colleagues here in the Penn math department have my utmost gratitude. Their friendship has helped to carry me through the rough times, and celebrate in the good times. I would particularly like to thank Joe-Kai Tsay for our daily conversations while we were officemates. I hope that we can one day have a real conversation in German! Thanks also to Andy Bressler with whom I have watched countless TV shows and movies, and played countless games of squash (of which I suspect he won more than I did). I would be remiss if I didn't mention by name Chris Jankowski, Colin Diemer, Dave Favero, Jen Hom, Pilar Herreros, Shea Vela-Vick, Tim Devries, Andrew Obus and Clay Shonkwiler. I am grateful also to our Quizo team. We will always be connected by the random facts that now represent the intersection of our knowledge of trivia.

Although they all live far from Philadelphia, my family has remained a continual source of strength for me. They have never doubted me even at times when I doubted myself. They have all taught me more than they probably realize.

Thanks also to the Penn math department administration: Janet, Monica, Paula and Robin. They work tirelessly every day to make sure that our lives as students run smoothly, and they do it with a smile, even in those time when we make it hard on them.

I would like to thank the numerous people who have seen parts of this work in its various stages and provided me with useful comments and suggestions: Anupam Datta, George Dinolt, Rachel Greenstadt, Joshua Guttman, Zack Ives, Pat Lincoln, John Mitchell, Jose Meseguer, Helen Nissenbaum, Tim Roughgarden, Natarajan Shankar, Paul Syverson and Steve Zdancewic.

Finally, I would like to acknowledge the financial support that I have received while at Penn. I have been partially supported by ONR Grant N00014-07-1-1039, by OSD/AFOSR MURI "Collaborative policies and assured information sharing", by OSD/ONR CIP/SW URI "Software Quality and Infrastructure Protection for Diffuse Computing" through ONR Grant N00014-01-1-0795 and OSD/ONR CIP/SW URI "Trustworthy Infrastructure, Mechanisms, and Experimentation for Diffuse Computing" through ONR Grant N00014-04-1-0725. Additional support was provided by NSF Grants CNS-0429689, CNS-0524059, and CNS-0830949.

ABSTRACT

POLICY COMPLIANCE, CONFIDENTIALITY AND COMPLEXITY IN COLLABORATIVE SYSTEMS

Paul D. Rowe

Andre Scedrov

Collaboration among organizations or individuals is common. While these participants are often unwilling to share all their information with each other, some information sharing is unavoidable when achieving a common goal. The need to share information and the desire to keep it confidential are two competing notions which affect the outcome of a collaboration. When collaborating agents share sensitive information to achieve a common goal it would be helpful to them to decide whether doing so will lead to an unwanted release of confidential data. These decisions are based on which other agents are involved, what those agents can do in the given context, and the individual confidentiality preferences of each agent.

This thesis proposes a formal model of collaboration which addresses confidentiality concerns. We draw on the notion of a plan which originates in the Artificial Intelligence literature. We use data confidentiality policies to specify the confidentiality concerns of each agent, and offer three ways of defining policy compliance. We also make a distinction between systems containing only well-balanced actions in which the pre- and post-conditions are of the same size, and general systems that contain un-balanced actions.

For each definition of policy compliance and for each type of system, we determine the decidability and complexity of scheduling a plan leading from a given initial state to a desired goal state while simultaneously deciding compliance with respect to the agents' policies.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

With information and resources becoming more distributed, interaction with external services is becoming more important. More and more software is being developed which is designed to foster collaboration through communication, resource sharing and data sharing. Collaborating agents are typically viewed as having a cooperative relationship in which they share a common goal. A common example can be found in companies which have a variety of departments working together to bring a product to market. In such situations confidentiality among departments is not always an obvious concern.

However, just because organizations or individuals are willing to collaborate does not mean they are ready to share all their information or resources. While much of the information they do share may be general and harmless, some of the information is sensitive, such as detailed sales reports at an organizational level or social security numbers and credit card numbers at an individual level. There is an interplay between achieving a collaborative goal on the one hand, and guarding sensitive information on the other. The decision about which pieces of information to share and which pieces to guard is usually based on a number of factors related to trust. Different participants can be trusted with different information in different contexts. Let us consider some examples.

When Alice eats at a restaurant she usually pays with a credit card. In order to pay for the meal she must give her card to the waiter, Bob, who takes it away for validation. She trusts the waiter, Bob, not to write down her card number or share it with anybody else. If she met Bob in a different social situation, however, she might not give him her credit card in that social context. Similarly, she doesn't give her credit card to her friends even though she may trust them. This is because her friends have no legitimate reason to learn her card number. Alice's decision to share her card number is based on necessity to achieve the goal of paying for dinner, and trust in the restaurant process. Both of these factors are dependent on the situation and the individual.

Consider another scenario where two competing companies make a temporary and partial strategic alliance. This might occur, for example, if they want to forecast the direction of the market in a collaborative way. Generally these companies will not trust each other with their business secrets, but they will have to share something to create an accurate forecast. They may first start with a contract that limits how the shared information will be used or who it can be shared with. This serves to change the context of the interaction. They build mutual trust by explicitly limiting their actions to ones which are acceptable by both sides.

Personal medical information is also quite sensitive. Hospitals take great care to ensure that information is shared and protected appropriately. There are quite detailed laws which specify exactly how a hospital may or may not distribute medical records. Some level of sharing is unavoidable. In order for a procedure to be covered by a patient's insurance company, the hospital must tell the insurance company what procedures were performed, as well as the diagnosis. Only then can the insurance company decide if it will cover the cost. But the insurance company should not be given other private information about the patient. Hospitals may also provide aggregate data on patients to students for educational purposes. This aggregate data should be appropriately sanitized for release to students. The same information, however, might not be acceptable for release to the general public.

Finally, when doing scientific research, researchers must find a balance between sharing too much information and not enough. On one hand, a principal goal of research is to obtain and publish results. On the other hand, if researchers provide their raw data to others too soon, another group may announce the results first. Data sharing is prominent in the fields of comparative genomics and systems biology where the focus is on integrating and analyzing large amounts of dynamic and growing biological data [67, 32]. Scientific data sharing also has the converse problem. Scientists often rely on data from outside sources. The choice of whether or not to use outside data depends on the source of the data. The choice also depends on whether the outside data is consistent with the local confidential data. Research groups may have informal policies, or practices that determine when they are willing to incorporate outside data into their results.

These examples serve to demonstrate that confidentiality of information is not an absolute notion. This may be viewed as a special case of privacy being dependent on context [6]. Not only does confidentiality depend on the context of an interaction, it also depends on the personal preferences of the agents involved. In these examples the agents have already implicitly or explicitly formulated an idea of which information flows are acceptable and which are unacceptable, or at least undesirable. Thus when attempting to model and analyze these interactions the confidentiality preferences or policies of the agents play a crucial role. The first question to ask about these scenarios is whether the agents are able to reach their common goal in a way that complies with each of the agents' confidentiality policies.

In this thesis we present a model of collaborative systems at an abstract level designed to act as a formal framework in which we might answer such questions. We draw on previous work in planning [11, 36, 51, 16, 21] and state transition systems and their connection to linear logic [25, 26]. We use local state transition systems in which we model local or confidential data through the use of a syntactic partition of predicate symbols. The global system configuration can then be thought of as a collection of local configurations (comprised of confidential or local facts) together with some shared public configuration (comprised of group or public facts). The confidential or local facts are only visible and accessible to the local agent (indicated syntactically), although another agent may have another copy stored in their local configuration. We force the global configuration to change incrementally through local transitions that affect at most one local configuration at a time. A collaborative plan is then a sequence of transitions which takes the system from some initial global configuration to an agreed goal configuration. In fact, there may be many configurations that satisfy some goal condition.

We assume that the agents are further constrained by their own confidentiality concerns. These are expressed explicitly as confidentiality policies, which are essentially sets of configurations that the agents deem undesirable. We call such configurations "critical". We offer three possible interpretations of the policies by giving three definitions of policy compliance, which may correspond to various levels of trust among the agents.

The first interpretation of policies is *system compliance.* A system is compliant if there is no way for the agents to reach a critical configuration. This might be appropriate for a situation in which the agents are generally untrusting such as the example above with competing companies. The contract they create can serve to limit each agent's actions in a way that makes any critical configuration unreachable. That is, any sequence of actions is safe to use because no sequence can create a critical configuration.

The second interpretation is *weak plan compliance.* A weakly compliant plan is one which avoids the critical configurations. Weak plan compliance is violated only when the agents actually reach a critical configuration while following a particular plan. This is a much weaker interpretation than system compliance which considers any possible future actions of the agents. This interpretation emphasizes the current knowledge of the agents along a single plan. Weak plan compliance is more appropriate for situations like the one above in which Alice uses a credit card at a restaurant. She trusts the waiter not to write down her card number within the restaurant process even though he can. So although a critical configuration, in which the waiter retains Alice's card number, is reachable, she trusts the waiter to follow the standard restaurant procedure and not use the actions available to him to create the critical configuration.

The final interpretation is simply called *plan compliance.* This interpretation provides an intermediate level of protection between system compliance and weak plan compliance. Intuitively, if a plan is compliant it protects those agents who follow the plan against those who may choose to deviate from the plan. In particular, if any subset of the agents deviate from a compliant plan then as long as the other agents abort, those who deviated can never reach a configuration which is viewed as critical by those who did not deviate. This may also be appropriate for situations involving competing companies. In contrast to system compliance, this interpretation considers only one plan at a time. Instead of implying that any sequence of actions is safe to use, it implies that a specific sequence is safe to use even if everybody else behaves differently.

The confidentiality policies themselves are quite flexible. In particular, they can express notions of current knowledge. For example, Alice's policy can specify that Bob should not be able to learn her *current* password. Her policy allows Bob to learn her old password as long as he does not learn her current password. This is an advantage of a state-based approach to confidentiality as opposed to the more trace-based approach of Non-Interference [60, 27, 52]. Declassification has to be handled very carefully in that setting [54, 61, 57].

Additionally, we consider a distinction between systems that only contain well-balanced actions, which have the same number of facts in both the pre-conditions and post-conditions, and general systems that allow un-balanced actions in which either the pre- or the post-conditions may have more facts. Intuitively, systems with well-balanced actions correspond to a setting in which the agents each have a database of a fixed, limited number of fields. The actions correspond to updating the fields instead of creating new fields. This restriction guarantees that the size of the global configuration remains constant throughout an execution of the system. Systems that contain un-balanced actions will have executions that grow and shrink the size of the global configuration. This distinction plays an important role in the results of this thesis.

The main results of this thesis are about the decidability and complexity of scheduling a plan which is compliant with all of the agents' confidentiality policies, that leads from an initial configuration to a goal configuration. We refer to this problem as the Collaborative Planning Problem with Confidentiality, and we determine its complexity with respect to each of the three definitions of policy compliance, considering first well-balanced systems which contain only well-balanced actions, then general systems which may contain un-balanced actions. This yields six main theorems, summarized in Table 1.1.

	System Compliance	Weak Plan Compliance	Plan Compliance
Well-Balanced Systems	PSPACE	PSPACE	PSPACE
General Systems	EXPSPACE	Undecidable	Undecidable

Table 1.1: Summary of the complexity results.

This figure helps illuminate the impact of various modeling decisions on the decidability and complexity of the Collaborative Planning Problem with Confidentiality. In particular, the problem is significantly more complex for general systems. Furthermore, in general systems the decidability is dependent upon which definition of policy compliance we use. In going from system compliance to (either version of) plan compliance in the general case, the problem goes from being decidable to being undecidable.

In addition to these decidability and complexity results, we also demonstrate the logical foundation of our approach in a variant of linear logic [25, 36]. While this is not necessary in determining the complexity of the planning problem, it relates our approach to a number of similar formalisms which have had success in the past. On the one hand, there is a wide literature on viewing the classical planning problem in logical settings [36, 49, 11, 55]. On the other hand, multiset rewriting formalisms have proved useful in the analysis of security protocols [20, 12] as well as in the field of computation [41, 38, 39, 37, 40].

Most of the results of this thesis have already been published. In particular, the reader can find the majority of this thesis in [33, 35, 34]. The original ideas were in [35], but the notion of policy compliance was only introduced in the full version of that paper [33]. Generally speaking, Chapters 3 and 5 contain the contents of [33] while Chapter 4 contains the contents of [34]. In Chapter 2 we have re-packaged the definitions from all three papers in order to unify the results.

1.2 Related Work

Before getting to the details of the current thesis, it may be useful to discuss some related work in order to understand the results that follow in their proper context. We discuss very briefly only those papers that have contributed the most to our picture of the field. It is certainly not a complete survey of all the relevant work. Generally speaking, the connections to other work can be found along two dimensions: similarity in motivation, and similarity in methods or formalism, with a sizable intersection of these two groups.

Let us begin by mentioning some work that shares similar motivations. On the applied side of the scale, there are numerous works about how to achieve confidentiality in specific settings such as scientific data sharing [32, 67], and the controlled release of medical information [68]. These approaches tend to address particular details of the situation in question, and therefore lack the abstractness necessary for capturing a wide range of scenarios.

While the focus of this thesis is on policy compliance, other authors have focused on defining metrics to measure the amount of privacy lost in specific algorithms. For example, [29, 28] analyze the privacy loss of a variety of algorithms designed to collaboratively schedule meetings. Their methods also apply to a broader class of distributed constraint optimization problems in which privacy issues may arise.

In [1, 17] the authors present a model that accounts for three types of participants: Byzantine, altruistic and rational. This is a framework meant to design and analyze protocols and processes in which some agents act in their own self interest, some can act in arbitrary ways, and some follow a deterministic set of instructions. The concept is general, but the incentives depend on the specific instance. Analyses are carried out on a case by case basis. At a more abstract level, [66] characterizes boolean functions that can be non-cooperatively computed. These are functions for which the participants will honestly state their input values to a central, trusted algorithm, assuming the agents care more about learning the outcome of the function than the input values of other agents. All of these papers incorporate explicit utility functions. While these differ from our confidentiality policies in detail, they represent similar confidentiality constraints that must be satisfied by "successful" executions.

While our notion of confidentiality depends on reaching a particular configuration or not, other notions have been considered which focus on an adversary's ability to distinguish between two traces. For example [2, 3] define secrecy in this way. This is very much in the spirit of non-interference [54, 60, 27, 52] in which secret, or High, variables should not leak to public, or Low, variables. These distinctions tend to be static and absolute, which makes them inflexible and hard to adapt to different contexts. Furthermore, the focus is on the past value of certain variables, while our confidentiality policies have the ability to express knowledge of the *current* configuration. This is one advantage of a state-based approach. Declassification must be handled very carefully in a trace-based setting [54, 61, 57]. On the other hand, in a state-based approach one must typically model all actions explicitly, while trace-based approaches consider more implicit means of reasoning.

At the intersection of work with similar motivation and work with similar methods, we come to a general, formal model for privacy in [6, 7] called Contextual Integrity. The authors focus on more than just confidentiality. They consider more general norms for the appropriate transmission of information. They consider both positive and negative norms which respectively generalize "allow" and “deny” rules in traditional access control. As the name suggests, the authors also emphasize the importance of context in determining which norms are applicable in a given situation. This determination may be imposed externally by law, or internally in the form of company privacy policies. Indeed the importance of context was brought to our attention by these papers. The underlying formalism is a different (although similar) state transition system called a concurrent game structure which provides the semantics for alternating time temporal logic [4].

Our formalism also shares some similarities with the Dolev-Yao model of security protocols [19, 20] in which a powerful intruder tries to disrupt communication between honest protocol participants. However, there are some key differences between the Dolev-Yao model and the model in this thesis which arise from differences in motivation. In the Dolev-Yao model used to analyze security protocols the attacker is much more powerful than the honest protocol participants. Our current work uses a model in which there is no external attacker, and intuitively, the participants are evenly matched, and they are more capable than the honest participants in security protocols, *e.g.* they may loop and they may have unbounded memory. This “closed-room” setting already provides interesting dynamics which are worth investigating.

We also contrast the undecidability results in this thesis with similar undecidability results in security protocol analysis [20]. While the Dolev-Yao model allows the participants and the attacker to create nonces, or fresh values, the model in this thesis does not currently allow for such fresh value creation. This is a notable difference because the undecidability results of [20] rely on the use of nonces, whereas we obtain undecidability in this work without them. In some ways, our current model is similar in spirit to some features of the Dolev-Yao model of contract signing protocols [14, 15] if *all* participants are “optimistic.”

Moving towards work with similar methods and formalisms, we point out that there may be strong connections between our complexity and undecidability results, and similar results from “classical” planning in artificial intelligence. The complexity of the planning problem has been shown to be sensitive to changes in the details of the problem statement [21, 16] regarding the use of function symbols, deletion lists and negative pre-conditions. We have not explored the possible connections and equivalences between the problems of this work and problems in classical planning.

Chapter 5 of this thesis demonstrates the connection between our formalism and affine logic which is a variant of linear logic. Linear logic has been shown to be useful in a number of areas of computer science, but or particular relevance is it usefulness in artificial intelligence and planning [36]. It has served well as a formalism for the discrete aspects of the planning problem. It has also been modified in [62] to incorporate constraints that correspond to the more continuous aspects of planning problems that one encounters in the real world. Perhaps ideas from this work could be incorporated into our formalism in a natural way.

Finally we would like to point out the similarity between our formalism and Petri nets. This connection is made more explicit in Section 4.1. However, that section only considers a very general case. It is possible that results from Petri net theory might be useful in settling future problems in the current setting of confidentiality in collaboration.

Chapter 2

Preliminaries

2.1 Defining the Model

We begin by describing the underlying structure of our model of collaboration. At a high level, the model has four main components. Being an evolving system, there must be a way of describing the configuration of the system at any given moment. Additionally, we describe how the agents transform the configurations from one to another via local actions. In our setting the agents also have some (common) goal of the collaboration, as well as (distinct) confidentiality concerns. These concerns are expressed as confidentiality policies. We now introduce each of these four components one at a time.

2.1.1 Configurations

Building the description of a configuration from the ground up, we have an underlying *signature* Σ of constants, variables, and predicate symbols. For the purposes of this thesis we assume that Σ is finite, that is it has finitely many symbols. A *fact* is a ground, atomic predicate over multi-sorted terms. Facts have the form, $P(\bar{t})$ where P is a predicate symbol and $\bar{t}$ is a tuple of terms containing no variables. A *configuration* is a multiset of facts. We use the letters $U, V, W, X, Y,$

and Z to represent multisets of facts. We take both XY and X, Y to denote the multiset union of multisets X and Y. For any fact $P(\bar{t})$, we use $P(\bar{t})^k$ to denote $\underbrace{P(\bar{t}), P(\bar{t}), \ldots, P(\bar{t})}_{\text{n times}}$ and $P(\bar{t})^0$ to indicate there are no instances of the fact $P(\bar{t})$.[1]

The global configuration is divided into different local configurations each of which is *accessible* to a specified agent. There is also a public configuration that is publicly accessible to all agents involved in the collaboration. We are considering interactions that take place in a "closed-room" setting, so we can ignore concerns about an outside intruder. The separation of the global configuration is done via a partition of the predicate symbols. We typically annotate local predicate symbols with the identity of the agent who owns it. So, for example, Alice's local facts will look like $P_A(\bar{t})$ or even perhaps just $A(\bar{t})$. The interpretation is that Alice has the information $\bar{t}$ kept locally. Any predicate symbol not explicitly annotated with an identity is public. We typically use N, R, or C as public predicates representing a public network, register or counter respectively. The predicate symbols act like fields of a database in which information may be stored, and the syntactic form of the predicate determines if the fact is local to a given agent or if it is public.

The global configuration is the multiset union of the local configurations and the public configuration:

$$Z = Z_{A_1}, Z_{A_2}, \ldots, Z_{A_n}, Z_{pub}$$

where Z_{A_i} is the local configuration of agent A_i and Z_{pub} is the public configuration. More generally, we use X_A to denote any multiset of facts all of which are local to agent A, and X_{pub} to denote any multiset of facts all of which are publicly accessible.

Let us start a running example to help demonstrate the definitions. This example is motivated by the concept of secret sharing [65]. Suppose Alice knows a secret number (say a passkey). For concreteness, let the number be 15. She also knows that Bob has an additive share of the passkey, say 7. The goal of their interaction will be for Alice to convey the secret passkey, 15, to Bob

[1] Because our signature does not have any function symbols, its finiteness implies that there are only finitely many facts which are expressible in the signature. We could alternatively allow function symbols but fix the nesting depth of function application. This would have the same effect and only Lemma 3.1.1 would change.

without revealing the secret in the public configuration (perhaps Charlie is also in the room).

We will say that the triple $(A, 15, pky)$ means that "Alice's passkey is 15", and that the triple $(B, 7, share)$ mean that "Bob's share is 7". Similarly, $(A, 8, share)$ means that "Alice's share is 8". Then the initial configuration W of the system might look like the following:

$$W \quad = \quad A_1(A, 15, pky), A_2(B, 7, share), B_1(B, 7, share)$$

That is, Alice knows both her passkey and Bob's share, while Bob only knows his share. If Charlie also begins with some information then we would need to explicitly add more facts to the configuration. Before continuing the example we must explain how the agents can transform the configuration.

