Security via Noninterference

Analyzing Information Flows

Dipl.-Math. Sebastian Eggert

Dissertation
zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
(Dr.-Ing.)
der Technischen Fakultät
der Christian-Albrechts-Universität zu Kiel
eingereicht im Jahr 2014

Kiel Computer Science Series (KCSS) 2014/5 v 1.0 dated 2014-10-28

ISSN 2193-6781 (print version)
ISSN 2194-6639 (electronic version)

Electronic version, updates, errata available via https://www.informatik.uni-kiel.de/kcss

The author can be contacted via http://www.theorie.informatik.uni-kiel.de

Published by the Department of Computer Science, Kiel University

Theoretical Computer Science

Please cite as:

▷ Sebastian Eggert. *Security via Noninterference – Analyzing Information Flows*. Number 2014/7 in Kiel Computer Science Series. Department of Computer Science, 2014. Dissertation, Faculty of Engineering, Kiel University.

```
@book{Eggert2014,
  author    = {Sebastian Eggert},
  title     = {Security via Noninterference -- Analyzing Information Flows},
  publisher = {Department of Computer Science, CAU Kiel},
  year      = {2014},
  number    = {2014/7},
  series    = {Kiel Computer Science Series},
  note      = {Dissertation, Faculty of Engineering,
               Kiel University.}
}
```

About this Series

The Kiel Computer Science Series (KCSS) covers dissertations, habilitation theses, lecture notes, textbooks, surveys, collections, handbooks, etc. written at the Department of Computer Science at Kiel University. It was initiated in 2011 to support authors in the dissemination of their work in electronic and printed form, without restricting their rights to their work. The series provides a unified appearance and aims at high-quality typography. The KCSS is an open access series; all series titles are electronically available free of charge at the department's website. In addition, authors are encouraged to make printed copies available at a reasonable price, typically with a print-on-demand service.

Please visit `http://www.informatik.uni-kiel.de/kcss` for more information, for instructions how to publish in the KCSS, and for access to all existing publications.

1. Gutachter:	Prof. Dr. Thomas Wilke Christian-Albrechts-Universität zu Kiel
2. Gutachter:	Prof. Dr. Peter Ryan University of Luxembourg

Datum der mündlichen Prüfung: 10. Juli 2014

Zusammenfassung

Die Sicherheit von Informationssystemen ist in der heutigen Zeit von zentraler Bedeutung. Die große Anzahl aufgedeckter Sicherheitslücken zeigt, dass neue Methoden zur Entwicklung sicherer Systeme notwendig sind. Um solche Methoden auf eine formale Basis zu stellen, wird geeignete Theorie zur Spezifikation von Sicherheit in Systemen benötigt. Ein weitverbreiteter Ansatz ist, die Begriffe Nichtinterferenz und Informationsflusssicherheit in den Mittelpunkt einer solchen Theorie zu stellen. Mit diesen lässt sich die Abwesenheit unerlaubter Informationsflüsse und verdeckter Informationskanäle beschreiben und analysieren. In dieser Arbeit wird die bestehende Theorie der Nichtinterferenz für zustandsbasierte, asynchrone Systeme erweitert und mit neuen Techniken angereichert, um sowohl ein tieferes Verständnis und eine breitere Anwendbarkeit zu erlangen, als auch eine formale Basis für die Entwicklung sicherer Informationssysteme zur Verfügung zu stellen.

Einen Schwerpunkt dieser Dissertation bilden neue Resultate zu dem Begriff der intransitiven Nichtinterferenz. Eines davon ist eine vollständige Charakterisierung durch Abwicklungsrelationen, durch die erst einen Polynomzeitalgorithmus zu deren Verifikation ermöglicht wird, der ebenfalls Teil dieser Arbeit ist.

Ein zweiter Schwerpunkt ist die Erweiterung der bisherigen Nichtinterferenzdefinitionen um sich während der Laufzeit ändernde Sicherheitsrichtlinien. Um den sich daraus ergebenden Sicherheitsanforderungen gerecht zu werden, wird eine neue Theorie der sogenannten dynamischen Nichtinterferenz entwickelt und mit bisherigen Ansätzen verglichen. Deren Anwendbarkeit wird durch verschiedene Beispiele und eine aufwändigere Sicherheitsanalyse eines verteilten, dynamischen Zugriffskontollsystems demonstriert.

Einen dritten Schwerpunkt dieser Arbeit bilden algorithmische Fragestellungen, insbesondere die Frage der Entscheidbarkeit und Komplexität der analysierten Sicherheitsbegriffe. Dies führt sowohl zu neuen Unent-

scheidbarkeitsresultaten für die Verifikation bisheriger Sicherheitsbegriffe, als auch zu neuen effizienten Algorithmen für die Analyse von sowohl existierenden, als auch in dieser Arbeit entwickelten unterschiedlichen Varianten von Nichtinterferenz.

Abstract

Nowadays, the security of information systems is of crucial importance. The large number of detected security vulnerabilities in many systems indicates that new methods for developing secure systems are necessary. These require an appropriate formal foundation. A widely used approach revolves around the notions noninterference and information flow. They allow to express and analyze the absence of illegal information flows and covert channels. In this thesis, the framework of noninterference for state-based asynchronous systems is extended and enriched with new techniques in order to gain a deeper understanding and a broader applicability. As a result, a formal foundation for developing secure systems is obtained.

First, new results for the notion of intransitive noninterference are obtained. In particular, a complete characterization by unwinding relations makes the development of a polynomial-time verification algorithm possible in the first place.

Second, the previous noninterference definitions are extended with support for policies changing during execution. To capture all resulting security requirements, a new theory of so-called dynamic noninterference is developed and compared to previous approaches. The applicability of this framework is demonstrated by several examples and a complex case study of a distributed dynamic access control system.

Third, algorithmic problems are examined, in particular with regard to the question of decidability and complexity of the analyzed security definitions. New undecidability results for some of the present security definitions are obtained, and new efficient algorithms for the verification of both the previously existing and in this thesis developed different notions of noninterference are established.

Acknowledgements

First of all, I would like to express my appreciation and thanks to my mentor Thomas Wilke for all his inspiration, help, and guidance during my time in his research group for theoretical computer science at Kiel University.

A special thanks goes to Henning Schnoor, not only for participating on all of my publications, also for an uncountable number of interesting discussions, without this thesis would have never become what it is.

I thank Ron van der Meyden for several constructive and insightful discussions on noninterference and supporting me with the opportunity to visit him twice at UNSW, Sydney.

I thank Peter Ryan for reviewing this thesis and for being in my thesis committee.

I thank my office mates Klaas Ole Kürtz and Sebastian Preugschat for having an enjoyable and entertaining time. Also I would like to thank all other members of the theoretical computer science group at Kiel University for having had a good time and for playing many rounds of darts.

Thanks are also due to my parents for the continuous support they have given me not only throughout my time at university. Finally, I would like to thank Tomma Radtke for always believing in me and her tireless support.

Contents

Chapter 1

Introduction

The security of computer systems is crucial in today's connected world. Nevertheless, for the year 2012, the Norton Cybercrime Report [Cor12] calculates the annual costs associated with global consumer cybercrime at US $110 billion. One reason for these costs is the high amount of attacks against several computer systems. The success of these attacks is often due to security vulnerabilities which stem from implementation or conceptual flaws in the affected systems. This illustrates the enormous importance of developing secure systems, however, the large number of detected security issues let one imagine the difficulty to do this successfully.

While attackers need to find only one vulnerability of a system to compromise it, developers must secure the system against all possible attacks—even those the developers themselves are not aware of. To rule out as many security flaws as possible in different systems, one needs a very general approach. Such an approach should enable the developer, as automatically as possible, to verify security and find potential vulnerabilities before deploying the system. In order to address this problem, we take a mathematical and formal approach for expressing the analyzed system and the security requirements.

On a very general and abstract level, one can formulate insecurity as a property that certain security sensitive information can get into places which it should not reach or that it can be manipulated in a prohibited manner. Therefore, one needs a certain understanding where information belongs and where it should not end up. This is expressed by a security specification, also called a policy, which has to capture all relevant security requirements of the system. For this purpose, the system is partitioned into so-called security domains. These form units depending on their confidentiality

and integrity of information. A policy states from which security domain information may reach another security domain. When information reaches another security domain, we speak of an information flow. For the security of a system, the absence of those information flows that are not allowed by the policy are relevant. The property that there is no information flow from one security domain to another is called noninterference.

This very general idea forms the basis of all security definitions in the field of information flow security and noninterference. It leads to many different interpretations. On the one hand, these differences are due to diverse types of systems and semantic models. To mention only a few of them, information flow security has been successfully adopted to state-based transition systems, process algebras, program languages, and reactive systems. On the other hand, the policy can also be interpreted in multiple ways. The interpretation of a policy in which a certain flow of information is either never or always allowed is usually either too restrictive to represent real systems or too coarse to capture all security related restrictions. This results in less restrictive but more expressive policies. These multifaceted directions explain why such a huge amount of research on information flow and noninterference has been carried out in the last decades.

Originally, noninterference was designed to uncover forbidden information flows in access control systems. Goguen and Meseguer [GM82] did the first work on this subject and used simple but very general deterministic state-based systems. This system model is specific enough to express real systems and sufficiently general to capture a large number of possible systems. Their definition is not limited to access control systems, and it also has been applied to various systems, for example to verify the security of small operating systems. Over time, their original definition has turned out to be too restrictive for expressing the security requirements of many systems. Several weakenings and generalizations of this initial idea have been proposed. One of the first and until today probably the most important generalization, introduced by Haigh and Young [HY87], is the notion of intransitive noninterference. While Goguen and Meseguer's definition only allowed to specify direct information flows between particular security domains, Haigh and Young's definition allowed information flows across multiple security domains. This led to a significant generalization

and better applicability of this concept. Rushby [Rus92] and afterwards van der Meyden [vdM07] formalized and extended the notion of intransitive noninterference, and the latter corrected it with regard to the application in completely asynchronous systems. In this thesis, we take up this previous theory of noninterference and contribute to a deeper understanding by providing new characterizations and extend them in particular with regard to the development of efficient algorithms for the verification of security.

Based on the developed framework, in this thesis, the previous notions of noninterference are extended to support policies that change during execution of the system. In the previously mentioned works on noninterference, it was basically assumed that the policy is a relation between security domains which does not change during a system run. But especially in discretionary access control systems, it is common that access rights can be changed during the system run. In the formalism used in this thesis, this will be expressed by dynamic policies. However, there is only little research on this subject—the only work that uses a similar system model as we do is [Les06]. In their definitions, we found inconsistencies which motivated us to develop new definitions for dynamic noninterference. In this thesis, we provide an extension of the previous theory that supports dynamic policies and reveals new kinds of undesired information flows which results from changes in the policy. We show the strength and flexibility of our theory by successfully applying it to a dynamic distributed access control system.

We also show the applicability of several of the analyzed and developed notions of noninterference by providing efficient algorithms for the detection of security flaws. According to the acceptance and practicability of security definitions, it is crucial that systems can be verified automatically. A focus of this work is to investigate this property and to examine the decidability and complexity of the considered noninterference properties. We provide efficient algorithms for those noninterference definitions, and we could show that the verification problem is in polynomial time.

1.1 Computer Security

In this section, we establish the context where information flow security and noninterference belong, and we introduce fundamental ideas and concepts of computer security.

Computer security is about the design and analysis of computer systems with respect to security properties. Due to the high number of attacks on computer systems and their news coverage, most people have an intuition of what computer security means or how a secure system should behave. However, for formal reasoning about security, a more precise definition about what security is and which security properties are relevant, is needed.

Very generally, computer security is about the protection and the security of computer systems, while information security is about the protection of information, mainly within some computer system. Since, at least in a computer system, none of these two can reasonably be considered in isolation, we do not make any distinction between them and use them interchangeably.

First, we will clarify which security properties this work addresses—we call this a *security model*. A security model is a formal statement about a system's *confidentiality*, *integrity*, or *availability* requirement. According to [usl11],

> "the term *information security* means protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction in order to provide—
>
> (A) integrity, which means guarding against improper information modification or destruction, and includes ensuring information nonrepudiation and authenticity;
>
> (B) confidentiality, which means preserving authorized restrictions on access and disclosure, including means for protecting personal privacy and proprietary information; and
>
> (C) availability, which means ensuring timely and reliable access to and use of information."

In this thesis, we focus mainly on the property of confidentiality, and due to its duality, on the property of integrity. Availability is essentially not considered in this work.

Confidentiality is about concealing data, information, or resources from someone who should not have access to it. In the next sections, we will detail this very informal statement by providing a clear description of what is protected and from whom. Keeping information secret is fundamental for governments and industry but also in every modern computer system which is connected to a network or users have access to. Several mechanisms have been established in the design of secure systems to achieve confidentiality properties. Here, we will mention only a few of them. One of the most general mechanism is an access control system which restricts the access to particular data. However, in general, data should not be kept secret from everyone nor at any time, since otherwise such a system would be useless. This requires discretionary access control systems, where the access rights may change during time. We will discuss access control systems in a later chapter in detail. Another very general way to enforce confidentiality of data is to encrypt it. Roughly speaking, encryption scrambles data into a cipher-text, which is incomprehensible for anyone that does not have the key to decrypt it. However, this mechanism shifts the problem of keeping data secret to the problem of keeping the key secret. Besides protecting data, hiding the use of system resources is also an aspect of confidentiality. If the use of system resources leaks information about confidential data, we call it a covert channel.

Integrity is about the trustworthiness of data, information, or resources, and one usually understands by that preventing unauthorized manipulation or changing of data. For integrity mechanisms, one distinguishes between prevention and detection mechanisms. Prevention mechanisms are used to enforce that data cannot be manipulated. This can be achieved by an access control mechanism which denies unauthorized altering of data. In contrast, a detection mechanism is used to reveal that data has been modified. A common mechanism are cryptographic hash functions which compute a

short value from a large amount of data and it is assumed that this value changes if the data has been changed or manipulated.

Availability is about the ability to use data, information, or resources. Availability depends highly on the used system and there are no standard mechanisms to achieve it. The attempt to block the availability of a system is generally denoted as a denial of service attack.

1.1.1 Security Policies

When dealing with security, it is required to have an understanding what is allowed and what is not allowed in a system to distinguish the desired system functionality from a security breach. This is usually achieved by setting up a security policy—a specification which states exactly what is allowed and what is forbidden. Such a specification can range from an informal description to a precise formal statement and may be given for different levels of abstraction and formalisms. Generally, we say that a system is "secure" if it satisfies the policy. Note that security has always been seen as a property relative to a policy and is never an absolute property of a system.

One of the earliest works defining security polices is Lampson's notion of confinement [Lam73], stating that a confined program cannot leak any information to any other program except its caller.

In the literature, several definitions of a security policy exist. One of the simplest is Bishop's [Bis03]:

> "A security policy is a statement of what is, and what is not, allowed."

In this work, we focus on security policies which are used to specify confidentiality or integrity requirements of a system. Later, we will provide a formalism for specifying policies used in this thesis to define allowed and forbidden information flows, and we will also provide precise semantics for formal reasoning about security.

1.1.2 Formal Analysis of Security

Most security flaws in modern IT systems stem from implementation or conceptual errors. To detect these flaws automatically or with the support of tools, formal methods are required which should be applicable at different states of the development process of a system and also afterwards. The use of formal methods helps one to have a better understanding of the designed system. In a large system, tool-based verification is desirable when a manual analysis becomes unfeasible.

For a formal specification of a system's security, a formal description of the system, which could be a transition system or a code of a program, and a formal security model is needed. In a formal analysis, one verifies that the system satisfies the security policy. The correctness of the policy itself, in the sense that it captures the desired security properties, is out of the scope of a formal analysis.

Due to this limitation, it is important to keep the policy simple and understandable for both the one who specifies the security properties and also a possible system designer who implements the policy in the system. Also a clear separation between the policy and the system is desirable to avoid that flaws in the system design influence the policy or that an unintended property of the policy is directly implemented into the system. Nevertheless, a clear separation of the policy and the system is often not possible—at least if a policy depends on the states of the system. However, in this case, it is difficult to overview the overall properties of such changing and state depending policies. Formal methods and automatic tools can then also provide a useful tool to tackle complex policies that are hard to understand.

Over time, several systematic criteria for the evaluation of security have been developed. One of the earliest handbooks for formal evaluation of computer systems is the *Orange Book* [US 85]. A more recent approach for security evaluations is the Common Criteria [com12]. A formal modeling of the system and a formal analysis are required on its highest evaluation assurance level.

1.1.3 Covert Channels

The nature of a computer system is to communicate with its environment, which could be a user interacting with a system locally, another device attached to the system, or another computer connected by a network. Therefore, every system has a communication interface necessary to provide some intended service in order to be a useful system. These kinds of communication channels are built in by the designer of the system.

From a security point of view, one can optimistically assume that the designer overviews the properties of the build-in communication channels and has carefully implemented mechanism to prevent any abuse or unintended use of these channels.

These communication channels can be seen as overt channels, as they are designed for communication and the system designer and any environment communicating with the system is aware of these channels.

Any communication channel which is not intended to be used for communication is a *covert channel*. Such a covert channel is relevant for security if it transmits secure information to some user that is not allowed to receive it, with respect to some security policy or at least the intended design of the system.

Bishop [Bis03] briefly defines a covert channel as:

> "A covert channel is a path of communication that was not designed to be used for communication."

The difficulty of detecting covert channels is that usually neither the designer of a system nor a security analyst is aware of all possible covert channels that might exist in a system.

Covert channels might exist in many different types, including the following.

Causal channels are covert channels that arise from the control structure of a system or from dependencies within a system. If some part of a system, considered to be non-confidential, has some causal dependencies on a part of system that is considered to be confidential. These causal channels may arise if different parts of a system share a common state space. This is the kind of covert channels this thesis addresses.

Timing channels are covert channels that leak information about confidential computations by the amount of time that a system needs to perform a particular task. A common example is the standard algorithm for modular exponentiation, whose run-time strongly depends on the number of ones in the exponent [Koc96]. Counter measures to rule out timing channels are introducing dummy actions or statements in programs that have the same running times independent of the values of some confidential data [ZAM12].

Termination channels may leak confidential information if the termination behavior of a system depends on it. However, in general one cannot decide, based on the observation, whether a run is non-terminating ,or it might need very long time and will terminate eventually. Hence since non-termination is not observable, one has to fix some bounded time interval and can observe if a system terminates within this interval. However, then this communication channel is a timing channel, rather a termination channel.

Resource exhaustion channels may leak information by exhausting some common resource such as memory or disk space or by consuming a great deal of computing power. In the latter case, it is again closely related to timing channels.

Physical channels are a vast number of possible communication channels, including all channels based on a physical level of a real system, rather than on the program or firmware running on a device. Such communication channels include the power consumption or the produced heat of a device depending on confidential computations. Other examples include the noises produced by printing on a dot matrix printer [BDG+10] or vibrations emitted by typing on a keyboard, measured by a mobile phone accelerometer [MVCT11].

1.2 Information Flow Security

Information flow security follows a very general approach to achieve confidentiality and integrity properties. The main idea is to partition the system

into security domains and to define in a policy between which security domains information may flow. Historically, the intuition was that the security domains are processes running on an operating system. Even though we do not want to restrict ourselves to the modeling of processes, this intuition fits best to the used framework. In our modeling, the security domains are active and can perform actions and that is why we call them agents. In general, it is possible that the agents share some system resources. In terms of processes, this could be the sharing of computing power as they run on the same CPU or sharing the system's memory. In our abstraction, we express this as a global state space. These shared resources can enable agents to communicate in an unintended way through a covert channel. If an agent can communicate with another agent, we say there is an information flow from one agent to another. Communication means the transmission of any information which can only be just a single bit. The opposite of an information flow is noninterference which is the security relevant property, since we are interested in the absence of illegal information flows. The concept of noninterference was introduced by Goguen and Meseguer [GM82] to provide a formal framework for reasoning about information flow properties and for specifying security policies in this context. Informally, they defined noninterference as:

> "one group of users, using a certain set of commands, is *noninterfering* with another group of users if what the first group does with those commands has no effect on what the second group of users can see."

This is essentially the main idea of all works about noninterference. Our interpretation of noninterference follows Rushby's [Rus92] more precise, but still informal definition of noninterference:

> "A security domain u is noninterfering with domain v if no action performed by u can influence subsequent outputs seen by v."

When considering security, it is essential to have a policy. In the context of information flow and noninterference, such a policy specifies from where to where information may flow, or conversely, which agent is not allowed to

interfere with another agent. An information flow policy, hereafter referred to as a policy, is a relation between the agents and describes from which agent to which other agent information is *allowed* to flow. That means that we have for agents u and v, the pair (u, v) is in the policy if and only if information may flow from u to v. Noninterference is then required by the absence of such a pair, i. e., if (u, v) is not in the policy, then u should be noninterfering with v. Since these kinds of information flow policies can only restrict the information flow *between* security domains, but not within a security domain, it is required that this relation is reflexive, i. e., it is always allowed that information flows from a security domain to itself. The information flow within a security domain has to be analyzed with either other techniques like model checking or with a more finely graded information flow analysis.

The simplest non-trivial policy consists of two security domains. They are usually denoted with H for high and L for low and the policy allows information flow from low to high but not in the other direction. A lot of the noninterference literature deals with this simple policy.

In early works, it was required that the policy is transitive. The underpinning argument for this requirement was that if information is allowed to flow from a domain u to a domain v and if it is allowed to flow from a domain v to a domain w, then it is necessary that it is also allowed to flow from u to w. However, for many situations, the restriction that a particular domain is always noninterfering with another domain is too restrictive. Both Goguen and Meseguer in [GM84] and Haigh and Young [HY87] recognized this and proposed weakenings to have more applicable definitions. The suggestion of Haigh and Young [HY87] was intransitive noninterference. This definition takes information flows through several security domain into account.

In its pure form, noninterference does not provide any restriction on which or how much information may flow if information flow is permitted. It only provides a coarse formulation whether information flow is permitted or not. In the case that information flow is allowed, there are no further restrictions which information is allowed to be transmitted—it can be seen as opening the flood doors.

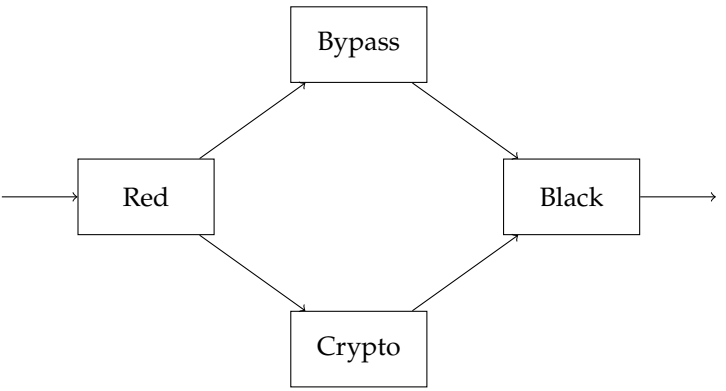

Figure 1.1. Information flow in an encryption controller

A classic example [Rus81] shows the limitations of this approach in an abstraction of an end-to-end encryption controller, depicted in Figure 1.1. Plain-text messages arrive at the Red interface. Then the body of the message is split from its header information and is passed to the Crypto part of the controller which encrypts it and forwards it to the Black output interface. The header information will not be encrypted and is passed through the Bypass device. In the Black part the information from the Bypass and from the Crypto part are reassembled and sent to the desired destination.

Such an abstract policy only guarantees that any information received by Black went through Bypass or Crypto. But it does neither guarantee that the whole body of a message is passed through the Crypto part nor that this part indeed encrypts the data. This needs a further security analysis of the Red part to make sure that splitting is done correctly and an analysis of the Crypto part to guarantee the desired functionality.

1.2.1 Enforcement of Noninterference

Despite the question of how to define and verify noninterference, in many practical systems it is required that a noninterference property is enforced.

The standard technique to prevent different parts of a system from communicating or interfering with each other is the use of a separation kernel [Rus81]. A separation kernel is a small piece of software and usually a part of the system's kernel. It supervises and enforces all processes running on the system to obey a particular policy and separates them to avoid any direct or indirect communication between them.

1.2.2 Attack Model

The security problem we address in this work relates to design or implementation flaws in systems. In our modeling and analyzing of systems and their security properties, we do not model an attacker explicitly. Implicitly, we assume that any agent can take the role of an attacker. We do not distinguish between an attacker and an ordinary user of a system. Hence, an attacker has the same possibilities that any user has. In our systems, these are to interact with the system by performing actions and to make observations during a run or about the output of the system. We do not take into account any attacks which modify or manipulate the system or its behavior. Following Kerckhoffs's principle [Ker83], we always assume that every user exactly knows the precise description and functionality of the system with which it interacts.

The attacks of interest are those in which the attack brings the system into a state in which the attacker gains information that is forbidden by the policy, about the behavior and the interaction of other agents with the system. However, we do not explicitly consider coalitions of different attackers which could possibly share their information or may have communication channels lying outside of the analyzed system.

1.2.3 Applications of Noninterference

Noninterference is a very general way to specify and analyze confidential and integrity properties and to uncover previously covert channels. At least historically, the main application of this theory relates to the detection of undesired information flows in access control systems. Besides this area, noninterference has been successfully adapted to other applications.

One classic area is the detection of information flows in (small) operating systems. Successfully analyzed systems include the seL4 microkernel [MMB+12], a separation kernel used by Motorola [MWTG00], the Infineon SLE66 smart card processor [vO04], and an operating system for multiapplicative smart cards [SRS+00].

Lafrance and Mullins [LM03] used information flow methods to detect denial of service attacks, which address the security property of availability, rather than confidentiality and integrity which are the properties usually addressed by information flow. There are also applications of applying noninterference techniques to the formalization of intrusion detections systems [KR02].

1.3 Contribution and Structure of this Thesis

After this introductory chapter, we give a systematic introduction to the used framework and to noninterference in Chapter 2. After having provided some prerequisites, we introduce the used system model formally. In the remainder of the chapter, we present general techniques for expressing information flow as we will apply them later to different noninterference definitions.

In Chapter 3, we review and characterize different notions of noninterference for systems with a static policy. These notions are Goguen and Meseguer's transitive noninterference, Haigh and Young's extension to intransitive noninterference, and van der Meyden's corrections of that. This thesis contributes especially to intransitive noninterference with several new characterizations, including sound and complete unwindings. Some of these results were published in [EvdMSW11] or are submitted for publication in [EvdMSW13].

In Chapter 4, we extend the results of the previous chapter to systems with dynamically changing policies. We introduce notions of transitive and intransitive noninterference for systems with dynamic policies. We also work out a notion which lies in-between these two definitions and also provide an alternative definition to our previously given intransitive notion of dynamic noninterference, which fits better with asynchronous

systems and to applications on dynamic access control systems. At the end of the chapter, we compare our new definition to the only previous one for state-based system we have found in the literature. Some of these results were published in [ESW13] and some appear in a technical report [ESW12].

In Chapter 5, we analyze the decidability and the complexity of the verification problem for our security notions on finite-state systems. For the cases where we have a characterization by a polynomial-size unwinding, we present efficient algorithms and analyze their running times. In particular, we present the first polynomial-time algorithm for verifying intransitive noninterference. For the notion of dynamic intransitive noninterference, we provide an NP-completeness result. For some other noninterference definition, where the allowed information depends on the observations of the agents, we show that the verification problem is undecidable. Additionally, for some of the noninterference definitions for static policies, we also show how to detect information flows and how to compute minimal policies. We also provide a very general construction idea for analyzing the complexity of the verification problem for noninterference in automata-like system models. Some of these result are published in [EvdMSW11] and [ESW13] or submitted for publication in [EvdMSW13]

In Chapter 6, we give an introduction to access control systems and strengthen our security definitions for systems with dynamic policies by applying it to a distributed access control system. A large part of the chapter is the modeling and the transformation of this access control system in our system model, which enables us to analyze its security.

Chapter 7 provides an overview about noninterference and information flow models, other then ours. This includes nondeterministic state-based systems, process algebras, computational models, and program language-based systems.

We conclude in Chapter 8.

Chapter 2

Systematic Introduction to Noninterference

In this chapter, we introduce the system and policy model used throughout this thesis and fundamental techniques for characterizing and expressing noninterference, needed in the next chapters for analyzing the different definitions for noninterference. In Section 2.1, the used system model is presented, and in Section 2.2, we provide the syntax of information flow policies. In Section 2.4, we present the syntax and semantics of the system diagrams, which we will use to depict the examples of systems in this thesis. The semantics of system runs and the observations of the agents are explained in Section 2.5. General techniques for expressing noninterference based on trace equivalence are introduced in Section 2.6. In the remainder of this chapter, we present the general idea of information flow in Section 2.7 and noninterference in Section 2.8, for the used system and policy model.

2.1 System Model

Throughout this thesis we use a deterministic state-based system model. This is the same formal framework as used in [vdM07], which is only a slight modification of that in [Rus92].

Formally, deterministic systems are labeled transition system with a unique initial state. To keep the results general, we do not restrict ourselves to finite-state systems. However, when we consider the verification problem and analyze its complexity, we will need a finite state space. The transitions are labeled with actions of some finite set (or alphabet) A. We explicitly

require that every state is reachable from the initial state. This will be important in several results in this thesis. Each action is assigned to a specific agent by the dom function, which assigns each action an agent. We say that the agent $\mathrm{dom}(a)$ owns the action a. The transition function is a function $\mathrm{step}\colon S \times A \to S$ and assigns each possible state and each action the follow-up state. Instead of $\mathrm{step}(s, a)$, we also write $s \cdot a$ and say the action a is performed in (the state) s. We stress that the system is input enabled, which means that every action can be performed in every state. A run of a system is a sequence of consecutively performed actions. Often runs are written as an alternating sequence of actions and states. However, since the system is deterministic, the reached states are uniquely defined by the sequence of actions. Thus, we denote a sequence of actions as runs or alternatively as traces.

In addition, these transition systems are equipped with an observation function obs which assigns every agent u and every state s the observation of agent u when the system is in the state s.

A deterministic state-based systems consist of

- a set of states S,
- an initial state $s^I \in S$,
- a finite set of actions A,
- a finite set of agents or security domains D,
- an action assignment $\mathrm{dom}\colon A \to D$,
- a transition function $\mathrm{step}\colon S \times A \to S$,
- a set of observations O,
- an observation function $\mathrm{obs}\colon D \times S \to O$.

If not otherwise specified, we will call these deterministic state-based systems just systems. We denote the elements of the set D mostly as agents, but also as security domains without any further semantic distinctions. Throughout this thesis, when we do not give a specific description of a system, we mean any arbitrary system and do not mention it explicitly.

2.2 Policies

A policy describes the allowed and forbidden information flow between agents. The precise semantics follows with the security definitions in the next chapters. In this section we will only give the syntax of information flow policies as they are used in this thesis.

A *policy* $\rightarrowtail$ is a reflexive binary relation on the set of agents D. For every $u, v \in D$, we write $u \rightarrowtail v$ if $(u, v) \in \rightarrowtail$ and $u \not\rightarrowtail v$ otherwise. Since $\rightarrowtail$ are the edges of a directed graph with vertex set D, we often adapt terminology from graph theory when we describe properties of a policy. For example, we denote $u \rightarrowtail v$ as an edge of the policy $\rightarrowtail$.

From Chapter 4 on, we will also consider families of policies $(\rightarrowtail_s)_{s \in S}$, where $\rightarrowtail_s$ is a policy in the previous sense. We name $(\rightarrowtail_s)_{s \in S}$ a *dynamic* policy, and if a clear distinction is needed, we denote a single policy $\rightarrowtail$, if it holds for a whole system, as a *static* or *global* policy. A single policy $\rightarrowtail_s$ of $(\rightarrowtail_s)_{s \in S}$ is denoted as a *local* policy (of the state s). If it is clear from the context to what kind of policy we refer, then we just call it a policy.

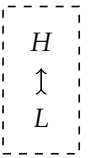

Figure 2.1. The HL policy

2.2.1 Example. The standard policy and probably the most common policy in the literature is the HL policy. This policy contains an agent H and and agent L. The agent L is allowed to interfere with H, but the policy prohibits any information flow from H to L. This is the simplest setting of a non-trivial noninterference policy. In our graphical notation, this policy is depicted in Figure 2.1. Since the agent H is not allowed to interfere with L, we call H the *high* agent and L the *low* agent.

2.3 Preliminaries and Notations

In this section, we introduce some notation we will use throughout this thesis. For any alphabet A, we denote with A^* the set of all finite sequences (or strings) of elements from A and with ϵ the empty sequence. We do not make a formal distinction between the elements of A and the words of length one in A^*. If A is the set of all actions of the system, we call the elements of A^* traces or (action) sequences. With ϵ, we denote the empty trace. For a trace α, $\alpha(i)$ denotes the ith entry of α, starting with 0. For any sequence of numbers $i_1, \ldots, i_k$ with $0 \leq i_j < |\alpha|$ and $i_j < i_{j+1}$, by $\alpha(i_1) \cdots \alpha(i_k)$ we denote a subsequence of α.

Concatenation of strings is not written explicitly: For strings α and β, the concatenation of α and β is written as $\alpha\beta$. For sets of strings B and C, we write BC for the set $\{\beta\gamma \mid \beta \in B \text{ and } \gamma \in C\}$. For singleton sets, we omit the curly brackets and write for example aB for $\{a\}B$. Due to associativity of concatenation, parentheses are usually not used.

For any subset $X \subseteq A$, we denote for any string α with $\alpha{\restriction_X}$ the restriction of α to elements from X, i. e., all symbols not in X are deleted in α. More precisely, the restriction operator can be defined inductively as

$$\epsilon{\restriction_X} = \epsilon$$
$$(a\alpha){\restriction_X} = \begin{cases} a(\alpha{\restriction_X}) & \text{if } a \in X \\ \alpha{\restriction_X} & \text{otherwise} \end{cases}.$$

For any set of agents $Y \subseteq D$, we abuse this notation and write $\alpha{\restriction_Y}$ for $\alpha{\restriction_{\{a \in A \mid \mathrm{dom}(a) \in Y\}}}$, i. e., restricting a trace to some set of agent means restricting the trace to the actions that can be performed by any of the agents from this set. For a singleton set of agents $\{u\}$, we may also write $\alpha{\restriction_u}$ instead of $\alpha{\restriction_{\{u\}}}$. For simplifying the notation, we write a dot for the transition function and its generalization to words. We define inductively $s \cdot \epsilon = s$ and $s \cdot \alpha a = \mathrm{step}(s \cdot \alpha, a)$ for any $s \in S$, $a \in A$, and $\alpha \in A^*$. The set of all symbols occurring in a trace α is denoted as $\mathrm{alph}(\alpha)$. For any agent $u \in D$, we denote with A_u the set of all actions a from A with $\mathrm{dom}(a) = u$. Since we usually apply the obs function to different states for the same agent, we put the name of the agent in subscript and write $\mathrm{obs}_u(s)$ rather then

$\mathrm{obs}(u, s)$.

For policies, we extend the interference relation to sets of agents and write for $U, V \subseteq D$: $U \rightarrowtail V$ iff there are $u \in U$ and $v \in V$ with $u \rightarrowtail v$. For singleton sets, we omit the curly brackets and mix the notations. For a given policy $\rightarrowtail$, with $u^{\rightarrowtail}$, we denote the set of all successors of u according to $\rightarrowtail$, i. e., $u^{\rightarrowtail} = \{v \in D \mid u \rightarrowtail v\}$ and with $u^{\leftarrowtail} = \{v \in D \mid v \rightarrowtail u\}$ the set of all predecessors of u. For a local policy $\rightarrowtail_s$, we write $u_s^{\leftarrowtail} = \{v \in D \mid v \rightarrowtail_s u\}$. Note that due to reflexivity of $\rightarrowtail$, we always have u in any of these sets $u^{\rightarrowtail}$, $u^{\leftarrowtail}$, and $u_s^{\leftarrowtail}$.

For any tuple t, $\pi_i(t)$ denotes the projection to the ith component of t and for tuples indexed by agents, $\pi_u(t)$ is the component that refers to agent u.

For any binary relation $\rightarrow$, we define $\xrightarrow{=}$ as the reflexive closure of $\rightarrow$ and $\xrightarrow{*}$ as the reflexive, transitive closure of $\rightarrow$.

2.4 Syntax and Semantics of System Diagrams

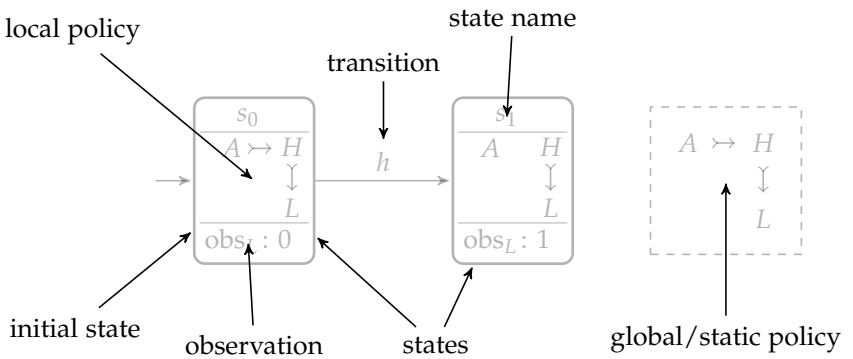

Figure 2.2. Explanation of a system diagram

Throughout this thesis, we will present several examples containing systems. To avoid a formal description of each example, we will provide syntax and semantics of our graphical representation of the systems. An

overview about the parts of the state diagrams is in Figure 2.2. The states of a system are represented by rectangles. Exactly one is well-marked as the initial state by an incoming edge from outside. The edges between the states denote the transitions of the system and are labeled with such an action that results in this transition. Transitions, which loop on states, are omitted from this representation. Since the system is input-enabled, i. e., for every state s and every action a, there is a state $s \cdot a$, we implicitly assume that $s \cdot a = s$ if there is no outgoing edge from the state representing s labeled with a. As a standard convention, we usually denote the agents with capital letters, e. g., $A, B, \ldots$ or H, D, L and the actions with lower letters $a, b, \ldots$ or h, d, l, and we always assume that any action is owned by the agent with the same letter, e. g., $\mathrm{dom}(a) = A$. If an agent has more than one action then we enumerate them by an index, e. g., h_1, h_2 are actions of agent H. If no action of an agent appears in the diagram, then we assume that the agent has no actions. This is often the case if an agent is only an observer.

In the representation of each state, additional information is provided, separated by horizontal lines. In the upper part, there is a unique identifier or name of the state, usually $s_0, s_1, \ldots$. If we do not refer to a particular state in an example, this part may be omitted. In the lower part, the observations of the agents at this state are depicted. For example $\mathrm{obs}_L : 0$ means that agent L observes 0 in this state. For any agent, its observations are either provided in all states or in no state. We assume implicitly that all agents not mentioned in this part have constant observation on all states and hence, the particular value of the observation function is not of interest. If there is only a single agent with non-constant observation, we just denote it as the observer or the observing agent.

Policies are depicted as a directed graph between the agents. Also in the representation of the policies, we omit self-loops since due to reflexivity, they would appear at each node. For systems with a static policy, the policy is usually depicted next to the system in a dashed box. In the case of a dynamic policy, the local policy that applies in the state appears as a third part within the state between the two previously mentioned parts. (However, we will never have both a dynamic policy in the states and a global policy next to the system as Figure 2.2 might suggest.)

2.5 Observations of System Runs

The observation of an agent of a run relies only on changes of the observations it obtains from the visited states. In general, an agent has only partial information about the current state, since in different states the observations may be the same. Formally, the observation that an agent makes from a system run α is the sequence of its observations in every state:

$$\mathrm{obs}_u(s^I) \mathbin{\hat{\circ}} \mathrm{obs}_u(s^I \cdot \alpha(0)) \mathbin{\hat{\circ}} \cdots \mathbin{\hat{\circ}} \mathrm{obs}_u(s^I \cdot \alpha(|\alpha| - 1)) \ .$$

Here $\hat{\circ}$ denotes the stutter-invariant concatenation. The general assumption is that the system is completely asynchronous and no agent has access to a global clock. If an agent makes the same observation twice without any other observation in-between, it does not know that these were two observations instead of a single observation. Sequences that only differ from each other in repetitions of a single observation are not distinguishable by an agent. Hence, in this sequence of observations, consecutive entries are different, since consecutive equal entries are canceled out. In fact, if the observation of an agent does not change, then it does not know whether any other agent has performed some action or not.

We generally assume that agents have perfect recall which means that they never forget anything they have seen. That is why we allow an agent to distinguish runs by the whole sequence of observations rather than by a part of it. However, we will see that for the security in deterministic systems it is sufficient to distinguish runs by the last observation of an agent.

Nonetheless, it is very restrictive that the sequence of observations is the only information that an agent has about a system run. It is reasonable that an agent is also aware of the actions that it has performed itself and also about the interleaving of these actions and the observations made during the run. For an agent u, the interleaving sequence of u's actions and its observation is denoted as u's view of a trace and is formally defined for every $a \in A$ and $\alpha \in A^*$ by

$$\mathrm{view}_u(\epsilon) = \mathrm{obs}_u(s^I)$$

$$\mathrm{view}_u(\alpha a) = \begin{cases} \mathrm{view}_u(\alpha)\ a\ \mathrm{obs}_u(s^I \cdot \alpha a) & \text{if } \mathrm{dom}(a) = u \\ \mathrm{view}_u(\alpha) \mathbin{\hat{\circ}} \mathrm{obs}_u(s^I \cdot \alpha a) & \text{otherwise .} \end{cases}$$

The view of a trace is the maximal information that an agent can get from a run of a system in an asynchronous system.

2.6 Trace-based Noninterference

Noninterference is about the absence of particular information flows during a run of a system. These runs are abstracted to sequences of actions, which we call traces. For the security in our framework, the importance of the actions is twofold. The performing of actions includes information that possibly has to be concealed from other agents, and actions are the parts of the system which transmit information by changing of the agents' observations.

In this section, we will introduce different techniques and formalisms for describing and analyzing trace-based information flow properties. Here, we will present the fundamental ideas of them on some abstract level and we will instantiate them in the next chapters with precise security definitions. We will do this at this point without reasoning about a precise definition of information flow, noninterference, or a semantics for the policy.

The security of systems with respect to a notion of noninterference strongly depends on the uncertainty that an agent has about a current run. If some action should be concealed from some agent, then this agent has to consider possible both the trace where the action appears and the one where the action does not appear.

The basic idea of describing trace-based information flow properties is that there are two equivalence relations on the set of all possible traces A^*.

One of these equivalence relations $\sim_u$ specifies which traces should be equivalent for the agent u, given an instantiation with a policy and a noninterference definition. This relation may also depend on the underlying system. The other equivalence relation $\approx_u$ specifies which traces effectively are indistinguishable for the agent u. The latter relation only depends on the underlying system and the ability of the agent to make observations

of the system behavior and is independent of any policy and any security definition.

Given these two equivalence relations for every agent u, the security of a system is defined as follows.

A system is secure if and only if for every $u \in D$ and every $\alpha, \beta \in A^*$, we have:

$$\text{if } \alpha \sim_u \beta \text{ then } \alpha \approx_u \beta \ .$$

This security definition captures the idea that if traces are not allowed to be distinguished by some agent, then it is not possible for this agent to distinguish these traces.

We assume that the equivalence relation $\approx_u$ is given by the analyzed or specified system. Throughout this work, we assume that the system is given and fixed. We will not analyze how to modify or synthesize systems for a given security specification. Modification of systems is considered in [YLBHA09], synthetization is considered in [CMR07].

On the other hand, the relation $\sim_u$ is induced by a policy and a security definition for a particular notion of noninterference. In this work, different notions of noninterference are developed and analyzed. Some of these notions are independent of the underlying system, while others depend on the state space or on the observations of agents.

Note that insecurity of a system can only be shown by two traces α, β with $\alpha \sim_u \beta$, but $\alpha \not\approx_u \beta$ and not by a single trace. Already McLean [McL94] mentioned that noninterference is not a trace-property. However, it is a trace-property of a product system of two modified copies of the original systems what is called a 2-trace property. The construction of these two systems follows the construction pattern in Section 5.6. More generally, noninterference is a property of a set of traces which is generalized to so-called hyperproperties [CS10].

2.6.1 Observational Equivalence

In the state-based system model used throughout this thesis, every agent makes observations during a run of the system. From these observations,

an agent can possibly distinguish different runs from each other and hence can gain information about things that have or have not happened during the run.

The two, possibly most intuitive ways of defining an observational equivalence are the equality of the observation in the reached state and the equivalence of the view of an agent as defined in Section 2.5. More precisely, the observational equivalence $\approx_u \subseteq A^* \times A^*$ is either defined as

$$\alpha \approx_u \beta \text{ iff } \mathrm{obs}_u(s^I \cdot \alpha) = \mathrm{obs}_u(s^I \cdot \beta) \ ,$$

or as

$$\alpha \approx_u \beta \text{ iff } \mathrm{view}_u(\alpha) = \mathrm{view}_u(\beta) \ .$$

We will focus on the former approach, due to its simplicity. In the next paragraph, we will see that for deterministic systems as used here, for all security properties considered in this thesis, these two ways of defining observational equivalence result in equivalent definitions of security. Note that this does not hold if one moves to nondeterministic systems, as pointed out in [EvdMZ12].

2.6.2 Inductively Defined Trace-Operators

The classic way of defining trace-based noninterference properties is by operators on traces. Such an operator $f\colon A^* \times D \to X$ is usually defined for any trace and any agent. For a particular agent, it transforms a trace into some object that encodes everything that the agent is allowed to know about the trace. In other words, the operator removes all the information from the trace that should be concealed from the agent. Any two traces should be indistinguishable for an agent if they contain the same allowed information for the agent. Hence, the indistinguishability relation on the trace is defined by

$$\alpha \sim_u \beta \text{ iff } f(\alpha, u) = f(\beta, u) \ .$$

In this and the following chapters, we will present several instantiations

for the operator f, depending on the security requirements and the security definition. All the security operators we use in this thesis are defined inductively.

We will show that with inductively defined operators and some side-conditions, security definitions based on observational equivalence, which are defined on just the observation function, are equivalent to those that are defined on the view function.

We express the function $f\colon A^* \times D \to X$ inductively, combined with a function $g\colon f(A^*, D) \times A^* \times A \times D \to Y$, with the following dependencies between them:

$$\begin{aligned} f(\epsilon, u) &= c \\ f(\alpha a, u) &= g(f(\alpha, u), \alpha, a, u) \ , \end{aligned}$$

where X and Y are some appropriate sets and c denotes some fixed value in X.

To show the equivalence of the observational equivalence based on obs and view, we need to restrict the structure of the functions f and g by the following two *influence conditions*:

1. For every $u \in D$ and $a \in A$ with $\mathrm{dom}(a) = u$ and every $\alpha \in A^*$, we have $f(\alpha a, u) \neq f(\alpha, u)$.
2. For every $u \in D$, $a, b \in A$, $\alpha, \beta \in A^*$ and $x, x' \in f(A^*, D)$ with $g(x, \alpha, a, u) = g(x', \beta, b, u)$, we have either $x = g(x', \beta, b, u)$, or $x = x'$ and $a = b$.

The first condition says that the own actions of an agent are encoded into the value of f. The function g is the previously collected information plus some new information from the last action. The second condition says that if two values of g are the same, after some action then either no information was added in the last step or in both cases the same information was added.

The next lemma summarizes the claimed equivalence.

2.6.1 Lemma. *Suppose that the two influence conditions above hold. Then for every $u \in D$ and $\alpha, \beta \in A^*$ the following two implications are equivalent:*

1. *If $f(\alpha, u) = f(\beta, u)$, then* $\mathrm{view}_u(\alpha) = \mathrm{view}_u(\beta)$.

2. *If* $f(\alpha, u) = f(\beta, u)$, *then* $\mathrm{obs}_u(s^I \cdot \alpha) = \mathrm{obs}_u(s^I \cdot \beta)$.

Proof. The implication from 1. to 2. follows directly from the definition of the view function.

For the other direction, let $u \in D$ and $\alpha, \beta \in A^*$ of minimal combined length with $f(\alpha, u) = f(\beta, u)$ and $\mathrm{view}_u(\alpha) \neq \mathrm{view}_u(\beta)$. Since at least one of α and β is not the empty trace, suppose that it is α and let $\alpha = \alpha' a$ for some $\alpha' \in A^*$ and $a \in A$.
Case 1: $f(\alpha' a, u) = f(\alpha', u)$.
From the minimality of α and β it follows $\mathrm{view}_u(\alpha') = \mathrm{view}_u(\beta)$ and hence $\mathrm{view}_u(\alpha' a) \neq \mathrm{view}_u(\alpha')$. From the first influence condition it follows $u \neq \mathrm{dom}(a)$ and hence $\mathrm{view}_u(\alpha' a) = \mathrm{view}_u(\alpha')\, \mathrm{obs}_u(s^I \cdot \alpha' a)$. Therefore, we have $\mathrm{obs}_u(s^I \cdot \alpha' a) \neq \mathrm{obs}_u(s^I \cdot \beta)$.
Case 2: $f(\alpha' a, u) \neq f(\alpha', u)$.
We can assume that $\beta = \beta' b$ and that $f(\beta' b, u) \neq f(\beta', u)$ since otherwise, we proceed with Case 1 with the roles of α and β swapped. From the second influence condition it follows $a = b$ and $f(\alpha', u) = f(\beta', u)$. By the minimality of α and β, we have $\mathrm{view}_u(\alpha') = \mathrm{view}_u(\beta')$. With $d = \epsilon$ if $\mathrm{dom}(a) \neq u$ and $d = a$ if $\mathrm{dom}(a) = u$, we have

$$\begin{aligned} \mathrm{view}_u(\alpha')\ d\ \mathrm{obs}_u(s^I \cdot \alpha' a) &= \mathrm{view}_u(\alpha) \\ &\neq \mathrm{view}_u(\beta) \\ &= \mathrm{view}_u(\beta')\ d\ \mathrm{obs}_u(s^I \cdot \beta' b)\ . \end{aligned}$$

It follows $\mathrm{obs}_u(s^I \cdot \alpha' a) \neq \mathrm{obs}_u(s^I \cdot \beta' b)$. □

One can easily see that all noninterference operators presented in this thesis can be expressed as an instantiation of the functions f and g and that they satisfy the influence-conditions above.

2.6.3 Unwinding

An *unwinding* is a famous way for defining or characterizing information flow properties. An unwinding is a combination of a family of unwinding relations and unwinding conditions. Unwinding relations are usually

equivalence relation on the elements of the analyzed structure. In this thesis, unwinding relations are relations either on the set of states or the set of traces of the system. In the first case, we call them *state-based unwinding* and in the latter case *trace-based unwinding*.

Unwinding relations are defined or characterized by a number of conditions, called *unwinding conditions*. Every equivalence relation that satisfies these conditions is then called an unwinding relation. An unwinding defines for every agent at least one unwinding relation. Usually, one considers the smallest relation that satisfies the unwinding conditions.

In a trace-based unwinding, an unwinding relation $\sim_u$ for each agent u is a relation on the set of traces. We say that such a relation is *observation consistent for u* if for every every $\alpha, \beta \in A^*$ the implication

$$\text{if } \alpha \sim_u \beta, \text{then } \text{obs}_u(s^I \cdot \alpha) = \text{obs}_u(s^I \cdot \beta)$$

holds. The equality of the observation is just the instantiation of the $\approx_u$ relation with the observational equivalence as described before.

In a state-based unwinding, an unwinding relation $\sim$ is a relation on the set of states of the system, parameterized with one or more agents. We call exactly one of these agents the observing agent u. We say that such a relation is *observation consistent for u* if for every $s, t \in S$ with $s \sim t$ it holds: $\text{obs}_u(s) = \text{obs}_u(t)$.

As we will see in this thesis, unwinding relations are an elegant way of characterizing noninterference properties. More importantly, on finite-state systems, state-based unwinding relations can directly be translated into a verification algorithm as we can see in Chapter 5.

2.6.4 Information Sets

Information sets are used in game theory to describe the set of all possible moves that could have taken place in a game so far, given what a particular player has observed. Here, we adopt this concept and understand an information set as a set of all traces that an agent considers possible from its maximal allowed knowledge. In other words, an information set of an agent is a set of all traces that an agent is not allow to distinguish. It will

turn out that information sets are exactly the equivalence classes of the corresponding operator-based or unwinding-based definition. However, in some cases, this set-based definition leads to a more elegant and possibly more intuitive way of describing the equivalence between traces than the previous approaches.

2.6.5 Securely Constructed Systems

The covert channels analyzed in this thesis stem from a global state space, i. e., a state space shared by all agents. Hence if every agent would have its own state space, all possible covert channels would be eliminated. However, it is necessary to make the desired interaction between the agents possible, as long as they obey the given policy. That is exactly the idea of a securely constructed system. Every agent has its own local state space, but any transition may also depend on the state space of other agents according to a given policy and a given security definition. This can be seen as the transition function is constructed from a given policy and hence, by construction, such a system is then secure. One benefit of this approach is that it gives the designer of a system a construction pattern, detailing how to implement a system that obeys a given security policy. Moreover, as we will see later for particular security definitions, this system structure is complete in the sense that any secure system is observationally equivalent to a securely constructed system.

We provide at this point the general structure of a securely constructed system. Only the local and the global transition functions depend on the security definition which should be satisfied by the construction of the system.

The common structure is that every agent u has its own local state space S_u and that the global state space is the product of the local state spaces. For each agent u the local transition function $\cdot_u$ describes the transition on u's local state space. This function may also depend on some other agent v's state space, depending on whether the security definition allows to transmit information about v to u. The global transition function is then defined by the local transition functions and defines whether a particular local transition function is triggered or not. A more precise

definition depends on a given policy from which the system is constructed and on the security definition which should be satisfied by the construction. To prevent any covert channels, the observation function obs_u then only depends on u's local state space. Note that the security of a securely constructed system does not depend on the choice of the observation function, except the property that the observation function for each agent only depends on the agents' local states.

As long as the transition function only transmits information which is allowed by a given policy, each local state space can only contain allowed information. Since the transition of each agent only depends on its local states and hence only on allowed information, this construction rules out any forbidden information flows and therefore, the constructed system is a secure system.

2.6.6 Knowledge-based Characterization

Secrecy and hence noninterference have a strong connection to epistemic logic—the logic of knowledge. When one says that something is kept secret from someone, then this means that this person does not know the secret. More generally, we consider a system as secure in the sense that it preserves secrecy. If an agent is not allowed to know some particular property about some system, then the agent, indeed, does not know this property.

We can express this general description of security in the formal framework of epistemic logic. We will only provide the most necessary definitions. A more elaborate introduction to the topic of epistemic logic is [FHMV95]. Epistemic logic is used to reason about the knowledge of agents. In our case, it extends propositional logic by a modal operator K_u for every agent u. Semantically, $K_u\varphi$ should usually be read as "agent u knows φ".

Since we deal with two different kinds of knowledge, the knowledge coming from the observations and the allowed knowledge, we introduce two different modal operators $K_u^{\sim}$ and $K_u^{\approx}$ for every agent u.

Let Φ be a set of atomic propositions. Then every $p \in \Phi$ is a formula and, inductively, if φ and ψ are formulas then $\neg\varphi$, $\varphi \vee \psi$ and for every $u \in D$: $K_u^{\sim}\varphi$ and $K_u^{\approx}\varphi$ are formulas.

The semantics follows those defined in [vdM07]. The atomic propo-

sitions express properties of traces given by an interpretation function $\pi\colon \Phi \to \mathcal{P}(A^*)$. An atomic proposition p holds in a system M after a trace α if and only if $\alpha \in \pi(p)$ and we write $M, \pi, \alpha \models p$. The semantics of formulas of the form $\neg\varphi$ and $\varphi \vee \psi$ is defined in a straightforward way—the semantics for formulas defined by the epistemic operators is defined for every agent u by

$$M, \pi, \alpha \models K_u^{\sim}\varphi \text{ iff } M, \pi, \beta \models \varphi \text{ for all traces } \beta \text{ with } \alpha \sim_u \beta \ ,$$
$$M, \pi, \alpha \models K_u^{\approx}\varphi \text{ iff } M, \pi, \beta \models \varphi \text{ for all traces } \beta \text{ with } \alpha \approx_u \beta \ .$$

With these two modal operators, the security of a system can be summarized with a family of short formulas. A system is secure if, and only if for every agent u and every formula φ, the following formula holds:

$$K_u^{\approx}\varphi \to K_u^{\sim}\varphi \ .$$

Intuitively, this formula says that if agent u actually knows φ then u is allowed to know φ, where "actually knows φ" means that u can conclude whether φ holds from its observations under the assumption that u has perfect recall, which is generally assumed throughout this thesis.