2.1.2 Local Actions

Each agent has a set of actions which transform the global configuration. Under the interpretation of the local predicates as accessibility restrictions it is natural to restrict the actions to conform to that interpretation. Thus the actions are local in the sense that each action can only depend on the local facts of at most one agent. They must have the form $X_A X_{pub} \rightarrow_A Y_A Y_{pub}$, where the agent who owns the action (indicated by the subscript on the arrow) is the same agent who owns any local facts mentioned (on both the right and left of the arrow). Another way of viewing this restriction is that agent A does not have either read or write access to any of the predicates which are local to other agents. Figure 2.1 gives a graphical representation of this locality restriction.

The actions work like re-writing rules. $X_A X_{pub}$ are the pre-conditions of the action and $Y_A Y_{pub}$ are the post-conditions of the action. The pre-conditions must be present in the configuration for the action to be *enabled*. By applying the action, the pre-conditions are erased and replaced with the post-conditions. The rest of the configuration remains untouched. Thus, we can apply the action $X_A X_{pub} \rightarrow_A Y_A Y_{pub}$ to the global configuration $V X_A X_{pub}$ to get the global configuration $V Y_A Y_{pub}$. Since several actions may be enabled simultaneously, we assume the actions are

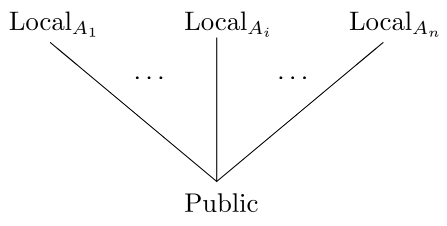


Figure 2.1: The flow of information with local actions.

applied nondeterministically.

Returning to our running example, if Alice wants to convey her secret passkey to Bob, she should publish her share of the passkey. That is, she should use the following action:

$$r_1 : A_1(A, 15, pky), A_2(B, 7, share) \quad \rightarrow_A \quad A_1(A, 15, pky), A_2(B, 7, share), N(A, 8, share)$$

Notice that this action only makes reference to her local predicates A_1 and A_2 and to the public predicate N. Also, in order for Alice to remember the passkey it also needs to appear in the post-conditions, otherwise it will be erased from her local configuration. Applying this action to the initial configuration W from above yields the configuration

$$A_1(A, 15, pky), A_2(B, 8, share), N(A, 7, share), B_1(B, 8, share).$$

In contrast to the above example which uses a *propositional action* which does not contain any free variables, we also allow *first-order actions* in which all free variables are implicitly universally quantified. A first-order action is enabled if there is a substitution of constants for variables in which the resulting propositional action is enabled. There may be several substitutions which enable the first-order action. Again, we assume the substitution is chosen nondeterministically.

Although some or all of the predicates in a first-order action will contain free variables, we will often abuse terminology and call these predicates facts. So for example, when we count the number of facts in the pre-condition of a first-order action, we are really counting the number of facts that occur under any substitution. Since this number does not change under different substitutions, there is no ambiguity.

Given a set of first-order actions, it is possible to propositionalize them by applying every possible substitution. If the underlying signature Σ is finite (which we assume in this work), then this process will expand the size of the set of actions exponentially. We will be explicit about whether an action is first-order or propositional whenever the distinction is important. In all other cases we will simply use the word action.

There is a small detail that deserves some attention. In the standard Dolev-Yao model, the intruder is always assumed to have the same set of basic actions which includes reading any message from the network. This is a useful assumption because it considers the worst-case scenario. In our setting, we do not force every agent to have such actions, although we certainly do not preclude such a situation. Thus, if a term appears in a public predicate, it may not be accessible to every agent. The idea is to have a system that is as general as possible by limiting the number of assumptions we have about it. Although we, as modelers, do not assume the agents have all the Dolev-Yao actions, the agents themselves might assume the others *do* have those actions. This assumption could be reflected their decision about which configurations are considered to violate their confidentiality (see Section 2.1.4).

2.1.3 Goal Configurations and Reachability

Since we are considering situations in which the agents are collaborating, we need a way of expressing the goals of collaboration as well as the reachability of those goals. Naturally, the reachability of a configuration depends on what actions are available to the agents. That is, systems with different actions behave differently, so we need a way to specify which system we are studying.

Definition 2.1.1. A *local state transition system* $T = (\Sigma, I, R)$ is a triple where Σ is the signature of the underlying language, $I = \{A_1, \ldots, A_n\}$ is a finite set of agents and R is a set of local actions owned by those agents. For convenience we let R^A denote the set of A's local actions. Hence $R = \bigcup_{A \in I} R^A$.

We often use the abbreviation LSTS for local state transition system, as well as abbreviating it to the one word "system". Once we know the system T, that we are studying we can specify what it means for one configuration to be reachable from another within that system. We let $X \rhd_T^n Y$ indicate that the configuration Y is exactly reachable from the configuration X by applying n actions from the system T. That is, using actions from T the system can transform from X into Y without any extra facts. $X \rhd_T^* Y$ means Y is exactly reachable from X by applying 0 or more actions. We will later need to talk about configurations which are reachable using the actions from all agents except for one. Thus we write $X \rhd_{-A_i}^* Y$ to indicate that Y can be reached exactly from X without using the actions of agent A_i, (the system T is implied). We also write $X \rhd_t^1 Y$ to mean that an action labeled by t performs the transformation in one step. We may drop the subscript T if the system is clear from the context.

Typically the agents will not care about the entire description of a goal configuration. It will usually be sufficient for a configuration to contain certain facts. For example, in contract negotiations the text of the contract is not known at the outset, but the goal is to have a signed contract. For this reason the notion of a partial goal is better suited for this setting. For this purpose we use the notation $X \leadsto_T^* Y$ to indicate that a configuration *containing* Y is reachable from X by applying 0 or more actions from the system T. That is, there is some multiset of facts U such that $X \rhd_T^* YU$. Similarly, we write $X \leadsto_{-A_i}^* Y$ to indicate that this partial goal is reachable without using the actions of A_i, and $X \leadsto_t^1 Y$ to indicate that action t performs the transformation in one step. In this thesis, all goals are partial goals, and reachability is interpreted in this weaker sense unless otherwise stated.

Definition 2.1.2. A *collaborative plan based on* T which leads from an initial configuration W to a partial goal configuration Z is a sequence of configurations $X_1; X_2; \ldots; X_n$ such that $X_1 = W$, $Z \subseteq X_n$ and $X_i \rhd_T^1 X_{i+1}$ for $1 \leq i < n$.

We also say that the configurations X_i are *contained* in the plan. We may also say that an action r is contained in the plan if r is the action that witnesses $X_i \rhd_T^1 X_{i+1}$ for some i. In other words, r is contained in the plan if $X_i \rhd_r^1 X_{i+1}$ for X_i and X_{i+1} contained in the plan. Notice that to say there exists a collaborative plan leading from W to Z is the same as stating $W \rightsquigarrow_T^* Z$. Similarly, we say that $W \rhd_T^* Z$ means that there is a collaborative plan leading from W exactly to Z. For this exact version of a plan, we simply require that $Z = X_n$ instead of $Z \subseteq X_n$.

Let us return to our running example in which Alice wants to convey her passkey to Bob. Recall, Alice has an action (labeled r_1) that publishes her share of the passkey. We assume that Bob has the following action that helps him recreate the passkey.

$$r_2 : B_1(B, 8, share), N(A, 7, share) \quad \rightarrow_B \quad B_2(A, 15, pky)$$

Also let Charlie have the first-order action which reads information off the network.

$$r_3 : N(\bar{x}) \quad \rightarrow_C \quad C_1(\bar{x})$$

We can now consider the system in which Alice, Bob and Charlie have actions r_1, r_2 and r_3 respectively. That is $T = (\Sigma, \{A, B, C\}, \{r_1, r_2, r_3\})$ for some appropriate signature Σ. It is clear that applying r_1 and r_2 in that order demonstrates that

$$A_1(A, 15, pky), A_2(B, 8, share), B_1(B, 8, share) \quad \rightsquigarrow_T^* \quad B_2(A, 15, pky)$$

In other words, there is a collaborative plan leading from the given initial configuration to the goal $Z = B_2(A, 15, pky)$. Also, since $N(A, 15, pky)$ can never appear we know that $C_1(A, 15, pky)$ is unreachable from the initial configuration. That is, no reachable configuration contains the fact $C_1(A, 15, pky)$. Thus, in this instance, Alice can successfully convey her passkey to Bob without Charlie learning it.

2.1.4 Confidentiality Policies

Throughout a typical collaboration, some information must be revealed and other information should be guarded. Also, the information which is shared is often only shared selectively with the agents who need it to successfully achieve the goals of collaboration.

In formalizing what it means for Alice's secrets to leak, we first need to know where Alice is willing to let her information flow. We will assume that each agent has a data confidentiality policy which specifies which pieces of data other agents are prohibited from learning. In order to motivate the definitions which follow, we consider a scenario from the medical field.

Scenario: Medical Test.

Consider a patient at a hospital who needs a medical test performed in order for her doctor to provide a diagnosis and possibly prescribe appropriate medication. Such a task will involve not only the patient and her doctor. It will also involve a nurse to draw the blood, a lab technician to perform the test, and a receptionist at the front desk to take care of the paperwork. Although there may be other agents typically involved in this task (such as an insurance company) we will limit ourselves to these agents for simplicity. Our focus in this scenario is on which pieces of information the patient is willing to share with the different agents, and in what combination.

In order for the test to be performed, the patient will request and schedule a test with the receptionist. The receptionist anonymizes the patient by giving her an ID number. The patient then goes to the nurse (with her ID number, not her name) to get a test sample taken. The nurse sends the test sample to a lab where the test is performed and the result is determined. The lab can then pass on the result to the doctor. We assume here that the doctor can recover the patient's name from the ID number. At this point the doctor can make a diagnosis and prescribe the appropriate medication.

We assume the patient has some idea of which information should not be learned by each of the agents. This information is specified in the patient's data confidentiality policy, a partial example of which can be found in Figure 2.2. For example, the receptionist has no need to learn the patient's name in combination with the test result. Similarly, the point of anonymizing the patient is so that the nurse does not learn both the patient's name and ID number together.

Agent	**Critical Information**
Receptionist	(Name, Test Result), ...
Nurse	(Name, ID ♯), ...
Lab Technician	(Name, Test Result), ...
Doctor	*No Restrictions*

Figure 2.2: Part of a patient's data confidentiality policy.

Although the lab technician must learn the result of the test, she should not know who the result pertains to. That is, she should not know both the name and the test result together, even though she can learn both the ID number and test result. In this scenario we assume the patient has no problem with the doctor learning any medical information.

The reader should not be concerned with the specifics of this scenario, but rather with the expressivity of data policies. The patient's policy is some way of indicating which configurations she considers as a violation of her confidential information. Although we only listed part of the patient's policy in Figure 2.2, each agent may have a similar policy. Indeed, there may be some external policy which is mandated by law.

Definition 2.1.3. The *confidentiality policy* of an agent is a set of partial configurations that the agent views as undesirable or bad. A configuration is called *critical for A* if it contains one of the partial configurations from A's policy. A configuration is simply called *critical* if it is critical for some agent.

Notice that since policies only specify which configurations must *not* occur, no conflict arises when combining policies. The combination of two policies simply specifies the union of their critical configurations.

When modeling real situations these policies serve as constraints in the model, which may not represent real constraints in the actual system. For example a policy which is mandated by law will impose real restrictions on the system, while a patient's policy may simply represent her personal preferences. As they are currently defined, policies only refer to the location of information and not to the manner in which it is transmitted. This may not be quite enough to capture real-world policies. For instance, a nurse may have access to sensitive health information, but often only the patient's doctor is allowed to convey that information to the patient. Exploring alternative definitions for policies may be a possible avenue for future research.

Notice that these policies can be very expressive. Returning once again to our running example, Alice may not want Bob to know her *current* passkey, but she does not mind if he learns her old passkey. Then her confidentiality policy may list the configuration $A_1(A, x, pky), B_2(A, x, pky)$ as critical. Notice that the variable x must be instantiated with the same constant. Thus a configuration containing $A_1(A, 21, pky), B_2(A, 15, pky)$ in which Bob's knowledge of Alice's passkey is not current, will not be critical for Alice. In this way, if Alice replaces her old action r_1 with the following action

$$r'_1 : A_1(A, 15, pky), A_2(B, 8, share) \quad \rightarrow_A \quad A_1(A, 21, pky), A_2(B, 8, share), N(A, 7, share)$$

in which she changes her passkey from 15 to 21 as she publishes her share of the old passkey, then Bob will still be able to reconstruct $B_2(A, 15, pky)$ without violating Alice's policy.

2.2 Well-Balanced Systems

In our running example from the previous section, Alice's and Bob's actions change the number of facts in the configuration. That is, when Alice publishes her share of her passkey, she adds a fact to the configuration. When Bob combines the two shares, he takes one fact away from the configuration. Charlie's action, on the other hand, replaces a public fact with a local fact. This difference among the actions motivates the following definition.

Definition 2.2.1. A *well-balanced* action has the same number of facts in the pre-condition as the post-condition, counting multiplicity. If an action is not well-balanced then it is *un-balanced.*

Additionally, actions with at least as many facts in the pre-condition than in the post-condition are called *non-lengthening*.

We then extend this definition to describe entire systems.

Definition 2.2.2. A system is called *well-balanced* if every action of the system is well-balanced. Otherwise, the system is called *un-balanced*.

Well-balanced actions do not change the number of facts in the global configuration when they are applied. Un-balanced actions either increase or decrease the number of facts in the global configuration. For this reason, well-balanced systems are useful in modeling situations with a fixed amount of total memory. Intuitively, this restriction forces each agent to have a buffer or database of a fixed size before the collaboration. The agents may update values in this buffer and erase values to leave empty slots, but they may not change the size of the buffer. Although using only well-balanced actions forces us to fix the number of fields of this database, there is no *a priori* bound on the number of fields we may choose.

In reality, restricting to well-balanced systems is more flexible than that. There is one global buffer of a fixed size, and the agents are free to release some fields for use by the group and claim others from the group. The model allows for a potential fight for space resources which could result in a form of denial of service. Since we are considering situations in which the agents are mostly cooperative and since our security focus is on the inappropriate release of information, we do not explore this dynamic of well-balanced systems.

While at first well-balanced systems might seem very restrictive compared to general systems which allow un-balanced actions, in practice we are still able to model most scenarios in a natural way using well-balanced systems. It is possible to transform any un-balanced action into a well-balanced one. Assume that our language has a special constant $*$. A predicate of the form $N(*)$ can be interpreted as saying that the field N is empty. We can then take any un-balanced action and "pad" the side that has fewer facts with instances of $N(*)$. For instance, if the pre-condition has fewer facts than the post-condition, we add instances of $N(*)$ until they have the same number of facts.

Let us apply this construction to our secret sharing example to see how it works. Alice's action may be rewritten in the following well-balanced form.

$$r_1'' : A_1(A, 15, pky), A_2(B, 8, share), N(*) \quad \rightarrow_A \quad A_1(A, 15, pky), A_2(B, 8, share), N(A, 7, share)$$

Thus, in order for the action to be enabled, there needs to be an empty slot on the network. Similarly Bob's action can be re-written as follows:

$$r_2' : B_1(B, 8, share), N(A, 7, share) \quad \rightarrow_B \quad B_2(A, 15, pky), N(*)$$

We assume Charlie retains is well-balanced action r_3.

Then if we take $T' = (\Sigma, \{A, B, C\}, \{r_1'', r_2', r_3\})$, the goal $Z = B_2(A, 15, pky)$ is no longer reachable from the initial configuration $W = A_1(A, 15, pky), A_2(B, 8, share), B_1(B, 8, share)$. This is because the plan from W to Z contains a configuration with four facts, but the well-balanced system T' can only create plans starting at W that have three facts in every configuration, because the configuration size is fixed in advance. However, if we consider $W' = W, N(*)$ then by applying r_1'' followed by r_2' we find that $W' \leadsto^*_{T'} Z$.

This is not a fluke. If Z is reachable from W in a general system, then we can always apply the above construction to get a well-balanced system, and pad W with enough instances of $N(*)$ to get W' so that Z is reachable from W' in the well-balanced system. That is the exactly what the following proposition says.

Proposition 2.2.3. *Let T be a general system, and let T' be the well-balanced system resulting from the above construction. If $W \leadsto^*_T Z$, then there is a natural number k such that $W, N(*)^k \leadsto^*_{T'} Z$.*

Proof. Let $X_1; X_2; \ldots; X_n$ be the configurations of the plan in T which leads from W to Z. Thus $W = X_1$ and $Z \subseteq X_n$. Let p_i be the number of facts in configuration X_i, and let $m = \max\{p_i\}$. Finally let $k = m - p_1$. We will show that $W, N(*)^k \leadsto^*_{T'} Z$.

To prove this we will use the sequence of actions in T' that correspond to the actions of T used in the plan there. We show that this yields a sequence of configurations $X'_1; X'_2; \ldots; X'_n$ where $X'_i = X_i, N(*)^{m-p_i}$ for each $1 \leq i \leq n$. This is true by construction for X'_1.

Now assume it is true for X'_i. If $r : V \to U$ is the action applied at X_i in the plan in T, then we first show that we can apply the corresponding well-balanced action r' at X'_i to get X'_{i+1} which satisfies the desired condition. There are two cases. The first case is when r' is $V \to U, N(*)^\ell$. Then r and r' have the same pre-conditions. The action r' is enabled at X'_i because $X_i \subseteq X'_i$ by the inductive hypothesis. When we apply r' to X'_i, we get $X_{i+1}, N(*)^{m-p_i+\ell}$. Because ℓ is the number that makes r' well-balanced, $\ell = p_i - p_{i+1}$. Thus, $X'_{i+1} = X_{i+1}, N(*)^{m-p_{i+1}}$ as desired.

The second case is when r' is $V, N(*)^\ell \to U$. Then for r' to be enabled we must check that $\ell \leq m - p_i$. This must be true because otherwise, the action r would add more than $m - p_i$ facts to X_i making X_{i+1} have more than m facts. This contradicts m being the maximum number of facts in a configuration of the plan. Thus r' is enabled at X'_i and applying it we get $X'_{i+1} = X_{i+1}, N(*)^{m-p_i-\ell}$. We can easily check that $\ell = p_{i+1} - p_i$ in this case, which means that $X'_{i+1} = X_{i+1}, N(*)^{m-p_{i+1}}$ as desired.

Therefore, in the well-balanced system T' we have found a plan from $W, N(*)^k$ to X'_n. But $Z \subseteq X_n \subseteq X'_n$ and so we have shown that $W, N(*)^k \rightsquigarrow^*_{T'} Z$. ■

2.3 Defining Policy Compliance

The agents' confidentiality policies are simply sets of configurations which the agents want to avoid. However, they may want to avoid them in a looser or stricter sense depending on the level of trust the agents have amongst themselves. We therefore propose several ways to define policy compliance. In this section we describe each of these interpretations and state the corresponding computational problems.

Definition 2.3.1 (System Compliance)**.** A system in initial configuration W is called *compliant* if the set of reachable configurations from W contains no configuration which is critical for any agent.

Compliant systems may be viewed as well-designed systems. Each agent has a guarantee that the actions of the system do not allow their critical configurations to be reached, whether by the malicious collusion of other agents or by the careless use of their own actions. A compliant system dispenses with all confidentiality concerns before the collaboration even starts, and the agents can then focus on the separate problem of finding a plan which leads to a goal configuration. Notice, however, that system compliance depends on the initial configuration of the system. It is possible for a system to be compliant in initial configuration W, and not be compliant in initial configuration $\widehat{W}$. Let us consider the simplest example of how system compliance might fail.

Suppose agent A has a term t which should never appear in the public setting. Then A's policy will indicate that any configuration containing $N(t)$ is critical since the predicate N is publicly accessible. However, A might have the first-order action $P_A(x) \rightarrow_A N(x)$ which publishes any value from the predicate P_A. In this case, the system is definitely not compliant when starting in an initial configuration which contains $P_A(t)$. Since A has the ability to publish t, she can single-handedly create a configuration which is critical for her.

Thus system compliance is a very strong notion. It protects each agent not only from any possible sequence of actions other agents can take, it also protects each agent from their own actions. This interpretation is most appropriate when the agents share a very low level of trust. For example, a company may ask employees or external collaborators to sign a non-disclosure agreement. This effectively deletes a whole class of actions from the system, severely restricting the reachability space. However, for many scenarios this interpretation of policies might be too strong. For example, Alice may want to include the action $P_A(x) \rightarrow_A N(x)$ in order to publish other pieces of information. For this reason it is useful to consider weaker definitions that can still provide some level of protection. We consider two other notions: plan compliance and weak plan compliance. We first introduce the weak version to motivate the stronger version.

Definition 2.3.2 (Weak Plan Compliance)**.** A plan is said to be *weakly compliant* if none of the configurations contained in the plan are critical for any agent.