In this thesis, we will not use this epistemic approach further, but at some point we will reason about the knowledge of agents on a more intuitive level. However, this formalism underpins the correctness of such reasoning by providing precise semantics.

2.7 Information Flow

In this section, we will explain what information flow is in a system.

With information flow, we denote the event that performing some action by some agent v is observable by some agent u. In this case, we say that there is an information flow from v to u.

We distinguish between direct and indirect information flows. By a direct information flow, we understand an information flow that is directly observable by some observer from performing a single action. For example, in the system in Figure 2.3 there is a direct information flow from the

agent H, which performs the action h, to the agent L, since L observes immediately that H performed an action by having an immediate change of its observation.

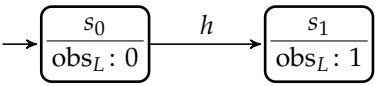

Figure 2.3. A direct information flow

An indirect information flow is an information flow that is not directly observable. The performed action will be revealed after a sequence of actions, but the observations lead to deductions about the not immediately observable event.

The system in Figure 2.4 has an action h that is not directly observable, but after a following action l the information that the action has been performed is revealed. Note that in this system there is a direct information flow from L to itself, since the action l performed in state s_1 changes L's observation. However, if the same action is performed in the state s_0 it does not change L's observation. If, in two runs of a system, L performs the same action l, then its observation depends on H's behavior. That is the reason why we say that H changes L's observations, or that there is an information flow from H to L.

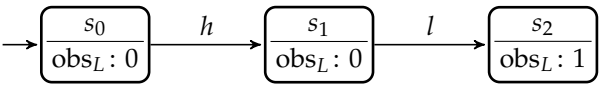

Figure 2.4. An indirect information flow

A policy and a security definition define which actions in a trace may be observable by some particular agent u. This gives then the maximal information about the performed actions in the trace that the agent u is allowed to observe. If this maximal information is the same in two traces, then these traces are required to be indistinguishable in the sense of the relation $\sim_u$ of the previous section.

2.8 Noninterference

Noninterference denotes the absence of an information flow. We say that an agent u is noninterfering with some other agent v if there is no information flow from u to v. It depends on a particular definition of noninterference if we refer to indirect or only to direct information flows. Often, with noninterference, we mean the requirement of an absence of an information flow from some agent to some other. This requirement is expressed by a policy, and a security definition provides the semantics and essentially defines the equivalence relation that states which traces are not allowed to be distinguished.

Chapter 3

Static Noninterference

In this chapter, we develop characterizations and methods for analyzing noninterference properties of systems with a static policy. A static policy is a policy which does not change during the run of a system and is required to hold for all runs. We recall the classical notion of noninterference introduced by Goguen and Meseguer [GM82] and its extensions of Haigh and Young [HY87], Rushby [Rus92], and van der Meyden [vdM07]. We apply new techniques to the previous definitions which give new insights in this theory, and our new characterizations make it possible to develop new efficient algorithms. This chapter does not only provide the theoretical background for our algorithms, it also leads to a deeper understanding of the security definition that is needed for further extensions and generalizations to a more complex setting in the next chapter.

After an introduction to transitive noninterference in Section 3.1, we enrich Goguen and Meseguer's definition of t-security by new characterizations in Section 3.2. In Section 3.3, we present the notion of intransitive noninterference. New characterizations for Haigh and Young's i-security are presented in Section 3.4. Van der Meyden's ta-security is analyzed in Section 3.5, and Section 3.6 provides a closer look at the notions of to-security and ito-security. In the remaining Section 3.7, we summarize the relation between all security definitions of this chapter.

Several results in this chapter were published in [EvdMSW11] or are submitted for publication in [EvdMSW13].

3.1 Transitive Noninterference

The definition of transitive noninterference, in the literature mostly just called noninterference, was introduced by Goguen and Meseguer [GM82, GM84]. Their works and definitions are the basis of most publications on the topic of noninterference. While our presentation and definitions are very close to theirs, we will see in Chapter 7 that a lot of work has been done, which is still only inspired by the work of Goguen and Meseguer, but also has major semantical differences.

A general assumption in most of the early work on this topic was that a security policy had to be a transitive relation. The argument for a transitive policy was that if an agent u is allowed to interfere with some agent v, and v is allowed to interfere with some agent w, then consequently, there is a possible interference from u to w and hence, this interference can not be restricted or prohibited. As we will see in the section about intransitive noninterference, this argument is too simple and a different interpretation of policies is more than reasonable. However, since noninterference policies were defined as transitive, a new name for policies that did not have this requirement was needed. Hence these policies without the transitivity requirement were called intransitive policies. However, the term *transitive* relates to the allowed interference relation, rather than to noninterference. However, we follow the denotation in the literature and call it transitive noninterference.

To have a clear distinction between these two interpretations we call Goguen and Meseguer's approach transitive noninterference. However, we do not require that the policy has to be transitive. Hence, syntactically, there is no distinction between an intransitive and a transitive policy, and we just call it a policy. However, the semantics differs: In transitive noninterference, an agent is only allowed to interfere with an agent u if there is an edge from this agent to u. This means that u is not allowed to observe or deduce anything regarding the actions of any agent that does not have an edge in the policy to u.

However, for every single agent u, the policy restricted to u's incoming edges is a transitive relation. At least from the point of view of each agent, it is reasonable to call the policy transitive, and therefore, we call this notion

of noninterference *transitive noninterference*.

From now on, we always consider a system together with a policy. The policy states from which agent to which other agent information may flow and where an information flow is forbidden. The interpretation of a policy is quite simple. If there is an edge from some agent u to v in the policy, then v is allowed to observe everything that u does. If there is no edge from u to v, the agent v is not allowed to observe or deduce anything u does or has done, including, whether u has done any action at all in a run of the system.

This idea is formalized as follows: Given a run $\alpha \in A^*$, if all actions v is not allowed to get any information about are removed from the trace α then this trace has to lead to the same observation for v as α. Since otherwise, v would notice that there are some actions missing and hence would get information about actions from agents from which it is not allowed to get information.

3.2 t-security

For formalizing transitive noninterference, Goguen and Meseguer [GM82] introduced a function which parameterized with an agent u, removes all actions of agents that not allowed to interfere with u from a trace. In the literature, this function is often called the *purge* function. To indicate that it relates to transitive noninterference, we call this function tpurge.

3.2.1 Definition (tpurge operator)**.** For every $u \in D$, $a \in A$, and $\alpha \in A^*$ define

$$\begin{aligned} \mathrm{tpurge}_u(\epsilon) &= \epsilon, \\ \mathrm{tpurge}_u(a\alpha) &= \begin{cases} a\ \mathrm{tpurge}_u(\alpha) & \text{if } \mathrm{dom}(a) \rightarrowtail u \\ \mathrm{tpurge}_u(\alpha) & \text{otherwise .} \end{cases} \end{aligned}$$

Instead of using an inductive definition, tpurge can be equivalently defined with the restriction operator on traces:

$$\mathrm{tpurge}_u(\alpha) = \alpha\restriction_{\{v \in D \mid v \rightarrowtail u\}} .$$

Intuitively, the tpurge function removes all information from the trace that u is not allowed to observe. Therefore, if two traces have the same allowed information—in our definition the same tpurge-value—then these traces should be indistinguishable for the observing agent. We denote this property, if it holds for all agents and all traces, as t-security: The security definition which we use for transitive noninterference.

3.2.2 Definition (t-security)**.** A system is t-secure iff for every $u \in D$, every $\alpha, \beta \in A^*$ with $\text{tpurge}_u(\alpha) = \text{tpurge}_u(\beta)$, we have $\text{obs}_u(s^I \cdot \alpha) = \text{obs}_u(s^I \cdot \beta)$.

Equivalently, this definition can be characterized by: A system is t-secure if for every $u \in D$ and every $\alpha \in A^*$, we have $\text{obs}_u(s^I \cdot \alpha) = \text{obs}_u(s^I \cdot \text{tpurge}_u(\alpha))$.

This means that the observations of an agent are only allowed to depend on the actions of agents that are allowed to interfere with this agent.

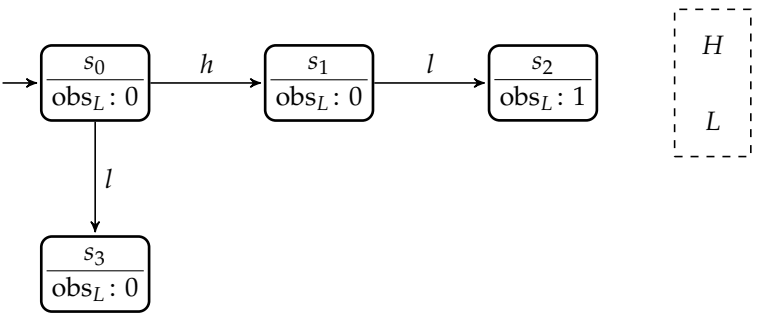

Figure 3.1. A *not* t-secure system

3.2.3 Example. To illustrate the idea of t-security, we consider the system in Figure 3.1. In this system, we have two agents H and L that are not allowed to interfere with each other, which is stated by the policy on the right-hand side next to the system. That means, what L observes should not depend on what H has done. This is clearly not the case in this system, since L only observes a 1 if H has performed the action h in the initial state. Formally,

this is expressed by

$$\text{tpurge}_L(hl) = l = \text{tpurge}_L(h) \ ,$$

but

$$\text{obs}_L(s^I \cdot hl) = 1 \neq 0 = \text{obs}_L(s^I \cdot l) \ .$$

It is not necessary to consider all possible traces α and β. If a system is not t-secure, then this is due to a single action a of an agent $\text{dom}(a)$ which is not allowed to interfere with u. Hence it is sufficient to compare the observations after a trace $\alpha\beta$ with those after $\alpha a\beta$. This is captured by the next lemma.

3.2.4 Lemma. *A system is t-secure iff for every $u \in D$, $a \in A$, $s \in S$ and $\beta \in A^*$ with $\text{dom}(a) \not\rightarrowtail u$, we have*

$$\text{obs}_u(s \cdot a\beta) = \text{obs}_u(s \cdot \beta) \ .$$

Proof. We will prove the contrapositive of this claim. Suppose there are $u \in D$, $a \in A$, $s \in S$, and $\alpha \in A^*$ with $\text{dom}(a) \not\rightarrowtail u$ and $\text{obs}_u(s \cdot a\alpha) \neq \text{obs}_u(s \cdot \alpha)$. By the general assumption that every state of the system is reachable from the initial state, there is some $\alpha \in A^*$ with $s = s^I \cdot \alpha$. Then we have $\text{tpurge}_u(\alpha a\beta) = \text{tpurge}_u(\alpha\beta)$. Hence the system is *not* t-secure.

For the other direction of the proof assume that the system is *not* t-secure. Then there are $\alpha, \beta \in A^*$ with $\text{tpurge}_u(\alpha) = \text{tpurge}_u(\beta)$ and $\text{obs}_u(s^I \cdot \alpha) \neq \text{obs}_u(s^I \cdot \beta)$ for some agent u. Assume that α and β are of minimal length for all possible choices satisfying this property. Let α' and β' be the longest prefix of α and β, respectively, such that $\alpha' = \beta'$. W.l.o.g. $\alpha' \neq \alpha$ and therefore there is some $a \in A$ and $\alpha'' \in A^*$ with $\alpha = \alpha' a \alpha''$. If $\text{dom}(a) \not\rightarrowtail u$, we have $\text{obs}_u(s^I \cdot \alpha' a \alpha'') \neq \text{obs}_u(s^I \cdot \alpha' \alpha'')$, due to the minimality of the length of α. Hence the claim holds for $s = s^I \cdot \alpha'$ and $\beta = \alpha''$. In the case of $\text{dom}(a) \rightarrowtail u$, we can apply the mentioned argumentation to β instead of α, since the suffix of β after β' has to start with an action of an agent that is not allowed to interfere with u. □

3.2.1 Unwinding for t-security

One way to characterize noninterference is by an unwinding. An unwinding relation is an equivalence relation that defines an indistinguishability relation on some set. In our case this will be either the set of states or the set of traces. Unwinding relations will be defined by rules called unwinding conditions, which have to be satisfied by the unwinding relation. Usually, one is interested in the smallest equivalence relation that satisfies the given conditions. Intuitively, the unwinding conditions are local conditions which define which state or trace are indistinguishable after performing a single action. Due to symmetry and transitivity of the unwinding relation, this relation then captures all states or traces that are required to be indistinguishable. These unwinding conditions strongly depend on the notion of security that is aimed to be expressed with the unwinding relations. However, the unwinding conditions and relations characterize which states or traces *should* be indistinguishable for some agent. When defining security, it is also required that they are in fact indistinguishable by the agent's observation, what we call observation consistency for the respective agent.

The first characterizations in the context of unwinding relations are trace-based unwinding relations. A trace-based unwinding relation is an unwinding relation on the set of traces and defines an indistinguishability relation that is required to hold for the corresponding security definition.

3.2.5 Definition (trace-based transitive unwinding)**.** A *trace-based transitive unwinding relation* for $u \in D$ is an equivalence relation $\sim_u^{\mathsf{tt}} \subseteq A^* \times A^*$ that for every $a \in A$ and $\alpha, \beta \in A^*$ satisfies the following two conditions:

(LR$^{\mathsf{tt}}$)*:* If $\mathrm{dom}(a) \not\rightsquigarrow u$, then $\alpha \sim_u^{\mathsf{tt}} \alpha a$.

(SC$^{\mathsf{tt}}$)*:* If $\alpha \sim_u^{\mathsf{tt}} \beta$, then $\alpha a \sim_u^{\mathsf{tt}} \beta a$.

The unwinding condition (LR$^{\mathsf{tt}}$) is denoted as *local respect*. It states that if an agent $\mathrm{dom}(a)$ performs some action a and the policy requires that $\mathrm{dom}(a)$ is not allowed to interfere with u, then after some arbitrary trace α, this trace and the trace αa have to be indistinguishable for u. The second condition (SC$^{\mathsf{tt}}$) is denoted as *step consistency*. It says that if two traces are indistinguishable for u, then after some agent $\mathrm{dom}(a)$ has performed the

same action after each of these traces, the resulting traces again have to be indistinguishable for u. Note that this is required regardless of whether $\mathrm{dom}(a)$ is allowed to interfere with u or not. However, if $\mathrm{dom}(a)$ is not allowed to interfere with u, then this condition already follows from ($\mathrm{LR}^{\mathsf{tt}}$) and the symmetry and transitivity of an equivalence relation.

Usually one considers the smallest unwinding relation which satisfies the two conditions. This smallest relation can be obtained by applying repeatedly the two conditions, starting with the identity relation. Clearly, a smallest unwinding relation exists and is unique.

The claimed characterization of the previous security definition follows from the next result.

3.2.6 Theorem. *A system is t-secure iff for every $u \in D$ there is a trace-based transitive unwinding relation $\sim_u^{\mathsf{tt}}$ that is observation consistent for u.*

Proof. Let $u \in D$. We will show that for every agent u, for every $\alpha, \beta \in A^*$, and for the smallest trace-based transitive unwinding relation $\sim_u^{\mathsf{tt}}$ it holds:

$$\mathrm{tpurge}_u(\alpha) = \mathrm{tpurge}_u(\beta) \text{ iff } \alpha \sim_u^{\mathsf{tt}} \beta \ .$$

We prove the implication from left to right by an induction on the combined length of α and β. Since the base case is clear, we proceed with the inductive step and assume that $\mathrm{tpurge}_u(\alpha) = \mathrm{tpurge}_u(\beta)$ and that the implication holds for all pairs of traces with smaller combined length. W.l.o.g., we can assume that $\alpha \neq \epsilon$. Hence there is some $a \in A$ and $\alpha' \in A^*$ with $\alpha = \alpha' a$.
Case 1: $\mathrm{dom}(a) \not\rightarrowtail u$.
In this case, we have $\mathrm{tpurge}_u(\alpha' a) = \mathrm{tpurge}_u(\alpha') = \mathrm{tpurge}_u(\beta)$ and by induction hypothesis, $\alpha' \sim_u^{\mathsf{tt}} \beta$. By the condition ($\mathrm{LR}^{\mathsf{tt}}$), we obtain $\alpha' \sim_u^{\mathsf{tt}} \alpha' a$ and hence, we have $\alpha \sim_u^{\mathsf{tt}} \beta$.
Case 2: $\mathrm{dom}(a) \rightarrowtail u$.
In this case, we have that $\beta \neq \epsilon$ and hence, there is some $b \in A$ and $\beta' \in A^*$ with $\beta = \beta' b$. If $\mathrm{dom}(b) \not\rightarrowtail u$, then we can apply Case 1 with the roles of α and β swapped. Hence, suppose $\mathrm{dom}(b) \rightarrowtail u$. This gives $a = b$ and $\mathrm{tpurge}_u(\alpha') = \mathrm{tpurge}_u(\beta')$. By applying the induction hypothesis, we get $\alpha' \sim_u^{\mathsf{tt}} \beta'$ and by applying the ($\mathrm{SC}^{\mathsf{tt}}$) condition, the result is $\alpha \sim_u^{\mathsf{tt}} \beta$.

We prove the other implication of this result by an induction on the

applications of the unwinding conditions ($\mathrm{LR}^{\mathsf{tt}}$) and ($\mathrm{SC}^{\mathsf{tt}}$). This is possible, since if an unwinding relation exists that satisfies ($\mathrm{LR}^{\mathsf{tt}}$), ($\mathrm{SC}^{\mathsf{tt}}$), and observation consistency, then also the smallest unwinding relation that satisfies ($\mathrm{LR}^{\mathsf{tt}}$) and ($\mathrm{SC}^{\mathsf{tt}}$) is observation consistent. We can assume that the unwinding condition is iteratively constructed by applying the ($\mathrm{LR}^{\mathsf{tt}}$) and ($\mathrm{SC}^{\mathsf{tt}}$) conditions.

Let $u \in D$ and let $\alpha, \beta \in A^*$ with $\alpha \sim_u^{\mathsf{tt}} \beta$. In the first case, suppose that $\alpha \sim_u^{\mathsf{tt}} \beta$ holds from the application of the ($\mathrm{LR}^{\mathsf{tt}}$) condition. Hence, due to symmetry, we can assume $\beta = \alpha a$ for some $a \in A$ with $\mathrm{dom}(a) \not\rightarrowtail u$. By the definition of tpurge, we get $\mathrm{tpurge}_u(\beta) = \mathrm{tpurge}_u(\alpha a) = \mathrm{tpurge}_u(\alpha)$. In the second case, assume that $\alpha \sim_u^{\mathsf{tt}} \beta$ holds from the application of the ($\mathrm{SC}^{\mathsf{tt}}$) condition. Hence, $\alpha = \alpha' a$ and $\beta = \beta' a$ for some $a \in A$ with $\alpha' \sim_u^{\mathsf{tt}} \beta'$. By induction hypothesis, we have $\mathrm{tpurge}_u(\alpha') = \mathrm{tpurge}_u(\beta')$ and hence $\mathrm{tpurge}_u(\alpha) = \mathrm{tpurge}_u(\beta)$. □

Another way of characterizing the indistinguishability requirement on a noninterference definition are information sets. Intuitively, an information set describes the allowed knowledge about the actual trace of an agent after this trace. If the actual run of a system in described by a trace α, then the agent u is only allowed to know that the actual trace was one of the traces in $I_u^{\mathsf{t}}(\alpha)$, but u is neither allowed to know which element of this set it actually was, nor to rule out any element of this set.

3.2.7 Definition (transitive information sets)**.** For any agent $u \in D$, any $a \in A$, and any $\alpha \in A^*$ the *transitive information sets* are inductively defined by

$$I_u^{\mathsf{t}}(\epsilon) = \{b \in A \mid \mathrm{dom}(b) \not\rightarrowtail u\}^*$$
$$I_u^{\mathsf{t}}(\alpha a) = \begin{cases} I_u^{\mathsf{t}}(\alpha)\ a\ I_u^{\mathsf{t}}(\epsilon) & \text{if } \mathrm{dom}(a) \rightarrowtail u \\ I_u^{\mathsf{t}}(\alpha) & \text{otherwise} \end{cases}.$$

Information sets are strongly related to the trace-based unwinding relations defined above. In fact, information sets are exactly the equivalence classes of the smallest trace-based transitive unwinding relation.

3.2.8 Lemma. *Let $u \in D$ and $\alpha \in A^*$. For the smallest trace-based transitive*

unwinding relation $\sim_u^{\mathsf{tt}}$*, we have* $I_u^{\mathsf{t}}(\alpha) = [\alpha]_{\sim_u^{\mathsf{tt}}}$.

Proof. We will show that we have for every $\alpha, \beta \in A^*$:

$$\alpha \in I_u^{\mathsf{t}}(\beta) \text{ iff } \alpha \sim_u^{\mathsf{tt}} \beta \ .$$

We prove the direction from left to right of this equivalence by an induction on the combined length of α and β. Assume that $\alpha \in I_u^{\mathsf{t}}(\beta)$. For the base case $\alpha = \beta = \epsilon$, the claim holds trivially. For the inductive step, let $\alpha = \alpha' a$ with $\alpha' \in A^*$ and $a \in A$.
Case 1: $\mathrm{dom}(a) \not\rightarrowtail u$.
Then we have $I_u^{\mathsf{t}}(\alpha' a) = I_u^{\mathsf{t}}(\alpha')$ and by applying the induction hypothesis $\alpha' \sim_u^{\mathsf{tt}} \beta$. From the ($\mathrm{LR}^{\mathsf{tt}}$) condition it follows $\alpha' \sim_u^{\mathsf{tt}} \alpha' a$ and therefore, by transitivity and symmetry of an equivalence relation, we have $\alpha \sim_u \beta$.
Case 2: $\mathrm{dom}(a) \rightarrowtail u$.
We assume that $\beta = \beta' b$ with $\beta' \in A^*$ and $b \in A$, since otherwise we may proceed with Case 1 with the roles of α and β swapped. Also assume that $\mathrm{dom}(b) \not\rightarrowtail u$. Then we have $\alpha \in I_u^{\mathsf{t}}(\beta' b) = I_u^{\mathsf{t}}(\beta')$. By applying the induction hypothesis, we obtain $\alpha \sim_u \beta$ with the same argument as before.

As the remaining case, we assume that also $\mathrm{dom}(b) \rightarrowtail u$. Then we have

$$\alpha' a \in I_u^{\mathsf{t}}(\beta' b) = I_u^{\mathsf{t}}(\beta')\ b\ I_u^{\mathsf{t}}(\epsilon) \ .$$

Since a is not in $I_u^{\mathsf{t}}(\epsilon)$, it follows $a = b$ and $\alpha' \in I_u^{\mathsf{t}}(\beta')$. By applying the induction hypothesis, we have $\alpha' \sim_u^{\mathsf{tt}} \beta'$ and from the ($\mathrm{LR}^{\mathsf{tt}}$) condition it follows $\alpha \sim_u^{\mathsf{tt}} \beta$.

We will prove the other direction by an induction on the inductive definition of the $\sim_u^{\mathsf{tt}}$ relation.
Case 1: Suppose $\alpha \sim_u^{\mathsf{tt}} \beta$ holds from an application of the ($\mathrm{LR}^{\mathsf{tt}}$) condition. Hence we have $\alpha = \beta a$ for some $a \in A$ with $\mathrm{dom}(a) \not\rightarrowtail u$. Then we have $\alpha \in I_u^{\mathsf{t}}(\beta a) = I_u^{\mathsf{t}}(\beta)$ and $\beta \in I_u^{\mathsf{t}}(\beta a)$.
Case 2: Suppose $\alpha \sim_u^{\mathsf{tt}} \beta$ holds from an application of the ($\mathrm{SC}^{\mathsf{tt}}$) condition. Then we have $\alpha = \alpha' a$ and $\beta = \beta' a$ for some $a \in A^*$. Therefore, we have $\alpha' \in I_u^{\mathsf{t}}(\beta')$ and hence $\alpha' a \in I_u^{\mathsf{t}}(\beta')\, a \subseteq I_u^{\mathsf{t}}(\beta' a)$. □

Note that the definitions of trace-based transitive unwinding relations and of information sets are characterizations of the tpurge function, or

more precisely of the indistinguishability induced by the equality of the values of the tpurge function on traces. Therefore, these characterizations are completely independent of the underlying machine model.

However, these characterizations are not helpful for verifying security. The verification problem for deterministic finite-state systems can be solved efficiently with the help of state-based unwinding relations. These are equivalence relations on the states of the systems rather than on traces. Similar to the trace-based unwinding relation, these unwinding relations are also defined by unwinding conditions. These conditions define which states of the system have to be undistinguishable for the observing agent.

3.2.9 Definition (state-based transitive unwinding)**.** A *state-based transitive unwinding relation* for $u \in D$ is an equivalence relation $\sim_u^{\mathsf{st}} \subseteq S \times S$ that satisfies the following conditions for every $a \in A$ and $s, t \in S$:

(LR$^{\mathsf{st}}$)*:* If $\mathrm{dom}(a) \not\mapsto u$, then $s \sim_u^{\mathsf{st}} s \cdot a$.

(SC$^{\mathsf{st}}$)*:* If $s \sim_u^{\mathsf{st}} t$, then $s \cdot a \sim_u^{\mathsf{st}} t \cdot a$.

The existence of observation consistent unwinding relations is equivalent to the security of the system.

3.2.10 Theorem ([GM82])**.** *A system is t-secure iff for every $u \in D$ there exists a state-based transitive unwinding relation $\sim_u^{\mathsf{st}}$ that is observation consistent for u.*

These unwinding conditions provide both a useful proof technique and a tool for efficient verification algorithms, as we will see later in Chapter 5.

3.2.2 Transitively Securely Constructed Systems

A structural approach of characterizing t-security is to express it as a securely constructed system. For a given policy, a securely constructed system implements the policy in a way that the system is t-secure by construction. The idea is that the global state space is partitioned into local state spaces for every single agent and that the (global) transition function is assembled by local transitions functions, each for every agent. These local transition functions are only applied if the policy allows an information

flow from the corresponding agent. This construction enforces that no information can be transmitted that is not allowed by the policy.

Suppose that we have some arbitrary, but fixed enumeration $D = \{0, \dots, n-1\}$ of the agents, and that we have some policy $\rightarrowtail$ as a relation between these agents. We say a system is transitively securely constructed (w.r.t. t-security and a given policy $\rightarrowtail$) if it satisfies the following construction pattern. For every agent i, the set S_i is the local state space for agent i and the (global) state space S is the product of the local state space, i. e., $S = S_0 \times \cdots \times S_{n-1}$. Every local state space has a local initial state s_i^I and the (global) initial state is then $s^I = (s_0^I, \dots, s_{n-1}^I)$. For every agent i, we suppose that a transition function $\cdot_i$ exists. This transition function $\cdot_i$ is defined on all local states of S_i and for all actions a with $\mathrm{dom}(a) \rightarrowtail i$.

The transition function is then defined as

$$(s_0, \dots, s_{n-1}) \cdot a = (s_0', \dots, s_{n-1}')$$
$$\text{with } s_i' = \begin{cases} s_i \cdot_i a & \text{if } \mathrm{dom}(a) \rightarrowtail i \\ s_i & \text{otherwise .} \end{cases}$$

Note that the transition function is only used for agents where an edge between them exists in the policy.

For the observation function, we require that the observations of each agent only depend on its local state. Hence for every agent i and every state $s, t \in S$ with $\pi_i(s) = \pi_i(t)$, we have $\mathrm{obs}_i(s) = \mathrm{obs}_i(t)$. Alternatively, one can define the observation functions for every agent i as a function defined on its local state space S_i.

From this construction it is obvious that a securely constructed system is t-secure, and we can skip a proof of the following result.

3.2.11 Lemma. *Every transitively securely constructed system is t-secure w.r.t. the policy from which it is constructed.*

The completeness of this construction is provided with the next result. From every t-secure system, we can construct a t-secure securely constructed system by taking the purged traces as states. Indeed, we can show a stronger result: The constructed system is observation equivalent for every agent,

i. e., for every agent i and every trace α, the agent i has the same observation after α in both systems.

3.2.12 Lemma. *For every t-secure system, there exists an observation equivalent system which is transitively securely constructed from the same policy.*

Proof. As mentioned above, we take the purged traces as the local states. For a given system M with agents $D = \{0, \ldots, n-1\}$ and policy $\rightarrowtail$, the system M' is defined by

- ▷ the local states of each agent i is $S_i = \{\mathrm{tpurge}_i(\alpha) \mid \alpha \in A^*\}$,
- ▷ the local initial state of each agent i is $s_i^I = \epsilon$,
- ▷ the local transition function of each agent i is defined for every $s \in S_i$ and every $a \in \{b \in A \mid \mathrm{dom}(b) \rightarrowtail i\}$ by $s \cdot_i a = sa$,
- ▷ for any agent i, the observation function $\mathrm{obs}_i'(s)$ is defined as $\mathrm{obs}_i(s^I \cdot \pi_i(s))$.

From the construction of the state space, we have for every $i \in D$ and every $\alpha, \beta \in A^*$ with $\mathrm{tpurge}_i(\alpha) = \mathrm{tpurge}_i(\beta)$ that $\mathrm{obs}_i'(\tilde{s}^I \cdot \alpha) = \mathrm{obs}_i'(\tilde{s}^I \cdot \beta)$, where $\tilde{s}^I$ is the initial state of the constructed system. By applying the definition of t-security to the original system, we have for every trace α that both α and $\mathrm{tpurge}_i(\alpha)$ lead to the same observation for i. Hence, the securely constructed system is observation equivalent to the system from which it is constructed. □

Summarizing the previous two results gives:

3.2.13 Theorem. *A system is t-secure iff there exists an observation equivalent, transitively securely constructed system.*

3.3 Intransitive Noninterference

While transitive noninterference is too restrictive for the security requirements of many systems, Haigh and Young [HY87] introduced a relaxed version of noninterference called intransitive noninterference. The basic

idea is that a particular agent, which one trusts assumably, is allowed to downgrade or declassify some information. Previously high or confidential information is then allowed to get to some other agent with a lower security level only if it passes the downgrading agent and in the following is released or declassified by it. Inspired by a military context, Figure 3.2 illustrates this idea. The edges in this diagram depict the allowed information flows. The downgrader is allowed to transmit any information from a higher context to a lower one.

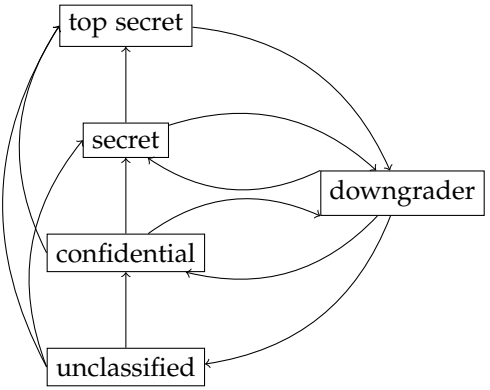

Figure 3.2. Trusted downgrader

For intransitive noninterference, we use the same kind of static policies as for transitive noninterference. However, the semantics of the policy differs. Intransitive noninterference relies on an interpretation of policies where paths of consecutive actions are considered. The simplest non-trivial intransitive policy which does not collapse to the transitive case, is the HDL policy depicted in Figure 3.3.

Intuitively, the interpretation of the HDL policy is that H is allowed to interfere with D and D is allowed to interfere with L, but a direct interference from H to L is prohibited. However, an indirect interference through D is allowed. This means that L is only allowed to observe anything about H's actions if D performed an action after H's actions. Hence, if there is no interaction of D, then L is not allowed to observe anything about H's

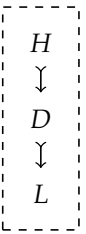

Figure 3.3. The HDL policy

behavior.

One intuition is that D's behavior is influenced by H's interaction, and since L can observe D's behavior, it can make conclusions about H's behavior.

Another intuition in the sense of information flow is that each action of an agent transmits information about all performed actions of those agents that are allowed to interfere with the agent. Since the security notions define the maximal allowed information flow, one is interested in the maximal information that a particular action can transmit. If the security notion allows it to transmit the performed actions of agents that are allowed to interfere, one can say that an action transmits all the previously performed actions that have reached the agent that performs the action. In a system with the policy in Figure 3.3, if there is an action h performed by H and an action d performed by D, then we say in the trace hd, the action d transmits the action h to L, since D is allowed to receive h from H and L is allowed to receive d from D.

3.3.1 Example. The system in Figure 3.4 has three agents H, D, and L and is considered in combination with the HDL policy of Figure 3.3.

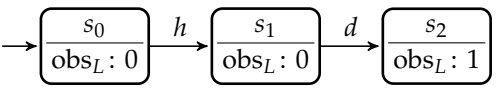

Figure 3.4. An i-secure, but not t-secure system

When the agent L observes 1, then it knows that there was an action h

performed in state s_0. Otherwise, the system would remain in s_0 and the observation of L remains 0. However, L can make this observation only if D performs an action after H's action. Otherwise the system would remain in state s_1 and L cannot make any conclusions about H's actions.

In the interpretation of intransitive noninterference, this system should be considered secure (w.r.t. the HDL policy), since L can only make conclusions about H if the information is transmitted by an action D.

However, according to the t-security definition of the previous section, this system is insecure. We have $\text{tpurge}_L(hd) = d = \text{tpurge}_L(d)$ but $\text{obs}_L(s_0 \cdot hd) = 1 \neq 0 = \text{obs}_L(s_0 \cdot d)$.

We will formalize this intuitive description of intransitive noninterference and we will give formal semantics to downgrading and transmission of actions.

3.3.1 Downgrading

The concept of downgrading is fundamental in the context of intransitive noninterference. The idea of downgrading is that an action is transmitted by a sequence of other actions. One intuition of this process is that with performing an action, an agent transmits all actions that it has received so far to all agents the agent is allowed to interfere.

Given a set of agents D, a policy $\rightarrowtail \subseteq D \times D$, a set of actions A, and a dom function that maps actions to agents, the action-labeled policy is a labeled graph with vertex set D. For every agent and each of its actions, there is an edge, labeled with this action, to every successor according to the policy. Formally, the edges of this labeled graph are given, by

$$\{(u, v, a) \mid \text{dom}(a) = u \text{ and } u \rightarrowtail v\} \quad .$$

We say an action a is downgraded by a sequence of actions α to an agent v if there is a subsequence β of $a\alpha$ such that β is the sequence of labels of a path in the corresponding labeled policy graph from $\text{dom}(a)$ to v.

Intuitively, α contains a subsequence of actions that transmits the action a from one agent to another, starting with $\text{dom}(a)$, to the observing agent v. This might also be seen as a casual chain of consecutive actions. In this

chain each action may depend on its previous action and therefore, by intransitivity, the end of the chain may depend on its first action.

3.3.2 Example. The action labeled policy of the HDL-policy in Figure 3.3 is depicted in Figure 3.5.

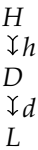

Figure 3.5. The action labeled HDL policy

To analyze whether the action h is downgraded to L in a trace hd, one has to verify that there is a path from $\mathrm{dom}(h) = H$ to L, labeled with hd. In this simple policy this is obviously the case. In contrast, if the trace is only h, then there is no path from H to L, labeled with h.

The next example shows why it is necessary to consider subsequences rather than the whole sequence as in the previous example

3.3.3 Example. The following policy consists of five agents H_1, H_2, D_1, D_2 and L, where each agent has exactly one action, except L. If we consider

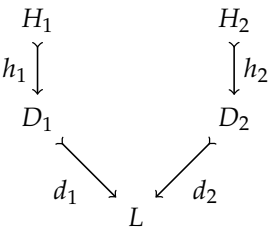

Figure 3.6. The action labeled $H_1H_2D_1D_2L$ policy

the trace $h_1h_2d_1d_2$ w.r.t. the policy in Figure 3.6, then there is no path from H_1 to L labeled with $h_1h_2d_1d_2$. However, there is a subsequence, namely h_1d_1 which labels such a path.

3.4 i-security

To formalize the idea of downgrading, Rushby [Rus92] introduced a function sources that collects all agents allowed to have influence on the observing agent. This means, an agent $\mathrm{dom}(a)$ is in the set $\mathrm{sources}_u(\alpha)$ if there is an action a in α that is downgraded by the remaining actions after the appearance of a in the trace to u, i. e., $\mathrm{dom}(a) = v$ and $\alpha = \beta a \gamma$ with a is downgraded by γ to u. Additionally, it is required that $u \in \mathrm{sources}_u(\alpha)$.

3.4.1 Definition (sources)**.** Define for every $u \in D$, $a \in A$, and $\alpha \in A^*$

$$\mathrm{sources}_u(\epsilon) = \{u\}$$
$$\mathrm{sources}_u(a\alpha) = \begin{cases} \mathrm{sources}_u(\alpha) \cup \{\mathrm{dom}(a)\} & \text{if } \mathrm{dom}(a) \rightarrowtail \mathrm{sources}_u(\alpha) \\ \mathrm{sources}_u(\alpha) & \text{otherwise .} \end{cases}$$

Note that the function $\mathrm{sources}_u(\alpha)$ is monotonically increasing in the parameter α, i. e., we have $\mathrm{sources}_u(\alpha) \subseteq \mathrm{sources}_u(a\alpha)$.

Originally, the intransitive purge function has be formulated by Haigh and Young [HY87], but instead, we use Rushby's simplified definition, based on the sources function [Rus92]. The intransitive purge function is defined next.

3.4.2 Definition (ipurge operator)**.** For every $u \in D$, $a \in A$, and $\alpha \in A^*$ define

$$\mathrm{ipurge}_u(\epsilon) = \epsilon$$
$$\mathrm{ipurge}_u(a\alpha) = \begin{cases} a\ \mathrm{ipurge}_u(\alpha) & \text{if } \mathrm{dom}(a) \in \mathrm{sources}_u(a\alpha) \\ \mathrm{ipurge}_u(\alpha) & \text{otherwise .} \end{cases}$$

The ipurge function removes all actions from a trace that are not downgraded to u. Hence, applied to a trace α, the resulting ipurge values consists of that subsequence of α, which exactly contains those actions that are allowed to influence u's observation.

Since $\mathrm{sources}_u(\alpha)$ is exactly the set of agents that have an action in the trace $\mathrm{ipurge}_u(\alpha)$ and the agent u itself, we can express the ipurge operator

directly without the sources function by

$$\mathrm{ipurge}_u(a\alpha) = \begin{cases} a\ \mathrm{ipurge}_u(\alpha) & \text{if } \mathrm{dom}(a) \rightarrowtail \mathrm{dom}(\mathrm{alph}(\mathrm{ipurge}_u(\alpha))) \\ & \qquad\qquad \text{or } \mathrm{dom}(a) \rightarrowtail u \\ \mathrm{ipurge}_u(\alpha) & \text{otherwise .} \end{cases}$$

Alternatively, the ipurge operator can also be defined on αa instead of $a\alpha$ by collecting the agents of the sources in a parameter, as defined as follows.

For every set of agents $X \subseteq D$, $a \in A$ and $\alpha \in A^*$ we "overload" the ipurge operator and define it for sets of agents by

$$\mathrm{ipurge}_X(\epsilon) = \epsilon$$
$$\mathrm{ipurge}_X(\alpha a) = \begin{cases} \mathrm{ipurge}_{X \cup \{\mathrm{dom}(a)\}}(\alpha)\ a & \text{if } \mathrm{dom}(a) \rightarrowtail X \\ \mathrm{ipurge}_X(\alpha) & \text{otherwise .} \end{cases}$$

This new ipurge operator relates to the previous one by $\mathrm{ipurge}_u(\alpha) = \mathrm{ipurge}_{\{u\}}(\alpha)$.

Analog to the previous security definition, i-security is defined as follows.

3.4.3 Definition (i-security)**.** A system is *i-secure* iff for every $u \in D$ and every $\alpha, \beta \in A^*$ the following implication holds:

$$\text{If } \mathrm{ipurge}_u(\alpha) = \mathrm{ipurge}_u(\beta), \text{ then } \mathrm{obs}_u(s^I \cdot \alpha) = \mathrm{obs}_u(s^I \cdot \beta) \ .$$

As in the transitive case, this definition can be rephrased. For every $u \in D$ and every $\alpha \in A^*$, we have

$$\mathrm{obs}_u(s^I \cdot \mathrm{ipurge}_u(\alpha)) = \mathrm{obs}_u(s^I \cdot \alpha) \ .$$

Another way to understand intransitive noninterference is that if there is an action that is not downgraded by some sequence of actions, then this single action should not have any visible influence on the observer.

3.4.4 Lemma. *A system is i-secure iff for every $u \in D$, $a \in A$, $s \in S$, and $\alpha \in A^*$, we have: if* $\mathrm{dom}(a) \notin \mathrm{sources}_u(a\alpha)$*, then* $\mathrm{obs}_u(s \cdot a\alpha) = \mathrm{obs}_u(s \cdot \alpha)$.

Proof. First, we show the direction from left to right of the equivalence. Let $s \in S$, $a \in A$, $u \in D$, and $\alpha \in A^*$ with $\mathrm{dom}(a) \notin \mathrm{sources}_u(\alpha)$. We show by an induction on the length of $\beta \in A^*$ that we have for every β:

$$\mathrm{ipurge}_u(\beta a \alpha) = \mathrm{ipurge}_u(\beta\alpha) \text{ and } \mathrm{sources}_u(\beta a \alpha) = \mathrm{sources}_u(\beta\alpha) \ .$$

For the base case, let $\beta = \epsilon$. From the assumption $\mathrm{dom}(a) \notin \mathrm{sources}_u(a\alpha)$ it follows $\mathrm{sources}_u(a\alpha) = \mathrm{sources}_u(\alpha)$ and hence $\mathrm{ipurge}_u(a\alpha) = \mathrm{ipurge}_u(\alpha)$.
For the inductive step, let $\beta = b\beta'$ with $b \in A$ and $\beta' \in A^*$ with $\mathrm{sources}_u(\beta' a\alpha) = \mathrm{sources}_u(\beta'\alpha)$ and $\mathrm{ipurge}_u(\beta' a\alpha) = \mathrm{ipurge}_u(\beta'\alpha)$.
Therefore, we have

$$\mathrm{dom}(b) \rightarrowtail \mathrm{sources}_u(\beta' a\alpha) \text{ iff } \mathrm{dom}(b) \rightarrowtail \mathrm{sources}_u(\beta'\alpha) \ .$$

By the definition of sources, we have

$$\mathrm{dom}(b) \in \mathrm{sources}_u(\beta' a\alpha) \text{ iff } \mathrm{dom}(b) \rightarrowtail \mathrm{sources}_u(\beta'\alpha) \ .$$

From the definition of ipurge, it follows $\mathrm{ipurge}_u(b\beta' a\alpha) = \mathrm{ipurge}_u(b\beta'\alpha)$. The claim follows by choosing some β with $s = s^I \cdot \beta$.

We show the other implication of the claim by an induction on the combined length of α and β. Since the base case $\alpha = \beta = \epsilon$ is obvious, we proceed with the inductive step. As induction hypothesis, we suppose that we have for all states $s \in S$ and all traces of shorter combined length $\alpha', \beta' \in A^*$ it holds:

$$\mathrm{ipurge}_u(\alpha') = \mathrm{ipurge}_u(\beta') \text{ implies } \mathrm{obs}_u(s \cdot \alpha') = \mathrm{obs}_u(s \cdot \beta') \ .$$

Let α and β in A^* with $\mathrm{ipurge}_u(\alpha) = \mathrm{ipurge}_u(\beta)$. Let $\alpha = a\alpha'$ for some $a \in A$ and $\alpha' \in A^*$.
Case 1: $\mathrm{dom}(a) \notin \mathrm{sources}_u(a\alpha')$.
In this case, we have $\mathrm{ipurge}_u(a\alpha') = \mathrm{ipurge}_u(\alpha') = \mathrm{ipurge}_u(\beta)$. By applying the induction hypothesis, we have for every state s that $\mathrm{obs}_u(s \cdot \alpha') = \mathrm{obs}_u(s \cdot \beta)$. And by the assumption that the left-hand side of the lemma holds, we have $\mathrm{obs}_u(s \cdot \alpha' a) = \mathrm{obs}_u(s \cdot \alpha')$. Hence, we have $\mathrm{obs}_u(s \cdot \alpha) = \mathrm{obs}_u(s \cdot \beta)$.

Case 2: $\mathrm{dom}(a) \in \mathrm{sources}_u(a\alpha')$.
We assume that $\beta = b\beta'$ with $b \in A$ and $\beta' \in A^*$ and $\mathrm{dom}(b) \in \mathrm{sources}_u(b\beta')$, since otherwise we proceed with Case 1 with the roles of α and β swapped. Hence, we have $\mathrm{ipurge}_u(a\alpha') = a\,\mathrm{ipurge}_u(\alpha')$ and $\mathrm{ipurge}_u(b\beta') = b\,\mathrm{ipurge}_u(\beta')$. Thus, we have $a = b$ and $\mathrm{ipurge}_u(\alpha') = \mathrm{ipurge}_u(\beta')$. Let s be an arbitrary state. Then, we can apply the induction hypothesis to the state $s \cdot a$ and obtain $\mathrm{obs}_u(s \cdot a\alpha') = \mathrm{obs}_u(s \cdot b\beta')$. □

The previous lemma states that for i-security, it is sufficient to consider a single action and determine if this action is downgraded. Therefore, for the definition of i-security with the ipurge operator, it is sufficient to consider traces that differ only in a single action. The next lemma is analogue to Lemma 3.2.4 in the transitive case.

3.4.5 Lemma. *A system is i-secure iff for every $u \in D$, $a \in A$, and $\alpha, \beta \in A^*$, we have if* $\mathrm{ipurge}_u(\alpha a\beta) = \mathrm{ipurge}_u(\alpha\beta)$*, then* $\mathrm{obs}_u(s^I \cdot \alpha a\beta) = \mathrm{obs}_u(s^I \cdot \alpha\beta)$.

Proof. We prove this lemma by contraposition. Assume that the right-hand side of the lemma does not hold. Then there are $u \in D$, $a \in A$, $\alpha, \beta \in A^*$ with $\mathrm{ipurge}_u(\alpha a\beta) = \mathrm{ipurge}_u(\alpha\beta)$ and $\mathrm{obs}_u(s^I \cdot \alpha a\beta) \neq \mathrm{obs}_u(s^I \cdot \alpha\beta)$. From $\mathrm{ipurge}_u(\alpha a\beta) = \mathrm{ipurge}_u(\alpha\beta)$, it follows that $\mathrm{dom}(a) \notin \mathrm{sources}_u(a\beta)$. Set $s = s^I \cdot \alpha$. Then we have $\mathrm{obs}_u(s \cdot a\beta) \neq \mathrm{obs}_u(s \cdot \beta)$. By applying Lemma 3.4.4, we obtain that the system is *not* i-secure.

For the other direction, assume that the system is *not* i-secure. By Lemma 3.4.4, there are $u \in D$, $s \in S$, $a \in A$, and $\beta \in A^*$ with $\mathrm{dom}(a) \notin \mathrm{sources}_u(a\beta)$ and $\mathrm{obs}_u(s \cdot a\beta) \neq \mathrm{obs}_u(s \cdot \beta)$. Due to reachability of every state, there is some $\alpha \in A^*$ with $s = s^I \cdot \alpha$. Then we have $\mathrm{ipurge}_u(\alpha\beta) = \mathrm{ipurge}_u(\alpha a\beta)$, which concludes the proof. □

3.4.1 Rushby's Unwinding

Rushby [Rus92] proposed the first unwinding for intransitive noninterference. However, his unwinding conditions are only sufficient but not necessary for i-security. Nonetheless, these conditions are of interest since if they are adopted to a trace-based unwinding then they are both a sound and complete characterization for ta-security as we will see later. Here, we name it after its inventor and call it a *Rushby unwinding*.

3.4.6 Definition (Rushby unwinding [Rus92])**.** An equivalence relation $\sim_u^{\mathsf{rushby}} \subseteq S \times S$ is a *state-based Rushby unwinding relation* for $u \in D$ if for every $a \in A$, $s, t \in S$ the following conditions are satisfied:

(LR$^{\mathsf{rushby}}$): If $\mathrm{dom}(a) \not\mapsto u$, then $s \sim_u^{\mathsf{rushby}} s \cdot a$.

(SC$^{\mathsf{rushby}}$): If $s \sim_u^{\mathsf{rushby}} t$ and $s \sim_{\mathrm{dom}(a)}^{\mathsf{rushby}} t$, then $s \cdot a \sim_u^{\mathsf{rushby}} t \cdot a$.

Rushby [Rus92] has shown that the existence of an observation consistent unwinding relation is sufficient for i-security.

3.4.7 Theorem ([Rus92])**.** *If for every $u \in D$ there is a state-based Rushby unwinding relation $\sim_u^{\mathsf{rushby}}$ that is observation consistent for u, then the system is i-secure.*

However, the state-based Rushby unwinding is not complete for i-security. The reason is that the unwinding relations on the states do not contain enough information about why the states are equivalent.

3.4.8 Example. The system depicted in Figure 3.7 is inspired by an example from [vdM07]. We will explain why for this system, it does not exist an observation-consistent state-based Rushby unwinding for L. Since $D_2 \not\mapsto D_1$, we have $s_0 \sim_{D_1}^{\mathsf{rushby}} s_1$ and since $H \not\mapsto L$, we have $s_0 \sim_L^{\mathsf{rushby}} s_1$, both by applying the (LR$^{\mathsf{rushby}}$) condition. Now, by applying (SC$^{\mathsf{rushby}}$) with the action d_1, we obtain $s_0 \sim_L^{\mathsf{rushby}} s_2$ and hence this relation is not observation consistent for L. However, since H cannot change L's observation without any interaction of D_1, this system is i-secure.

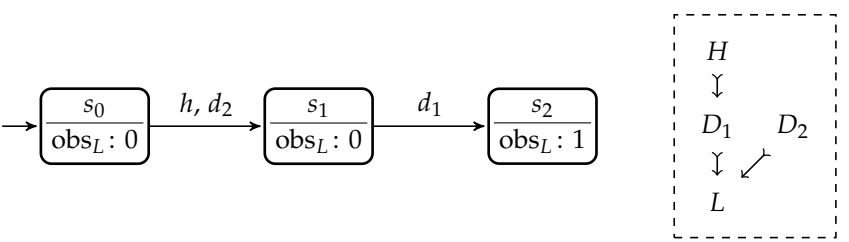

Figure 3.7. An i-secure system without an observation consistent Rushby unwinding

3.4.2 Policy Cuts

The basic idea of policy cuts can be described as follows. Given an intransitive policy and agents v and u such that there is a path from v to u in the policy, but u is not a direct successor of v. Then a necessary condition for security is that there is no information flow from v to u if the agents forming a cut (in a graph-theoretic sense) between v and u in the policy are removed from the system.

In [BP03], Backes and Pfitzmann have a definition of intransitive noninterference based on cuts in the policy called *blocking non-interference*. The main difference to our technique is that they use a completely different semantic model than ours and that they require to consider all possible cuts between every pair of agents. We will see that the latter is not necessary in our setting.

The following theorem shows that it is both necessary and sufficient to only take all successors of v as a cut if it is done for all possible pairs of agents u and v. Recall that $v^{\rightarrowtail}$ is the set of all successors of v in the policy graph, including v itself.

3.4.9 Theorem. *A system is i-secure iff for every $u \in D$, all $s \in S$, all $a \in A$, and all $\alpha \in A^*$ such that*

$$\mathrm{ipurge}_u(a\alpha) = \mathrm{ipurge}_u(\alpha) \textit{ and } \mathrm{dom}(a)^{\rightarrowtail} \cap \mathrm{dom}(\mathrm{alph}(\alpha)) = \emptyset \ ,$$

we have $\mathrm{obs}_u(s \cdot a\alpha) = \mathrm{obs}_u(s \cdot \alpha)$.

Proof. We prove this theorem by contraposition. Assume first that there are $u \in D$, $a \in A$, $\alpha \in A^*$, and $s \in S$ with $\mathrm{ipurge}_u(a\alpha) = \mathrm{ipurge}_u(\alpha)$, $\mathrm{dom}(a)^{\rightarrowtail} \cap \mathrm{dom}(\mathrm{alph}(\alpha)) = \emptyset$, and $\mathrm{obs}_u(s \cdot a\alpha) \neq \mathrm{obs}_u(s \cdot \alpha)$. Then clearly, $\mathrm{dom}(a) \notin \mathrm{sources}_u(a\alpha)$ and hence, the system is *not* i-secure.

For the other direction of the proof, assume that the system is *not* i-secure. By Lemma 3.4.4, we have that there are $u \in D$, $s \in S$, $a \in A$, and $\alpha \in A^*$ with $\mathrm{dom}(a) \notin \mathrm{sources}_u(a\alpha)$, but $\mathrm{obs}_u(s \cdot a\alpha) \neq \mathrm{obs}_u(s \cdot \alpha)$. We choose α of minimal length for all possible choices of a and s with this property.

Assume that there are some $b \in A$ and $\beta, \gamma \in A^*$ with $\alpha = \beta b \gamma$ and $\mathrm{dom}(a) \rightarrowtail \mathrm{dom}(b)$. We will show that the existence of such an action b

would contradict the minimality of the length of α.
Case 1: $\mathrm{obs}_u(s \cdot \beta b \gamma) \neq \mathrm{obs}_u(s \cdot \beta\gamma)$.
In this case, we could choose $s' = s \cdot \beta$, $a' = b$, and $\alpha' = \gamma$ instead of s, a, and α. This contradicts the minimal length of α.
Case 2: $\mathrm{obs}_u(s \cdot a\beta b\gamma) \neq \mathrm{obs}_u(s \cdot a\beta\gamma)$.
With the same argument as in the previous case, we have $s' = s \cdot a\beta$.
Case 3: This is the remaining case, i. e., $\mathrm{obs}_u(s \cdot \beta b\gamma) = \mathrm{obs}_u(s \cdot \beta\gamma)$ and $\mathrm{obs}_u(s \cdot a\beta b\gamma) = \mathrm{obs}_u(s \cdot a\beta\gamma)$. Hence, we have

$$\mathrm{obs}_u(s \cdot \beta b\gamma) = \mathrm{obs}_u(s \cdot \beta\gamma) \neq \mathrm{obs}_u(s \cdot a\beta b\gamma) = \mathrm{obs}_u(s \cdot a\beta\gamma) \ .$$

Here, we can choose $s' = s$, $a' = b$, and $\alpha' = \beta\gamma$. Then, the condition $\mathrm{ipurge}_u(\beta\gamma) = \mathrm{ipurge}_u(a\beta\gamma)$ is satisfied, and again we have a contradiction to the minimality of the length of α. □

For a pair of agents $v = \mathrm{dom}(a)$ and u as in Theorem 3.4.9, the agent v is isolated from all other agents (if we ignore v's incoming edges). Since this is universally quantified over all agents $v \not\rightarrowtail u$, we can allow that all agents, except v itself and the removed agents $\{w \in D \mid v \rightarrowtail w, v \neq w\}$, are allowed to interfere with u directly. Since the resulting policy is transitive, we can apply the simpler definition of t-security rather than i-security on this modified system.

We formalize the described construction by introducing a system $\mathcal{C}(v, u)$ for agents v and u with $v \not\rightarrowtail u$, which has the same states as the original system. Let $C = \{w \in D \mid v \rightarrowtail w, v \neq w\}$ be the already mentioned cut in the policy, i. e., the set of all proper successors of v. The set of agents of $\mathcal{C}(v, u)$ is the set $D \setminus C$. The actions of the agents from C are removed from the action set of $\mathcal{C}(v, u)$ and the transitions corresponding to these actions are deleted. The observations of u in $\mathcal{C}(v, u)$ are the same as in the original system and all other agents have constant observations. The policy that applies to $\mathcal{C}(v, u)$ is the policy that has the edges $w \rightarrowtail u$ for all $w \in D \setminus C$ with $w \neq v$.