This definition may be appropriate for a set of agents who are mutually trusting. If a system has a compliant plan leading to a goal then each agent knows two things. First, they know that there is plan leading from the initial state to the goal. Secondly, they know that none of their critical configurations will be reached *as long as everybody follows the plan.* This means that the agents must trust each other to follow the plan because if they deviate from the plan, then all guarantees of compliance are lost. To say that a system contains a weakly compliant plan leading to a goal configuration is simply to say that goal reachability does not require any agent to pass through one of their critical configurations.

This definition emphasizes the path that is taken to reach the goal. A policy is violated only when a critical configuration is actually reached along the particular plan being considered. This is a big difference from system compliance in which policies are violated as long as a critical configuration is reachable. In other words, system compliance focuses on the reachable configurations themselves. Weak plan compliance focuses on the particular way in which a configuration is reached. We will emphasize this difference when considering the decidability and complexity results. We can also state how system compliance and weak plan compliance are related.

Lemma 2.3.3. *A system T in initial configuration W is compliant if and only if every plan in the system which starts at W is weakly compliant.*

Proof. The forward direction is immediate. If the system is compliant, then no critical configuration is reachable from W. Therefore along any given plan in the system which starts at W, its configurations will not be critical. For the reverse direction we show the contrapositive. Suppose that $W \rhd_T^* Z$ where Z is critical. Then there is a plan in T from W which leads (exactly) to Z. Since this plan ends in a critical configuration, it is not weakly compliant. ■

A natural situation where weak plan compliance might be appropriate is when Alice uses a credit card to pay for a meal at a restaurant. She certainly does not want the waiter to write down her credit card number to use later, but there is no mechanism in place to prevent the waiter from doing so. In this scenario Alice can pay for her meal without the waiter learning her card number, but she must trust him not to write it down. A proper formalization of this system would not be compliant according to Definition 2.3.1, since it would contain an action in which the waiter copies the credit card number. However the system does have a weakly compliant plan which allows Alice to pay for her meal, in which the waiter does not write down the card number.

In order for the notion of weak plan compliance to be appropriate the agents need to have a level of trust that will not be present in more sensitive settings, so using the definition of weak plan compliance may be too weak. For this reason we introduce our last definition of policy compliance which provides an intermediate level of protection between system compliance and weak plan compliance.

Definition 2.3.4 (Plan Compliance)**.** A plan is said to be *compliant* if it is weakly compliant and if for each agent A_i, and for each configuration Z contained in the plan, whenever $Z \rhd^*_{-A_i} V$, then V is not critical for A_i.

A compliant plan gives each agent a stronger guarantee than a weakly compliant plan. Let us consider this guarantee from the perspective of the single agent A. First, she knows that the plan itself does not contain any configurations critical for her because it is also weakly compliant. Secondly, she knows that starting from any configuration Z in the plan, all configurations which are reachable from Z using only the other agents' actions will also not be critical for her. These guarantees hold for each agent.

Thus each agent knows that *as long she follows the plan* the other agents cannot collude to create a configuration critical for her. The agents no longer have to trust one another. As soon as one agent deviates from the plan, the other agents may choose to stop their participation. They can do so with the knowledge that the remaining agents will never create a configuration critical for those agents that aborted.

Such a plan has the flavor of a Nash equilibrium in game theory. No agent has any incentive to deviate from the given plan. If agent A does deviate, she only introduces the possibility that the other agents can now collude to create a configuration critical for her, because the guarantee of compliance only exists for the specified plan. Exploring the connection between Nash equilibria and this current interpretation of policies is an interesting avenue of investigation, however, we leave it for future research as it will likely involve some modifications of our formalism.

We give a similar connection between system compliance and plan compliance as we did between system compliance and weak plan compliance.

Lemma 2.3.5. *A system T in initial configuration W is compliant if and only if every plan in T starting from W is compliant.*

Proof. In the forward direction, if the system is compliant then whenever $W \rhd_T^* V$ we know that V is not critical. Now consider a plan starting from W. It is weakly compliant by Lemma 2.3.3. Now let Z be a configuration of that plan. If $Z \rhd_{-A_i}^* V$ we can conclude that $W \rhd_T^* V$ and thus V is not critical. Since this is true for any agent A_i, the plan is compliant. For the other direction we simply note that if every plan in T starting at W is compliant then every plan in T starting at W is also weakly compliant. Thus by Lemma 2.3.3, the system is compliant. ■

These definitions differ somewhat from our original confidentiality condition found in [35]. We used the following definition instead.

Definition 2.3.6. We say that a local state transition system in initial configuration W, *protects the privacy of agent A* if every term t which, in the initial configuration W, occurs only in local predicates of A, also occurs only in local predicates of A in any reachable configuration.

In fact, we can express this condition as a data confidentiality policy which is entirely determined by the initial configuration. We use system compliance to determine whether this condition is met. The critical configurations of this policy are those configurations which contain predicates of the form $P_B(t, \bar{u})$, where t is a term which is to be protected, B is an agent not equal to A, and $\bar{u}$ is an arbitrary (and possibly empty) tuple of terms. Similarly, configurations containing $P'(t, \bar{u})$ would be critical. In particular, this policy does not distinguish between other agents. We assumed that agent A was unwilling to reveal any secret term t to any agent.

In the end we considered this condition to be too restrictive, so we generalized the notion to data confidentiality policies which are much more flexible. This is also what caused us to consider the weaker notions of plan compliance and weak plan compliance. Even somewhat liberal policies may be too strong to satisfy system compliance. In Section 3.3 we discuss the effect these changes had on our complexity results and show that Corollary 6.6 in [35] is actually a corollary of Theorem 3.3.1 in this thesis.

2.4 The Computational Problems

Now that we have introduced all the relevant definitions we can state the computational problems for which we would like to determine the computational complexity. They can each be considered special cases of the following general problem.

The Collaborative Planning Problem with Confidentiality. Let T be a local state transition system with a finite set of actions $\mathcal{T}$, in initial configuration W, with a finite set of goals $\mathcal{G}$, and a finite set of critical configurations $\mathcal{C}$. Determine whether T has a plan leading from W to one of the goals in $\mathcal{G}$, which complies with all the agents' confidentiality policies.

We can make two independent choices. First, we must choose which definition of policy compliance we would like to use. Second, we choose whether or not to assume the system T

is well-balanced. This yields six computational problems which we list below. We give each problem a name which is provided in parentheses.

Problem 1 (Well-Balanced System Compliance). *Let T be a well-balanced local state transition system with a finite set of actions $\mathcal{T}$, in initial configuration W, with a finite set of goals $\mathcal{G}$, and a finite set of critical configurations $\mathcal{C}$. Determine whether the system T is compliant in configuration W and if $W \rightsquigarrow_T^* Z$ for $Z \in \mathcal{G}$.*

Problem 2 (Well-Balanced Weak Plan Compliance). *Let T be a well-balanced local state transition system with a finite set of actions $\mathcal{T}$, in initial configuration W, with a finite set of goals $\mathcal{G}$, and a finite set of critical configurations $\mathcal{C}$. Determine whether the system T has a weakly compliant plan leading from W to some $Z \in \mathcal{G}$.*

Problem 3 (Well-Balanced Plan Compliance). *Let T be a well-balanced local state transition system with a finite set of actions $\mathcal{T}$, in initial configuration W, with a finite set of goals $\mathcal{G}$, and a finite set of critical configurations $\mathcal{C}$. Determine whether the system T has a compliant plan leading from W to some $Z \in \mathcal{G}$.*

Problem 4 (General System Compliance). *Let T be a local state transition system with a finite set of (possibly un-balanced) actions $\mathcal{T}$, in initial configuration W, with a finite set of goals $\mathcal{G}$, and a finite set of critical configurations $\mathcal{C}$. Determine whether the system T is compliant in configuration W and if $W \rightsquigarrow_T^* Z$ for $Z \in \mathcal{G}$.*

Problem 5 (General Weak Plan Compliance). *Let T be a local state transition system with a finite set of (possibly un-balanced) actions $\mathcal{T}$, in initial configuration W, with a finite set of goals $\mathcal{G}$, and a finite set of critical configurations $\mathcal{C}$. Determine whether the system T has a weakly compliant plan leading from W to some goal $Z \in \mathcal{G}$.*

Problem 6 (General Plan Compliance). *Let T be a local state transition system with a finite set of (possibly un-balanced) actions $\mathcal{T}$, in initial configuration W, with a finite set of goals $\mathcal{G}$, and* *a finite set of critical configurations $\mathcal{C}$. Determine whether the system T has a compliant plan leading from W to some goal $Z \in \mathcal{G}$.*

Chapter 3

Complexity in Well-Balanced Systems

In this chapter we determine exactly the complexity class of Well-Balanced System Compliance, Well-Balanced Weak Plan Compliance, and Well-Balanced Plan Compliance. In Section 3.2 we demonstrate a PSPACE lower bound for these problems. Section 3.3 demonstrates a PSPACE upper bound for all three of these problems. Finally, in Section 3.4 we show that these problems can all be solved in polynomial time if you fix a finite signature in advance. We begin, however, by carefully describing how to represent the inputs to the problems. With only a few exceptions, the results in this chapter can be found in [33]. We bring attention to those results that appear for the first time in this thesis.

3.1 Representing the Input

In determining the complexity of a problem, one needs to understand how an instance of the problem is represented. It is especially important for this chapter because the overall complexity of our algorithms in Section 3.3 vary significantly depending on how we measure the size of the input parameters. Let us start by exploring why this is so.

Consider two hypothetical algorithms. The first algorithm assumes that the actions in the system are represented by a direct binary encoding of the set of all propositional instances of the actions, and it works in polynomial space, $p(n)$. The second algorithm works directly with a binary encoding of the first-order actions, and it also works in polynomial space, $p(n)$ (where p is the *same* polynomial). These algorithms actually have very different complexities when both are measured with respect to the size of the set of first-order actions. The difference is hidden in the fact that when a set of first-order actions is propositionalized, its size may increase exponentially. Thus if the first-order representation has size n, then the corresponding propositional representation could have size 2^n. Viewed in this way, the first algorithm works in space $p(2^n)$ which is exponential in n, not polynomial. Thus the second algorithm is more efficient than the first algorithm with respect to the size of the first-order actions.

For this reason, the most efficient algorithms are those which are efficient for the most compact representation of the inputs. Thus, instead of using a direct binary encoding of the propositional instances of the actions, we assume we have some program, $\mathcal{T}$, which recognizes the set of propositional actions of the system T. That is, we assume that $\mathcal{T}$ is a program satisfying $\mathcal{T}(r) = 1$ if $r \in R_T$, and $\mathcal{T}(r) = 0$ otherwise. We make no other assumptions about the program $\mathcal{T}$, so in particular, $\mathcal{T}$ could be the smallest such program. This amounts roughly to using the Kolmogorov descriptive complexity of the set of actions [45].

We rely on same idea when representing the goal configurations and the critical configurations. That is, we take the smallest program $\mathcal{G}$ such that $\mathcal{G}(Z) = 1$ if Z contains a goal, and $\mathcal{G}(Z) = 0$ otherwise. We also let $\mathcal{C}$ be the smallest program such that $\mathcal{C}(Z) = 1$ if the configuration Z is critical for some agent, and $\mathcal{C}(Z) = 0$ otherwise. In order for our algorithms to run in PSPACE, we must assume that each of these programs $\mathcal{T}, \mathcal{G}$, and $\mathcal{C}$ runs in space which is polynomial in the size of their inputs.

That brings us to the last input to our problems: the initial configuration W. For this input we will use a direct binary encoding. The size of its description depends on two main factors. The first is the number of facts (including multiplicity) that are present in a configuration. Since we are in the well-balanced case, this will be a fixed number m for every configuration. The second factor is the size of a description of a fact. If we let S_T denote the number of facts expressible in the finite signature Σ, then this will be on the order of $\log_2(S_T)$. Thus a direct binary encoding of a configuration can be done in $O(m \cdot \log_2(S_T))$ space, since each fact requires $O(\log_2(S_T))$ space and the configuration has m facts. Thus, both $\mathcal{G}$ and $\mathcal{C}$ work in space which is polynomial in $|W|$, since they must actually look at a configuration to decide if it is a goal or if it is critical respectively.

We can similarly represent each propositional instance of an action in $O(2m \cdot \log_2(S_T))$ space since an applicable action is just a list of two configurations each requiring space $O(m \cdot \log_2(S_T))$. Therefore $\mathcal{T}$ also works in space which is polynomial in $|W|$, since it must actually look at a propositional action to decide whether it is part of the system or not.

Notice that the size of W depends on the signature Σ that we use. We therefore give an upper bound for S_T in terms of aspects of Σ. We also give an upper bound, $L_T(m)$ for the number of configurations expressible in Σ with at most m facts. We use $L_T(m)$ because it has a nice formula and as the following lemma shows, we can say that $|W| = O(\log_2(L_T(m)))$. Thus even though it overestimates the number of configurations we must consider, it does not overestimate it by too much. We omit the proof because it follows from some standard counting arguments, and from established formulas for counting multisets.

Lemma 3.1.1. *Given a local state transition system T over a finite signature Σ,*

(i) *let S_T be the total number of facts in signature Σ, and*

(ii) *let $L_T(m)$ denote the total number of configurations in signature Σ whose number of facts (counting repetitions) is bounded by m.*

Then

(a) *$S_T \leq (J \cdot D^a)$, where:*

  (a1) *J is the total number of predicate symbols in signature Σ,*

  (a2) *a is an upper bound of their arity, and*

  (a3) *D is the total number of constants in signature Σ.*

(b) *For any m:*

$$L_T(m) \leq \begin{pmatrix} m + S_T \\ \\ m \end{pmatrix} = \frac{(m+S_T)!}{m! \cdot (S_T)!} \leq \min\{(S_T+1)^m, (m+1)^{S_T}\}.$$

Before moving on the main theorems, we discuss a small point regarding our ability to produce a plan if it exists. Although all six of our problems are stated as decision problems, in the well-balanced case we prove more than just PSPACE decidability. Ideally we would also be able to generate a plan in PSPACE when there is a solution. Unfortunately, the number of actions in the plan may already be exponential in the size of the inputs, precluding PSPACE membership of plan generation. For this reason we introduce the notion of "scheduling" a plan in which an algorithm will also take an input i and output the i-th step of the plan.

Definition 3.1.2. An algorithm is said to *schedule* a plan if it

(i) finds a plan if one exists, and

(ii) on input i, if the plan contains at least i actions, then it outputs the i-th action of the plan otherwise it outputs $\perp$.

3.2 Lower PSPACE Bounds

We start with a reduction from the PSPACE-complete problem IN-PLACE ACCEPTANCE for non-deterministic Turing machines [56].

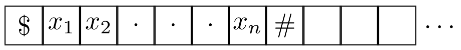
Proposition 3.2.1. *Any non-deterministic Turing machine M that accepts in space n can be faithfully represented within well-balanced local state transition systems.*

Proof. Let M be a non-deterministic machine that accepts in space n.

Without loss of generality, we assume the following:

(a) M has only one tape, which is one-way infinite to the right. The leftmost cell (numbered by 0) contains the marker \$ unerased.

(b) Initially, an *input* string, say $x_1x_2..x_n$, is written in cells 1, 2,..,n on the tape. In addition, a special marker # is written in the $(n+1)$-th cell.



(c) The program of M contains no instruction that could erase either \$ or #. There is no instruction that could move the head of M either to the right when M scans symbol #, or to the left when M scans symbol \$. As a result, M acts in the space between the two unerased markers.

(d) Finally, M has only one *accepting* state, and, moreover, all *accepting* configurations in space n are of one and the same form.

For each n, we design a local state transition system T_n (with a single agent A) as follows. We introduce the following 0-ary predicates (propositions), which are assumed to be local predicates of A, and hence there is no need to annotate them with As identity:

(a) $R_{i,\xi}$ stands for *"the i-th cell contains symbol ξ"*,

here $i=0,1,..,n+1$, ξ is a symbol of the tape alphabet of M,

(b) $S_{j,q}$ means *"the j-th cell is scanned by M in state q"*,

here $j=0,1,..,n+1$, q is a state of M.

Given an *instantaneous description* (*configuration*) of M in space n - that M scans j-th cell in state q, when a string $\xi_0\xi_1\xi_2..\xi_i..\xi_n\xi_{n+1}$ is written left-justified on the otherwise blank tape, we will represent it by a configuration of T_n of the form

$$S_{j,q}R_{0,\xi_0}R_{1,\xi_1}R_{2,\xi_2}\cdots R_{n,\xi_n}R_{n+1,\xi_{n+1}}. \tag{3.2.1}$$

Necessarily, $\xi_0 = \$$, and $\xi_{n+1} = \#$.

Each of the instructions of M, $q\xi \to q'\eta D$:

> *"if in state q looking at symbol ξ, replace it by η, move the tape head one cell in direction D along the tape, and go into state q'",*

is specified by the set of $n{+}2$ actions of the form, $i{=}0,1,..,n{+}1$:

$$S_{i,q} R_{i,\xi} \to_A S_{i_D,q'} R_{i,\eta}, \tag{3.2.2}$$

where $i_D := i{+}1$ if D is *right*, and $i_D := i{-}1$ if D is *left*, and $i_D := i$, otherwise.

(Neither $i_{right} = n{+}2$, nor $i_{left} = -1$ can occur.)

We denote the set of all these actions by R_{T_n}.

Lemma 3.2.2. *Given an input string $x_1x_2..x_n$ of length n, let W_n be a configuration of T_n of the form (3.2.1), which represents the* initial *configuration of M with the input string $x_1x_2..x_n$, and let Z_n be a configuration of T_n of the form (3.2.1), which represents the unique* accepting *configuration of M of length n.*

The following propositions are pairwise equivalent:

(a) *The input string $x_1x_2..x_n$ is accepted by M.*

(b) $W_n \rhd^*_{T_n} Z_n$

(c) $W_n \rightsquigarrow^*_{T_n} Z_n$

(d) *There exists a finite plan based on T_n that (exactly) leads from W_n to Z_n.*

Proof. Our encoding proceeds in a straight course, so that given any successful non-deterministic computation in space n that leads from the initial configuration represented by W_n to the accepting configuration represented by Z_n we can rewrite it as a sequence of actions from T_n leading from W_n to Z_n. This is done simply by writing (in order) the actions from T_n which correspond to the transitions used in the successful computation. This shows that $W_n \rhd^*_{T_n} Z_n$. Hence (a) implies (b). Item (c) follows from (b) by the definitions, and we noted earlier that (d) and (b) are equivalent.

The most complicated direction is that such a straightforward encoding is *faithful.* Suppose that $W_n \rightsquigarrow^*_{T_n} Z_n$. Then, for some V_1 we have $W_n \rhd^*_{T_n} Z_n V_1$. That is, there is a finite plan $\mathcal{P}$ based on T_n that exactly leads from W_n to $Z_n V_1$.

Since each of the actions (3.2.2) is well-balanced, and the number of facts in both W_n and Z_n is just the same $n+3$, this V_1 must be empty, which means that $W_n \rhd^*_{T_n} Z_n$, and so (c) implies (b). On the other hand, because of the specific form of our actions (3.2.2) every configuration X_i in $\mathcal{P}$ is of the form (3.2.1), and, hence, represents a configuration of M in space n.

We prove by induction that if M is ever in a configuration represented by X_i, for X_i in the plan $\mathcal{P}$, then M has a successful non-deterministic computation leading from that configuration to the accepting configuration. The basis step of $X_n = Z$ is trivial.

Induction Step:

Suppose M can lead to the accepting configuration whenever it finds itself in a configuration represented by X_i for $k < i \leq n$. Then we show it can lead to the accepting configuration from the configuration represented by X_k. Our construction is such that if $X_k \rhd^1_r X_{k+1}$ then M can apply the transition corresponding to r and transform from the configuration represented by X_k into the configuration represented by X_{k+1}. Since, by the inductive hypothesis, M can successfully finish when it starts in the configuration represented by X_{k+1}, then it can also successfully finish when starting in the configuration represented by X_k. In particular, we can conclude that the given input string $x_1 x_2 .. x_n$ is accepted by M, and so (b) implies (a). This completes the proof of Lemma 3.2.2. ✓

We now need to verify the efficiency of the reduction. In translating the Turing machine transitions into LSTS actions, we expand each transition into $n+2$ actions. Thus we expand the number of actions by a factor of n which is on the order of the size of the input string. The input string to the Turing machine is represented by the initial configuration of the LSTS. Under appropriate binary encodings this will not expand the size by more than a fixed constant. Finally, the representation of the goal configuration has size on the order of the initial configuration, so both initial and goal configurations can have representations on the order of the input string of the Turning machine. Thus, this translation can be performed in polynomial time with respect to the size of the input to the IN-PLACE ACCEPTANCE problem. ■

Notice that the signature of T_n consists only of $O(n)$ 0-ary predicate symbols. The number of the constants invoked is zero. If we confine ourselves both to a fixed number of predicate symbols and to a fixed number of constants, Theorem 3.2.1 fails because of the *polytime* algorithms presented in Section 3.4. Nevertheless, the next proposition show that we get `PSPACE`-hardness also at the opposite end of the "scale" *with a fixed number of predicate symbols and $O(n)$ constants.* Considering both ends of the spectrum allows us to pinpoint the cause of the PSPACE-hardness.