3.4.10 Theorem. *A system is i-secure iff for every $v, u \in D$ with $v \not\rightarrowtail u$ the system $\mathcal{C}(v, u)$ is t-secure.*

Proof. We prove this result by contraposition. Suppose first that the right-hand side does not hold. Hence, there are $v, u \in D$ with $v \not\rightarrow u$ such that $\mathcal{C}(v,u)$ is *not* t-secure. By Lemma 3.2.4, there are $w \in D$, $a \in A$, $s \in S$, and $\beta \in \{a \in A \mid v \not\rightarrow \mathrm{dom}(a)\}^*$ with $\mathrm{dom}(a) \not\rightarrow w$ and $\mathrm{obs}_w(s \cdot a\beta) \neq \mathrm{obs}_w(s \cdot \beta)$. We also assume that β is of minimal length for all possible choices of w, a, and s. This guarantees that β does not contain any action of $\mathrm{dom}(a)$. Since u is the only agent that has non-constant observations, we have $u = w$, and since v is the only agent that is not allowed to interfere with u, we have $v = \mathrm{dom}(a)$. From $v \not\rightarrow \mathrm{dom}(\mathrm{alph}(\beta))$, it follows $v \notin \mathrm{sources}_u(a\beta)$. By Lemma 3.4.4, the system is *not* i-secure.

For proving the other direction of this result, suppose that the system is *not* i-secure. By Theorem 3.4.9, there are $u \in D$, $s \in S$, $a \in A$, and $\alpha \in A^*$ with $\mathrm{ipurge}_u(a\alpha) = \mathrm{ipurge}_u(\alpha)$, $\mathrm{dom}(a)^{\rightarrowtail} \cap \mathrm{dom}(\mathrm{alph}(\alpha)) = \emptyset$, and $\mathrm{obs}_u(s \cdot a\alpha) \neq \mathrm{obs}_u(s \cdot \alpha)$. From $\mathrm{ipurge}_u(a\alpha) = \mathrm{ipurge}_u(\alpha)$, it follows $\mathrm{dom}(a) \not\rightarrow u$, and from $\mathrm{dom}(a)^{\rightarrowtail} \cap \mathrm{dom}(\mathrm{alph}(\alpha)) = \emptyset$, it follows $\alpha \in \{a \in A \mid v \not\rightarrow \mathrm{dom}(a)\}^*$. By Lemma 3.2.4, the system $\mathcal{C}(\mathrm{dom}(a), u)$ is *not* t-secure. □

This is some kind of reduction from i-security to t-security. Instead of analyzing a single system with respect to i-security, it is sufficient to analyze several systems with respect to t-security. However, this number of systems is quadratic in the number of agents in the original system.

3.4.3 State-based Unwinding for i-security

We use the technique of policy cuts to obtain a sound and complete state-based unwinding for i-security. Each unwinding relation is defined for a pair of agents v and u. The agent u has the role of an observer. The role of the agent v is to perform actions that should not be revealed to u. Since the transition relating to v's action should be hidden, the corresponding states are treated to be equivalent for u. To guarantee that v's actions are not downgraded, only agents w with $v \not\rightarrow w$ are allowed to perform actions. These actions are performed synchronously in equivalent states. This is essentially the same as the state-based transitive unwinding from Definition 3.2.9 applied to each of the systems $\mathcal{C}(v,u)$ as defined above.

3.4.11 Definition (state-based intransitive unwinding)**.** A *state-based intransitive unwinding relation* for $v, u \in D$ is an equivalence relation $\sim^{\mathsf{si}}_{v,u} \subseteq S \times S$ such that for every $s, t \in S$, $a \in A$ the following conditions hold

(LR$^{\mathsf{si}}$)*:* If $v \not\rightarrowtail u$ and $\mathrm{dom}(a) = v$, then $s \sim^{\mathsf{si}}_{v,u} s \cdot a$.

(SC$^{\mathsf{si}}$)*:* If $s \sim^{\mathsf{si}}_{v,u} t$ and $v \not\rightarrowtail \mathrm{dom}(a)$ then $s \cdot a \sim^{\mathsf{si}}_{v,u} t \cdot a$.

The soundness and completeness of this unwinding-based definition is established in the next result.

3.4.12 Theorem. *A system is i-secure iff for every $u, v \in D$ there is a state-based intransitive unwinding relation $\sim^{\mathsf{si}}_{v,u}$ that is observation consistent for u.*

Proof. First, we prove the direction from left to right. Assume that the system is i-secure. For every $u, v \in D$ define a relation $\sim^{\mathsf{si}}_{v,u}$ for every $s, t \in S$ by

$$s \sim^{\mathsf{si}}_{v,u} t \text{ iff for every } \alpha \in \{a \in A \mid v \not\rightarrowtail \mathrm{dom}(a)\}^* \text{ it holds:}$$
$$\mathrm{obs}_u(s \cdot \alpha) = \mathrm{obs}_u(t \cdot \alpha) \ .$$

We will show that $\sim^{\mathsf{si}}_{v,u}$ is a state-based intransitive unwinding relation for v and u. Clearly, $\sim^{\mathsf{si}}_{v,u}$ is an equivalence relation on S. Define $X = \{a \in A \mid v \not\rightarrowtail \mathrm{dom}(a)\}^*$. Since $\epsilon \in X$, we have that $\sim^{\mathsf{si}}_{v,u}$ is observation consistent for u.

For showing (LR$^{\mathsf{si}}$), suppose that $v \not\rightarrowtail u$ and let $a \in A$ with $\mathrm{dom}(a) = v$. Then for every $\alpha \in X$, we have $\mathrm{dom}(a) \not\rightarrowtail \mathrm{dom}(\mathrm{alph}(\alpha))$ and hence $\mathrm{dom}(a) \notin \mathrm{sources}_u(a\alpha)$. From i-security and Lemma 3.4.4, it follows $\mathrm{obs}_u(s \cdot a\alpha) = \mathrm{obs}_u(s \cdot \alpha)$. Therefore, we have $s \cdot a \sim^{\mathsf{si}}_{v,u} s$.

For showing (SC$^{\mathsf{si}}$), let $s, t \in S$ with $s \sim^{\mathsf{si}}_{v,u} t$ and $a \in A$ with $v \not\rightarrowtail \mathrm{dom}(a)$. From $s \sim^{\mathsf{si}}_{v,u} t$, it follows that for every $\alpha \in X$ it holds $\mathrm{obs}_u(s \cdot \alpha) = \mathrm{obs}_u(t \cdot \alpha)$. But also $a\alpha$ is in X and hence, we have $\mathrm{obs}_u(s \cdot a\alpha) = \mathrm{obs}_u(t \cdot a\alpha)$ and therefore $s \cdot a \sim^{\mathsf{si}}_{v,u} t \cdot a$.

For the other direction of the proof, assume that the system is *not* i-secure. By Theorem 3.4.9, there are $a \in S$, $a \in A$, and $\alpha \in A^*$ with $\mathrm{ipurge}_u(a\alpha) = \mathrm{ipurge}_u(\alpha)$, $\mathrm{dom}(a)^{\rightarrowtail} \cap \mathrm{dom}(\mathrm{alph}(\alpha)) = \emptyset$, and $\mathrm{obs}_u(s \cdot a\alpha) \neq \mathrm{obs}_u(s \cdot \alpha)$. Set $v = \mathrm{dom}(a)$ and assume that $\sim^{\mathsf{si}}_{v,u}$ satisfies (LR$^{\mathsf{si}}$) and (SC$^{\mathsf{si}}$). By (LR$^{\mathsf{si}}$),

we have $s \cdot a \sim^{\mathsf{si}}_{v,u} s$ and by ($\mathrm{SC}^{\mathsf{si}}$) it follows $s \cdot a\alpha \sim^{\mathsf{si}}_{v,u} s \cdot \alpha$. But, since $\mathrm{obs}_u(s \cdot a\alpha) \neq \mathrm{obs}_u(s \cdot \alpha)$, the relation $\sim^{\mathsf{si}}_{v,u}$ is not observation consistent for u. □

We will see in Chapter 5 that this result can be used to obtain an efficient algorithm for verifying i-security.

3.5 ta-security

Van der Meyden [vdM07] pointed out some drawbacks of the definition of i-security. The main concern is that the definition of i-security allows the observer to observe the interleaving of actions which should not be observable by any single agent under the general assumption that the system is completely asynchronous. We illustrate and discuss this property with the next example.

3.5.1 Example. In Figure 3.8 is the policy from Figure 3.6 (but without the labels on the edges). Again, we suppose that there is some system with actions h_1, h_2, d_1, and d_2 of the agent H_1, H_2, D_1, and D_2, respectively. If we consider the trace $h_1h_2d_1d_2$ and the trace $h_2h_1d_1d_2$, then we see that in both traces, every action is downgraded to L. Hence, applying ipurge_L to any of these traces does not remove any action from these traces. That means that L is allowed to distinguish these two traces by its observations. However, no agent is allowed to observe which of the two actions h_1 and h_2 has been performed first. For instance, agent D_1 only observes that h_1 has been performed but does not know if there was some action h_2 before of after this action h_1. An analogue argument holds for D_2. Hence, neither D_1 nor D_2 can transmit information about the ordering of h_1 and h_2. Therefore, L cannot reconstruct the ordering of h_1 and h_2 from the received information. Therefore, we need to require that L is not allowed to distinguish the traces $h_1h_2d_1d_2$ and $h_2h_1d_1d_2$. However, the ordering of the action d_1 and d_2 should be allowed to be distinguished by L, since L immediately gets the action when it is performed and hence learns which of these actions was first.

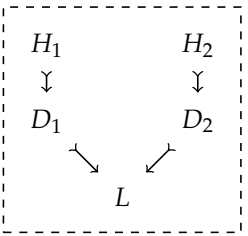

Figure 3.8. The $H_1H_2D_1D_2L$ policy

To capture this issue, van der Meyden [vdM07] introduced the ta operator[1] that transforms traces into trees. These trees have the property that the information about the unobservable ordering of the actions has been removed. Therefore, two traces which differ only in the ordering of those actions that are not allowed to be observed directly by any single agent have the same trees.

Formally, the ta operator is defined as follows.

3.5.2 Definition (ta operator)**.** Define for every $u \in D$, every $a \in A$, and every $\alpha \in A^*$

$$\mathrm{ta}_u(\epsilon) = \epsilon$$
$$\mathrm{ta}_u(\alpha a) = \begin{cases} (\mathrm{ta}_u(\alpha), \mathrm{ta}_{\mathrm{dom}(a)}(\alpha), a) & \text{if } \mathrm{dom}(a) \rightarrowtail u \\ \mathrm{ta}_u(\alpha) & \text{otherwise .} \end{cases}$$

The value $\mathrm{ta}_u(\alpha)$ is also called the ta_u-tree of α.

The function ta transforms a trace into a binary tree. The expression $(\mathrm{ta}_u(\alpha), \mathrm{ta}_{\mathrm{dom}(a)}(\alpha), a)$ should be read as a binary tree with the root vertex a and a left subtree encoded by $\mathrm{ta}_u(\alpha)$ and a right subtree encoded by $\mathrm{ta}_{\mathrm{dom}(a)}(\alpha)$. Hence, the inner vertices of such a tree are labeled by actions of the trace, all leaves are labeled with ϵ. The labeling of the leaves with ϵ is not essential for this definition and may be omitted, however, we want to stay consistent with the definition in the literature.

[1] ta stands for *transmission* of information about *actions*

3.5.3 Example. The ta_L value of the trace $h_1h_2d_1d_2$ w.r.t. the policy in Figure 3.8 is

$$\begin{aligned}\mathrm{ta}_L(h_1h_2d_1d_2) &= \big(\mathrm{ta}_L(h_1h_2d_1), \mathrm{ta}_{D_2}(h_1h_2d_1), d_2\big)\\ &= \big(\big(\mathrm{ta}_L(h_1h_2), \mathrm{ta}_{D_1}(h_1h_2), d_1\big), \mathrm{ta}_{D_2}(h_1h_2), d_2\big)\\ &= \big(\big(\mathrm{ta}_L(h_1), \mathrm{ta}_{D_1}(h_1), d_1\big), \big(\mathrm{ta}_{D_2}(h_1), \mathrm{ta}_{H_2}(h_1), h_2\big), d_2\big)\\ &= ((\epsilon, (\epsilon, \epsilon, h_1), d_1), (\epsilon, \epsilon, h_2), d_2) \quad .\end{aligned}$$

Similarly, computing the ta_L value of the trace with h_1 and h_2 swapped, gives

$$\begin{aligned}\mathrm{ta}_L(h_2h_1d_1d_2) &= \big(\mathrm{ta}_L(h_2h_1d_1), \mathrm{ta}_{D_2}(h_2h_1d_1), d_2\big)\\ &= \big(\big(\mathrm{ta}_L(h_2h_1), \mathrm{ta}_{D_1}(h_2h_1), d_1\big), \mathrm{ta}_{D_2}(h_2h_1), d_2\big)\\ &= \big(\big(\mathrm{ta}_L(h_2), \big(\mathrm{ta}_{D_1}(h_2), \mathrm{ta}_{H_1}(h_2), h_1\big), d_1\big), \mathrm{ta}_{D_2}(h_2), d_2\big)\\ &= ((\epsilon, (\epsilon, \epsilon, h_1), d_1), (\epsilon, \epsilon, h_2), d_2) \quad .\end{aligned}$$

As we see, both traces have the same ta_L value. The information about the ordering of h_1 and h_2 is removed from this representation of the traces. Figure 3.9 gives a graphical representation of the ta_L-tree constructed from any of these two traces.

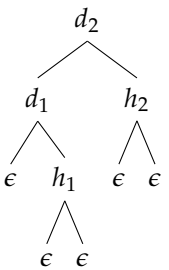

Figure 3.9. The ta_L-tree of Example 3.5.3

Instead of defining the ta operator from right to left, it is also possible to define it from left to right. However, this requires one to get through

the ta-tree from the root to the leafs for adding an action at a leaf. This is performed by the att function (att is the abbreviation for add-to-tree), which takes an action a, an agent u, and a ta-tree as argument.

3.5.4 Definition (att function)**.** For every $u \in D$, $a, b \in A$ and t, t' ta-trees, define

$$\begin{aligned} \mathrm{att}(a, u, \epsilon) &= \begin{cases} (\epsilon, \epsilon, a) & \text{if } \mathrm{dom}(a) \rightarrowtail u \\ \epsilon & \text{otherwise} \end{cases} \\ \mathrm{att}(a, u, (t, t', b)) &= \big(\mathrm{att}(a, u, t), \mathrm{att}(a, \mathrm{dom}(b), t'), b\big) \end{aligned}$$

With this function, it is possible to give a definition of $\mathrm{ta}_u(a\alpha)$ instead of $\mathrm{ta}_u(\alpha a)$.

3.5.5 Lemma. *For every $u \in D$, $a \in A$, $\alpha \in A^*$, we have*

$$\mathrm{ta}_u(a\alpha) = \mathrm{att}(a, u, \mathrm{ta}_u(\alpha)) \ .$$

Proof. We prove this lemma by an induction on α. For the base case $\alpha = \epsilon$, we have

$$\mathrm{ta}_u(a) = \begin{cases} (\epsilon, \epsilon, a) & \text{if } \mathrm{dom}(a) \rightarrowtail u \\ \epsilon & \text{otherwise} \ , \end{cases}$$

which is exactly the definition of $\mathrm{att}(a, u, \epsilon)$.

For the inductive step, let $\alpha = \beta b$ with $\beta \in A^*$ and $b \in A$ and assume as induction hypothesis that for every $v \in D$ it holds:

$$\mathrm{ta}_v(a\beta) = \mathrm{att}(a, v, \mathrm{ta}_v(\beta)) \ . \tag{I.H.}$$

Let $u \in D$. There are two cases.
Case 1: $\mathrm{dom}(b) \not\rightarrowtail u$.
In this case, we have

$$\begin{aligned} \mathrm{att}(a, u, \mathrm{ta}_u(\beta b)) &= \mathrm{att}(a, u, \mathrm{ta}_u(\beta)) \\ &\overset{\text{(I.H.)}}{=} \mathrm{ta}_u(a\beta) \ . \end{aligned}$$

Case 2: $\mathrm{dom}(b) \rightarrowtail u$.
This gives

$$\begin{aligned}\mathrm{att}(a,u,\mathrm{ta}_u(\beta b)) &= \mathrm{att}(a,u,(\mathrm{ta}_u(\beta),\mathrm{ta}_{\mathrm{dom}(b)}(\beta),b))\\ &= (\mathrm{att}(a,u,\mathrm{ta}_u(\beta)),\mathrm{att}(a,\mathrm{dom}(b),\mathrm{ta}_{\mathrm{dom}(b)}(\beta)),b)\\ &\overset{\text{(I.H.)}}{=} (\mathrm{ta}_u(a\beta),\mathrm{ta}_{\mathrm{dom}(b)}(a\beta),b)\\ &= \mathrm{ta}_u(a\beta b)\ .\end{aligned}$$ □

Analogue to the previous security definitions, ta-security is defined in the usual way.

3.5.6 Definition (ta-security)**.** A system is *ta-secure* iff for every $u \in D$ and every $\alpha, \beta \in A^*$ such that $\mathrm{ta}_u(\alpha) = \mathrm{ta}_u(\beta)$, we have $\mathrm{obs}_u(s^I \cdot \alpha) = \mathrm{obs}_u(s^I \cdot \beta)$.

The next example illustrates the difference between i-security and ta-security.

3.5.7 Example. We will see that a minimal example of an i-secure but not ta-secure system only requires three agents. Note that for systems with only two agents, both i-security and ta-security are equivalent to t-security.

As agents, we take H, D, and L and the policy that, except the self-loops, only contains the edges $H \rightarrowtail D$ and $D \rightarrowtail L$. We consider the traces hld and lhd from the point of view of L. These two traces have different ipurge_L values since ipurge does not remove anything from these traces. However, the ta_L values are the same, since we have

$$\begin{aligned}\mathrm{ta}_L(hld) &= (\mathrm{ta}_L(hl),\mathrm{ta}_D(hl),d)\\ &= ((\epsilon,\epsilon,l),(\epsilon,\epsilon,h),d)\ ,\end{aligned}$$

and

$$\begin{aligned}\mathrm{ta}_L(lhd) &= (\mathrm{ta}_L(lh),\mathrm{ta}_D(lh),d)\\ &= ((\epsilon,\epsilon,l),(\epsilon,\epsilon,h),d)\ .\end{aligned}$$

Intuitively, L does not know if it has performed its l action before or after the action h since no agent has this information. The action d only transmits the information that the action h has been performed before the action d.

An example for an i-secure but *not* ta-secure system with this policy is depicted in Figure 3.10.

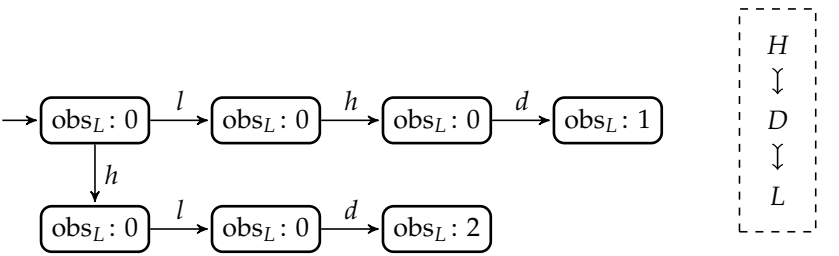

Figure 3.10. An i-secure, but not ta-secure system

The relation between i-security and ta-security is that ta-security implies i-security. The reason is that the ta operator possibly removes more information from a trace than the ipurge operator.

3.5.8 Lemma ([vdM07])**.** *For every agent $u \in D$ and all traces $\alpha, \beta \in A^*$ with* $\mathrm{ipurge}_u(\alpha) = \mathrm{ipurge}_u(\beta)$*, we have* $\mathrm{ta}_u(\alpha) = \mathrm{ta}_u(\beta)$*. Moreover, every ta-secure system is i-secure.*

3.5.1 Trace-based Unwinding for ta-security

Alternatively to the definition of ta-security by the use of the ta operator, ta-security has a very elegant representation by a trace-based unwinding.

3.5.9 Definition (trace-based intransitive unwinding)**.** A *trace-based intransitive unwinding relation* for $u \in D$ is an equivalence relation $\sim_u^{\mathsf{ti}} \subseteq A^* \times A^*$ that satisfies for every $a \in A$ and every $\alpha, \beta \in A^*$:

(LR$^{\mathsf{ti}}$)*:* If $\mathrm{dom}(a) \not\rightarrowtail u$, then $\alpha \sim_u^{\mathsf{ti}} \alpha a$.

(SC$^{\mathsf{ti}}$)*:* If $\alpha \sim_u^{\mathsf{ti}} \beta$ and $\alpha \sim_{\mathrm{dom}(a)}^{\mathsf{ti}} \beta$, then $\alpha a \sim_u^{\mathsf{ti}} \beta a$.

The unwinding conditions of this trace-based unwinding look very similar to the unwinding conditions of Rushby's state-based unwinding. However, van der Meyden [vdM07] has shown that it is complete and sound for ta-security.

3.5.10 Theorem ([vdM07])**.** *A system is ta-secure iff for every agent u, there is a trace-based intransitive unwinding $\sim_u^{\mathsf{ti}}$ which is observation consistent for u.*

The relation between the trace-based intransitive unwinding and the ta operator can be stated more precisely than in the previous result: The equivalence between traces with respect to a smallest $\sim_u^{\mathsf{ti}}$ is the same as with respect to the equality of the ta_u values.

3.5.11 Lemma ([vdM07])**.** *Let $u \in D$ and let $\sim_u^{\mathsf{ti}}$ be the smallest trace-based intransitive unwinding relation for u. Then for every $\alpha, \beta \in A^*$, we have*

$$\alpha \sim_u^{\mathsf{ti}} \beta \textit{ iff } \mathrm{ta}_u(\alpha) = \mathrm{ta}_u(\beta) \ .$$

3.5.2 Information Sets for ta-security

Again it is possible to give a characterization in terms of information sets.

3.5.12 Definition (information sets for intransitive noninterference)**.** *Information sets for intransitive noninterference* are defined for every $u \in D$ by

$$I_u^{\mathsf{i}}(\epsilon) = \{b \in A \mid \mathrm{dom}(b) \not\rightarrowtail u\}$$
$$I_u^{\mathsf{i}}(\alpha a) = \begin{cases} (I_u^{\mathsf{i}}(\alpha) \cap I_{\mathrm{dom}(a)}^{\mathsf{i}}(\alpha))\ a\ I_u^{\mathsf{i}}(\epsilon) & \text{if } \mathrm{dom}(a) \rightarrowtail u \\ I_u^{\mathsf{i}}(\alpha) & \text{otherwise .} \end{cases}$$

As we will show with the next theorem, $I_u^{\mathsf{i}}(\alpha)$ is the set of all traces, which u is not allowed to distinguish from α. This can be seen as u's allowed knowledge after the trace α. The expression

$$(I_u^{\mathsf{i}}(\alpha) \cap I_{\mathrm{dom}(a)}^{\mathsf{i}}(\alpha))\ a\ I_u^{\mathsf{i}}(\epsilon)$$

should be read as follows: if $\mathrm{dom}(a)$ is allowed to interfere with u, then with performing the action a, $\mathrm{dom}(a)$ transmits its whole knowledge, namely

$I^{\mathsf{i}}_{\mathrm{dom}(a)}(\alpha)$, to u. The agent u combines its previous knowledge $I^{\mathsf{i}}_u(\alpha)$ with the new knowledge received from $\mathrm{dom}(a)$. Additionally, u knows that the action a has been performed, hence the a is appended to every trace. But there is also some uncertainty about actions possibly performed after the action a of agents that are not allowed to interfere with u. These are the traces in $I^{\mathsf{i}}_u(\epsilon)$, which are appended to all traces.

3.5.13 Theorem. *For every $u \in D$ and all $\alpha, \beta \in A^*$, we have $\alpha \in I^{\mathsf{i}}_u(\beta)$ iff* $\mathrm{ta}_u(\alpha) = \mathrm{ta}_u(\beta)$.

Proof. Let $u \in D$. We prove the implication from left to right by an induction on the combined length of α and β. Let $\alpha, \beta \in A^*$ such that $\alpha \in I^{\mathsf{i}}_u(\beta)$. The claim holds trivially for the base case $\alpha = \beta = \epsilon$. In the case of $\alpha = \alpha' a$ with $\mathrm{dom}(a) \not\rightarrowtail u$, we have $\alpha' \in I_u(\beta)$ and by induction hypothesis, it follows $\mathrm{ta}_u(\alpha) = \mathrm{ta}_u(\alpha' a) = \mathrm{ta}_u(\alpha')$. In the case of $\beta = \beta' b$ with $\mathrm{dom}(b) \not\rightarrowtail u$, we have $\alpha \in I^{\mathsf{i}}_u(\beta) = I^{\mathsf{i}}_u(\beta')$. Again by induction hypothesis, we have $\mathrm{ta}_u(\alpha) = \mathrm{ta}_u(\beta') = \mathrm{ta}_u(\beta)$. Now consider the case that $\alpha = \alpha' a$ and $\beta = \beta' b$ with $\mathrm{dom}(a) \rightarrowtail u$ and $\mathrm{dom}(b) \rightarrowtail u$. Since

$$\alpha' a \in I^{\mathsf{i}}_u(\beta' b) = (I^{\mathsf{i}}_u(\beta') \cap I^{\mathsf{i}}_{\mathrm{dom}(b)}(\beta'))\ b\ I^{\mathsf{i}}_u(\epsilon)\ ,$$

we have $a = b$ and $\alpha' \in I_u(\beta')$ and $\alpha' \in I_{\mathrm{dom}(a)}(\beta')$. Applying the induction hypothesis, we have $\mathrm{ta}_u(\alpha') = \mathrm{ta}_u(\beta')$ and $\mathrm{ta}_{\mathrm{dom}(a)}(\alpha') = \mathrm{ta}_{\mathrm{dom}(a)}(\beta')$. This gives $\mathrm{ta}_u(\alpha' a) = \mathrm{ta}_u(\beta' b)$.

For the other implication, we consider α, β with $\mathrm{ta}_u(\alpha) = \mathrm{ta}_u(\beta)$. If $\alpha = \alpha' a$ or $\beta = \beta' b'$ with $\mathrm{dom}(a) \not\rightarrowtail u$ or $\mathrm{dom}(b) \not\rightarrowtail u$, then we can apply the induction hypothesis directly with the same arguments as above. For the rest of the proof, consider $\alpha = \alpha' a$ and $\beta = \beta' b$ with $\mathrm{dom}(a) \rightarrowtail u$ and $\mathrm{dom}(b) \rightarrowtail u$ and $\mathrm{ta}_u(\alpha' a) = \mathrm{ta}_u(\beta' b)$. From the definition of ta, we have $a = b$ and $\mathrm{ta}_u(\alpha') = \mathrm{ta}_b(\beta')$ and $\mathrm{ta}_{\mathrm{dom}(a)}(\alpha') = \mathrm{ta}_{\mathrm{dom}(b)}(\beta')$. By induction hypothesis, we have $\alpha' \in I^{\mathsf{i}}_u(\beta')$ and $\alpha' \in I^{\mathsf{i}}_{\mathrm{dom}(a)}(\beta')$. Therefore, $\beta a \in (I^{\mathsf{i}}_u(\beta') \cap I^{\mathsf{i}}_{\mathrm{dom}(a)}(\beta'))\ a\ I^{\mathsf{i}}_u(\epsilon)$. □

Information sets provide a clear and intuitive way how the allowed knowledge of agent evolves during a run of the system. But they do not provide a tool for verifying security, since every information set is an infinite

set. For that, a state-based unwinding is an appropriate tool in the case of finite-state systems.

3.5.3 State-based Unwinding for ta-security

Before defining a state-based unwinding, some further notation is necessary for describing the effects showing the difference between traces purged by the ipurge operator and ta-trees.