Proposition 3.2.3. *Any non-deterministic Turing machine M that accepts in space n can be faithfully represented in well-balanced local state transition systems with a fixed number of unary predicates.*

Proof. For each n, we will design a local state transition system T'_n (with a single agent A) by modifying the construction of Theorem 3.2.1 as follows.

Assume a list of constants:

$$c_0, c_1, c_2, \ldots, c_n, c_{n+1}.$$

We introduce the following unary predicate symbols, which are assumed to be private predicates of A:

(a) $\widehat{R}_\xi$, for every ξ from the tape alphabet of M.

The intended meaning of $\widehat{R}_\xi$ is:

$$\widehat{R}_\xi(c_i) = R_{i,\xi}.$$

(b) $\widehat{S}_q$, for every q from the set of states of M.

The intended meaning of $\widehat{S}_q$ is:

$$\widehat{S}_q(c_j) = S_{j,q}.$$

Given an *instantaneous description* (*configuration*) of M in space n - that M scans j-th cell in state q, when a string $\xi_0\xi_1\xi_2..\xi_i..\xi_n\xi_{n+1}$ is written left-justified on the otherwise blank tape, now it is represented as (cf. (3.2.1)):

$$\widehat{S}_q(c_j)\widehat{R}_{\xi_0}(c_0)\widehat{R}_{\xi_1}(c_1)\cdots\widehat{R}_{\xi_n}(c_n)\widehat{R}_{\xi_{n+1}}(c_{n+1}). \tag{3.2.3}$$

Each of the actions of the form (3.2.2) is rewritten as:

$$\widehat{S}_q(c_i)\widehat{R}_\xi(c_i) \to_A \widehat{S}_{q'}(c_{i_D})\widehat{R}_\eta(c_i). \tag{3.2.4}$$

The *faithfulness* of the modified encoding is established by

Lemma 3.2.4. *Given an input string $x_1x_2..x_n$ of length n, let W_n be a configuration of T'_n of the form (3.2.3) that represents the* initial *configuration of M with the input string $x_1x_2..x_n$, and let Z_n be a configuration of T'_n of the form (3.2.3) that represents the unique* accepting *configuration of M of length n.*

The following propositions are pairwise equivalent:

(a) *The input string $x_1x_2..x_n$ is accepted by M.*

(b) $W_n \rhd^*_{T'_n} Z_n$

(c) $W_n \leadsto^*_{T'_n} Z_n$

(d) *There exists a finite plan based on T'_n that (exactly) leads from W_n to Z_n.*

Proof. The proof mimics exactly the proof of Lemma 3.2.2. ■

Theorems 3.2.1 and 3.2.3 say that as long as either the number of predicate symbols or the number of constants in the signature is not bounded in advance, then goal reachability is PSPACE-hard. As the next result shows, this implies the PSPACE-hardness of all three versions of the collaborative planning problem with compliance to a confidentiality policy.

Theorem 3.2.5. *The problems of Well-Balanced System Compliance, Well-Balanced Weak Plan Compliance, and Well-Balanced Plan Compliance are all* `PSPACE`*-hard.*

Proof. The embeddings given in the proofs of Theorems 3.2.1 and 3.2.3 are into systems with a single agent whose confidentiality policy does not view any configuration as being critical. Thus the hardness is already achieved in the single-agent case where policy compliance is vacuously true for each of the three definitions. ■

3.3 Upper PSPACE Bounds

We are now ready to demonstrate algorithms for the three versions of the collaborative planning problem with confidentiality. The following theorem says that the problem of determining system compliance and scheduling a plan is in PSPACE.

Theorem 3.3.1 (Well-Balanced System Compliance)**.** *There is an algorithm which takes as inputs: an initial configuration W, programs $\mathcal{T}, \mathcal{G}$, and $\mathcal{C}$ as described in Section 3.1, and a natural number $0 < i \leq L_T(m)$, which behaves as follows:*

(a) *If the system is compliant, and if $W \rightsquigarrow^*_T Z$ where $\mathcal{G}(Z) = 1$, then it outputs "yes" and schedules the plan, otherwise it outputs "no".*

(b) *It runs in* `polynomial space` *with respect to $|\mathcal{T}|$, $|W|$, $|\mathcal{G}|$ and $|\mathcal{C}|$.*

Proof. The proof will rely on several critical facts about PSPACE. Namely, we rely on the equivalence of PSPACE, NPSPACE and COPSPACE [56].

Policy Compliance:

The algorithm first checks that none of the critical configurations are reachable from W. That is, we must check that every reachable configuration Z satisfies $\mathcal{C}(Z) = 0$. To do this, we give a nondeterministic algorithm which accepts exactly when a critical configuration *is* reachable. We then apply Savitch's Theorem to determinize the algorithm. Finally, we swap the accept and reject conditions to get a deterministic algorithm which accepts exactly when *all* critical configurations are *unreachable*.

We begin with $W_0 := W$. For any $t \geq 0$, we first check if $\mathcal{C}(W_t) = 1$. If so, then we return "yes". Otherwise we proceed to guess an action $r : X_A X_{pub} \rightarrow_A Y_A Y_{pub}$ such that $\mathcal{T}(r) = 1$ and such that r is enabled in configuration W_t. If no such action exists we return "no". If such an action does exist then we let W_{t+1} be the configuration which results from applying action r to configuration W_t, and we replace W_t by W_{t+1}. A critical configuration is reachable if and only if there is a sequence of at most $L_T(m)$ guesses that ends in a critical configuration. This is because if a plan loops back to a previously visited configuration and then proceeds to a critical configuration, then it could have avoided the loop the first time around. Thus, at worst, a plan must visit each of the $L_T(m)$ configurations.[1] This means we should stop guessing actions and return "no" after having already guessed $L_T(m)$ of them.

It is clear that if a critical configuration is reachable from W then there is a sequence of guesses which will result in the algorithm returning "yes". Otherwise the algorithm will always return "no". Thus the algorithm correctly decides the reachability of critical configurations. It only remains to show that this algorithm uses space which is polynomial in $|\mathcal{T}|$, $|W|$, and $|\mathcal{C}|$.

Let us now determine the space required for each t-th step. We need to record the configuration W_t. Since the configuration size never grows, it follows that $|W_t| = O(|W|)$ for any configuration W_t. Program $\mathcal{C}$ can verify $\mathcal{C}(W_t) = 1$ in space polynomial in $|W_t|$ by the assumption about the program $\mathcal{C}$. Our algorithm will need to keep track of $\mathcal{C}$'s functioning which can be done

[1] This is actually an overestimate as we have already seen above.

in space which is polynomial in $|\mathcal{C}|$. We then need to record the action r which is guessed. In the worst case an action r will have m facts in both the pre- and post-conditions. This amounts to representing two complete configurations which can be done in $O(|W|)$ space. Program $\mathcal{T}$ can verify $\mathcal{T}(r) = 1$ in space which is polynomial in $|r|$. Since $|r| = O(|W|)$ this verification can be done in space polynomial in $|W|$. Our algorithm will need to keep track of $\mathcal{T}$'s functioning which can be done in space which is polynomial in $|\mathcal{T}|$. In replacing W_t by W_{t+1} we may have to record two configurations at once, but we never record more than two. Finally, we should keep track of how many actions we have guessed so far. Since the algorithm stops after guessing $L_T(m)$ actions, this can be maintained on $\log_2(L_T(m))$ space. By Lemma 3.1.1, this is less than $m \cdot \log_2(S_T + 1)$ which we already saw is $O(|W|)$. Since the space at each step is polynomial in $|\mathcal{T}|$, $|W|$ and $|\mathcal{C}|$, the total space is also polynomial in those parameters. Thus the problem is found to be in NPSPACE. As stated above, we can determinize this algorithm and swap the accept and reject conditions to obtain the algorithm we are looking for, which remains in PSPACE.

Scheduling the Plan:

We now have to give an algorithm to schedule a plan if it exists, and show this algorithm is also in PSPACE. This is easily done by adapting the algorithm given above. We can replace every occurrence of $\mathcal{C}$ with $\mathcal{G}$, and we get a (nondeterministic) algorithm which decides the existence of a plan leading from W to one of the goals $Z_1, \ldots, Z_k$. In order for the algorithm to schedule a plan (and not just decide the existence of a plan) the algorithm will remember the guess for the i-th action. This only adds $O(|W|)$ space to the complexity. Thus this algorithm will run in space which is polynomial in $|\mathcal{T}|$, $|W|$ and $|\mathcal{G}|$. This time we only need to determinize the algorithm to conclude that it is in PSPACE.

Combining the Algorithms:

Finally, we note that we can combine these two algorithms in sequence to get the correct behavior. If either the first or the second parts fail, then we output "no". Otherwise we output "yes" along with the i-th action of the plan. This combined algorithm will run in space which is polynomial in $|\mathcal{T}|$, $|W|$, $|\mathcal{G}|$ and $|\mathcal{C}|$. ■

The previous theorem combines with Theorem 3.2.5 to imply that the collaborative planning problem with system compliance is PSPACE-complete. Due to the use of data confidentiality policies, this is actually a generalization of our PSPACE result in [35]. The following corollary is the formal statement of this fact. (We modify the bound on k, the number of partial goals in order to keep the input itself within the size bound.)

Corollary 3.3.2. *There is an algorithm applicable to the following inputs: an initial configuration W, a program $\mathcal{T}$ as described in Section 3.1, and a list $Z_1, \ldots, Z_k$ of partial goal configurations for $k = O(poly(|W|))$, which behaves as follows:*

(a) *If the system preserves the privacy of all agents according to Definition 2.3.6, and if $W \rightsquigarrow_T^* Z_i$ for some Z_i, outputs "yes", otherwise it outputs "no".*

(b) *It runs in* `polynomial space` *with respect to $|\mathcal{T}|$, and $|W|$ (which is $O(m \cdot \log_2(S_T))$).*

Proof. The algorithm is essentially the same as in the previous proof. The key difference is that there is no policy which is input to the problem. Instead, the policy is implicitly encoded in the initial configuration. We can replace the program $\mathcal{C}$ with a process that looks at each term in the current configuration and determines if it is one of the terms that should be protected according to Definition 2.3.6. It then determines if the current occurrence of the term violates Definition 2.3.6 by looking at who owns the predicate in which it appears. This is easily done in space which is polynomial in the size of a configuration (*i.e.* in $|W|$).

Notice also that the complexity is no longer with respect to $|\mathcal{G}|$, since we input an explicit encoding of the partial goals $Z_1, \ldots, Z_k$. Essentially, we replace $\mathcal{G}$ with this list. At each step we can check the current configuration against those partial goals on the list in space which is polynomial in $|W|$. Assuming $k = O(\text{poly}(|W|))$ is a special case of assuming $|\mathcal{G}| = O(\text{poly}(|W|))$, which allows us to put the polynomial complexity only in terms of $|W|$. ■

Intuitively, the corollary follows because the policy can be extracted from the initial configuration in polynomial space, and because we restrict the explicit representation of the partial goals as a list to be small. By using data confidentiality policies we gain a great deal of expressiveness at the small cost of an extra input parameter. Similarly, by considering more possibilities for the partial goal configurations we must pay a small cost. The corollary is just a special case in which $|\mathcal{G}|$ and $|\mathcal{C}|$ are essentially restricted to be polynomial in $|W|$. Thus these extra parameters only come into play when their size greatly exceeds that of $|W|$.

The next theorem states that the problem of scheduling a weakly compliant plan is also in PSPACE.

Theorem 3.3.3 (Well-Balanced Weak Plan Compliance)**.** *There is an algorithm applicable to the following inputs: an initial configuration W, programs $\mathcal{T}, \mathcal{G}$, and $\mathcal{C}$ as described in Section 3.1, and a natural number $0 < i \leq L_T(m)$, which behaves as follows:*

(a) *If there is a weakly compliant plan leading from W to Z where $\mathcal{G}(Z) = 1$, then it outputs "yes" and schedules the plan, otherwise it outputs "no".*

(b) *It runs in* `polynomial space` *with respect to $|\mathcal{T}|$, $|W|$, $|\mathcal{G}|$ and $|\mathcal{C}|$.*

Proof. We can adapt the proof of Theorem 3.3.1 for our situation here. At each step, the nondeterministic algorithm first checks if $\mathcal{C}(W_t) = 1$. If so, then the algorithm outputs "no". Otherwise it checks if $\mathcal{G}(W_t) = 1$. If so, then it outputs "yes". If not, then it guesses an action r in order to generate the next configuration W_{t+1}. In this way, we only continue along a path if it avoids the critical configurations. It is clear that if there is a weakly compliant plan then there is a sequence of guesses which will conclude with the algorithm outputting "yes". Otherwise the algorithm will always return "no". Thus the algorithm correctly decides the existence of a weakly compliant plan. As we did above, we can remember the i-th action using space which is $O(|W|)$. Since we can store each configuration and action in PSPACE as well as perform the necessary checks in PSPACE, we again find that the entire procedure runs in PSPACE with respect to the size of the inputs. Finally, we determinize the algorithm according to the proof of Savitch's Theorem. ■

This theorem combines with Theorem 3.2.5 to imply that the collaborative planning problem with weak plan compliance is PSPACE-complete.

Our next result gives an algorithm to find a compliant plan in PSPACE if such a plan exists. This result appears for the first time in this thesis. Due to the nature of the definition of plan compliance, it is not enough for our program $\mathcal{T}$ to recognize actions of the system. It must also recognize who it belongs to. That way from any Z along the plan it can find every V such that $Z \rhd^*_{-A_i} V$. Thus we will assume that $\mathcal{T}(i, r) = 1$ if r is an instance of one of the actions of agent A_i, and $\mathcal{T}(i, r) = 0$ otherwise. Similarly, we assume $\mathcal{C}(i, Z) = 1$ if configuration Z is critical for agent A_i and $\mathcal{C}(i, Z) = 0$ otherwise. For reasonable systems, the number of facts m in the initial configuration W is much larger than the number n of agents. Thus we can assume that $\mathcal{T}$ and $\mathcal{C}$ both still work in space which is polynomial in $|W|$. We assume $\mathcal{G}$ is just as described in Section 3.1.

Theorem 3.3.4 (Well-Balanced Plan Compliance)**.** *There is an algorithm applicable to the following inputs: an initial configuration W, programs $\mathcal{T}, \mathcal{G}$, and $\mathcal{C}$ as described directly above, and a natural number $0 < i \leq L_T(m)$, which behaves as follows:*

(a) *If there is a compliant plan leading from W to Z where $\mathcal{G}(Z) = 1$, then it outputs "yes" and schedules the plan, otherwise it outputs "no".*

(b) *It runs in* `polynomial space` *with respect to $|\mathcal{T}|$, $|W|$, $|\mathcal{G}|$ and $|\mathcal{C}|$.*

Proof. We present another nondeterministic algorithm that works in a way very similar to the algorithm in the proof of the previous theorem. The strategy is to describe a main procedure $\mathcal{P}$

which guesses actions of a plan leading to the goal. At each configuration Z along the way, $\mathcal{P}$ will call sub-procedures $\mathcal{P}_i$ that determine if there is a configuration V such that $Z \rhd^*_{-A_i} V$ and $\mathcal{C}(i, V) = 1$.

Sub-Procedures:

We again rely on the fact that NPSPACE, PSPACE, and COPSPACE are all the same complexity class. We begin with nondeterministic procedures $\mathcal{P}'_i$ for each agent A_i. Each sub-procedure $\mathcal{P}'_i$ begins in some configuration $Z_0 = Z$. For each $0 \leq t \leq L_T(m)$ it checks first if $\mathcal{C}(i, Z_t) = 1$. If so, it returns "yes". Otherwise, it guesses an action r and an agent $A_j \neq A_i$ such that $\mathcal{T}(j, r) = 1$ and such that r is enabled in configuration Z_t. If no such action exists, then return "no". Otherwise, apply it applies r to Z_t to obtain Z_{t+1}, and replaces Z_t by Z_{t+1} and repeats. After guessing $L_T(m)$ actions, if it has not yet returned anything, it returns "no". We can then reverse the accept and reject conditions and use Savitch's Theorem to get a deterministic procedure $\mathcal{P}_i$ which returns "yes" if every V satisfying $Z \rhd^*_{-A_i} V$ also satisfies $\mathcal{C}(i, V) = 0$, and returns "no" otherwise.

Main Procedure:

The main procedure $\mathcal{P}'$ begins at $W_0 = W$, the initial configuration. For each $0 \leq t \leq L_T(m)$ $\mathcal{P}'$ calls each of the sub-procedures $\mathcal{P}_i(W_t)$. If any of them returns "no", then $\mathcal{P}'$ returns "no". Otherwise, it checks if $\mathcal{G}(W_t) = 1$. If so, then it returns "yes". If not, it guesses an action r such that $\mathcal{T}(i, r) = 1$ for some agent A_i and such that r is enabled in W_t. If no such action exists, $\mathcal{P}'$ returns "no". Otherwise, it applies r to W_t to obtain W_{t+1} and it replaces W_t with W_{t+1} and repeats. After guessing $L_T(m)$ actions, if it has not yet returned anything, it returns "no". We can use Savitch's Theorem to get a deterministic procedure $\mathcal{P}$ which outputs "yes" if there is a compliant plan leading from W to some goal configuration, and "no" otherwise. Again, we can have $\mathcal{P}$ record the $i - th$ action in order to schedule the plan.

Let us analyze the space used in this algorithm. Let's start with the sub-procedures. Just as in the previous theorems, they only remember two configurations at a time plus a counter to know when it has looked at $L_T(m)$ of them. This requires linear space in $|W|$. Checking if $\mathcal{T}(j, r) = 1$ can be done in space polynomial in $|\mathcal{T}|, |j|$ and $|r|$. We have already seen that $|r|$ is polynomial in $|W|$. Since the number of agents n is much less than m, we can record each agent j in space $log_2(n)$ which is within $O(|W|)$. Similarly, $\mathcal{C}(i, V)$ can be evaluated in space polynomial in $|W|$. Thus each sub-procedure works in space polynomial in $|W|, |\mathcal{T}|$, and $|\mathcal{C}|$.

The main procedure also only remembers at most three configurations at one time. Since it calls each sub-procedure $\mathcal{P}_i$ in sequence, those calls can be done in polynomial space, as just described above. Again, the extra calls to $\mathcal{T}$ and $\mathcal{G}$ can be done in polynomial space in $|T|, |G|$ and the size of their inputs which is $O(|W|)$. Finally, it can use $log_2 L_T(m) = O(|W|)$ space to count up to the maximum $L_T(m)$ actions before deciding the plan is too long. Adding this all up, the whole procedure remains in polynomial space with respect to the input parameters. ■

This theorem combines with Theorem 3.2.5 to imply that the collaborative planning problem with plan compliance is PSPACE-complete.

3.4 Polynomial Time Algorithms

We end this chapter by considering the complexity of the planning problems with confidentiality under one further restriction. Instead of viewing the signature Σ of the transition system as an input to the problem, we fix it *in advance*. This may be a reasonable thing to consider if the agents involved participate in distinct yet related collaborations on a periodic basis. For example, a group of companies may wish to perform quarterly collaborative forecasting. The language of the problem can be viewed as a fixed aspect of the system instead of an input parameter. Under this assumption we show that both versions of the collaborative planning problem are solvable in polynomial time with respect to the size of their input. Although the time complexity of the problems is exponential, the following theorems also serve to isolate the source of the exponential complexity namely, the number of facts expressible by the signature.

The algorithms will work the same as the algorithms in the previous section. Thus in order for them to work in polynomial time, the programs $\mathcal{T}, \mathcal{G}$, and $\mathcal{C}$ must also work in polynomial time. In practice, the programs we used in the previous section were already likely to work in polynomial time, so this is not a very strong restriction.

Theorem 3.4.1 (Well-Balanced System Compliance)**.** *Let Σ be a fixed finite signature (consisting of a finite number of predicate symbols with their arity and of a finite number of constants). Then there is an algorithm applicable to the following inputs: an initial configuration W, and programs $\mathcal{T}, \mathcal{G}$, and $\mathcal{C}$ as described in Section 3.1 that work in polynomial time, which behaves as follows:*

(a) *If the system is compliant, and if $W \rightsquigarrow_T^* Z$ such that $\mathcal{G}(Z) = 1$, then it outputs the plan, otherwise it outputs "no".*

(b) *It runs in* `polynomial time` *with respect to $|\mathcal{T}|$, $|W|$, $|\mathcal{G}|$, and $|\mathcal{C}|$.*

Proof. The crucial fact to keep in mind throughout this proof is that the total number of configurations is $L_T(m) \leq (m+1)^{S_T}$. Since we have fixed a signature *in advance*, S_T is viewed as a constant. Thus $L_T(m)$ is polynomial in m. Similarly, $|W| = O(m \cdot \log_2(S_T)) = O(m)$, so we find that $L_T(m)$ is polynomial in $|W|$.

The algorithm works the same as the determinized algorithm in the proof of Theorem 3.3.1, except that it records all the configurations of the plan if there is one. The size of the reachability tree is polynomial in $L_T(m)$. Thus the number of steps is polynomial in $|W|$. Since we assume that each of $\mathcal{T}$, $\mathcal{G}$ and $\mathcal{C}$ run in polynomial time (with respect to $|W|$), the whole algorithm now runs in polynomial time with respect to the size of the four input parameters. ■

Theorem 3.4.2 (Well-Balanced Weak Plan Compliance)**.** *Let Σ be a fixed finite signature (consisting of a finite number of predicate symbols with their arity and of a finite number of constants). Then there is an algorithm applicable to the following inputs: an initial configuration W, and programs $\mathcal{T}, \mathcal{G}$, and $\mathcal{C}$ as described in Section 3.1 that work in polynomial time, which behaves as follows:*

(a) *If there is a weakly compliant plan leading from W to some Z such that $\mathcal{G}(Z) = 1$ then it outputs the plan, otherwise it outputs "no".*

(b) *It runs in* `polynomial time` *with respect to $|\mathcal{T}|$, $|W|$, $|\mathcal{G}|$, and $|\mathcal{C}|$.*

Proof: This algorithm works the same as the determinized algorithm in the proof of Theorem 3.3.3, except that it records all the configurations of the plan if there is one. Since the reachability tree is polynomial in $L_T(m)$ which is polynomial in $|W|$, we can conclude that the whole algorithm runs in polynomial time with respect to the size of the four input parameters. ■

While the previous two theorems appear in [33], the corresponding theorem for Well-Balanced Plan Compliance appears here for the first time.

Theorem 3.4.3 (Well-Balanced Plan Compliance)**.** *Let Σ be a fixed finite signature (consisting of a finite number of predicate symbols with their arity and of a finite number of constants). Then there is an algorithm applicable to the following inputs: an initial configuration W, and programs $\mathcal{T}, \mathcal{G}$, and $\mathcal{C}$ as described in Section 3.1 that work in polynomial time, which behaves as follows:*

(a) *If there is a compliant plan leading from W to some Z such that $\mathcal{G}(Z) = 1$ then it outputs the plan, otherwise it outputs "no".*

(b) *It runs in* `polynomial time` *with respect to $|\mathcal{T}|$, $|W|$, $|\mathcal{G}|$, and $|\mathcal{C}|$.*

Proof. This algorithm works the same as the determinized algorithm in the proof of Theorem 3.3.4, except that it records all the configurations of the plan if there is one. Since the total reachability tree from any configuration is polynomial in $L_T(m)$ which is polynomial in $|W|$, so is the reachability tree when we disallow actions from agent A_i in determining whether $\mathcal{C}(i, V)$ for all V such that $Z \rhd^*_{-A_i} V$. Thus each sub-procedure can be performed in time polynomial in $|W|, |\mathcal{T}|$, an $|\mathcal{C}|$. Thus, each step of the main procedure can be completed in time $n \cdot \text{poly}(|W|, |\mathcal{T}|, |\mathcal{C}|) + \text{poly}(|\mathcal{G}|)$. Since $n < m$ we know that $n = O(|W|)$. Thus each step can be performed in polynomial time in the size of the inputs. Since the reachability tree has size polynomial in $L_T(m)$, this complexity grows at most polynomially, making the whole procedure run in polynomial time. ■

Chapter 4

Complexity of General Systems

In this chapter, we consider how the complexity and indeed even the decidability of the different versions of the collaborative planning problem with confidentiality are affected when we do not assume our systems are well-balanced. Section 4.1 proves that the version for system compliance is EXPSPACE-complete. In Section 4.2 we show that both versions of plan compliance cause the problem to be undecidable. The contents of this chapter appear in [34].

4.1 System Compliance

In this section we demonstrate that General System Compliance is EXPSPACE-complete. The main step on the way to proving this is to show that the coverability problem for Petri nets is equivalent to the reachability problem for LSTSs. In order to prove this we rely on reductions in both directions between the coverability problem for Petri nets, and the reachability problem for LSTSs. The coverability problem was shown to be EXPSPACE-hard by Lipton [48], and shown to have an exponential space algorithm by Rackoff [58]. Both of these results are actually about vector addition systems [43], but vector addition systems have been shown to be computationally equivalent to Petri nets [30, 69].

This equivalence arises in the general case in which actions are not necessarily well-balanced. In order to make the correspondence in the well-balanced case, we would need to identify the appropriate boundedness restriction on Petri nets. Searching through the literature, it appears that Petri nets which are 1-conservative correspond most closely to well-balanced LSTSs. Our search through the literature revealed that Petri net *reachability* is PSPACE-complete for 1-conservative nets. We did not run across the corresponding complexity result for the *coverability* problem, although we do not want to claim that no such result has been published. Since we did not thoroughly investigate the connection between well-balanced LSTSs and certain restricted Petri nets, we did not perform a complete examination of all the relevant Petri net literature. The possibility of connections between certain restricted Petri nets and certain restricted LSTSs could be fruitful for both formalisms. However, going forward we will only consider the correspondence in the general case.

In order to demonstrate the correspondence we must review some definitions from Petri nets. A number of equivalent definitions exist but we use the ones found in [69].

Petri Nets. A *Petri net* is a tuple $N = (P, T, \phi)$ where P is a finite set of *places*, $P = \{p_1, \dots, p_k\}$, T is a finite set of transitions, $T = \{t_1, \dots, t_s\}$, and ϕ is a flow function $\phi : (P \times T) \cup (T \times P) \to \mathbb{N}$. A *marking* μ of the Petri net is a function $\mu : P \to \mathbb{N}$ which assigns some number of *tokens* to each place. For two markings μ, μ' we write $\mu \geq \mu'$ whenever $\mu(p) \geq \mu'(p)$ for all $p \in P$.

A transition t is *enabled* at μ if and only if for all $p \in P$, $\phi(p,t) \leq \mu(p)$. When t is enabled then it may *fire* by removing $\phi(p,t)$ tokens from each place p and by adding $\phi(t,p)$ tokens to each place p. We then write $\mu \xrightarrow{t} \mu'$ where $\mu'(p) = \mu(p) - \phi(p,t) + \phi(t,p)$ for all $p \in P$. (This notation also implies that t is enabled at μ.) Thus the set $\{\phi(p,t) \mid p \in P\}$ act like pre-conditions for the transition t and the set $\{\phi(t,p) \mid p \in P\}$ act like post-conditions for the transition t. A firing sequence $\sigma = t_1 \dots t_n$ is enabled at μ_0 if and only if $\mu_0 \xrightarrow{t_1} \mu_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} \mu_n$, for some markings

$\mu_1, \ldots, \mu_n$. In that case we write $\mu_0 \xrightarrow{\sigma} \mu_n$.

A marking μ is *reachable* from μ_0 if $\exists \sigma \in T^*$ such that $\mu_0 \xrightarrow{\sigma} \mu$. The reachability set of a marking μ_0 is $R(N, \mu_0) = \{\mu \mid \exists \sigma \in T^*, \mu_0 \xrightarrow{\sigma} \mu\}$. The marking μ is *coverable* from μ_0 if $\exists \mu' \in R(N, \mu_0)$ such that $\mu' \geq \mu$. The coverability set of μ_0 is $C(N, \mu_0) = \{\mu \mid \exists \mu' \in R(N, \mu_0), \mu' \geq \mu\}$. The coverability problem for Petri nets is the following. Given a Petri net N and two markings μ_0, μ_1, decide if $\mu_1 \in C(N, \mu_0)$.

In order to show the equivalence of the coverability problem for Petri nets and the reachability problem for LSTSs, we must demonstrate reductions both ways between Petri nets and LSTSs. The details follow, but in order to help the reader see the big picture we give a high-level overview of how the reductions work. Intuitively, places of the Petri net correspond to facts of the LSTS. Petri net markings μ correspond to LSTS configurations $\hat{\mu}$. Petri net transitions t_i correspond to LSTS actions $\hat{t}_i$ in such a way that if markings μ and μ' correspond to configurations $\hat{\mu}$ and $\hat{\mu}'$ respectively, then $\mu \xrightarrow{t_i} \mu'$ if and only if $\hat{\mu} \rhd^1_{\hat{t}_i} \hat{\mu}'$.

The reason that LSTS reachability corresponds to coverability instead of Petri net reachability is that we focus on *partial* goals. In other words, our use of reachability according to $\rightsquigarrow^*_T$ allows the goal configuration to contain extra facts as long as it contains at least those facts represented by the goal. Similarly, Petri net coverability allows the final marking to have more tokens in any given place, as long as each place contains a certain minimum number. This distinction between Petri net reachability and coverability is crucial to our complexity result because although Petri net reachability has been shown to be decidable [50], the most reasonable upper bound on the complexity is primitive recursive in the Ackermann function. Thus, EXPSPACE membership relies crucially on our use of partial goals instead of exact goals.

In the presentation that follows we use notation for LSTSs that is similar to notation from Petri nets in order to highlight the correspondence more clearly.

4.1.1 Reduction from Petri nets to LSTSs

Let $N = (P, T, \phi)$ be a Petri net. We will associate a LSTS T_N to this Petri net. For each place p_i we will include a propositional constant (0-ary predicate) F_i in the signature of T_N. For any marking μ of N we associate the configuration

$$\hat{\mu} = F_1^{\mu(p_1)}, \ldots, F_k^{\mu(p_k)}.$$

For each transition t_i we include the (propositional) action

$$\hat{t}_i = F_1^{\phi(p_1,t_i)}, \ldots, F_k^{\phi(p_k,t_i)} \to F_1^{\phi(t_i,p_1)}, \ldots, F_k^{\phi(t_i,p_k)}.$$

This action is enabled in a configuration $\hat{\mu}$ iff $\hat{\mu}(p_j) \geq \phi(p_j, t_i)$ for all $1 \leq j \leq k$ iff t_i is enabled in μ. Thus the result of applying this action to the configuration $\hat{\mu}$ is

$$\hat{\mu}' = F_1^{\mu(p_1)-\phi(p_1,t_i)+\phi(t_i,p_1)}, \ldots, F_k^{\mu(p_k)-\phi(p_k,t_i)+\phi(t_i,p_k)}$$

where $\mu \xrightarrow{t_i} \mu'$.

Note that T_N has finitely many propositional actions. Notice also that the size of the LSTS reachability problem (as measured by the total length of the binary descriptions of the actions, and the initial and final configurations) is of the same order as the size of the Petri net coverability problem (measured by the total length of the binary descriptions of the flow function and the initial and final markings). Under appropriate binary encodings this translation can be performed in polynomial time.

Proposition 1. *Under the above translation, if $\mu_1 \in C(N, \mu_0)$ then $\hat{\mu}_0 \rightsquigarrow^*_{T_N} \hat{\mu}_1$.*

Proof. The proof is by induction on the length of the firing sequence σ which satisfies $\mu_0 \xrightarrow{\sigma} \mu$ such that $\mu \geq \mu_1$. The basis case of length 0 is trivial.

Induction Step:

Suppose that if μ is coverable from μ_0 via a firing sequence of length n then $\hat{\mu}_0 \rightsquigarrow^*_{T_N} \hat{\mu}$. Suppose also that μ_1 is coverable from μ_0 via a firing sequence of length $n + 1$. Then there are markings μ, μ' such that $\mu_0 \xrightarrow{\sigma} \mu \xrightarrow{t_i} \mu'$ where σ is a firing sequence of length n and $\mu' \geq \mu_1$. By the inductive hypothesis, $\hat{\mu}_0 \rightsquigarrow^*_{T_N} \hat{\mu}$. It is easy to check that since $\mu \xrightarrow{t_i} \mu'$, we also get that $\hat{\mu} \rhd^1_{t_i} \hat{\mu}'$. Also since $\mu' \geq \mu_1$ we know that $\hat{\mu}'$ contains $\hat{\mu}_1$. Thus, $\hat{\mu} \rightsquigarrow^*_{T_N} \hat{\mu}_1$. It is easily seen that $\rightsquigarrow^*_{T_N}$ is a transitive relation, thus $\hat{\mu}_0 \rightsquigarrow^*_{T_N} \hat{\mu}_1$ as desired. ■

Proposition 2. *Under the above translation, if $\hat{\mu}_0 \rightsquigarrow^*_{T_N} \hat{\mu}_1$ then $\mu_1 \in C(N, \mu_0)$.*

Proof. The proof is by induction on the length of the sequence of actions which witnesses $\hat{\mu}_0 \rightsquigarrow^*_{T_N} \hat{\mu}_1$. The basis case of a length 0 sequence is trivial.

Induction Step:

Suppose that if $\hat{\mu}_0 \rightsquigarrow^n_{T_N} \hat{\mu}$ then $\mu \in C(N, \mu_0)$. Suppose also that $\hat{\mu}_0 \rightsquigarrow^{n+1}_{T_N} \hat{\mu}_1$. Then there is a configuration $\hat{\mu}$ such that $\hat{\mu}_0 \rightsquigarrow^n_{T_N} \hat{\mu} \rightsquigarrow^1_{\hat{t}_i} \hat{\mu}_1$. By the inductive hypothesis we know that $\mu \in C(N, \mu_0)$. Also, by definition, there is a configuration $\hat{\mu}'$ such that $\hat{\mu} \rhd^1_{\hat{t}_i} \hat{\mu}'$ and $\hat{\mu}'$ contains $\hat{\mu}_1$. It is easy to check that $\mu \xrightarrow{t_i} \mu'$ and $\mu' \geq \mu_1$. Thus $\mu_1 \in C(N, \mu)$. It is not too difficult to see that coverability is transitive and so $\mu_1 \in C(N, \mu_0)$ as desired. ■

Because this reduction is efficient and since Lipton showed an EXPSPACE lower bound for Petri net coverability [48], we can conclude that the LSTS reachability problem is EXPSPACE-hard.

4.1.2 Reduction from LSTSs to Petri Nets

Suppose we are given a LSTS T with finitely many propositional actions, and two configurations μ_0 and μ_1. Then there are only finitely many facts which can appear in any plan: those facts in the configurations μ_0 and μ_1 and those facts mentioned in one of the actions. Suppose there are k facts total, and for convenience let us rename them $F_1, \ldots, F_k$. Then we will associate to this LSTS a Petri net $N_T = (\widehat{P}, \widehat{T}, \phi)$ with places $\widehat{P} = \{p_1, \ldots, p_k\}$. Any configuration of the form

$$\mu = F_1^{\mu^1}, \ldots, F_k^{\mu^k}$$

corresponds to a marking of the Petri net $\hat{\mu}$ such that $\hat{\mu}(p_i) = \mu^i$ for $1 \leq i \leq k$.

If T has actions $\{t_1, \dots, t_s\}$ then the Petri net N_T will have transitions $\widehat{T} = \{\hat{t}_1, \dots, \hat{t}_s\}$. If the action t_i is

$$t_i : F_1^{t^-_{i,1}}, \dots, F_k^{t^-_{i,k}} \to F_1^{t^+_{i,1}}, \dots, F_k^{t^+_{i,k}}$$

where $t^-_{i,j}, t^+_{i,j} \in \mathbb{N}$ for all $1 \leq j \leq k$, then ϕ satisfies

$$\phi(p_j, \hat{t}_i) = t^-_{i,j} \text{ and } \phi(\hat{t}_i, p_j) = t^+_{i,j} \text{ for all } 1 \leq j \leq k.$$

Then action t_i is enabled in configuration μ if and only if $t^-_{i,j} \leq \mu^j$ for all $1 \leq j \leq k$ if and only if transition $\hat{t}_i$ is enabled in marking $\hat{\mu}$. Note also that $\mu \rhd^1_{t_i} \mu'$ iff $\hat{\mu} \xrightarrow{\hat{t}_i} \hat{\mu}'$.

Notice that the size of the Petri net coverability problem is of the same order as the size of the LSTS reachability problem. Under an appropriate binary encoding the translation can be performed in polynomial time.

Proposition 3. *Under the above translation, if* $\mu_0 \rightsquigarrow^*_T \mu_1$ *then* $\hat{\mu}_1 \in C(N_T, \hat{\mu}_0)$.

Proof. The proof is by induction on the length of the sequence of actions witnessing $\mu_0 \rightsquigarrow^*_T \mu_1$. The basis case of a length 0 sequence is trivial.

Induction Step:

Suppose that if $\mu_0 \rightsquigarrow^n_T \mu$ then $\hat{\mu} \in C(N_T, \hat{\mu}_0)$. Suppose also that $\mu_0 \rightsquigarrow^{n+1}_T \mu_1$. Then there is a configuration μ such that $\mu_0 \rightsquigarrow^n_T \mu \rightsquigarrow^1_{t_i} \mu_1$. By the inductive hypothesis $\hat{\mu} \in C(N_T, \hat{\mu}_0)$. Also, by definition, there is a configuration μ' such that $\mu \rhd^1_{t_i} \mu'$ where μ' contains μ_1. It is easy to check that $\hat{\mu} \xrightarrow{\hat{t}_i} \hat{\mu}'$ and $\hat{\mu}' \geq \hat{\mu}_1$. Thus $\hat{\mu}_1 \in C(N_T, \hat{\mu})$. By the transitivity of coverability we see that $\hat{\mu}_1 \in C(N_T, \hat{\mu}_0)$ as desired. ■

Proposition 4. *Under the above translation, if* $\hat{\mu}_1 \in C(N_T, \hat{\mu}_0)$ *then* $\mu_0 \rightsquigarrow^*_T \mu_1$.

Proof. The proof is by induction on the length of the firing sequence $\hat{\sigma}$ such that $\hat{\mu}_0 \xrightarrow{\hat{\sigma}} \hat{\mu}$ with $\hat{\mu} \geq \hat{\mu}_1$. The basis case of a length 0 firing sequence is trivial.

Induction Step:

Suppose that if $\hat{\mu}$ is coverable from $\hat{\mu}_0$ in n steps then $\mu_0 \rightsquigarrow_T^* \mu$. Suppose also that $\hat{\mu}_1$ is coverable from $\hat{\mu}_0$ in $n+1$ steps. Then there are markings $\hat{\mu}$ and $\hat{\mu}'$ such that $\hat{\mu}_0 \xrightarrow{\hat{\sigma}} \hat{\mu} \xrightarrow{\hat{t}_i} \hat{\mu}'$ with $\hat{\mu}' \geq \hat{\mu}_1$ and $\hat{\sigma}$ a firing sequence of length n. Then by the inductive hypothesis $\mu_0 \rightsquigarrow_T^* \mu$. It is also easy to see that since $\hat{\mu} \xrightarrow{\hat{t}_i} \hat{\mu}'$ then $\mu \rhd_{t_i}^1 \mu'$, and since $\hat{\mu}' \geq \hat{\mu}_1$ we know that μ' contains μ_1. Thus $\mu \rightsquigarrow_T^* \mu_1$. By the transitivity of $\rightsquigarrow_T^*$ we conclude that $\mu_0 \rightsquigarrow_T^* \mu_1$ as desired. ■

Because this reduction is efficient, and because the coverability problem can be solved in EXPSPACE by Rackoff's upper bound [58], we can conclude that LSTS reachability can be determined in EXPSPACE as well.

4.1.3 EXPSPACE-Completeness of System Compliance

The previous sections proved that the LSTS reachability problem is EXPSPACE-complete. We extend this complexity of the LSTS reachability problem to determining system compliance and goal reachability in general systems.

Theorem 4.1.1 (General System Compliance)**.** *Let T be a local state transition system with a finite set of propositional actions, $\mathcal{Z}$ a finite set of goal configurations, $\mathcal{C}$ a finite set of critical configurations, and W an initial configuration. It is EXPSPACE-complete to determine if the system is compliant, and if so to determine if there is a plan leading from W to one of the goal configurations.*

Proof. We already achieve EXPSPACE-hardness in the case where $\mathcal{C}$ is empty and $\mathcal{Z}$ has a single element Z. This is simply a direct application of the EXPSPACE-hardness of goal reachability in local state transition systems.

In order to show EXPSPACE membership we rely on the fact that EXPSPACE = COEXPSPACE ([63] chapter 8). This means that given a single critical configuration C, it is possible to determine in exponential space whether or not C is unreachable, that is, whether or not the system is compliant with respect to C. In order to determine system compliance with respect to a finite set $\mathcal{C} = \{C_1, \ldots, C_m\}$ of critical configurations, we can simply create a process which checks compliance with each of the configurations in order. If this process ever determines that the system is not compliant with respect to some C_i then it outputs "no". If it determines that the system is compliant with respect to all of the configurations, then it proceeds to determine if one of the goals is reachable.

Given the set $\mathcal{Z}$ of goal configurations, another process can determine, in order, whether each configuration is reachable. If the process finds a goal which is reachable, it outputs "yes". If all the goals are unreachable, it outputs "no". Since the checks for (non-)reachability are done sequentially, and each one is performed within exponential space, the whole process can be performed in exponential space. ■

Compare Theorem 4.1.1 with Theorem 3.3.1. These theorems represent the first column of Table 1.1. Lifting the restriction of well-balanced actions to allow un-balanced actions causes the problem to jump from PSPACE to EXPSPACE. Viewed another way, we already discussed how EXPSPACE membership relies crucially on using *partial* goals instead of exact goals. When we make the further assumption of using only well-balanced actions we reduce the complexity to PSPACE. It might be interesting to search for further reasonable restrictions that can make the problem more tractable. Also note that using the techniques from Section 3.3 we can schedule a plan if one exists without increasing the complexity.