3.5.14 Definition (swappable actions)**.** Let $\alpha, \alpha' \in A^*$ and $a, b \in A$ and $u \in D$. The actions a and b are *swappable* in $\alpha a b \alpha'$ (w.r.t. the agent u), written as $\alpha a b \alpha' \leftrightarrow_u^{\mathsf{swap}} \alpha b a \alpha'$ iff

$$\mathrm{dom}(a)^{\rightarrowtail} \cap \mathrm{dom}(b)^{\rightarrowtail} \cap (\{u\} \cup \{\mathrm{dom}(c) | c \in \mathrm{alph}(ab\alpha')\}) = \emptyset \ .$$

Intuitively, two actions are swappable if no successor of their corresponding agents performs any action after these two actions that are downgraded to the observing agent.

Traces are defined to be order indistinguishable if one can be transformed into the other by a sequence of swap operations.

3.5.15 Definition (order indistinguishable traces)**.** Traces $\alpha, \alpha' \in A^*$ are *order indistinguishable for* u, written as $\alpha \equiv_u^{\mathsf{oi}} \alpha'$, if $\alpha \overset{*}{\leftrightarrow}_u^{\mathsf{swap}} \alpha'$.

The following lemma shows that if traces can be transformed into each other by only swap operations of swappable actions, then these traces have the same ta value.

3.5.16 Lemma. *Let $u \in D$ and $\alpha, \beta \in A^*$. If we have $\alpha \leftrightarrow_u^{\mathsf{swap}} \beta$, then* $\mathrm{ta}_u(\alpha) = \mathrm{ta}_u(\beta)$.

Proof. Due to Lemma 3.5.11, it is sufficient to show that if $\alpha \leftrightarrow_u^{\mathsf{swap}} \beta$, then $\alpha \sim_u^{\mathsf{ti}} \beta$. By the definition of $\leftrightarrow_u^{\mathsf{swap}}$, there are $\gamma, \delta \in A^*$ and $a, b \in A$ with $\alpha = \gamma a b \delta$ and $\beta = \gamma b a \delta$. Let $v \in \{u\} \cup \mathrm{dom}(\mathrm{alph}(ab\delta))$.

We prove this lemma by an induction on the length of δ. For the base case, we have that $\mathrm{dom}(a)$ or $\mathrm{dom}(b)$ does not interfere with v. Due to symmetry, we consider just the case that $\mathrm{dom}(a) \not\rightarrowtail v$. By ($\mathrm{LR}^{\mathsf{ti}}$), we have

$\gamma a \sim_v^{\mathsf{ti}} \gamma$ and from $\mathrm{dom}(a) \not\rightarrowtail \mathrm{dom}(b)$ it follows $\gamma a \sim_{\mathrm{dom}(b)}^{\mathsf{ti}} \gamma$. By ($\mathrm{SC}^{\mathsf{ti}}$), we have then $\gamma ab \sim_v^{\mathsf{ti}} \gamma b$ and again by ($\mathrm{LR}^{\mathsf{ti}}$), we have $\gamma ab \sim_v^{\mathsf{ti}} \gamma ba$.

For the inductive step, consider a trace δc and assume that for every $v \in \{u\} \cup \mathrm{dom}(\mathrm{alph}(ab\delta))$ it holds $\gamma ab\delta \sim_v^{\mathsf{ti}} \gamma ba\delta$. Moreover, we have $\gamma ab\delta \sim_{\mathrm{dom}(c)}^{\mathsf{ti}} \gamma ba\delta$. By ($\mathrm{SC}^{\mathsf{ti}}$), we have $\gamma ab\delta c \sim_v^{\mathsf{ti}} \gamma ba\delta c$. □

If the ipurge values of two traces are order indistinguishable, then the ta-values of these traces are the same.

3.5.17 Corollary. *Let $u \in D$ and $\alpha, \beta \in A^*$. If we have $\mathrm{ipurge}_u(\alpha) \equiv_u^{\mathsf{oi}} \mathrm{ipurge}_u(\beta)$, then $\mathrm{ta}_u(\alpha) = \mathrm{ta}_u(\beta)$ holds.*

Proof. This follows directly from Lemma 3.5.8 and Lemma 3.5.16. □

This can be summarized as follows: The ta operator removes the same actions from a trace as the ipurge operator. Additionally it removes the information about the order of swappable actions from the trace. The resulting information is exactly the information encoded in the ta-tree.

3.5.18 Lemma. *Let $u \in D$ and $\alpha, \beta \in A^*$. Then*

$$\mathrm{ta}_u(\alpha) = \mathrm{ta}_u(\beta) \textit{ iff } \mathrm{ipurge}_u(\alpha) \equiv_u^{\mathsf{oi}} \mathrm{ipurge}_u(\beta) \ .$$

Proof. For the proof from left to right, assume that $\mathrm{ta}_u(\alpha) = \mathrm{ta}_u(\beta)$ holds. Since $\mathrm{ipurge}_u(\alpha) = \mathrm{ipurge}_u(\beta)$ implies $\mathrm{ta}_u(\alpha) = \mathrm{ta}_u(\beta)$, without loss of generality, we can assume that the traces α and β are already purged, i. e., $\alpha = \mathrm{ipurge}_u(\alpha)$ and $\beta = \mathrm{ipurge}_u(\beta)$.

Assume that $\alpha \not\equiv_u^{\mathsf{oi}} \beta$. Since α and β have the same ta_u-trees, the same actions are contained in α and β. Let $\gamma \in A^*$ with $\gamma \equiv_u^{\mathsf{oi}} \beta$ such that γ has a common prefix with α of maximal length for all possible choices of γ. Hence, there are $\alpha', \alpha'', \gamma' \in A^*$, and $a \in A$ with $\alpha = \alpha' a \alpha''$, $\gamma = \alpha' \gamma'$, and $a\alpha'' \neq \gamma'$.

We assume that the position of a in γ' is the left-most position among all possible choices for γ'. Then γ' is of the form $\delta ba\delta'$ with $\delta ba\delta' \not\leftrightarrow_u^{\mathsf{swap}} \delta ab\delta'$. Therefore, there is some $v \in \mathrm{sources}_u(\delta')$ with $\mathrm{dom}(a) \rightarrowtail v$ and $\mathrm{dom}(b) \rightarrowtail v$. Hence, $\mathrm{ta}_v(\alpha' \delta ba)$ is a subtree of $\mathrm{ta}_u(\alpha)$. Because of the corresponding number of occurrences of a in α, the corresponding subtree

would be $\mathrm{ta}_v(\alpha' a)$. But the numbers of corresponding bs in this two trees do not match. Hence, we have a contradiction to the assumption that $\alpha \not\equiv_u^{\mathsf{oi}} \beta$.

The other direction of the proof directly follows from Corollary 3.5.17. □

These two properties, namely, i-security and the property that it is allowed to swap swappable actions in a trace, are exactly the properties that characterize ta-security.

3.5.19 Theorem. *A system is ta-secure iff*

1. *it is i-secure, and*
2. *for every $s \in S$, every $u \in D$, every $\alpha \in A^*$, and every $a, b \in A$ such that a and b are swappable in $ab\alpha$ w.r.t. u, we have $\mathrm{obs}_u(s \cdot ab\alpha) = \mathrm{obs}_u(s \cdot ba\alpha)$.*

Proof. For the proof of the implication from left to right, assume that the system is ta-secure. Since ta-security implies i-security, condition 1. is satisfied. Assume that the condition 2. is not satisfied. Then there exist $s \in S$, $u \in D$, $\alpha \in A^*$, and $a, b \in A$ such that a and b are swappable in $ab\alpha$ and $\mathrm{obs}_u(s \cdot ab\alpha) \neq \mathrm{obs}_u(s \cdot ba\alpha)$. Since Lemma 3.5.18 implies $\mathrm{ta}_u(ab\alpha) = \mathrm{ta}_u(ba\alpha)$, we have that the system is not ta-secure which contradicts our assumption.

For proving the other direction by contraposition, we assume that the system is *not* ta-secure. Additionally, suppose that the system is i-secure. Since the system is *not* ta-secure, there are $u \in D$ and $\alpha, \beta \in A^*$ with $\mathrm{ta}_u(\alpha) = \mathrm{ta}_u(\beta)$ and $\mathrm{obs}_u(s^I \cdot \alpha) \neq \mathrm{obs}_u(s^I \cdot \beta)$. By Lemma 3.5.18, we have $\mathrm{ipurge}_u(\alpha) \equiv_u^{\mathsf{oi}} \mathrm{ipurge}_u(\beta)$. Hence, α arose from β by removing actions according to ipurge_u and by swapping of swappable actions. More precisely, there is a finite sequence of traces $\alpha_0, \ldots, \alpha_{n-1}$ with

$$\mathrm{ipurge}_u(\alpha) = \alpha_0 \leftrightarrow_u^{\mathsf{swap}} \cdots \leftrightarrow_u^{\mathsf{swap}} \alpha_{n-1} = \mathrm{ipurge}_u(\beta) \ .$$

From i-security it follows $\mathrm{obs}_u(s^I \cdot \alpha) = \mathrm{obs}_u(s^I \cdot \mathrm{ipurge}_u(\alpha))$ and $\mathrm{obs}_u(s^I \cdot \beta) = \mathrm{obs}_u(s^I \cdot \mathrm{ipurge}_u(\beta))$. Hence, there is some $0 \leq i < n-1$ with $\mathrm{obs}_u(s^I \cdot \alpha_i) \neq \mathrm{obs}_u(s^I \cdot \alpha_{i+1})$. Due to the definition of the $\leftrightarrow_u^{\mathsf{swap}}$ relation, we have $\alpha_i = \gamma ab\delta$ and $\alpha_{i+1} = \gamma ba\delta$ for some $a, b \in A$ and $\gamma, \delta \in A^*$ with a and b are swappable in $ab\delta$ w.r.t. u. The claim follows with $s = s^I \cdot \gamma$. □

The previous lemma showed that ta-security is characterized as a conjunction of two properties. We will exploit this property for a characterization in terms of a state-based unwinding. For the first condition, a state-based unwinding was already provided in Definition 3.4.11. Therefore, it is sufficient to formulate unwinding conditions for the second property.

The state-based unwinding relations $\sim^{\mathsf{sta}}_{v,w,u}$ are defined for any triple of agents v, w, and u. The agent u takes the role of an observer. The agents v and w may perform actions that are guaranteed to be swappable w.r.t. u, since no agent that is allowed to receive both v's and w's actions is involved in any considered trace.

3.5.20 Definition (state-based unwinding for ta-security)**.** A *state-based unwinding relation for ta-security* for agents $u, v, w \in D$ with $v \neq w$ is an equivalence relation $\sim^{\mathsf{sta}}_{v,w,u} \subseteq S \times S$ such that for every $s, t \in S$ and $a \in A$ the following conditions hold:

(SWAP$^{\mathsf{sta}}$)*:* If $\mathrm{dom}(a) = v$ and $\mathrm{dom}(b) = w$ and $v \not\rightarrowtail w$ and $w \not\rightarrowtail v$, and $v \not\rightarrowtail u$ or $w \not\rightarrowtail u$, then $s \cdot ab \sim^{\mathsf{sta}}_{v,w,u} s \cdot ba$.

(SC$^{\mathsf{sta}}$)*:* If $s \sim^{\mathsf{sta}}_{v,w,u} t$ and $a \in A$ with $v \not\rightarrowtail \mathrm{dom}(a)$ or $w \not\rightarrowtail \mathrm{dom}(a)$, then $s \cdot a \sim^{\mathsf{sta}}_{v,w,u} t \cdot a$.

The next theorem shows the soundness and completeness of the state-based unwinding for ta-security if it is applied to an i-secure system. Hence for checking ta-security, it is necessary to verify both i-security and the existence of a state-based unwinding relation for ta-security that is observation consistent for the observing agent.

3.5.21 Theorem. *A system is ta-secure if and only if it is i-secure, and for every agent u, v, and w there exists a state-based unwinding relation for ta-security $\sim^{\mathsf{sta}}_{v,w,u}$ that is observation consistent for u.*

Proof. First, we proof the implication from left to right. Suppose that the system is ta-secure. This directly implies that it is i-secure. Let $u \in D$. We will show the existence of a state-based unwinding relation $\sim^{\mathsf{sta}}_{v,w,u}$ for ta-security for agents $u, v \neq w$ that is observation consistent for u. Define for all states $s, t \in S$

$s \sim^{\mathsf{sta}}_{v,w,u} t$ iff for all $\alpha \in \{a \in A \mid v \not\rightarrowtail \mathrm{dom}(a) \text{ or } w \not\rightarrowtail \mathrm{dom}(a)\}^*$ it holds:

$$\mathrm{obs}_u(s \cdot \alpha) = \mathrm{obs}_u(t \cdot \alpha) \ .$$

The observation consistency for u is immediate from the choice of $\alpha = \epsilon$. For showing the condition (SWAP$^{\mathsf{sta}}$), let $s \in S$, $a, b \in A$, and $v, w \in D$ with $\mathrm{dom}(a) = v$, $\mathrm{dom}(b) = w$, $v \not\rightarrow w$, $w \not\rightarrow v$, and $v \not\rightarrow u$ or $w \not\rightarrow u$. Let $\alpha \in \{a \in A \mid v \not\rightarrow \mathrm{dom}(a) \text{ or } w \not\rightarrow \mathrm{dom}(a)\}^*$. Then the actions a and b are swappable in $ab\alpha$. By Theorem 3.5.19, we have $\mathrm{obs}_u(s \cdot ab\alpha) = \mathrm{obs}_u(s \cdot ba\alpha)$.

To show (SC$^{\mathsf{sta}}$), let $s, t \in S$ with $s \sim^{\mathsf{sta}}_{v,w,u} t$. Then we have for every $\alpha \in \{b \in A \mid v \not\rightarrow \mathrm{dom}(b) \text{ or } w \not\rightarrow \mathrm{dom}(b)\}^*$ that $\mathrm{obs}_u(s \cdot \alpha) = \mathrm{obs}_u(t \cdot \alpha)$. Let $a \in A$ with $v \not\rightarrow \mathrm{dom}(a)$ or $w \not\rightarrow \mathrm{dom}(a)$. Hence $a\alpha \in \{b \in A \mid v \not\rightarrow \mathrm{dom}(b) \text{ or } w \not\rightarrow \mathrm{dom}(b)\}^*$. Therefore $\mathrm{obs}_u(s \cdot a\alpha) = \mathrm{obs}_u(t \cdot a\alpha)$ and hence $s \cdot a \sim^{\mathsf{sta}}_{v,w,u} t \cdot a$.

For the other direction of the proof, assume that the system is *not* ta-secure, but i-secure. We will show that every state-based unwinding relation for ta-security satisfying the conditions (SWAP$^{\mathsf{sta}}$) and (SC$^{\mathsf{sta}}$) is not observation consistent.

Since the system is not ta-secure, by Theorem 3.5.19, there are $u \in D$, $s \in S$, $a, b \in A$, and $\alpha \in A^*$ with a and b swappable in $ab\alpha$, but $\mathrm{obs}_u(s \cdot ab\alpha) \neq \mathrm{obs}_u(s \cdot ba\alpha)$. Set $v = \mathrm{dom}(a)$ and $w = \mathrm{dom}(b)$. By the condition (SWAP$^{\mathsf{sta}}$), we have $s \cdot ab \sim^{\mathsf{sta}}_{v,w,u} s \cdot ba$. Since a and b are swappable in $ab\alpha$, for every $c \in \mathrm{alph}(\alpha)$, we have $v \not\rightarrow \mathrm{dom}(c)$ or $w \not\rightarrow \mathrm{dom}(c)$. Hence, by applying the condition (SC$^{\mathsf{sta}}$) on all actions of α, we obtain $s \cdot ab\alpha \sim^{\mathsf{sta}}_{v,w,u} s \cdot ba\alpha$. From $\mathrm{obs}_u(s \cdot ab\alpha) \neq \mathrm{obs}_u(s \cdot ba\alpha)$, it follows that the relation $\sim^{\mathsf{sta}}_{v,w,u}$ is not observation consistent for u. □

We will use this result in our algorithms, since we have a characterization for i-security in terms of a state-based unwinding. Together with the state-based unwinding for ta-security, it is possible to verify ta-security on finite-state systems.

3.5.4 Intransitively Securely Constructed Systems

As in the transitive case, in the intransitive case as well, we can characterize noninterference by a structural description of a secure system or, more precisely, of an observational equivalent system.

The main difference between transitively and intransitively securely constructed systems are the local transition functions. In the transitive case, the local transition function $\cdot_i$ only depends on the local states of u_i. In the intransitive case, it depends on both, the local state space of u_i and of the local state space of the agent that performs the action. More precisely, each local transition function is of the form

$$\cdot_i \colon \{(s_j, s_i, a) \mid s_j \in S_j, s_i \in S_i, a \in A \text{ with } \mathrm{dom}(a) = u_j \text{ and } u_j \rightarrowtail u_i\} \to S_i.$$

Suppose again, the agents denoted as numbers $0, \ldots, n-1$ and a policy $\rightarrowtail$ is given. For any state $(s_0, \ldots, s_{n-1}) \in S = S_0 \times \cdots \times S_{n-1}$, the transition function is defined as

$$(s_0, \ldots, s_{n-1}) \cdot a = (s'_0, \ldots, s'_{n-1})$$
$$\text{with } s'_i = \begin{cases} (s_j, s_i) \cdot_i a & \text{if } \mathrm{dom}(a) \rightarrowtail i \\ s_i & \text{otherwise .} \end{cases}$$

This global transition function ensures two properties.

First, if an action a is performed, then only those local transitions are triggered that perform changes on the local state spaces of the agents with which $\mathrm{dom}(a)$ is allowed to interfere.

Second, if a local transition is performed then this transition only depends on the local state of the agent who performs the action and on the local state of the agent who receives the action. If we assume inductively that the local state space of each agent only contains information that the agent is allowed to own, then any agent that receives any information is allowed to get it, since the sender is allowed to own this information and by the construction of the transition function, the receiver is allowed to receive it.

As usual the observations of each agent only depend on its local state space and hence all possible forbidden information flows are ruled out in this system. This informal argumentation is stated more precisely in the next lemma, which shows that this construction builds ta-secure systems.

3.5.22 Lemma. *In every intransitively securely constructed system, it holds for*

every $i \in D$ *and for every* $\alpha, \beta \in A^*$ *with* $\mathrm{ta}_i(\alpha) = \mathrm{ta}_i(\beta)$ *that* $\pi_i(s^I \cdot \alpha) = \pi_i(s^I \cdot \beta)$.

Proof. Let $i \in D$. We prove this lemma by an induction on the combined length of α and β. Since the base case is clear, we proceed with the inductive step. Let $\alpha, \beta \in A^*$ with $\mathrm{ta}_i(\alpha) = \mathrm{ta}_i(\beta)$ and assume that for all traces with smaller combined length, the claim holds. Since at least one of α and β is not equal to ϵ, we can assume that $\alpha = \alpha' a$ for some $a \in A$ and $\alpha' \in A^*$.
Case 1: $\mathrm{dom}(a) \not\rightarrowtail i$.
In this case, we have $\pi_i(s^I \cdot \alpha a) = \pi_i(s^I \cdot \alpha)$, and since $\mathrm{ta}_i(\beta) = \mathrm{ta}_i(\alpha a) = \mathrm{ta}_i(\alpha)$, we have, by applying the induction hypothesis, that $\pi_i(s^I \cdot \alpha) = \pi_i(s^I \cdot \beta)$.
Case 2: $\mathrm{dom}(a) \rightarrowtail i$.
In this case, we can assume that $\beta = \beta' b$ with $\mathrm{dom}(b) \rightarrowtail i$, since otherwise we could proceed with Case 1 with the roles of α and β swapped. From $\mathrm{ta}_i(\beta' b) = \mathrm{ta}_i(\alpha' a)$, it follows $a = b$, $\mathrm{ta}_i(\alpha') = \mathrm{ta}_i(\beta')$, and $\mathrm{ta}_{\mathrm{dom}(a)}(\alpha') = \mathrm{ta}_{\mathrm{dom}(a)}(\beta')$. By applying the induction hypothesis, we have $\pi_i(s^I \cdot \alpha') = \pi_i(s^I \cdot \beta')$ and $\pi_{\mathrm{dom}(a)}(s^I \cdot \alpha') = \pi_{\mathrm{dom}(a)}(s^I \cdot \beta')$. Hence, we have

$$\begin{aligned}\pi_i(s^I \cdot \alpha' a) &= (\pi_{\mathrm{dom}(a)}(s^I \cdot \alpha'), \pi_i(s^I \cdot \alpha')) \cdot_i a \\ &= (\pi_{\mathrm{dom}(a)}(s^I \cdot \beta'), \pi_u(s^I \cdot \beta')) \cdot_i b \\ &= \pi_i(s^I \cdot \beta' b) \ .\end{aligned}$$

□

The following corollary follows immediately from the previous result, since the observations of each agent depend only on its local states.

3.5.23 Corollary. *Every intransitively securely constructed system is ta-secure w.r.t. the policy from which it is constructed.*

The completeness of intransitively securely constructed systems is established by the next lemma. The constructed system takes the ta values of each agent as the states of its local state space.

3.5.24 Lemma. *For every ta-secure system, there exists an observation equivalent system which is intransitively securely constructed from the same policy.*

Proof. As mentioned above, we take the ta values as the local states. For a given system M with agents $D = \{0, \ldots, n-1\}$ and policy $\rightarrowtail$ construct a system M' where

- ▷ the local state space of each agent i is $S_i = \{\mathrm{ta}_i(\alpha) \mid \alpha \in A^*\}$,
- ▷ the local initial state for every agent i is $s_i^I = \epsilon$,
- ▷ the local transition functions are for every $i, j \in D$, every $a \in A$, and every $s_i \in S_i$ and $s_j \in S_j$ defined by $(s_j, s_i) \cdot_i a = (s_j, s_i, a)$,
- ▷ for any agent i, the observation function is defined by $\mathrm{obs}_i'(s) = \mathrm{obs}_i(s^I \cdot \alpha)$ for some $\alpha \in A^*$ with $s = \mathrm{ta}_i(\alpha)$. Since the system M is ta-secure, every α with $\mathrm{ta}_i(\alpha) = s$ leads to the same observation for i, and hence, the observation function is well-defined.

From the construction of the state space, we have for every $i \in D$ and every $\alpha, \beta \in A^*$ with $\mathrm{ta}_i(\alpha) = \mathrm{ta}_i(\beta)$ that $\mathrm{obs}_i'(\tilde{s}^I \cdot \alpha) = \mathrm{obs}_i'(\tilde{s}^I \cdot \beta)$, where $\tilde{s}^I$ is the initial state of the constructed system. Since the ta values are the states of the constructed system, and since in the original system the observations only depend on the ta values, the constructed system is observation equivalent to the system from which it is constructed. □

Summarizing the previous two results give:

3.5.25 Theorem. *A system is ta-secure iff there exists an observation-equivalent, intransitively securely constructed system.*

This result shows again that ta-security captures the intuition of intransitive noninterference much better than i-security. A similar pattern of securely constructed systems for i-security is not possible in this way, except that there is a global clock or at least a global order of the actions, which is transmitted along with the actions.

3.6 Noninterference with Transmission of Observations

The definitions of i-security and ta-security allow the transmission of previously performed actions according to some restriction given by the policy. However, this transmitted information is the maximal allowed information that an agent may have and this is generally more information then the agent actually has. This was one of the criticisms of Roscoe and Goldsmith [RG99] about previous security definitions. They proposed that agents are only allowed to transmit information that they actually have and not information that they are allowed to have. Van der Meyden [vdM07] formalized their ideas and adapted them to state-based systems that we use here. He proposed two trace-based operators and corresponding security definitions: to-security and ito-security. The abbreviation to stands for *transmission of observation* and ito for *immediate transmission of observation*. Both operators allow the transmission of previously made observations instead of the collected actions. The difference between these definitions is how fast information is transmitted.

3.6.1 to-security

The first of these two security definitions is to-security. This notion is defined by the to operator which builds a tree which contains the views of those agents that are allowed to interfere with the observing agent.

3.6.1 Definition (to operator)**.** For every agent $u \in D$, every action $a \in A$, and every $\alpha \in A^*$ define

$$\mathrm{to}_u(\epsilon) = \mathrm{obs}_u(s^I)$$
$$\mathrm{to}_u(\alpha a) = \begin{cases} (\mathrm{to}_u(\alpha), \mathrm{view}_{\mathrm{dom}(a)}(\alpha), a) & \text{if } \mathrm{dom}(a) \rightarrowtail u \\ \mathrm{to}_u(\alpha) & \text{otherwise .} \end{cases}$$

Intuitively, with this operator, the maximal information that is allowed to be transmitted is the observed information of those agents that perform

an action and are allowed to interfere with the observing agent u. The corresponding security definition is:

3.6.2 Definition (to-security)**.** A system is *to-secure* iff for every $u \in D$, $\alpha, \beta \in A^*$ with $\mathrm{to}_u(\alpha) = \mathrm{to}_u(\beta)$, we have $\mathrm{obs}_u(s^I \cdot \alpha) = \mathrm{obs}_u(s^I \cdot \beta)$.

The next example illustrates the differences between to-security and ta-security.

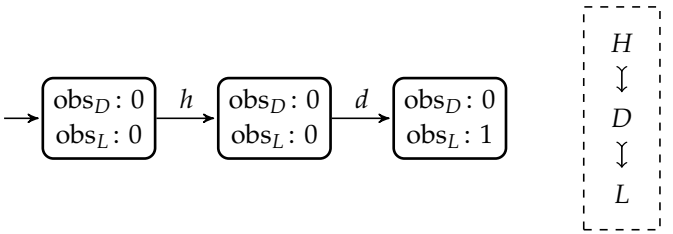

Figure 3.11. A ta-secure, but not to-secure system

3.6.3 Example. For analyzing to-security, we do not only need the observations of the observing agent, we also need the observations of the downgrading agent. Hence, in the system in Figure 3.11, also the observations of agent D are specified. Intuitively, in this system, L observes whether H has performed an action h because agent D performs an action afterwards. With respect to ta-security, the policy allows it and hence, this system is ta-secure. However, agent D has constant observations and does not observe whether H has performed any action. According to to-security, L is not allowed to observe any information about H, since D does not observe anything about H and hence, cannot transmit any information about H to L. More formally, the traces hd and d have the same to value:

$$\begin{aligned} \mathrm{to}_L(hd) &= (\mathrm{to}_L(h), \mathrm{view}_D(h), d) \\ &= (0, 0, d) \\ &= \mathrm{to}_L(d) \ . \end{aligned}$$

Since the to values of the traces hd and d are the same, but L's observations after these two traces are different, the system is not to-secure.

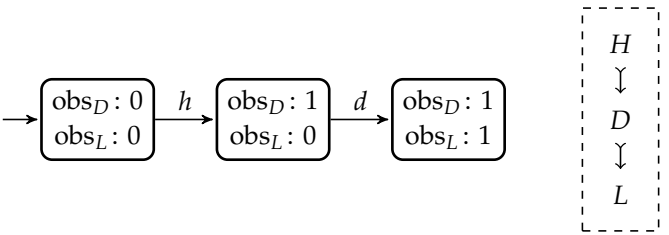

Figure 3.12. A to-secure system

In contrast, if D actually observes whether H has performed an action in the initial state, then the system would be to-secure. In the system in Figure 3.12, the action h, performed in the initial state, changes D's observations and hence, this information can be downgraded by D. Now, the to values of the traces hd and d are different:

$$\begin{aligned} \mathrm{to}_L(hd) &= (\mathrm{to}_L(h), \mathrm{view}_D(h), d) \\ &= (0, 0\, h\, 1, d) \ , \end{aligned}$$

but

$$\mathrm{to}_L(d) = (0, 0, d) \ .$$

This modified system, shown in Figure 3.12, is to-secure.

It is possible to express to-security without the tree-like structure of the to operator. Moreover, this tree structure may contain some redundancies, since if an agent v that is allowed to interfere with the agent u performs several actions, the corresponding views of v in u's to value are all prefixes of the last one. Therefore, it is sufficient to have the value of the view function of the largest prefix which ends with an action of the corresponding agent. In this sense, let $\mathrm{tview}_u(\alpha)$ be the largest prefix of $\mathrm{view}_u(\alpha)$ that ends in an action a with $\mathrm{dom}(a) = u$. With this definition of tview, it is sufficient to consider the tpurge_u value and the tview_v values of all agents v interfering with u but other than u itself.

3.6.4 Lemma ([vdM07]). *A system is to-secure iff for every $u \in D$, every $\alpha, \beta \in$*

D with $\mathrm{tpurge}_u(\alpha) = \mathrm{tpurge}_u(\beta)$ *and* $\mathrm{tview}_v(\alpha) = \mathrm{tview}_v(\beta)$ *for all* $v \neq u$ *with* $v \rightarrowtail u$*, we have* $\mathrm{obs}_u(s^I \cdot \alpha) = \mathrm{obs}_u(s^I \cdot \beta)$.

3.6.2 ito-security

Similar to the idea of to-security is the idea of ito-security. This notion of security was also introduced by van der Meyden [vdM07] in order to compare his other notions of security with those of Roscoe and Goldsmith [RG99]. Compared to the to operator, the ito operator transmits information coming from other agents faster. If an agent, other than the observing agent, performs an actions, then the ito operator allows the receiver to obtain the corresponding view including the observation stemming from the last action in the trace. This last observation was not transmitted by the to operator. This last observation is not transmitted if the last action is one of the observing agent, since otherwise the following security definition would not make any sense—every system would be secure.

3.6.5 Definition (ito operator)**.** For every $u \in D$, $a \in A$, and $\alpha \in A^*$ define

$$\mathrm{ito}_u(\epsilon) = \mathrm{obs}_u(s^I)$$

$$\mathrm{ito}_u(\alpha a) = \begin{cases} \mathrm{ito}_u(\alpha) & \text{if } \mathrm{dom}(a) \not\rightarrowtail u \\ (\mathrm{ito}_u(\alpha), \mathrm{view}_{\mathrm{dom}(a)}(\alpha), a) & \text{if } \mathrm{dom}(a) = u \\ (\mathrm{ito}_u(\alpha), \mathrm{view}_{\mathrm{dom}(a)}(\alpha a), a) & \text{otherwise .} \end{cases}$$

The security definition based on the ito operator follows the pattern of the previous definitions.

3.6.6 Definition (ito-security)**.** A system is *ito-secure* iff for every $u \in D$ and every $\alpha, \beta \in A^*$ with $\mathrm{ito}_u(\alpha) = \mathrm{ito}_u(\beta)$, we have $\mathrm{obs}_u(s^I \cdot \alpha) = \mathrm{obs}_u(s^I \cdot \beta)$.

The next example shows the differences between to-security and ito-security.

3.6.7 Example. The system in Figure 3.13 is only a slight modification of the previous example. Essentially with the same arguments as for the system in Figure 3.11, this system is not to-secure. However, the ito values differ

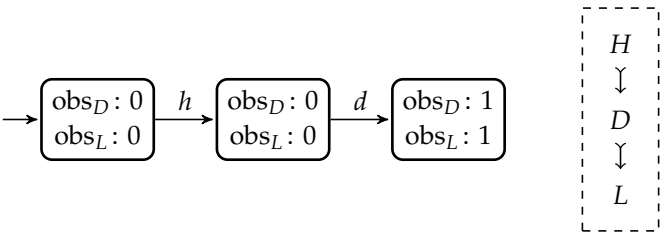

Figure 3.13. An ito-secure, but not to-secure system

for the traces hd and d:

$$\begin{aligned} \mathrm{ito}_L(hd) &= (\mathrm{ito}_L(h), \mathrm{view}_D(hd), d) \\ &= (0, 0\,d\,1, d) \ , \end{aligned}$$

but

$$\begin{aligned} \mathrm{ito}_L(d) &= (\mathrm{ito}_L(\epsilon), \mathrm{view}_D(d), d) \\ &= (0, 0\,d\,0, d) \ . \end{aligned}$$

Since these values are different, the system is ito-secure.

Similar to the characterization of to-security, we give a characterization of ito-security that does not need the tree-like structure of the ito operator. Instead, it is based on tpurge and a modification of the view function. This modification of the view function, called ftview_u, gives the longest prefix of the view_u value that ends *after* the first observation of u right after u's last action.

First, define a function $\mathrm{lpre}_u \colon A^* \to A^*$, which gives the largest prefix of a string that ends in an action a with $\mathrm{dom}(a) = u$. That is

$$\begin{aligned} \mathrm{lpre}_u(\epsilon) &= \epsilon \\ \mathrm{lpre}_u(\alpha a) &= \begin{cases} \alpha a & \text{if } \mathrm{dom}(a) = u \\ \mathrm{lpre}_u(\alpha) & \text{otherwise} \ . \end{cases} \end{aligned}$$

Now, ftview_u is defined for all $u \in D$ and all $\alpha \in A^*$ by

$$\mathrm{ftview}_u(\alpha) = \mathrm{view}_u(\mathrm{lpre}_u(\alpha)) \ .$$

The value of ftview_u may contain more information than the value of tview_u, since tview_u ends after the last action of u and ftview_u also contains u's observation obtained from its last action.

Similar to Lemma 3.6.4, we show that ito-security can be characterized by a flatter representation.

3.6.8 Lemma. *A system is ito-secure iff for every $u \in D$, every $\alpha, \beta \in A$ with $\mathrm{tpurge}_u(\alpha) = \mathrm{tpurge}_u(\beta)$ and $\mathrm{ftview}_v(\alpha) = \mathrm{ftview}_v(\beta)$ for every $v \neq u$ with $v \rightarrowtail u$, we have $\mathrm{obs}_u(s^I \cdot \alpha) = \mathrm{obs}_u(s^I \cdot \beta)$.*

Proof. From left to right. Suppose that the system is ito-secure. Let u be in D and define $V = \{v \in D \mid v \rightarrowtail u \text{ and } v \neq u\}$. We will prove by an induction on the combined length of α and β that if $\mathrm{tpurge}_u(\alpha) = \mathrm{tpurge}_u(\beta)$ and if for every $v \in V$ it holds $\mathrm{ftview}_v(\alpha) = \mathrm{ftview}_v(\beta)$, then $\mathrm{ito}_u(\alpha) = \mathrm{ito}_u(\beta)$. As induction hypothesis, we assume that this claim holds for all traces of smaller combined length. Since the claim clearly holds for the base case, we proceed with the inductive step. Let $\alpha = \alpha' a$ for some $a \in A$ and $\alpha' \in A^*$
Case 1: $\mathrm{dom}(a) \not\rightarrowtail u$.
Then we have

$$\mathrm{tpurge}_u(\alpha') = \mathrm{tpurge}_u(\alpha' a) = \mathrm{tpurge}_u(\beta) \ .$$

Since $\mathrm{dom}(a) \notin V$, we have for every $v \in V$:

$$\mathrm{ftview}_v(\alpha') = \mathrm{ftview}_v(\alpha' a) = \mathrm{ftview}_v(\beta) \ .$$

By applying the induction hypothesis, we get $\mathrm{ito}_u(\alpha') = \mathrm{ito}_u(\beta)$ and from the definition of the ito operator it follows $\mathrm{ito}_u(\alpha' a) = \mathrm{ito}_u(\alpha')$.
Case 2: $\mathrm{dom}(a) = u$.
We can assume that $\beta = \beta' b$ for some $\beta \in A^*$ and $b \in A$ with $\mathrm{dom}(b) \rightarrowtail u$, since otherwise we can proceed with Case 1. From

$$\mathrm{tpurge}_u(\alpha')\, a = \mathrm{tpurge}_u(\alpha' a) = \mathrm{tpurge}_u(\beta' b) = \mathrm{tpurge}_u(\beta')\, b$$

it follows $a = b$ and $\mathrm{tpurge}_u(\alpha') = \mathrm{tpurge}_u(\beta')$. Since $\mathrm{dom}(a) \notin V$, we have for every $v \in V$

$$\mathrm{ftview}_v(\alpha') = \mathrm{ftview}_v(\alpha' a) = \mathrm{ftview}_v(\beta' b) = \mathrm{ftview}_v(\beta') \ .$$

By induction hypothesis, we have $\mathrm{ito}_u(\alpha') = \mathrm{ito}_u(\beta')$ and by ito-security and Lemma 2.6.1, we have $\mathrm{view}_u(\alpha') = \mathrm{view}_u(\beta')$. Hence,

$$\begin{aligned} \mathrm{ito}_u(\alpha' a) &= (\mathrm{ito}_u(\alpha'), \mathrm{view}_u(\alpha'), a) \\ &= (\mathrm{ito}_u(\beta'), \mathrm{view}_u(\beta'), b) \\ &= \mathrm{ito}_u(\beta' b) \ . \end{aligned}$$

Case 3: $\mathrm{dom}(a) \rightarrowtail u$ and $\mathrm{dom}(a) \neq u$.
With the same argument as in the previous case, we have that $\beta = \beta' b$. Then for the tpurge values, we have

$$\mathrm{tpurge}_u(\alpha')\, a = \mathrm{tpurge}_u(\alpha' a) = \mathrm{tpurge}_u(\beta' b) = \mathrm{tpurge}_u(\beta')\, b \ .$$

This immediately gives $a = b$ and $\mathrm{tpurge}_u(\alpha') = \mathrm{tpurge}_u(\beta')$. For every $v \in V$ with $v \neq \mathrm{dom}(a)$, we have

$$\mathrm{ftview}_v(\alpha') = \mathrm{ftview}_v(\alpha' a) = \mathrm{ftview}_v(\beta' b) = \mathrm{ftview}_v(\beta') \ ,$$

and for the agent $\mathrm{dom}(a)$, the ftview value of these two traces are

$$\begin{aligned} \mathrm{view}_{\mathrm{dom}(a)}(\alpha')\, a\, \mathrm{obs}_{\mathrm{dom}(a)}(s^I \cdot \alpha' a) &= \mathrm{view}_{\mathrm{dom}(a)}(\alpha' a) \\ &= \mathrm{ftview}_{\mathrm{dom}(a)}(\alpha' a) \\ &= \mathrm{ftview}_{\mathrm{dom}(a)}(\beta' a) \\ &= \mathrm{view}_{\mathrm{dom}(a)}(\beta' a) \\ &= \mathrm{view}_{\mathrm{dom}(a)}(\beta')\, b\, \mathrm{obs}_{\mathrm{dom}(a)}(s^I \cdot \beta' a) \ . \end{aligned}$$

This shows both $\mathrm{view}_{\mathrm{dom}(a)}(\alpha') = \mathrm{view}_{\mathrm{dom}(a)}(\beta')$ and $\mathrm{view}_{\mathrm{dom}(a)}(\alpha' a) = \mathrm{view}_{\mathrm{dom}(a)}(\beta' b)$. By induction hypothesis, we have $\mathrm{ito}_u(\alpha') = \mathrm{ito}_u(\beta')$ and

hence,

$$\begin{aligned}\mathrm{ito}_u(\alpha' a) &= (\mathrm{ito}_u(\alpha'), \mathrm{view}_{\mathrm{dom}(a)}(\alpha' a), a) \\ &= (\mathrm{ito}_u(\beta'), \mathrm{view}_{\mathrm{dom}(a)}(\beta' b), b) \\ &= \mathrm{ito}_u(\beta' b) \ .\end{aligned}$$

For the other direction of the proof, as induction hypothesis, we assume that if $\mathrm{ito}_u(\alpha) = \mathrm{ito}_u(\beta)$, then $\mathrm{tpurge}_u(\alpha) = \mathrm{tpurge}_u(\beta)$ and for every $v \in V$ it holds $\mathrm{ftview}_v(\alpha) = \mathrm{ftview}_v(\beta)$.
Case 1: $\mathrm{dom}(a) \not\rightarrowtail u$.
We have

$$\mathrm{ito}_u(\beta) = \mathrm{ito}_u(\alpha' a) = \mathrm{ito}_u(\alpha') \ ,$$

and hence by induction hypothesis

$$\mathrm{tpurge}_u(\alpha' a) = \mathrm{tpurge}_u(\alpha') = \mathrm{tpurge}_u(\beta) \ .$$

Since $\mathrm{dom}(a) \notin V$, we have for every $v \in V$:

$$\mathrm{ftview}_v(\alpha' a) = \mathrm{ftview}_v(\alpha') = \mathrm{ftview}_v(\beta) \ .$$

Case 2: $\mathrm{dom}(a) \rightarrowtail u$.
With the same argument as in the other direction of the proof, we can suppose $\beta = \beta' b$ with $\mathrm{dom}(b) \rightarrowtail u$. From $\mathrm{ito}_u(\alpha' a) = \mathrm{ito}_u(\beta' b)$ it follows $a = b$ and $\mathrm{ito}_u(\alpha') = \mathrm{ito}_u(\beta')$. It also follows

$$\mathrm{view}_{\mathrm{dom}(a)}(\alpha') = \mathrm{view}_{\mathrm{dom}(a)}(\beta') \quad \text{if } \mathrm{dom}(a) = u \ ,$$

and

$$\mathrm{view}_{\mathrm{dom}(a)}(\alpha' a) = \mathrm{view}_{\mathrm{dom}(a)}(\beta' a) \quad \text{if } \mathrm{dom}(a) \rightarrowtail u \text{ and } \mathrm{dom}(a) \neq u \ .$$

Hence, by applying the induction hypothesis, we have

$$\text{tpurge}_u(\alpha' a) = \text{tpurge}_u(\alpha')\ a = \text{tpurge}_u(\beta')\ b = \text{tpurge}_u(\beta' b)\ .$$

For every $v \in V$ with $v \neq \text{dom}(a)$, we have

$$\text{ftview}_v(\alpha' a) = \text{ftview}_v(\alpha') = \text{ftview}_v(\beta') = \text{ftview}_v(\beta' b)\ ,$$

and for the agent $\text{dom}(a)$, we have

$$\begin{aligned} \text{ftview}_{\text{dom}(a)}(\alpha' a) &= \text{view}_{\text{dom}(a)}(\alpha' a) \\ &= \text{view}_{\text{dom}(a)}(\beta' b) \\ &= \text{ftview}_{\text{dom}(a)}(\beta' b)\ . \end{aligned}$$

□

3.7 Relations between the Definitions for Static Noninterference

Van der Meyden has already worked out the relations between these noninterference definitions in [vdM07]. We will just recall them for providing an outline. All the security definitions of this chapter are in a linear order and all of these implications are strict. However, we do not recall the separating examples here.

$$\text{t-security} \Rightarrow \text{to-security} \Rightarrow \text{ito-security} \Rightarrow \text{ta-security} \Rightarrow \text{i-security}\ .$$

Chapter 4

Dynamic Noninterference

In this section, we introduce and discuss the notion of dynamic noninterference. This includes an extension of transitive noninterference and two different extensions of intransitive noninterference to systems with dynamic policies.

After an introduction to the general idea of dynamic noninterference in Section 4.1, we introduce our first dynamic noninterference definition dt-security, an adaption of t-security to the dynamic setting, in Section 4.2. Before presenting the idea of dynamic intransitive noninterference in Section 4.4, we introduce the intermediate notion of dot-secure in Section 4.3. Our security definitions for intransitive noninterference are di-security, a generalization of i-security, in Section 4.5, and dta-security, an adaption of ta-security, in Section 4.6. In Section 4.7, we compare all these security definitions with each other and in Section 4.8, we explain how our definitions relate to Leslie's definition [Les06] of dynamic intransitive noninterference.

Several results on dt-security and di-security are published in [ESW13] and those on dot-security can be found in [ESW12].

4.1 Introduction to Dynamic Noninterference

Dynamic noninterference applies to systems with dynamic polices: systems where the policy may change during the run of the system. This is a natural generalization of the noninterference definitions of the previous chapter. The requirement for dynamic policies appears in many system and is typical in discretionary access control systems. In security specifications where rights are passed by the agents, or where the rights of an agent changes during a run, dynamic policies are necessary. An example of the latter is a

program that has more rights if the system is connected to a secure network, but limited rights if it is connected to an unprotected network.

However, changes of policy may lead to new covert channels in the case that policy changes can be used to communicate information in an unintended way. In our formalization, we generally do not restrict how the policy can be changed in advance or who may perform the changes. Instead, as a result, we find out which changes contradict other security requirements and are forbidden in order to avoid unauthorized information leakage. These new kinds of covert channels cannot be treated in isolation, since they may appear in combination with possible covert channels analyzed in the previous chapter.

As we have already seen in the analysis of noninterference with static policies, high[1] agents are generally not allowed to visibly change the system's state.

To formalize dynamic policies, we just define a (local) policy for every state of the system. Then a family of local policies $(\rightarrowtail_s)_{s \in S}$ in combination with the dynamics of the system reflects the dynamic changes of the policy. For any state s, we write $u \rightarrowtail_s v$ if there is an edge from u to v in the policy which applies to the state s. We say that an action a changes the policy if the policy in the state $s \cdot a$ is different from the policy in the state s. In particular, if we have $u \rightarrowtail_s v$ and $u \not\rightarrowtail_{s \cdot a} v$, then we say that the action a has removed the edge from u to v. The intuition of a local policy in a state s is that it specifies whose observations are allowed to be influenced by an action, performed in the state s.

However, the question is what happens if a high action changes the part of the policy that relates to low, given that high is not allowed to interfere with low. This can lead to security problems if the policy changes have visible effects to low. Hence, a general question is which policy should be applied. When an action h has been performed in a state s where H is not allowed to have any influence on any other agent, should we apply the policy of state s or the one of state $s \cdot h$? The simplest answer is always to take the current state of the system. However, if the policy differs in

[1] With "high", we mean an agent that is not allowed to interfere with some fixed observing agent, or in the intransitive setting, an agent whose actions are not downgraded to the observing agent.

the state s and $s \cdot h$ and we take the policy of the state $s \cdot h$, then some information about performing the action h is encoded in the policy, which might be revealed later on.

However, in systems with a dynamic policy, the question of what is a high action is more subtle than in the static case, since if in some states, there is an edge from high to low and in some other state, there is no edge, then in general one cannot say that a particular agent is high.

This discussion should be more clear with the next example.

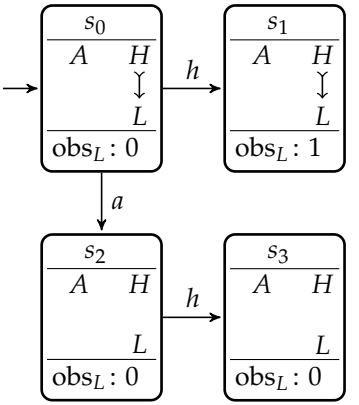

Figure 4.1. An insecure system with a dynamic policy

4.1.1 Example. In systems with a dynamic policy, the local policy that applies to a particular state is depicted within the state.

We consider the system in Figure 4.1. For example, the local policy in state s_0 allows information flow from H to L. If H performs an action h in the state s_0, L is possibly allowed to observe it. We will provide a precise security definition and semantics later. However, we want to argue here, on an intuitive level why this system should be rejected as insecure.

The local policy in state s_0 might suggest that the change of L's observation is allowed, since it is done by H's action and H is allowed to interfere with L. In the case where H is not allowed to interfere with L, namely in state s_2, H's action has no visible effect to L.

However, the agent A is never allowed to interfere with any of the agents. Therefore, A's actions should not change the observations of any other agent. But in this system, this is not the case, since if A performs an action in the initial state, afterwards, H cannot change L's observations, but if L observes the value 1, it knows that A has not performed an action before H's first action. In this case, L gains information about A's actions, which is forbidden by every local policy.

Another argument why we should treat this system as insecure is the following: For any reasonable security definition, it should hold that if a system is insecure, then it remains insecure with respect to a more restrictive policy. Conversely, we consider this system with a less restrictive policy where in all states H is allowed to interfere with L, but A is still not allowed to interfere with any other agent. Since we now have the same policy in all states, we can apply the noninterference definitions of the previous chapter. However, according to any noninterference definition of the last chapter, the system is insecure. Back in the original system, by removing the previously added edges, the system should now remain insecure by the assumption of the monotony as mentioned before.

According to both arguments, any reasonable noninterference definition for systems with dynamic policies should treat this system as insecure.

An intransitive interpretation of information flow in dynamic policies leads to new kinds of downgrading effects. Due to the fact that information flow from one agent is allowed in some states and may be not allowed in some other states, actions performed in states where the information flow is not allowed may be downgraded later in states where the information flow is allowed. Hence downgrading information has a temporal component additionally to the property of downgrading via agents as in the previous chapter. Similarly to the previous chapter, we will present different definitions for noninterference depending on if downgrading is a desired property. For dynamic intransitive noninterference, we will provide different incomparable definitions which reflect that different reasonable generalizations of intransitive noninterference to a dynamic setting are possible.

4.2 dt-security

In this section, we introduce the first of our noninterference definitions for systems with a dynamic policy. We call this notion dynamic transitive noninterference, since it is a direct adaption of the notion of transitive noninterference to systems with a dynamic policy. The idea of this very restrictive security definition is that if the local policy in the state s forbids the interference of an agent $\mathrm{dom}(a)$ with the agent u, then the agent u is neither allowed to observe nor to deduce that this action has been performed when the system was in the state s.

This can be directly formalized as:

4.2.1 Definition (dt-security)**.** A system is *dt-secure* iff for all $u \in D$, $s \in S$, $a \in A$, and $\alpha \in A^*$, the following implication holds:

$$\text{If } \mathrm{dom}(a) \not\rightarrowtail_s u, \text{then } \mathrm{obs}_u(s \cdot a\alpha) = \mathrm{obs}_u(s \cdot \alpha) \ .$$

4.2.2 Example. We continue with the system in Figure 4.1 of Example 4.1.1. We have $\mathrm{dom}(a) \not\rightarrowtail_{s_0} L$, but $\mathrm{obs}_L(s_0 \cdot ah) \neq \mathrm{obs}_L(s_0 \cdot h)$ and hence, the system is not dt-secure.

Like in the case of a static policy, it is possible to formulate dt-security in terms of a purge-like operator. Obviously, since the policy is state-dependent, a purge operator has to keep track of the state from where the policy should be taken. This state can differ from the state where the system is in the actual run. When an action is purged, this action must not influence the observing agent. This means that the run of the system should be indistinguishable to the run where this action has not been performed. In this case, no transition would be performed and the system would remain in the same state. This is adopted by the following purge operator, by *not* performing the transition if the corresponding action is purged.

4.2.3 Definition (dtpurge operator)**.** For every $u \in D$, $s \in S$, $a \in A$, and $\alpha \in A^*$ define

$$\mathrm{dtpurge}(\epsilon, u, s) = \epsilon$$

$$\text{dtpurge}(a\alpha, u, s) = \begin{cases} a\ \text{dtpurge}(\alpha, u, s \cdot a) & \text{if } \text{dom}(a) \rightarrowtail_s u \\ \text{dtpurge}(\alpha, u, s) & \text{otherwise .} \end{cases}$$

Next, we summarize some properties of the dtpurge operator to show that it behaves similarly to the tpurge operator.

4.2.4 Lemma. *Let $u \in D$, $\alpha, \beta \in A^*$, and $s \in S$. Then we have*

1. $\text{dtpurge}(\text{dtpurge}(\alpha, u, s), u, s) = \text{dtpurge}(\alpha, u, s)$, *and*
2. $\text{dtpurge}(\alpha\beta, u, s) = \text{dtpurge}(\alpha, u, s)\ \text{dtpurge}(\beta, u, s \cdot \text{dtpurge}(\alpha, u, s))$.

Proof. 1. We show this claim by an induction on the length of α. Since it holds trivially for the base case $\alpha = \epsilon$, we proceed with the inductive step. Consider $a\alpha$ and assume that the claim holds for α. First, suppose $\text{dom}(a) \not\rightarrowtail_s u$. Then we have

$$\begin{aligned} \text{dtpurge}(\text{dtpurge}(a\alpha, u, s), u, s) &= \text{dtpurge}(\text{dtpurge}(\alpha, u, s), u, s) \\ &= \text{dtpurge}(\alpha, u, s) \ . \end{aligned}$$

Second, suppose $\text{dom}(a) \rightarrowtail_s u$. In this case, we have

$$\begin{aligned} \text{dtpurge}(\text{dtpurge}(a\alpha, u, s), u, s) &= \text{dtpurge}(a\ \text{dtpurge}(\alpha, u, s \cdot \alpha), u, s) \\ &= a\ \text{dtpurge}(\text{dtpurge}(\alpha, u, s \cdot \alpha), u, s \cdot a) \\ &= a\ \text{dtpurge}(\alpha, u, s \cdot a) \\ &= \text{dtpurge}(a\alpha, u, s \cdot a) \ . \end{aligned}$$

2. We show this claim by an induction on the length of α, too. Since the base case clearly holds for $\alpha = \beta = \epsilon$, we proceed with the inductive step. Again, consider a trace $a\alpha$ and assume that the claim holds for α. In the case of $\text{dom}(a) \not\rightarrowtail_s u$, we obtain

$$\begin{aligned} &\text{dtpurge}(a\alpha\beta, u, s) \\ &\qquad = \text{dtpurge}(\alpha\beta, u, s) \\ &\qquad = \text{dtpurge}(\alpha, u, s)\ \text{dtpurge}(\beta, u, s \cdot \text{dtpurge}(\alpha, u, s)) \\ &\qquad = \text{dtpurge}(a\alpha, u, s)\ \text{dtpurge}(\beta, u, s \cdot \text{dtpurge}(a\alpha, u, s)) \ . \end{aligned}$$

In the case of $\mathrm{dom}(a) \rightarrowtail_s u$, we have

$$\begin{aligned}
&\mathrm{dtpurge}(a\alpha\beta, u, s) \\
&\qquad = a\ \mathrm{dtpurge}(\alpha\beta, u, s \cdot a) \\
&\qquad = a\ \mathrm{dtpurge}(\alpha, u, s \cdot a)\ \mathrm{dtpurge}(\beta, u, s \cdot a\ \mathrm{dtpurge}(\alpha, u, s \cdot a)) \\
&\qquad = \mathrm{dtpurge}(a\alpha, u, s)\ \mathrm{dtpurge}(\beta, u, s \cdot \mathrm{dtpurge}(a\alpha, u, s))\ . \qquad \square
\end{aligned}$$

The first property shows that the dtpurge operator is idempotent: If it is applied to an already purged trace, it has no effect. The second property states that traces can be split into two parts and each trace can be purged on its own if the purging of the second part starts in the state where the purged value of the first part ends.

Like in the static case, if two traces have the same purge value, it is required that they lead to the same observation. However, in the dynamic case, it is required to start purging from every possible state.

4.2.5 Theorem. *A system is dt-secure iff for every $u \in D$, $s \in S$, and $\alpha, \beta \in A^*$ with* $\mathrm{dtpurge}(\alpha, u, s) = \mathrm{dtpurge}(\beta, u, s)$, *we have* $\mathrm{obs}_u(s \cdot \alpha) = \mathrm{obs}_u(s \cdot \beta)$.

Proof. First, we prove this theorem from left to right by contraposition. Hence, there are $u \in D$, $s \in S$, and $\alpha, \beta \in A^*$ with

$$\mathrm{dtpurge}(\alpha, u, s) = \mathrm{dtpurge}(\beta, u, s)\ ,$$

and

$$\mathrm{obs}_u(s \cdot \alpha) \neq \mathrm{obs}_u(s \cdot \beta)\ .$$

Additionally, assume that the combined length of α and β is minimal for all possible choices of u and s with this property. At least one of α and β is not the empty trace, hence, w.l.o.g., we can assume that $\alpha \neq \epsilon$. Therefore, $\alpha = a\alpha'$ for some $a \in A$ and $\alpha' \in A^*$.
Case 1: $\mathrm{dom}(a) \not\rightarrowtail_s u$.
In this case, we have $\mathrm{dtpurge}(a\alpha', u, s) = \mathrm{dtpurge}(\alpha', u, s)$. If $\mathrm{obs}_u(s \cdot a\alpha') = \mathrm{obs}_u(s \cdot \alpha')$, then we could take α' instead of α, which contradicts the assumption about the minimality. Hence, we have $\mathrm{dom}(a) \not\rightarrowtail_s u$ and

$\text{obs}_u(s \cdot a\alpha) \neq \text{obs}_u(s \cdot \alpha)$, which shows that the system is not dt-secure.
Case 2: $\text{dom}(a) \rightarrowtail_s u$.
In this case, we have $\text{dtpurge}(a\alpha', u, s) = a\ \text{dtpurge}(\alpha', u, s \cdot a)$ and we can assume that $\beta = b\beta'$ for some $b \in A$, $\beta \in A^*$ with $\text{dom}(b) \rightarrowtail_s u$, since otherwise, we would proceed with Case 1 with the roles of α and β swapped. But in this case, we have

$$\text{dtpurge}(\alpha', u, s \cdot b) = \text{dtpurge}(\beta', u, s \cdot b) \ ,$$

and still

$$\text{obs}_u(s \cdot a\alpha') \neq \text{obs}_u(s \cdot b\beta') \ .$$

Again, this contradicts the minimality of the combined length of α and β. Therefore, this case is not possible.