4.2 Undecidability

In this section we show that General Weak Plan Compliance and General Plan Compliance are both undecidable. The proof is by reduction from two-counter Minsky machines. The reduction we use is modeled after a similar reduction by Kanovich [42] to prove the undecidability of pure monadic linear logic. There are several key differences however. First, Kanovich's proof is about first-order linear logic, and hence takes advantage of the existential quantifier to "create new values." We have no such operator, thus our proof corresponds more closely to a propositional setting. In particular, this is *not* a reduction from Kanovich's result, but rather a direct reduction from Minsky machines.

Second, we point out that since our notion of reachability refers to *partial* goals, reachability more closely corresponds to derivability of certain sequents in affine logic as described in Chapter 5. While full propositional linear logic is known to be undecidable [46], it has been shown that full propositional affine logic is decidable [44]. Thus, although our setting relates to propositional affine logic, the satisfaction of both plan compliance and weak plan compliance differs enough from simple reachability to affect the decidability of the problem.

4.2.1 Minsky Machines

We use a standard two-counter Minsky machine M of a certain form. We assume the instructions alternate between instructions for register 1 and instructions for register 2. Instructions labeled by a_i will be 'run' by Alice, instructions labeled by b_j will be 'run' by Bob.

No two instructions are labeled with the same label. States a_1 and a_0 are the *initial* and *final* states of M, respectively. Furthermore, a_0 is a halting state so it is distinct from the label of any of M's instructions. *M's configuration* where M is in state m, and k_1 and k_2 are the current values of counters r_1 and r_2, respectively, is denoted by $(m; k_1, k_2)$. A *computation* performed by M is a sequence of M's configurations such that each step is made by one of the above instructions:

$$(a_1; n, 0) \xrightarrow{a_1} \cdots \longrightarrow (a_i; k_1, k_2) \xrightarrow{a_i} (b_j; k_1', k_2') \xrightarrow{b_j} \cdots$$

A *terminating* computation is one that ends in a configuration $(a_0; *, *)$, that is, the final state a_0 with any values in the counters.

Jump	a_i: **goto** b_j;
Add	a_i: $r_1 := r_1 + 1$; **goto** b_j;
Subtract	a_i: $r_1 := r_1 - 1$; **goto** b_j;
0-test	a_i: **if** $(r_1 = 0)$ **goto** b_j **else goto** b_k;
Jump	b_j: **goto** a_k;
Add	b_j: $r_2 := r_2 + 1$; **goto** a_k;
Subtract	b_j: $r_2 := r_2 - 1$; **goto** a_k;
0-test	b_j: **if** $(r_2 = 0)$ **goto** a_k **else goto** a_l;

Table 4.1: The form of Minsky machine instructions.

Furthermore, for encoding purposes, each instruction of the form

$$a_i : \textbf{ if } (r_1 = 0) \textbf{ goto } b_j \textbf{ else goto } b_k;$$

is replaced by the following set of instructions:

$$\begin{aligned} &a_i : \textbf{ if } (r_1 = 0) \textbf{ goto } b_{\hat{\jmath}} \textbf{ else goto } b_k; \\ &b_{\hat{\jmath}} : \textbf{ goto } a_{\hat{\imath}}; \\ &a_{\hat{\imath}} : \textbf{ goto } b_j; \end{aligned} \tag{4.2.1}$$

and each instruction of the form b_j : **if** $(r_2 = 0)$ **goto** a_k **else goto** a_l; is replaced by the following set of instructions:

$$\begin{aligned} &b_j : \textbf{ if } (r_2 = 0) \textbf{ goto } a_{\tilde{k}} \textbf{ else goto } a_l; \\ &a_{\tilde{k}} : \textbf{ goto } b_{\tilde{\jmath}}; \\ &b_{\tilde{\jmath}} : \textbf{ goto } a_k; \end{aligned} \tag{4.2.2}$$

where $b_{\hat{\jmath}}$, $a_{\hat{\imath}}$, $a_{\tilde{k}}$, $b_{\tilde{\jmath}}$ are fresh, unique labels. It is important to note that instructions labeled by $b_{\hat{\jmath}}$ and $a_{\hat{\imath}}$ can be applied only in a row, and only in the case when the corresponding value of counter r_1 is zero. Similarly instructions labeled by $a_{\tilde{k}}$ and $b_{\tilde{\jmath}}$ can be applied only in a row, and only in the case when the corresponding value of counter r_2 is zero.

4.2.2 Minsky Machine Translation

We can now describe how to embed a given Minsky machine M into a local state transition system which we denote T_M. The system T_M will have two participants, Alice and Bob. They will each limit their own local configuration to contain only one fact. Alice will use public tokens to track the value of register 1, and Bob will similarly track the value of register 2.

States of agent A are encoded by private propositions: r^A, s_0^A, s_1^A,.., s_i^A,.... States of agent B are encoded by private propositions: r^B, s_0^B, s_1^B,.., s_j^B,.... A public proposition R_1 means that "One public resource unit is allocated to Alice." A public proposition R_2 means that "One public resource unit is allocated to Bob." A public predicate $C_1(x)$ means "x is the label of an instruction to be run by Alice". A public predicate $C_2(x)$ means "x is the label of an instruction to be run by Bob". M's configuration of the form $(a_i; k_1, k_2)$ is encoded by $r^A, R_1^{k_1}, r^B, R_2^{k_2}, C_1(a_i)$. M's configuration of the form $(b_j; k_1, k_2)$ is encoded by $r^A, R_1^{k_1}, r^B, R_2^{k_2}, C_2(b_j)$.

Alice's actions roughly correspond to M's instructions on register one. For every instruction of the form a_i : **goto** b_j; Alice has the corresponding jump action:

$$s_i^A \quad \to_A \quad r^A, C_2(b_j). \tag{4.2.3}$$

This action does not change the public resources, and it writes the public fact $C_2(b_j)$ to tell Bob which instruction to perform next. Similarly, for every instruction of the form $a_i : r_1 := r_1+1;$ **goto** b_j; Alice has the corresponding addition action:

$$s_i^A \quad \to_A \quad r^A, R_1, C_2(b_j). \tag{4.2.4}$$

For every instruction of the form $a_i : r_1 := r_1-1;$ **goto** b_j; Alice has the corresponding subtraction action:

$$s_i^A, R_1 \quad \to_A \quad r^A, C_2(b_j). \tag{4.2.5}$$

For every instruction of the form a_i : **if** $(r_1 = 0)$ **goto** $b_{\hat{\jmath}}$ **else goto** b_k; Alice has two actions:

$$s_i^A \quad \to_A \quad r^A, C_2(b_{\hat{\jmath}}) \tag{4.2.6}$$

$$s_i^A, R_1 \quad \to_A \quad r^A, R_1, C_2(b_k). \tag{4.2.7}$$

Notice that we introduce nondeterminism here. Without any way of controlling this nondeterminism, an agent may use action (4.2.6) when there is an instance of R_1. The machine M cannot take this direction because M first checks if $r_1 = 0$ and if not, it follows the other branch of the instruction. The use of critical configurations play an essential role in controlling this nondeterminism as we discuss below. Also note that although actions of types (4.2.3) and (4.2.6) have a similar form, they are differentiated syntactically by their labels. In particular, actions of type (4.2.6) publish one of the distinguished labels $b_{\hat{\jmath}}$ that were introduced in construction (4.2.1).

Finally, Alice has an action that does not directly correspond to any of M's instructions. This action explicitly links the labels a_i with the corresponding propositions s_i^A. For each corresponding pair, Alice has the receive action

$$r^A, C_1(a_i) \quad \to_A \quad s_i^A \tag{4.2.8}$$

in which Alice reads the label of an instruction from the network and enters a state ready to perform that instruction.

Bob has actions on his side that correspond to instructions on M's register 2. For completeness, we list those actions in the corresponding order below.

$$s_j^B \quad \to_B \quad r^B, C_1(a_k) \tag{4.2.9}$$

$$s_j^B \quad \to_B \quad r^B, R_2, C_1(a_k) \tag{4.2.10}$$

$$s_j^B, R_2 \quad \to_B \quad r^B, C_1(a_k) \tag{4.2.11}$$

$$s_j^B \quad \to_B \quad r^B, C_1(a_{\tilde{k}}) \tag{4.2.12}$$

$$s_j^B, R_2 \quad \to_B \quad r^B, R_2, C_1(a_l) \tag{4.2.13}$$

$$r^B, C_2(b_j) \quad \to_B \quad s_j^B \tag{4.2.14}$$

Thus associated to any Minsky machine M is the local state transition system T_M with actions of types (4.2.3)–(4.2.14). The initial configuration of M is $(a_1; n, 0)$ and the corresponding initial configuration of T_M is $r^A, R_1^n, r^B, C_1(a_1)$.

In order to fully specify the problem of finding a compliant or a weakly compliant plan we must describe what the goal configurations and confidentiality policies are. We specify a single goal of $C_1(a_0)$. This simply corresponds to an accepting state of M. Thus any configuration containing $C_1(a_0)$ is a goal configuration. We need to take some care in choosing what confidentiality policies to use because they will be used to control the potentially undesirable behavior caused by the nondeterminism introduced in the translation of the 0-test instructions. If we do not specify any policies (or specify the empty policies) then the translation will not be faithful. That is, a plan might follow action (4.2.6) when there is an instance of R_1.

We choose critical configurations which designate this behavior as undesirable. Namely, Alice's critical configurations are those of the form $C_1(a_{\hat{\imath}}), R_1$ and Bob's critical configurations are those of the form $C_2(b_{\hat{\jmath}}), R_2$ where $a_{\hat{\imath}}$ and $b_{\hat{\jmath}}$ are the fresh unique labels from constructions (4.2.1) and (4.2.2) respectively. Intuitively, since the labels $a_{\hat{\imath}}$ and $b_{\hat{\jmath}}$ should only occur when $r_1 = 0$ and $r_2 = 0$ respectively, these critical configurations serve as a signal that the wrong branch was taken in a plan. Thus if compliant plans and weakly compliant plans are supposed to accurately represent machine computations, they should never reach these configurations.

Notice that these critical configurations do not take full advantage of the expressive power of confidentiality policies. In particular they consist only of public facts. This means that both Alice and Bob can recognize when a configuration is critical for either agent. We show undecidability even in this more restrictive case.

It remains to show that this translation allows T_M to faithfully simulate terminating computations of M. The next subsection provides the necessary soundness and completeness results.

4.2.3 Soundness and Completeness

In this section we show that, under our translation of Minsky machines into LSTSs (with confidentiality policies), the machine M has a terminating computation on $(a_1; n, 0)$ leading to $(a_0; *, *)$ if and only if T_M has a weakly compliant plan leading from $r^A, R_1^n, r^B, C_1(a_1)$ to $C_1(a_0)$ if and only if T_M has a compliant plan leading from $r^A, R_1^n, r^B, C_1(a_1)$ to $C_1(a_0)$.

We first show the soundness of our translation with respect to plan compliance. Since plan compliance implies weak plan compliance the corresponding soundness result with respect to weak plan compliance is an immediate corollary. We then show completeness of the translation with respect to weak plan compliance. Again, since plan compliance implies weak plan compliance, we get the completeness with respect to plan compliance as an immediate corollary.

Proposition 5 (Soundness)**.** *Given a Minsky machine M and its translation T_M, if M's computation on $(a_1; n, 0)$ terminates in $(a_0; *, *)$, then T_M has a compliant plan leading from $r^A, R_1^n, r^B, C_1(a_1)$ to the partial goal $C_1(a_0)$.*

Proof. We show that whenever M and T_M are in corresponding configurations, if M's instructions lead to a configuration of the form $(s_i; k_1, k_2)$, then T_M can reach the corresponding configuration which represents $(s_i; k_1, k_2)$. We then show that the resulting plan is compliant. The argument proceeds by induction on the length of the M-computation with the basis step being an M-computation of only 1 step. (The 0-step case is trivial.)

Basis Step: $(a_i; k_1, k_2) \rightarrow_M^1 (b_j; k_1', k_2')$

Case: a_i: **goto** b_j*;*

M goes from $(a_i; k_1, k_2)$ to $(b_j; k_1, k_2)$. Then T_M satisfies

$$r^A, R_1^{k_1}, r^B, R_2^{k_2}, C_1(a_i) \quad \rhd^* \quad r^A, R_1^{k_1}, r^B, R_2^{k_2}, C_2(b_j)$$

by applying a receive action of type (4.2.8) followed by a jump action of type (4.2.3). Notice that the configuration on the right represents the configuration $(b_j; k_1, k_2)$ as desired.

Case: $a_i : r_1 := r_1 + 1$; **goto** b_j;

M goes from $(a_i; k_1, k_2)$ to $(b_j; k_1 + 1, k_2)$. Then T_M satisfies

$$r^A, R_1^{k_1}, r^B, R_2^{k_2}, C_1(a_i) \quad \rhd^* \quad r^A, R_1^{k_1+1}, r^B, R_2^{k_2}, C_2(b_j)$$

by applying a receive action of type (4.2.8) followed by an addition action of type (4.2.4). Notice that the configuration on the right represents the configuration $(b_j; k_1 + 1, k_2)$ as desired.

Case: $a_i : r_1 := r_1 - 1$; **goto** b_j;

M goes from $(a_i; k_1, k_2)$ to $(b_j; k_1 - 1, k_2)$. Since M can perform this instruction we know that $k_1 > 0$. Thus T_M satisfies

$$r^A, R_1^{k_1}, r^B, R_2^{k_2}, C_1(a_i) \quad \rhd^* \quad r^A, R_1^{k_1-1}, r^B, R_2^{k_2}, C_2(b_j)$$

by applying a receive action of type (4.2.8) followed by a subtraction action of type (4.2.5) (which is applicable because $k_1 > 0$). Notice that the configuration on the right represents the configuration $(b_j; k_1 - 1, k_2)$ as desired.

Case: a_i: **if** $(r_1 = 0)$ **goto** $b_{\hat{\jmath}}$ **else goto** b_k;

M goes from $(a_i; k_1, k_2)$ to either $(b_{\hat{\jmath}}; k_1, k_2)$ if $k_1 = 0$ or $(b_k; k_1, k_2)$ if $k_1 > 0$. In the first case T_M satisfies:

$$r^A, R_1^{k_1}, r^B, R_2^{k_2}, C_1(a_i) \quad \rhd^* \quad r^A, R_1^{k_1}, r^B, R_2^{k_2}, C_2(b_{\hat{\jmath}})$$

by applying a receive action of type (4.2.8) followed by an action of type (4.2.6). In the latter case, T_M satisfies

$$r^A, R_1^{k_1}, r^B, R_2^{k_2}, C_1(a_i) \quad \rhd^* \quad r^A, R_1^{k_1}, r^B, R_2^{k_2}, C_2(b_k)$$

by applying a receive action of type (4.2.8) followed by an action of type (4.2.7) (which is enabled because $k_1 > 0$ in this case). Notice that the configurations on the right represent $(b_{\hat{\jmath}}; k_1, k_2)$ and $(b_k; k_1, k_2)$ respectively as desired.

We conclude that any single instruction on M's register 1 can be simulated by T_M's actions. The cases in which M's instruction modifies register 2 are completely analogous, and so we omit the details.

Induction Step: This follows immediately from the inductive hypothesis by splitting a length n computation into a length $n-1$ computation followed by a length 1 computation. Therefore computations of any length can be simulated by T_M. In particular, if $(a_1; n, 0) \to_M^* (a_0; *, *)$ then we can conclude that T_M satisfies

$$r^A, R_1^n, r^B, C_1(a_1) \quad \rightsquigarrow^* \quad C_1(a_0).$$

We must now show why this plan is compliant. Recall that Alice's critical configurations are of the form $C_1(a_{\hat{\imath}}), R_1$ where $a_{\hat{\imath}}$ is a label which appears only in an instruction sequence resulting from the transformation (4.2.1). We first show why none of the configurations in the plan are critical. Every configuration of the form $r^A, R_1^{k_1}, r^B, R_2^{k2}, C_1(a_{\hat{\imath}})$ in the plan corresponds to a configuration $(a_{\hat{\imath}}; k_1, k_2)$ along M's computation. We remarked at the end of Section 4.2.1, that this only happens when $k_1 = 0$. Thus none of the configurations of the plan are critical for Alice. A similar argument shows none of the configurations are critical for Bob.

Now suppose that Bob tries to deviate from the plan to create a configuration which is critical for Alice. Without loss of generality we may assume he starts from a configuration of the form $r^A, R_1^{k_1}, r^B, R_2^{k_2}, C_2(b_j)$.

He must create a fact $C_1(a_{\hat{\imath}})$ (with a distinguished label) in order to succeed. By inspection we can see that Bob can only do this if $b_j = b_{\hat{\jmath}}$. But that means that the machine configuration is $(b_{\hat{\jmath}}; k_1, k_2)$ which we already remarked happens only when $k_1 = 0$. Thus, if Bob succeeds in creating the fact $C_1(a_{\hat{\imath}})$ it means that there are no instances of R_1 in the configuration, and the result is not critical for Alice. A similar argument shows that Alice cannot create a critical configuration for Bob by herself, if she starts from a configuration in the plan. ■

Thus the translation is sound with respect to both plan compliance and weak plan compliance. In order to show that the translation is complete, we show that an arbitrary weakly compliant plan leading from $r^A, R_1^n, r^B, C_1(a_1)$ to $C_1(a_0)$ corresponds to the terminating computation of M on $(a_1; n, 0)$. This direction requires some care because we do not know *a priori* that a weakly compliant plan has a form which clearly corresponds to a computation of M. Thus we first show that weakly compliant plans do have a nice form passing only through configurations which we call regular.

Definition 4.2.1. If T_M is the translation of a Minsky machine M, then a configuration is called *regular* if it has one of the four following forms:

$$r^A, R_1^{k_1}, r^B, R_2^{k_2}, C_1(a_i) \tag{I}$$

$$s_i^A, R_1^{k_1}, r^B, R_2^{k_2} \tag{II}$$

$$r^A, R_1^{k_1}, r^B, R_2^{k_2}, C_2(b_j) \tag{III}$$

$$r^A, R_1^{k_1}, s_j^B, R_2^{k_2} \tag{IV}$$

Configurations of type (I) and (III) correspond directly to Minsky machine configurations as we saw before. Configurations of type (II) and (IV) are the "intermediate" configurations that arise from the use of the receive actions (4.2.8) and (4.2.14). In the previous proof, every configuration of the plan was regular. The next lemma states that this must be the case for every plan which starts in a regular configuration.

Lemma 4.2.2. *Let T_M be the translation of a Minsky machine M. If T_M begins in a regular* *configuration then every configuration in a plan is also regular. Furthermore, the configurations cycle through types (I), (II), (III) and (IV) in that order.*

Proof. It suffices to show that if T_M is in any of the four regular configuration types, then any action which is enabled will transform T_M into the next type in the cycle. We analyze each case separately.

Case I:

T_M is in a regular configuration of type (I). By inspection the only action which is enabled is one of Alice's receive actions of type (4.2.8). It is easy to check that applying this action will yield a regular configuration of type (II).

Case II:

T_M is in a configuration of type (II). By inspection, the only possible actions which can be enabled are Alice's actions of type (4.2.3)–(4.2.7). One easily checks that applying any of these actions will result in a regular configuration of type (III).

Case III:

T_M is in a regular configuration of type (III). By inspection, the only action which is enabled is one of Bob's receive actions of type (4.2.14). It is easy to check that applying this action will yield a regular configuration of type (IV).

Case IV:

T_M is in a regular configuration of type (IV). By inspection, the only actions which are enabled are Bob's actions of type (4.2.9)–(4.2.13). Just as in the first case, it is easy to check that applying any of these actions will yield a regular configuration of type (I). ■

Using this result we can now show how to simulate an arbitrary weakly compliant plan by a computation of M.

Proposition 6 (Completeness). *Given a Minsky machine M and its translation T_M, if T_M has a weakly compliant plan leading from $r^A, R_1^n, r^B, C_1(a_1)$ to $C_1(a_0)$ then M's computation leads from $(a_1; n, 0)$ to $(a_0; *, *)$.*

Proof. We prove more generally that if the weakly compliant plan leads from $r^A, R_1^n, r^B, C_1(a_1)$ to $r^A, R_1^{k_1}, r^B, R_2^{k_2}, C_1(a_i)$ then M's computation leads from $(a_1; n, 0)$ to $(a_i; k_1, k_2)$. Since the configuration $r^A, R_1^n, r^B, C_1(a_1)$ is regular, Lemma 4.2.2 implies that the plan proceeds to enter a cycle of regular configurations. Since a regular configuration which contains $C_1(a_i)$ is of type (I) we know that the plan must start and end in a configuration of type (I) and therefore must consist of some number of complete cycles (and no partial cycles). We can prove the result by induction on the number of cycles in the plan. Recall that a full cycle consists of four actions, two actions from Alice followed by two actions from Bob.

Basis Step: 0 Cycles

If the plan has 0 cycles then it starts in the ending state and hence $a_1 = a_i$, $n = k_1$ and $k_2 = 0$ so M also starts in the configuration $(a_i; k_1, k_2)$.

Induction Step:

The inductive hypothesis is that a weakly compliant plan with $n-1$ cycles can be simulated by an M-computation. We must then show that an n-cycle plan can also be simulated by an M-computation.

After the first $n-1$ cycles of the plan, T_M is in a regular configuration of type (I) that looks like the following:

$$r^A, R_1^{k_1}, r^B, R_2^{k_2}, C_1(a_i)$$

By the inductive hypothesis we know that M leads to the corresponding configuration $(a_i; k_1, k_2)$. We must only show how M simulates the last cycle of the plan. By Lemma 4.2.2 we know that the cycle begins with a receive action of type (4.2.8) followed by an action of one of the types (4.2.3)–(4.2.7).

Case I:

Alice applies a receive action followed by an action of type (4.2.3). This transforms T_M into a configuration of the form

$$r^A, R_1^{k_1}, r^B, R_2^{k_2}, C_2(b_j).$$

Since the second action is of type (4.2.3), the label a_i is the label for a jump instruction, and M can apply the instruction and end up in the configuration $(b_j; k_1, k_2)$ which is represented by T_M's resulting configuration.

Case II:

Alice applies a receive action followed by an action of type (4.2.4). This transforms T_M into a configuration of the form

$$r^A, R_1^{k_1+1}, r^B, R_2^{k_2}, C_2(b_j).$$

Since the second action is of type (4.2.4), the label a_i is the label for an addition instruction, and M can apply the instruction and end up in the configuration $(b_j; k_1 + 1, k_2)$ which is represented by T_M's resulting configuration.

Case III:

Alice applies a receive action followed by an action of type (4.2.5). This transforms T_M into a configuration of the form

$$r^A, R_1^{k_1-1}, r^B, R_2^{k_2}, C_2(b_j).$$

Since the second action is of type (4.2.5), the label a_i is the label for a subtraction instruction. Since T_M's action requires the existence of an occurrence of R_1 we are guaranteed that $k_1 > 0$. Thus M can successfully apply the instruction and end up in the configuration $(b_j; k_1 - 1, k_2)$ which is represented by T_M's resulting configuration.

Case IV:

Alice applies a receive action followed by an action of type (4.2.6). This transforms T_M into a configuration of the form

$$r^A, R_1^{k_1}, r^B, R_2^{k_2}, C_2(b_{\hat{\jmath}}).$$

Since the second action is of type (4.2.6), the label a_i is the label for a 0-test instruction. Furthermore, we know there are no occurrences of R_1 because the plan is assumed to be weakly compliant. An occurrence of R_1 would cause the cycle to finish with Bob publishing the fact $C_1(a_{\hat{\imath}})$ creating a critical configuration contrary to the weak compliance of the plan. Thus $k_1 = 0$ and M's previous configuration is actually $(a_i; 0, k_2)$. Hence M's test for 0 in register 1 is successful. M then ends up in the configuration $(b_{\hat{\jmath}}; 0, k_2)$ which is represented by T_M's resulting configuration.

Case V:

Alice applies a receive action followed by an action of type (4.2.7). This transforms T_M into a configuration of the form

$$r^A, R_1^{k_1}, r^B, R_2^{k_2}, C_2(b_k).$$

Since the second action is of type (4.2.7), the label a_i is the label for a 0-test instruction. Furthermore, we know $k_1 > 0$ because the action is only enabled when there is an occurrence of R_1. Thus when M performs the 0-test, it finds $k_1 > 0$ and transitions into the configuration $(b_k; k_1, k_2)$ which is represented by T_M's resulting configuration.

The cycle is completed by Bob applying a receive action of type (4.2.14) followed by an action of one of the types (4.2.9)–(4.2.13). The analysis of each of these cases is completely analogous to the previous cases completing the induction. We therefore conclude that M can simulate any weakly compliant plan which completes some number of full cycles. In particular, if the weakly compliant plan leads from $r^A, R_1^n, r^B, C_1(a_1)$ to $C_1(a_0)$, then M's computation leads from $(a_1; n, 0)$ to $(a_0; *, *)$. ■

Since every compliant plan is also weakly compliant, it follows that the translation is complete with respect to plan compliance as well. The soundness and completeness results combine to imply that Problem 2 (weak plan compliance) and Problem 3 (plan compliance) are both undecidable.

Theorem 4.2.3 (General Weak Plan Compliance)**.** *Let T be a local state transition system with a finite set of propositional actions, $\mathcal{Z}$ a finite set of goal configurations, $\mathcal{C}$ a finite set of critical configurations, and W an initial configuration. The problem of determining if T has a weakly compliant plan leading from W to one of the goal configurations is undecidable.*

Proof. The translation from Minsky machines creates a local state transition system T_M with only finitely many propositional actions, a single goal configuration, and a finite set of critical configurations. The soundness and completeness of the translation imply the undecidability of the problem. ■

Theorem 4.2.3 is in contrast with Theorem 3.3.3. These theorems represent the second column of Table 1.1. Going from well-balanced systems to general systems alters the decidability of the problem. Also note that in the well-balanced case, system compliance and weak plan compliance have the same complexity, but in the general case system compliance is decidable and weak plan compliance becomes undecidable. Therefore, the undecidability arises from a combination of the possibility of having un-balanced actions and the importance of remembering the path that is followed to reach a goal as embodied by the definition of weak plan compliance.

Theorem 4.2.4 (General Plan Compliance)**.** *Let T be a local state transition system with a finite set of propositional actions, $\mathcal{Z}$ a finite set of goal configurations, $\mathcal{C}$ a finite set of critical configurations, and W an initial configuration. The problem of determining if T has a compliant plan leading from W to one of the goal configurations is undecidable.*

Proof. This proof is identical to the proof of Theorem 4.2.4. ■

We also want to point out the differences between Theorems 4.2.3 and 4.2.4 and the undecidability results of [20] and [42]. The results of [20] are about secrecy in the context of security protocols with an external intruder who has considerably more power than the honest protocol participants. In that setting, undecidability is only attained with the ability to create an unbounded number of fresh values, or nonces, combined with the unbounded memory of the intruder. In the current work, we take advantage of the unbounded memory of the agents by allowing un-balanced actions, but we work in a propositional setting in which no new values are ever created.

Similarly, the proof of undecidability for pure monadic linear logic [42] relies on the ability to use the existential quantifier to create an unbounded number of new values. Also, linear logic derivability roughly correspond to exact reachability in our setting. We lose some control by using our weaker notion of reachability, and by not allowing new values to be created, but we regain that control through the interpretation of confidentiality policies.

Chapter 5

Foundation in Logic

In this chapter we present the logical foundation of our approach. It is desirable to have such a logical foundation because we are often able to gain new insight by thinking in terms of a well established formalism. For example we may be able to apply existing tools, which work with some logical formalism, to the problem at hand. Here we demonstrate a connection to a variation of linear logic known as affine logic. The contents of this chapter can be found in [35] and the full proofs first appear in [33].

5.1 Linear and Affine Logic

Linear logic (LL), which was introduced in [25], is a resource-sensitive refinement of traditional logic. It is presented as a Gentzen style sequent calculus. A sequent of the form $\Gamma \vdash \Delta$ says roughly that the resources in the multiset Γ can be used to produce the multiset Δ. Linear logic differs in a number of ways from traditional logic, but most notably it does not allow the rules of weakening or contraction.

The rule of contraction is given by

$$\frac{A, A, \Gamma \vdash \Delta}{A, \Gamma \vdash \Delta}$$

It says that if we can produce Δ using Γ and two copies of A, then we can also produce Δ with Γ and one copy of A. Weakening is the opposite rule and is given by

$$\frac{\Gamma \vdash \Delta}{A, \Gamma \vdash \Delta}$$

It says that if we can produce Δ using Γ, the we can also produce Δ with Γ and A, for any A.

The effect of disallowing the contraction rule is that it forces us to use each resource *at most* once. The effect of disallowing the weakening rule is that it forces us to use each resource *at least* once. This is why linear logic is popular when trying to model resource-sensitive systems. It allows the available resources to grow and shrink, in contrast to traditional logic in which the resources grow monotonically. Linear logic also provides a mechanism which allows certain resources to be used any number of times. This will be useful when we use logic to model actions.

At a lower level, linear logic splits the traditional connectives defining conjunction and disjunction into two forms. For example, the traditional conjunction $\wedge$ is split into $\otimes$ (multiplicative conjunction) and & (additive conjunction). A derivation of a $\otimes$ conjunction forbids any sharing of the resources used to derive each conjunct. In contrast, a derivation of a & conjunction requires all resources to be shared.

Affine logic (AL) is the variation of linear logic which allows weakening, but still disallows contraction. Thus, in affine logic, we may use each resource either once or not at all. Affine logic is therefore an appropriate logic to use when we want to model the reachability of partial goals. It allows us to work with the relevant resources in arbitrary contexts.

5.2 Reachability and Derivability

There is a well-known correspondence between state transition systems and linear logic. We extend this connection to one between local state transition systems and affine logic. By encoding the objects of local state transition systems as logical formulas we are able to relate partial goal reachability with derivability of certain Gentzen sequents within affine logic. Table 5.1 shows the rules of affine logic that are most relevant to our correspondence.

Our translation of local state transition systems into affine logic follows closely the translation of ordinary state transition systems into linear logic. There is one notable difference which deserves mention here. If we want our translation to preserve all key elements of local state transition systems, we must preserve the information which allows us to determine which action is allowed to be used by which participant. For this reason we translate actions into linear implications with a special constant which acts as a "lock", as described below.

For a local state transition system T, multisets of facts are encoded using the multiplicative conjunction, $\otimes$ (which is commutative).

$$\ulcorner S \urcorner = \bigotimes S.$$

The empty multiset is encoded as the multiplicative unit.

$$\ulcorner \cdot \urcorner = \mathbf{1}.$$

Transition rules are encoded as linear implication. This is where we use the "lock". For each agent A we have a constant symbol q_A. We then encode transition rules using linear implication as follows.

$$\ulcorner X_A X_{pub} \to_A Y_A Y_{pub} \urcorner = q_A \otimes \ulcorner X_A X_{pub} \urcorner \multimap q_A \otimes \ulcorner Y_A Y_{pub} \urcorner$$

This technique forces derivations in affine logic to contain all the relevant information about which agent is applying an action at a given point. Now, for the rule set R_T we can define

$$\ulcorner R_T \urcorner = \{\ulcorner r \urcorner : r \in R_T\}.$$

Since locks of the form q_A are part of the linear implications we must also include these constants as part of the global configuration. If I is the set of participants of T, then we define

$$\ulcorner I \urcorner = \bigotimes \{q_A : A \in I\}.$$

Structural rules

$$\frac{}{\Gamma; A_1, \ldots, A_n \vdash A_i}\ \text{id} \qquad \frac{\Gamma, A; \Delta, A \vdash C}{\Gamma, A; \Delta \vdash C}\ \text{clone}$$

Cut rules

$$\frac{\Gamma; \Delta_1 \vdash A \qquad \Gamma; \Delta_2, A \vdash C}{\Gamma; \Delta_1, \Delta_2 \vdash C}\ \text{cut} \qquad \frac{\Gamma; \cdot \vdash A \qquad \Gamma, A; \Delta \vdash C}{\Gamma; \Delta \vdash C}\ \text{cut!}$$

Left rules	**Right rules**
$\frac{\Gamma; \Delta \vdash C}{\Gamma; \Delta, \mathbf{1} \vdash C}\ \mathbf{1}\ell$	$\frac{}{\Gamma; \cdot \vdash \mathbf{1}}\ \mathbf{1}r$
$\frac{\Gamma; \Delta, A_1, A_2 \vdash C}{\Gamma; \Delta, A_1 \otimes A_2 \vdash C}\ \otimes\ell$	$\frac{\Gamma; \Delta_1 \vdash C_1 \qquad \Gamma; \Delta_2 \vdash C_2}{\Gamma; \Delta_1, \Delta_2 \vdash C_1 \otimes C_2}\ \otimes r$
$\frac{\Gamma; \Delta_1 \vdash A \qquad \Gamma; \Delta_2, B \vdash C}{\Gamma; \Delta_1, \Delta_2, A \multimap B \vdash C}\ \multimap \ell$	$\frac{\Gamma; \Delta, A \vdash C}{\Gamma; \Delta \vdash A \multimap C}\ \multimap r$
$\frac{\Gamma; \Delta, A \vdash C}{\Gamma, \Delta, \exists x.A \vdash C}\ \exists\ell$	$\frac{\Gamma; \Delta \vdash [t/x]C}{\Gamma; \Delta \vdash \exists x.C}\ \exists r$

Table 5.1: Some rules of affine logic.

We use a notion of provability which is based on an intuitionistic version of affine logic in which the sequents have the form

$$\Gamma; \Delta \vdash C$$

where Δ is a linear context, Γ is an unrestricted context in which the resources may be used as many times as necessary, and C is a single formula (see [13]). The relevant rules of affine logic are given in Table 5.1. Under our encoding of local state transition systems we are able to achieve the following results.

Theorem 5.2.1. (Soundness) *For every pair of states W, Z, and every rule set R_T defining a* *participant set I, if $W \rightsquigarrow_T^* Z$ then $\ulcorner R_T \urcorner ; \ulcorner I \urcorner \otimes \ulcorner W \urcorner \vdash \ulcorner I \urcorner \otimes \ulcorner Z \urcorner$ is derivable in affine logic.*

Theorem 5.2.2. (Completeness) *For every pair of states W, Z and every rule set R_T defining a participant set I, if $\ulcorner R_T \urcorner ; \ulcorner I \urcorner \otimes \ulcorner W \urcorner \vdash \ulcorner I \urcorner \otimes \ulcorner Z \urcorner$ is derivable in affine logic then $W \rightsquigarrow_T^* Z$.*

In order to prove these results we will pass through soundness and completeness results which relate exact reachability to derivability in linear logic (Lemmas 5.2.3 and 5.2.4 respectively). We then prove Lemma 5.2.5 which relates derivability of sequents in linear logic to derivability of sequents in affine logic. This last lemma allows us to lift the soundness and completeness results with respect to linear logic to the corresponding results with respect to affine logic. Since we are concerned with sequents in both linear logic and affine logic we will write $\Gamma; \Delta \vdash_{LL} C$ for sequents which are meant to be derived in linear logic, and we write $\Gamma; \Delta \vdash_{AL} C$ for sequents which are meant to be derived in affine logic.

Lemma 5.2.3. *For every pair of states W, Z and every rule set R_T defining a participant set I, if $W \rhd_T^* Z$ then $\ulcorner R_T \urcorner ; \ulcorner I \urcorner \otimes \ulcorner W \urcorner \vdash_{LL} \ulcorner I \urcorner \otimes \ulcorner Z \urcorner$ is derivable.*

Proof: (by induction on the length of the action sequence) We consider two basis cases:

Length 0 transition sequence: $W \rhd_T^0 Z$.

That means that $W = Z$ and $\ulcorner R_T \urcorner ; \ulcorner I \urcorner \otimes \ulcorner W \urcorner \vdash_{LL} \ulcorner I \urcorner \otimes \ulcorner W \urcorner$ is the conclusion of the identity rule.

Length 1 transition sequence: $W \rhd_T^1 Z$.

So Z is reachable from W using only one rule from R_T. Assume it is $r : X_\alpha X_{pub} \rightarrow_\alpha Y_\alpha Y_{pub}$. We get the derivation seen in Figure 5.1.

In the above derivation V appears after (above) the occurrence of the rule $\otimes\ell$. This V represents $(\ulcorner I \urcorner \otimes \ulcorner W \urcorner) \setminus (q_\alpha \otimes \ulcorner X_\alpha \urcorner \otimes \ulcorner X_{pub} \urcorner)$. This means that the top right sequent can be separated

$$\cfrac{\cfrac{\cfrac{\ulcorner R_T\urcorner; q_\alpha \otimes \ulcorner X_\alpha\urcorner \otimes \ulcorner X_{pub}\urcorner \vdash_{LL} q_\alpha \otimes \ulcorner X_\alpha\urcorner \otimes \ulcorner X_{pub}\urcorner \qquad \ulcorner R_T\urcorner; q_\alpha \otimes \ulcorner Y_\alpha\urcorner \otimes \ulcorner Y_{pub}\urcorner, V \vdash_{LL} \ulcorner I\urcorner \otimes \ulcorner Z\urcorner}{\ulcorner R_T\urcorner; q_\alpha \otimes \ulcorner X_\alpha\urcorner \otimes \ulcorner X_{pub}\urcorner \multimap q_\alpha \otimes \ulcorner Y_\alpha\urcorner \otimes \ulcorner Y_{pub}\urcorner, q_\alpha \otimes \ulcorner X_\alpha\urcorner \otimes \ulcorner X_{pub}\urcorner, V \vdash_{LL} \ulcorner I\urcorner \otimes \ulcorner Z\urcorner}\ \multimap\ell}{\ulcorner R_T\urcorner; q_\alpha \otimes \ulcorner X_\alpha\urcorner \otimes \ulcorner X_{pub}\urcorner \multimap q_\alpha \otimes \ulcorner Y_\alpha\urcorner \otimes \ulcorner Y_{pub}\urcorner, \ulcorner I\urcorner \otimes \ulcorner W\urcorner \vdash_{LL} \ulcorner I\urcorner \otimes \ulcorner Z\urcorner}\ \otimes\ell}{\ulcorner R_T\urcorner; \ulcorner I\urcorner \otimes \ulcorner W\urcorner \vdash_{LL} \ulcorner I\urcorner \otimes \ulcorner Z\urcorner}\ \text{clone}$$

Figure 5.1: Derivation corresponding to a single transition.

by the use of $\otimes r$ to reduce it to two applications of the identity. This shows that the lemma holds for length 1 transition sequences.

Induction Step:

Assume it holds for transition sequences of length $\leq n$. Now assume that $W \rhd_T^{n+1} Z$. Then we know that there is some U such that $W \rhd_T^n U$ and $U \rhd_T^1 Z$. Therefore our induction hypothesis tells us that we have two derivations ending in $\ulcorner R_T\urcorner; \ulcorner I\urcorner \otimes \ulcorner W\urcorner \vdash_{LL} \ulcorner I\urcorner \otimes \ulcorner U\urcorner$ and in $\ulcorner R_T\urcorner; \ulcorner I\urcorner \otimes \ulcorner U\urcorner \vdash_{LL} \ulcorner I\urcorner \otimes \ulcorner Z\urcorner$. We can apply the cut rule with these two conclusions to obtain a derivation with the conclusion $\ulcorner R_T\urcorner; \ulcorner I\urcorner \otimes \ulcorner W\urcorner \vdash_{LL} \ulcorner I\urcorner \otimes \ulcorner Z\urcorner$ as desired. ■

Lemma 5.2.4. *For every pair of states W, Z and every rule set R_T defining a participant set I, if $\ulcorner R_T\urcorner; \ulcorner I\urcorner \otimes \ulcorner W\urcorner \vdash_{LL} \ulcorner I\urcorner \otimes \ulcorner Z\urcorner$ is derivable then $W \rhd_T^* Z$.*

Proof: Completeness is much more complicated than soundness. The reason for this is that there may be many different LL-derivations of a given formula. We cannot simply assume that the derivation which is assumed to exist has a nice form that corresponds to the form of a derivation which arises in the proof of soundness. We will briefly sketch the ideas here noting that the key to this argument is the ability to permute certain rules above or below others. We direct the reader to [47] for the proofs of these permutation rules. We also note that LL completeness has been shown for rewriting systems before in [12, 13].

By cut elimination there is a cut-free derivation of the sequent

$$\ulcorner R_T\urcorner; \ulcorner I\urcorner \otimes \ulcorner W\urcorner \vdash_{LL} \ulcorner I\urcorner \otimes \ulcorner Z\urcorner.$$

In particular, this derivation satisfies the sub-formula property which states that any formula occurring in the derivation must be a sub-formula of the formulas occurring in the conclusion. This allows us to restrict our attention to derivations with only the following rules: id, clone, $\otimes\ell$, $\otimes r$, $\mathbf{1}\ell$, $\mathbf{1}r$, $\multimap \ell$ and $\multimap r$. Furthermore, we can rule out derivations which use $\multimap r$ since we do not use nested implications. As mentioned above, these rules may be permuted in restricted ways as proven in [47]. These permutation results allow us to assume that clone appears below all other rules, $\otimes r$ occurs just below id or $\mathbf{1}r$ which occur at or near the leaves, and that $\otimes\ell$ and $\mathbf{1}\ell$ are applied greedily (when reading the derivation from bottom to top). We must respect the nesting structure, so that in decomposing $(A \otimes B) \multimap C$ we must apply the $\multimap \ell$ rule below the $\otimes\ell$ rule. The derivation can thus be assumed to have the following form:

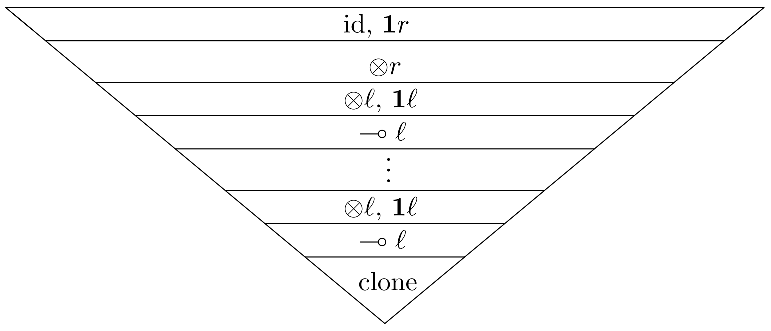


Figure 5.2: The form of a permuted proof.

Note that the conclusion of each $\multimap \ell$ is of the form $\ulcorner R_T \urcorner; \ulcorner \bar{r_i} \urcorner, \ulcorner I \urcorner \otimes \ulcorner W_i \urcorner \vdash_{LL} \ulcorner I \urcorner \otimes \ulcorner Z \urcorner$, where $\ulcorner \bar{r_i} \urcorner$ is a multiset of m - i encodings of actions which have yet to be decomposed, and where there are m applications of $\multimap \ell$ in the derivation. In the conclusion of the bottom application of $\multimap \ell$ is $W_0 = W$. In the right premise of the top application of $\multimap \ell$ is $W_m = Z$.

Claim: There is a transition sequence $r_1, \ldots, r_m$ such that

$$W = W_0 \rhd_{r_1} W_1 \rhd_{r_2} \cdots \rhd_{r_m} W_m = Z$$

allowing us to conclude $W \rhd_T^* Z$.

The claim follows from the simple observation that if the occurrence of $\multimap \ell$ between W_i and W_{i+1} decomposes $q_\alpha \otimes \ulcorner X_\alpha \urcorner \otimes \ulcorner X_{pub} \urcorner \multimap q_\alpha \otimes \ulcorner Y_\alpha \urcorner \otimes \ulcorner Y_{pub} \urcorner$ then $W_i = X_\alpha X_{pub} V$ for some V and $W_{i+1} = Y_\alpha Y_{pub} V$ for the same V. Thus $W_i \rhd_T W_{i+1}$ via $X_\alpha X_{pub} \to_\alpha Y_\alpha Y_{pub}$. This observation is easily turned into an inductive proof. ■

Lemma 5.2.5. *$\Gamma; W \vdash_{AL} Z$ is derivable if and only if there is some U such that $\Gamma; W \vdash_{LL} Z \otimes U$ is derivable.*

Proof: This lemma is an immediate corollary to Lemma 3.1 in [36]. The *if*-direction is immediate since every LL-derivation is also an AL-derivation. The *only if*-direction requires a simple induction on the structure of the LL-derivation. We direct the reader to [36] for more details. ■

Proof: *(of Theorems 5.2.1 and 5.2.2)* The three lemmas combine in the following way to imply soundness and completeness with respect to AL.

$$W \leadsto_T^* Z \Leftrightarrow \text{(by definition)}$$

$$\exists U \text{ s.t. } W \rhd_T^* ZU \Leftrightarrow \text{(Lemmas 5.2.3, 5.2.4)}$$

$$\exists U \text{ s.t. } \ulcorner R_T \urcorner; \ulcorner I \urcorner \otimes \ulcorner W \urcorner \vdash_{LL} \ulcorner I \urcorner \otimes \ulcorner Z \urcorner \otimes \ulcorner U \urcorner \Leftrightarrow \text{(Lemma 5.2.5)}$$

$$\ulcorner R_T \urcorner; \ulcorner I \urcorner \otimes \ulcorner W \urcorner \vdash_{AL} \ulcorner I \urcorner \otimes \ulcorner Z \urcorner$$

■

The above theorems relate the existence of a plan to derivability in pure affine logic. It is also possible to translate local state transition systems into affine logic *theories*. Namely, for every action of the form $X \to_A Y$ we include a new axiom of the form $q_A \otimes X \vdash q_A \otimes Y$. The set of axioms of a theory T is denoted by $\mathtt{Ax}_T$. Theories with axioms of this form are called (pure) *Horn theories*. It can be easily seen that finite Horn theories can be faithfully represented in the formalism above, by listing the non-logical axioms, $\mathtt{Ax_T}$, in the unrestricted context, Γ.

Under this interpretation, the existence of a plan leading from an initial state W to an exact goal Z can be shown to be equivalent to derivability of the sequent $\ulcorner I \urcorner \otimes W \vdash Z$ in the corresponding Horn LL-theory. Also, $\ulcorner I \urcorner \otimes W \vdash Z$ is derivable in the corresponding AL-theory if and only if $\ulcorner I \urcorner \otimes W \vdash Z \otimes U$ is derivable in the LL-theory for some U [36].

The definition of system compliance is essentially an unreachability condition. Namely, we expect that every critical configuration should be unreachable. Under our translation into affine logic, this is the same as saying that the corresponding sequents should not be derivable. Translating the definition of plan compliance is more subtle. It corresponds to a certain kind of normal form proof not containing sequents whose left hand side represents a critical configuration. We do not investigate this relation further.

Chapter 6

Conclusion and Future Directions

6.1 Conclusion

In this thesis we have presented an abstract model for collaboration which addresses the inherently competing notions of protecting and releasing resources. We have discussed what it means to generate a collaborative plan and maintain the participants' confidentiality of information or resources relative to a data confidentiality policy. We provided three definitions of policy compliance with respect to a given set of data confidentiality policies: system compliance, weak plan compliance, and plan compliance. Since the level of trust among collaborating agents may differ depending on the agents and the context, each interpretation of the agents' confidentiality policies is appropriate for a different level of trust among the agents.

We examined the complexity and decidability of the Collaborative Planning Problem with Confidentiality with respect to each of the three definitions of policy compliance, for both well-balanced systems and for general systems which may have un-balanced actions. We showed that in the well-balanced case, all three versions are PSPACE-complete. We also showed that if we fix *in advance* the finite signature we want to use, then these problems can be solved in polynomial time for well-balanced systems. In the general case, however, system compliance is EXPSPACE-complete, and both plan compliance and weak plan compliance are undecidable. This undecidability is due to the importance of remembering the path taken to reach a configuration of the system in both versions of plan compliance. This is in contrast with other undecidability results in the security literature [20] which require the creation of fresh values. Finally, we have demonstrated the logical foundation of our approach with a translation into affine logic which relates the existence of a collaborative plan to derivability.

6.2 Future Directions

Currently we have considered systems with a finite signature that remains unchanged throughout an execution. In practice, agents are continually creating and destroying values. We would like to consider ways to model the generation of fresh values. This would be appropriate when, for example, creating new passwords. The ability to create new values leads naturally to a more thorough investigation of our formalism's ability to distinguish between current and obsolete knowledge. Can our notion of confidentiality policy easily accommodate this distinction?

While data confidentiality policies specify where certain data may or may not go, it does not explicitly specify rules of transmitting the data. For example, the results of some medical tests may be known by both the doctor and the nurse, but only the doctor is allowed to pass the results on to the patient. We hope to integrate this type of information flow policy into our work. Such policies may also provide us with the ability to trace information leaks back to their source. This would provide a sort of auditing mechanism.

The interplay between confidentiality and goal reachability suggests agents who make rational trade-offs between the two. We may be able to gain more insight into the situation by enriching our formalism with an explicit notion of rationality. We would like to consider such "rational adversaries" both in contrast to and in combination with malicious adversaries.

Finally, this work and all the suggestions for future work can also be put into a synchronous context in which there is an explicit notion of time. The existence of such a global clock may affect the properties of systems in interesting ways.

Looking towards possible directions of future research, we note that the computational problems considered in this thesis require full information about each agent's actions and policies. As a participant in the collaboration, however, one would like to be able to make some local assessment of the confidentiality guarantees provided by the system. One advantage to this approach is that each agent could choose the interpretation of their confidentiality policy that best suits their needs. It may be that methods from mechanism design can help to decentralize the analysis. Such an approach is likely to entail the introduction of explicit utility functions and incentives to the model. This may be the right setting in which to explore connections between the definition of plan compliance and types of Nash equilibria as mentioned in Section 2.3.

Some other directions aim at increasing the expressibility of the model. Many algorithms and protocols require the use of random values or fresh values. In the past, the use of the existential quantifier has proven successful as a tool for modeling them [20]. We would like to explore the complexity of the three problems discussed in this thesis, in both the unbalanced and well-balanced cases, when existentials are used.

It would be interesting to consider lifting some of the restrictions about the locality of the actions. Namely, we might consider restrictions that correspond to "read" and "write" privileges with the structure of a lattice [18]. The extra structure could be potentially useful for more natural models of communication and for expressing even richer confidentiality policies.

The confidentiality policies we use in this thesis do not place any explicit restrictions on the actions. Many real-world policies have rules about when an agent may or must perform an action and when an action is prohibited. We may be able to express such policies by imposing parametric conditions on the actions, including boolean conditions. This would allow the model to align more closely with real-world scenarios. Similarly, we might be able to introduce parameters that help determine the context. Currently each context has a corresponding transition system. Parameters could determine which actions are available and which configurations are critical, allowing the model to consider the impact on an agent of changing contexts in various ways.

Finally, time can be an important factor in some places like financial institutions. Using an asynchronous model such as the one in this thesis is not well-suited to these time-sensitive situations. We want to explore a similar synchronous model which has some notion of explicit time. We believe this could provide a rich exploration of confidentiality concerns which are not expressible in the current model.

Bibliography

[1] A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth. Bar fault tolerance for cooperative services. *SIGOPS Oper. Syst. Rev.*, 39(5):45–58, 2005.

[2] R. Alur, P. Černý, and S. Chaudhuri. Model checking on trees with path equivalences. In *TACAS 2007*, pages 664–678. Springer, 2007.

[3] R. Alur, P. Černý, and S. Zdancewic. Preserving secrecy under refinement. In *ICALP '06: Proceedings (Part II) of the 33rd International Colloquium on Automata, Languages and Programming*, pages 107–118. Springer, 2006.

[4] R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Journal of the ACM*, pages 100–109. IEEE Computer Society Press, 1997.

[5] P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter. Enterprise privacy authorization language (EPAL 1.2), 2003. `http://www.zurich.ibm.com/security/enterprise-privacy/epal/Specification/index.html`.

[6] A. Barth, A. Datta, J. C. Mitchell, and H. Nissenbaum. Privacy and conextual integrity: Framework and applications. In *27th IEEE Symposium on Security and Privacy*, May 2006.

[7] A. Barth, A. Datta, J. C. Mitchell, and S. Sundaram. Privacy and utility in business processes. In *Proceedings of 20th IEEE Computer Security Foundations Symposium*, July 2007.

[8] A. Barth, J. C. Mitchell, A. Datta, and S. Sundaram. Privacy and utility in business processes. In *20th IEEE Computer Security Foundations Symposium (CSF 20)*, Venice, Italy, July 2007.

[9] D. E. Bell. Looking back at the Bell-La Padula model. In *Proceedings of the 21st Annual Computer Security Applications Conference*, pages 337–351, December 2005.

[10] D. E. Bell and L. J. L. Padula. Secure computer systems: Mathematical foundations. Technical report, MTR2547, Vol. I, The MITRE Corporation, Bedford, MA, May 1973. (ESD-TR-73-278-I).

[11] W. Bibel. A deductive solution for plan generation. *New Generation Computing*, 4(2):115–132, 1986.

[12] I. Cervesato, N. Durgin, M. Kanovich, and A. Scedrov. Interpreting strands in linear logic. In H. Veith, N. Heintze, and E. Clark, editors, *2000 Workshop on Formal Methods and Computer Security –FMCS'00*, Chicago, IL, 2000.

[13] I. Cervesato and A. Scedrov. Relating state-based and process-based concurrency through linear logic. In R. de Queiroz, editor, *Thirteenth Workshop on Logic, Language, Information and Computation — WoLLIC'06*, pages 145–176, Stanford, CA, 18–21 July 2006. Elsevier ENTCS 165.

[14] R. Chadha, M. I. Kanovich, and A. Scedrov. Inductive methods and contract-signing protocols. In *ACM Conference on Computer and Communications Security*, pages 176–185, 2001.

[15] R. Chadha, J. C. Mitchell, A. Scedrov, and V. Shmatikov. Contract signing, optimism, and advantage. *J. Log. Algebr. Program.*, 64(2):189–218, 2005.

[16] D. Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32(3):333–377, 1987.

[17] A. Clement, H. C. Li, J. Napper, J.-P. Martin, L. Alvisi, and M. Dahlin. Bar primer. In *DSN*, pages 287–296, 2008.

[18] D. E. Denning. A lattice model of secure information flow. *Commun. ACM*, 19(5):236–243, 1976.

[19] D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[20] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Multiset rewriting and the complexity of bounded security protocols. *Journal of Computer Security*, 12(2):247–311, 2004.

[21] K. Erol, D. S. Nau, and V. S. Subrahmanian. Complexity, decidability and undecidability results for domain-independent planning. *Artif. Intell.*, 76(1-2):75–88, 1995.

[22] D. Garg, L. Bauer, K. D. Bowers, F. Pfenning, and M. K. Reiter. A linear logic of authorization and knowledge. In *Proceedings of the 11th European Symposium on Research in Computer Science (ESORICS'06)*, volume 4189 of *Springer Lecture Notes in Computer Science*, pages 297–312. Springer-Verlag, 2006.

[23] D. Garg and F. Pfenning. Non-interference in constructive authorization logic. In *Proc. of the IEEE Computer Security Foundations Workshop (CSFW)*, pages 283–296, 2006.

[24] V. Gehlot and C. Gunter. Normal process representatives. In *Proc. of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 200–207, Philadelphia, PA, 1990.

[25] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–102, 1987.

[26] J.-Y. Girard. Linear logic: Its syntax and semantics. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, volume 222 of *London Mathematical Society Lecutre Notes*, pages 1–42. Cambridge University Press, 1995.

[27] J. A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pages 11–20, 1982.

[28] R. Greenstadt, J. P. Pierce, and M. Tambe. Analysis of privacy loss in distributed constraint optimization. In *21st Conference on Artificial Intelligence (AAAI)*, Boston, Massachusetts, July 2006.

[29] R. Greenstadt and M. D. Smith. Collaborative scheduling: Threats and promises. In *Fifth Annual Workshop on Economics and Information Security*, Cambridge, England, June 2006.

[30] J. Hopcroft and J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Thoeretical Computer Science*, 8(2):135–159, 1979.

[31] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[32] Z. G. Ives, N. Khandelwal, A. Kapur, and M. Cakir. ORCHESTRA: Rapid, collaborative sharing of dynamic data. In *Conference on Innovative Data Systems Research (CIDR)*, pages 107–118, 2005.

[33] M. Kanovich, P. Rowe, and A. Scedrov. Collaborative planning with confidentiality. *Submitted*. `http://www.math.upenn.edu/~rowep/CPWC_full.pdf`.

[34] M. Kanovich, P. Rowe, and A. Scedrov. Policy compliance in collaborative systems. Preliminary version. Revised version to appear in *22nd IEEE Computer Security Foundations Symposium,* 2009. `http://www.math.upenn.edu/~rowep/Policy_Compliance.pdf`.

[35] M. Kanovich, P. Rowe, and A. Scedrov. Collaborative planning with privacy. In *20th IEEE Computer Security Foundations Symposium (CSF 20)*, Venice, Italy, July 2007.

[36] M. Kanovich and J. Vauzeilles. The classical AI planning problems in the mirror of Horn linear logic: semantics, expressibility, complexity. *Mathematical Structures in Computer Science*, 11(6):689–716, 2001.

[37] M. I. Kanovich. Horn programming in linear logic is NP-complete. In *Proc. 7-th Annual IEEE Syposium on Logic in Computer Science, Santa Cruz*, pages 200–210, 1992.

[38] M. I. Kanovich. The complexity of Horn fragments of linear logic. *Annals of Pure and Applied Logic*, 69:195–241, 1994.

[39] M. I. Kanovich. Linear logic as a logic of computations. *Annals of Pure and Applied Logic*, 67:183–212, 1994.

[40] M. I. Kanovich. The direct simulation of Minsky machines in linear logic. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Advances in Linear Logic*, volume 222 of *London Mathematical Society Lecture Notes*, pages 123–145, 1995.

[41] M. I. Kanovich. Petri nets, Horn programs, linear logic and vector games. *Annals of Pure and Applied Logic*, 75(1-2):107–135, 1995.

[42] M. I. Kanovich. The expressive power of horn monadic linear logic. In *CSL*, pages 39–53, 2001.

[43] R. M. Karp and R. E. Miller. Parallel program schemata. *Journ. Computer and System Sciences*, 3(2):147–195, May 1969.

[44] A. P. Kopylov. Decidability of linear affine logic. In *LICS '95: Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science*, page 496, Washington, DC, USA, 1995. IEEE Computer Society.

[45] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications.* Springer, New York, NY, USA, 1997.

[46] P. Lincoln, J. Mitchell, A. Scedrov, and N. Shankar. Decision problems for propositional linear logic. *Annals Pure Applied Logic*, 56:239–311, 1992.

[47] P. D. Lincoln and N. Shankar. Proof Search in First-order Linear Logic and Other Cut-free Sequent Calculi. In *Ninth Annual Symposium on Logic in Computer Science* (Paris, France), pages 282–291. IEEE Computer Society Press, 1994.

[48] R. J. Lipton. The reachability problem requires exponential space. Technical report, Department of Computer Science, Research Report 62, Yale University, 1976.

[49] M. Masseron, C. Tollu, and J. Vauzeilles. Generating plans in linear logic I: Actions as proofs. *Theoretical Computer Science*, 113(2):349–370, 1993.

[50] E. W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM J. Comput.*, 13(3):441–460, 1984.

[51] D. McDermott and J. Hendler. Planning: What it is, what it could be, An introduction to the special issue on planning and scheduling. *Artificial Intelligence*, 76:1–16, 1995.

[52] J. McLean. Security models. In J. Marciniak, editor, *Encyclopedia of Software Engineering*. John Wiley & Sons, 1994.

[53] J. Meseguer. Conditional rewriting logic as a unified model of concurrency. In *Selected papers of the Second Workshop on Concurrency and compositionality*, pages 73–155, Essex, UK, 1992. Elsevier Science Publishers Ltd.

[54] A. C. Myers, A. Sabelfeld, and S. Zdancewic. Enforcing robust declassification and qualified robustness. *Journal of Computer Security*, 14(2):157–196, 2006. Extended abstract in CSFW pages 172-186, 2004.

[55] N. J. Nilsson. *Principles of Artificial Intelligence*. Springer, Berlin, 1980.

[56] C. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, Inc., Reading, MA, USA, 1994.

[57] S. Pinsky. Absorbing covers and intransitive non-interference. In *SP '95: Proceedings of the 1995 IEEE Symposium on Security and Privacy*, page 102, Washington, DC, USA, 1995. IEEE Computer Society.

[58] C. Rackoff. The covering and boundedness problem for vector addition systems. *Theoretical Computer Science*, 6:223–231, 1978.

[59] J. Reagle and L. F. Cranor. The platform for privacy preferences. *Communications of the ACM*, 42(2):48–55, 1999.

[60] J. C. Reynolds. Syntactic control of interference. In *Symposium on Principles of Programming Languages (POPL)*, pages 39–46, 1978.

[61] A. W. Roscoe. What is intransitive noninterference. In *In Proc. of the 12th IEEE Computer Security Foundations Workshop*, pages 228–238, 1999.

[62] U. Saranli and F. Pfenning. Using constrained intuitionistic linear logic for hybrid robotic planning problems. In *ICRA*, pages 3705–3710, 2007.

[63] J. E. Savage. *Models of Computation: Exploring the Power of Computing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1997.

[64] A. Scedrov. Linear logic and computation: A survey. In H. Schwichtenberg, editor, *Proof and Computation, Proceedings Marktoberdorf Summer School 1993*, pages 281–298. NATO Advanced Science Institutes, Series F, Springer-Verlag, Berlin, 1994.

[65] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[66] Y. Shoham and M. Tennenholtz. Non-cooperative computation: boolean functions with correctness and exclusivity. *Theor. Comput. Sci.*, 343(1-2):97–113, 2005.

[67] N. E. Taylor and Z. G. Ives. Reconciling while tolerating disagreement in collaborative data sharing. In *ACM SIGMOD Conference on Management of Data*, pages 13–24, 2006.

[68] G. Wiederhold, M. Bilello, V. Sarathy, and X. Qian. Protecting collaboration. In *Proc. 19th NIST-NCSC National Information Systems Security Conference*, pages 561–569, 1996.

[69] H.-C. Yen. Introduction to Petri net theory. *Recent Advances in Formal Languages and Applications*, 25:343–373, 2006.

[70] S. Zdancewic and A. C. Myers. Robust declassification. In *Proc. 14th IEEE Computer Securtiy Foundations Workshop (CSFW)*, pages 15–23, 2001.