For the other direction of the proof, suppose that the system is *not* dt-secure. Therefore, there are $u \in D$, $a \in A$, $s \in S$, $\alpha \in A^*$ with $\text{dom}(a) \not\rightarrowtail_s u$ and $\text{obs}_u(s \cdot a\alpha) \neq \text{obs}_u(s \cdot \alpha)$. Hence, we have

$$\text{dtpurge}(a\alpha, u, s) = \text{dtpurge}(\alpha, u, s) \ ,$$

and

$$\text{obs}_u(s \cdot a\alpha) \neq \text{obs}_u(s \cdot \alpha) \ . \qquad \square$$

As we saw in the previous theorem, for t-security, it is necessary to start purging from every single state. That it is not sufficient to start in the initial state only is shown with the next example.

4.2.6 Example. The system in Figure 4.2 is a *not* dt-secure system, since $\text{dom}(h) \not\rightarrowtail_{s_1} L$, but $\text{obs}_L(s_1) = 0 \neq 1 = \text{obs}_L(s_1 \cdot h) = \text{obs}_L(s_2)$. However, if we only required to start purging from the initial state, then we would have for the values of the tpurge function applied to the traces ah and h:

$$\text{dtpurge}(ah, L, s_0) = h = \text{dtpurge}(h, L, s_0) \ ,$$

but also $\text{obs}_L(s_0 \cdot ah) = 1 = \text{obs}_L(s_0 \cdot h)$. Also the traces a and ϵ have the

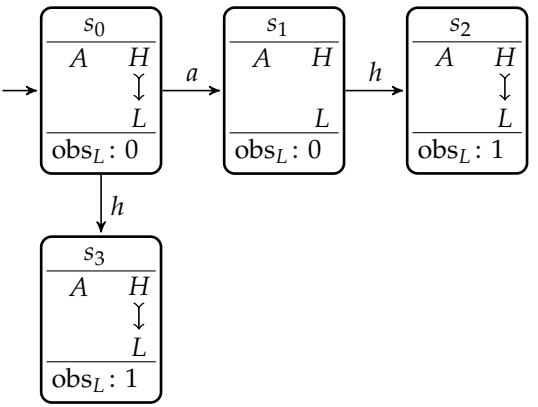

Figure 4.2. A *not* dt-secure system, but purging form the initial state only is not sufficient

same dtpurge value and lead to the same observations. Hence the system would be secure. We see that purging only from the initial state leads to a strictly weaker security definition, which is too weak in our understanding.

4.2.1 Unwinding for dt-security

Similar to the static case, state-based unwinding relations can be defined for dt-security. The only difference to the state-based transitive unwinding of Definition 3.2.9 for systems with a static policy is that the ($\mathrm{LR}^{\mathsf{sdt}}$) condition depends on the state where it is applied.

4.2.7 Definition (state-based dynamic transitive unwinding)**.** A *state-based dynamic transitive unwinding relation* for $u \in D$ is an equivalence relation $\sim_u^{\mathsf{sdt}} \subseteq S \times S$ such that for every $a \in A$ and every $s, t \in S$, the following conditions are satisfied:

($\mathrm{LR}^{\mathsf{sdt}}$)*:* If $\mathrm{dom}(a) \not\rightarrow_s u$, then $s \sim_u^{\mathsf{sdt}} s \cdot a$.

($\mathrm{SC}^{\mathsf{sdt}}$)*:* If $s \sim_u^{\mathsf{sdt}} t$, then $s \cdot a \sim_u^{\mathsf{sdt}} t \cdot a$.

The soundness and completeness of these unwinding relations for dt-security is shown in the next theorem.

4.2.8 Theorem. *A system is dt-secure iff for every $u \in D$, there exists a state-based dynamic transitive unwinding relation $\sim_u^{\mathsf{sdt}}$ that is observation consistent for u.*

Proof. For the proof of the implication from left to right, suppose that the system is dt-secure. Let $u \in D$. Define a binary relation on the set of states for every $s, t \in S$ by

$$s \sim_u^{\mathsf{sdt}} t \text{ iff for all } \alpha \in A^* \text{ it holds: } \mathrm{obs}_u(s \cdot \alpha) = \mathrm{obs}_u(t \cdot \alpha) \ .$$

Clearly, the relation $\sim_u^{\mathsf{sdt}}$ is an equivalence relation. By taking $\alpha = \epsilon$, this relation is observation consistent for u.

For showing ($\mathrm{LR}^{\mathsf{sdt}}$), let $s \in S$ and $a \in A$ with $\mathrm{dom}(a) \not\rightarrow_s u$. Since the system is dt-secure, we have for every $\alpha \in A^*$: $\mathrm{obs}_u(s \cdot a\alpha) = \mathrm{obs}_u(s \cdot \alpha)$. Hence, we have $s \sim_u^{\mathsf{sdt}} s \cdot a$.

For showing ($\mathrm{SC}^{\mathsf{sdt}}$), let $s, t \in S$ with $s \sim_u^{\mathsf{sdt}} t$ and let $a \in A$. Since we have for every $\alpha \in A^*$: $\mathrm{obs}_u(s \cdot \alpha) = \mathrm{obs}_u(t \cdot \alpha)$, this also holds for the states $s \cdot a$ and $t \cdot a$.

For the other direction of the proof, let $u \in D$ and suppose that there is a state-based dynamic transitive unwinding relation that is observation consistent for u. Let $\sim_u^{\mathsf{sdt}}$ be such a smallest unwinding relation. By an induction on the combined length of α and β, we will show that for every $s \in S$ the implication

$$\text{if } \mathrm{dtpurge}(\alpha, u, s) = \mathrm{dtpurge}(\beta, u, s), \text{ then } s \cdot \alpha \sim_u^{\mathsf{sdt}} s \cdot \beta$$

holds.

The base case $\alpha = \beta = \epsilon$ is obvious.

For the inductive step, let $\alpha = a\alpha'$ for some $a \in A$ and $\alpha' \in A^*$ and suppose that $\mathrm{dtpurge}(\alpha, u, s) = \mathrm{dtpurge}(\beta, u, s)$ holds.
Case 1: $\mathrm{dom}(a) \not\rightarrow_s u$.
In this case, we have

$$\mathrm{dtpurge}(\beta, u, s) = \mathrm{dtpurge}(a\alpha', u, s) = \mathrm{dtpurge}(\alpha', u, s) \ .$$

By applying the induction hypotheses, we have $s \cdot \alpha' \sim_u^{\mathsf{sdt}} s \cdot \beta$. From the property ($\mathrm{LR}^{\mathsf{sdt}}$) it follows that $s \sim_u^{\mathsf{sdt}} s \cdot a$ and by applying ($\mathrm{SC}^{\mathsf{sdt}}$), we obtain $s \cdot \alpha' \sim_u^{\mathsf{sdt}} s \cdot a\alpha'$. By symmetry and transitivity of $\sim_u^{\mathsf{sdt}}$, we have $s \cdot a\alpha' \sim_u^{\mathsf{sdt}} s \cdot \beta$.
Case 2: $\mathrm{dom}(a) \not\rightarrowtail_s u$.
Let $\beta = b\beta'$ for some $b \in A$ and $\beta \in A^*$. If $\mathrm{dom}(b) \not\rightarrowtail_s u$, we may apply Case 1 with the roles of α and β swapped. Hence, assume that $\mathrm{dom}(b) \rightarrowtail_u u$ holds and therefore, $\mathrm{dtpurge}(\alpha', u, s \cdot a) = \mathrm{dtpurge}(\beta', u, s \cdot a)$ and $a = b$. Applying the induction hypothesis to the traces α' and β' and the state $s \cdot a$, we obtain $s \cdot a\alpha' \sim_u^{\mathsf{sdt}} s \cdot b\beta'$.

Now, the desired claim follows from the fact that $\sim_u^{\mathsf{sdt}}$ is observation consistent for u. □

As we will see in Chapter 5, the definition of the unwinding relations combined with this theorem provides both a useful proof technique and an efficient verification procedure.

For a better comparison to the other noninterference definitions, we provide a definition of a trace-based unwinding for dt-security. The unwinding conditions of the trace-based unwinding for dynamic transitive noninterference are similar to those of the trace-based unwinding for t-security in Definition 3.2.5.

4.2.9 Definition (trace-based dynamic transitive unwinding)**.** A *trace-based dynamic transitive unwinding relation* for $u \in D$ is an equivalence relation $\sim_u^{\mathsf{tdt}} \subseteq A^* \times A^*$ such that for every $a \in A$ and every $\alpha, \beta \in A^*$, the following conditions are satisfied:

($\mathrm{LR}^{\mathsf{tdt}}$)*:* If $\mathrm{dom}(a) \not\rightarrowtail_{s^I \cdot \alpha} u$, then $\alpha \sim_u^{\mathsf{tdt}} \alpha \cdot a$.

($\mathrm{SC}^{\mathsf{tdt}}$)*:* If $\alpha \sim_u^{\mathsf{tdt}} \beta$, then $\alpha a \sim_u^{\mathsf{tdt}} \beta a$.

The soundness and completeness of this trace-based unwinding is established by:

4.2.10 Theorem. *A system is dt-secure iff for every $u \in D$ there exists a trace-based transitive unwinding relation $\sim_u^{\mathsf{tdt}}$ that is observation consistent for u.*

Proof. From left to right. Let $u \in D$, $s \in S$, and $a \in A$ such that $\mathrm{dom}(a) \not\rightarrow_s u$. Let $\alpha \in A^*$. Then there is some trace $\beta \in A^*$ with $s = s^I \cdot \beta$, and by ($\mathrm{LR}^{\mathsf{tdt}}$), we have $\beta \sim_u^{\mathsf{tdt}} \beta a$, and by ($\mathrm{SC}^{\mathsf{tdt}}$), we have $\beta\alpha \sim_u^{\mathsf{tdt}} \beta a\alpha$. By observation consistency for u, we have $\mathrm{obs}_u(s^I \cdot \beta\alpha) = \mathrm{obs}_u(s^I \cdot \beta a\alpha)$.

From right to left by contraposition. Suppose that the system is not dt-secure. Then there are $u \in D$, $s \in S$, $a \in A$, and $\alpha \in A^*$ with $\mathrm{dom}(a) \not\rightarrow_s u$ and $\mathrm{obs}_u(s \cdot a\alpha) \neq \mathrm{obs}_u(s \cdot \alpha)$. For any equivalence relation satisfying ($\mathrm{LR}^{\mathsf{tdt}}$) and ($\mathrm{SC}^{\mathsf{tdt}}$), it holds $\beta\alpha \sim_u^{\mathsf{tdt}} \beta a\alpha$ for every $\beta \in A^*$ with $s = s^I \cdot \beta$. Then we have that $\sim_u^{\mathsf{tdt}}$ is not observation consistent for u. □

4.2.2 dt-useless Edges and dt-uniformity

As we have seen, some edges in a local policy cannot be used, since they contradict some others edges. In these cases, such contradicting edges can be removed from the local policies without affecting security. We call these edges dt-useless.

Since some edges in a local policy cannot be used, an edge $v \rightarrowtail_s u$ should be interpreted as follows: The information flow from v to u is not explicitly forbidden in the state s. However, there might be some reasons why such an information flow is forbidden anyway. Conversely, if $v \not\rightarrow_s u$ appears in the local policy of state s, the information flow from v to u is explicitly forbidden in this state.

Intuitively, the incoming edges of an agent are only allowed to be changed by agents that are allowed to interfere with the agent. Additionally, in indistinguishable states, the policy changes have to be the same.

4.2.11 Example. Again, we continue with the system in Figure 4.1. One can say that the action a performed in the initial state removes the edge from H to L. However, since A is not allowed to interfere with L, this change is "not allowed" in the sense that only in one of the two for L indistinguishable states interference from H to L is allowed, and in the other it is forbidden. This is some kind of contradiction between these two local policies, and we always obey the more restrictive policy. Hence, the edge from H to L in the initial state is useless and can safely be removed without affecting security.

However, if the policy changes are only performed by agents that are allowed to interfere, the system is secure and no useless edges exist.

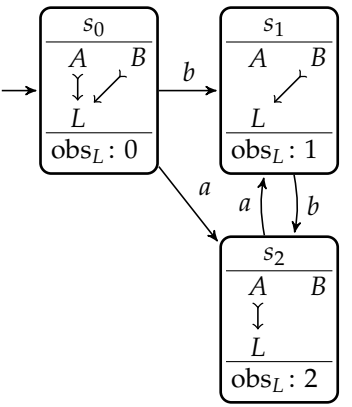

Figure 4.3. A dt-secure system without useless edges

4.2.12 Example. In the system in Figure 4.3, the incoming edges of L are only changed by actions of those agents that are allowed to interfere. The system is dt-secure and in this system there are no useless edges.

Before formalizing useless edges, we define an equivalence relation on the states of the system.

4.2.13 Definition (dt-similarity)**.** States s and t are *dt-similar* for an agent $u \in D$, denoted by $s \approx_u^{\mathsf{dt}} t$, if there exist $\tilde{s} \in S$, $a \in A$, and $\alpha \in A^*$ such that $\text{dom}(a) \not\rightarrowtail_{\tilde{s}} u$, $s = \tilde{s} \cdot a\alpha$, and $t = \tilde{s} \cdot \alpha$.

Clearly, two states are dt-similar for an agent u if u is required to have the same observations in these two states according to the dt-security definition. One can also observe that $\approx_u^{\mathsf{dt}}$ is the smallest state-based dynamic transitive unwinding relation for agent u. However, we define this relation explicitly to have a clearer analogy to dynamic intransitive noninterference later in this chapter. We use this equivalence relation to define useless edges. An edge of a local policy is said to be dt-useless if in an equivalent state, this edge does not exist.

4.2.14 Definition (dt-useless edge)**.** An edge $v \rightarrowtail_s u$ is *dt-useless* if there is a state t with $s \approx_u^{\mathsf{dt}} t$ and $v \not\rightarrowtail_t u$.

The next theorem shows that dt-useless edges are indeed useless in the sense that their removal does not affect security.

4.2.15 Theorem. *Let $(\rightarrowtail_s)_{s\in S}$ be the dynamic policy of some given system. Define the dynamic policy $(\rightarrowtail'_s)_{s\in S}$ for every $s \in S$ by*

$$\rightarrowtail'_s = \rightarrowtail_s \setminus \{v \rightarrowtail_s u \mid v \rightarrowtail_s u \text{ is dt-useless}\} \ .$$

Then the system is dt-secure w.r.t. $(\rightarrowtail_s)_{s\in S}$ iff it is dt-secure w.r.t. $(\rightarrowtail'_s)_{s\in S}$.

Proof. For the proof from left to right, assume that the system is dt-secure w.r.t. $(\rightarrowtail_s)_{s\in S}$. Then there is an observation consistent state-based dynamic transitive unwinding relation for u. Let $\sim_u^{\mathsf{sdt}}$ be such a smallest unwinding relation. Let $\sim_u^{\mathsf{sdt}'}$ be the smallest state-based dynamic transitive unwinding for u w.r.t. the policy $(\rightarrowtail'_s)_{s\in S}$. We will show that $\sim_u^{\mathsf{sdt}'} \subseteq \sim_u^{\mathsf{sdt}}$. Let $s, t \in S$ with $s \sim_u^{\mathsf{sdt}'} t$ and $t = s \cdot a$ for some $a \in A$ with $\mathrm{dom}(a) \not\rightarrowtail'_s u$. Then there is some $s' \in S$ with $s' \sim_u^{\mathsf{sdt}} s$ and $\mathrm{dom}(a) \not\rightarrowtail_{s'} u$. From $s' \sim_u^{\mathsf{sdt}} s \cdot a$ and $s' \cdot a \sim_u^{\mathsf{sdt}} s \cdot a$ it follows $s \sim_u^{\mathsf{sdt}} t$.

The other direction of the equivalence follows from the fact that the dynamic policy $(\rightarrowtail'_s)_{s\in S}$ is at least as restrictive as the dynamic policy $(\rightarrowtail_s)_{s\in S}$. □

The dynamic policy $(\rightarrowtail'_s)_{s\in S}$, which results from removing all dt-useless edges, has the property that all edges in the local policies represent allowed information flows. That means that no edge in a local policy contradicts any other edge of another local policy. Hence, the information flow in a state is allowed if and only if the local policy in that state allows it.

Since on all dt-similar states, every agent has the same incoming edges in the local policies, the agent knows (better say, is allowed to know) its incoming edges. We call such dynamic policies dt-uniform. Recall that $u_s^{\leftarrowtail}$ denotes the set of all agents v with $v \rightarrowtail_s u$.

4.2.16 Definition (dt-uniformity)**.** A dynamic policy $(\rightarrowtail_s)_{s\in S}$ is *dt-uniform* iff for every $u \in D$ and every $s, t \in S$ with $s \approx_u^{\mathsf{dt}} t$, we have $u_s^{\leftarrowtail} = u_t^{\leftarrowtail}$.

Note that a dynamic policy is dt-uniform if and only if it does not contain any dt-useless edges.

In systems with a dt-uniform dynamic policy, the dtpurge operator behaves very naturally in the combination with a state-based dynamic transitive unwinding.

4.2.17 Lemma. *Consider a system with a dt-uniform dynamic policy. Let $u \in D$ and let $\sim_u^{\mathsf{sdt}}$ be the smallest state-based dynamic transitive unwinding relation. Then we have for every $s, t \in S$ with $s \sim_u^{\mathsf{sdt}} t$ and every $\alpha \in A^*$:*

$$s \cdot \alpha \sim_u^{\mathsf{sdt}} t \cdot \mathrm{dtpurge}(\alpha, u, t) \text{ and } \mathrm{dtpurge}(\alpha, u, s) = \mathrm{dtpurge}(\alpha, u, t) \ .$$

Proof. We prove this claim by an induction on the length of α. It clearly holds for the base case with $\alpha = \epsilon$. Consider a trace $a\alpha$ and assume that the claim holds for α and all possible states. First, assume that $\mathrm{dom}(a) \not\rightarrowtail_s u$. Since the dynamic policy is dt-uniform, we have $\mathrm{dom}(a) \not\rightarrowtail_t u$. In this case we have

$$\begin{aligned} \mathrm{dtpurge}(a\alpha, u, s) &= \mathrm{dtpurge}(\alpha, u, s) \\ &= \mathrm{dtpurge}(\alpha, u, t) \\ &= \mathrm{dtpurge}(a\alpha, u, t) \ , \end{aligned}$$

and

$$\begin{aligned} s \cdot a\alpha &\sim_u^{\mathsf{sdt}} s \cdot \alpha \\ &\sim_u^{\mathsf{sdt}} t \cdot \mathrm{dtpurge}(\alpha, u, t) \\ &\sim_u^{\mathsf{sdt}} t \cdot \mathrm{dtpurge}(a\alpha, u, t) \ . \end{aligned}$$

Second, assume that $\mathrm{dom}(a) \rightarrowtail_s u$. Due to a dt-uniform dynamic policy, we have $\mathrm{dom}(a) \rightarrowtail_t u$. This gives

$$\begin{aligned} \mathrm{dtpurge}(a\alpha, u, s) &= a\ \mathrm{dtpurge}(\alpha, u, s \cdot a) \\ &= a\ \mathrm{dtpurge}(\alpha, u, t \cdot a) \\ &= \mathrm{dtpurge}(a\alpha, u, t) \ , \end{aligned}$$

and

$$\begin{aligned} s \cdot a\alpha &\sim_u^{\mathsf{sdt}} t \cdot a\alpha \\ &\sim_u^{\mathsf{sdt}} t \cdot a \ \mathrm{dtpurge}(\alpha, u, t \cdot a) \\ &\sim_u^{\mathsf{sdt}} t \cdot \mathrm{dtpurge}(a\alpha, u, t) \ . \end{aligned}$$ □

Since in dt-uniform policies no contradicting edges exist, for purging, it is sufficient to start in the initial state instead of starting from every state.

4.2.18 Theorem. *A system with a dt-uniform dynamic policy is dt-secure iff for every $u \in D$ and all $\alpha, \beta \in A^*$ with $\mathrm{dtpurge}(\alpha, u, s^I) = \mathrm{dtpurge}(\beta, u, s^I)$, we have $\mathrm{obs}_u(s^I \cdot \alpha) = \mathrm{obs}_u(s^I \cdot \beta)$.*

Proof. For the proof of the claim from left to right, assume that the system is dt-secure and by Theorem 4.2.8, for every agent u, there is a state-based dynamic transitive unwinding relation $\sim_u^{\mathsf{sdt}}$ that is observation consistent for u. Let $\alpha, \beta \in A^*$ with $\mathrm{dtpurge}(\alpha, u, s^I) = \mathrm{dtpurge}(\beta, u, s^I)$. Since the system has a dt-uniform policy, by Lemma 4.2.17, we have $s^I \cdot \alpha \sim_u^{\mathsf{sdt}} s^I \cdot \mathrm{dtpurge}(\alpha, u, s^I)$ and $s^I \cdot \beta \sim_u^{\mathsf{sdt}} s^I \cdot \mathrm{dtpurge}(\beta, u, s^I)$. From observation consistency for u it follows

$$\begin{aligned} \mathrm{obs}_u(s^I \cdot \alpha) &= \mathrm{obs}_u(s^I \cdot \mathrm{dtpurge}(\alpha, u, s^I)) \\ &= \mathrm{obs}_u(s^I \cdot \mathrm{dtpurge}(\beta, u, s^I)) \\ &= \mathrm{obs}_u(s^I \cdot \beta) \ . \end{aligned}$$

For the other direction of the proof, assume that the right-hand side of the claim holds. Let $u \in D$, $s \in S$, and $a \in A$ with $\mathrm{dom}(a) \not\rightarrowtriangle_s u$. Then there is some $\gamma \in A^*$ with $s = s^I \cdot \gamma$. Let α be in A^*. By Lemma 4.2.4, we have $\gamma \sim_u^{\mathsf{sdt}} \mathrm{dtpurge}(\gamma, u, s^I)$ and hence, $\mathrm{dom}(a) \not\rightarrowtriangle_{s^I \cdot \mathrm{dtpurge}(\gamma, u, s^I)} u$. With the properties of dtpurge from Lemma 4.2.4, the dtpurge values are:

$\mathrm{dtpurge}(\gamma a\alpha, u, s^I)$

$$
\begin{aligned}
&= \mathrm{dtpurge}(\gamma, u, s^I)\,\mathrm{dtpurge}(a\alpha, u, s^I \cdot \mathrm{dtpurge}(\gamma, u, s^I)) \\
&= \mathrm{dtpurge}(\gamma, u, s^I)\,\mathrm{dtpurge}(\alpha, u, s^I \cdot \mathrm{dtpurge}(\gamma, u, s^I)) \\
&= \mathrm{dtpurge}(\gamma\alpha, u, s^I) \ .
\end{aligned}
$$

Therefore, we have

$$
\mathrm{obs}_u(s \cdot a\alpha) = \mathrm{obs}_u(s^I \cdot \gamma a\alpha) = \mathrm{obs}_u(s^I \cdot \gamma\alpha) = \mathrm{obs}_u(s \cdot \alpha) \ . \qquad \square
$$

4.2.3 Dynamic Transitively Securely Constructed Systems

The idea of securely constructed systems is to enforce noninterference by the use of local state spaces for each agent, which contain only allowed information and by the use of local transition function, which can only transmit this locally stored information. If we want to apply this technique to systems with dynamic policies, we have to ensure that policy changes do not introduce new covert channels. Therefore, the policy changes may depend only on the allowed information of those agents whose policy edges are affected by the change. More precisely, for each agent, the incoming edges of the policy have to be uniquely determined by the agent's local states. We define this property formally by:

4.2.19 Definition (transitively local state dependent policy)**.** For a system with local state spaces $S_0, \ldots, S_{n-1}$ and a global state space $S = S_0 \times \cdots \times S_{n-1}$, a dynamic policy $(\rightarrowtail_s)_{s \in S}$ is *transitively local state dependent* if for every $i, j \in D$ and every $s, t \in S$ with $\pi_i(s) = \pi_i(t)$ we have

$$
j \rightarrowtail_s i \text{ iff } j \rightarrowtail_t i \ .
$$

The definition of a dynamic transitively securely constructed system is similar to the corresponding construction of systems with a static policy. However, for the composition of these local transition functions to a global transition function, we require that the dynamic policy is transitively local state dependent. With this restriction on the dynamic policy, the global transition function is defined as

$$
(s_0, \ldots, s_{n-1}) \cdot a = (s'_0, \ldots, s'_{n-1})
$$

$$\text{with } s_i' = \begin{cases} s_i \cdot_i a & \text{if } \mathrm{dom}(a) \rightarrowtail_{(s_0,\ldots,s_{n-1})} i \\ s_i & \text{otherwise .} \end{cases}$$

The local transition functions are only applied if the corresponding edge in the local policy exists. Since the existence of an edge depends only on the local state of the receiving agent, the agent does not gain more information about the existence of a policy edge than what is already encoded in its own local state space. Therefore, such a system is dt-secure if the observations of every single agent depend only on its local states.

From this construction, it is clear that a dynamic transitively securely constructed system does not contain any illegal information flow and hence, it is dt-secure. Thus, we can skip to prove the following result.

4.2.20 Lemma. *Every dynamic transitively securely constructed system is dt-secure w.r.t. the dynamic policy from which it is constructed.*

The completeness result for this construction shows that the requirement of transitively local state dependent policies does not restrict the expressiveness of securely constructed systems. Every dt-secure system can be transformed into an observation equivalent dynamic transitively securely constructed system. Since the state space of a system will be modified by such a transformation, we need a definition of equivalence between the policies of these two systems. Intuitively, in both systems, after running the same traces the same policy should apply.

We define dynamic policies $(\rightarrowtail_s)_{s\in S}$ and $(\rightarrowtail'_{\tilde{s}})_{\tilde{s}\in S'}$ to be *trace-equivalent* iff for the respective initial states $s^I \in S$ and $\tilde{s}^I \in S'$ and every trace $\alpha \in A^*$ it holds $\rightarrowtail_{s^I\cdot\alpha} = \rightarrowtail'_{\tilde{s}^I\cdot\alpha}$.

4.2.21 Lemma. *For every dt-secure system, there exists an observation equivalent system that is dynamic transitively securely constructed from a trace-equivalent policy.*

Proof. Let M be a dt-secure system with agents $D = \{0,\ldots,n-1\}$, states S, and dynamic policy $(\rightarrowtail_s)_{s\in S}$. Due to Theorem 4.2.15, we can assume that the dynamic policy $(\rightarrowtail_s)_{s\in S}$ is dt-uniform. As states for the constructed system, we take the purged traces for each agent, starting from the initial state. The system M' is constructed from M as follows:

- ▷ the local state space of each agent i is $S_i = \{\text{dtpurge}(\alpha, i, s^I) \mid \alpha \in A^*\}$ and the global state space is $S' = S_0 \times \cdots \times S_{n-1}$,
- ▷ the local initial states are $s_i^I = \epsilon$, the global initial state is $\tilde{s}^I = (s_0^I, \ldots, s_{n-1}^I)$,
- ▷ the local transition function is for every $i \in D$, every $a \in A$, and every $s \in S$ defined by $s \cdot_i a = sa$,
- ▷ the edges of the dynamic policy $(\rightarrowtail'_s)_{s \in S'}$ are for every $i, j \in D$ and every $\alpha \in A^*$ defined by $j \rightarrowtail'_{\text{dtpurge}(\alpha,i,s^I)} i$ iff $j \rightarrowtail_{s^I \cdot \text{dtpurge}(\alpha,i,s^I)} i$, (the dtpurge values are computed in both expressions w.r.t. $(\rightarrowtail_s)_{s \in S}$,)
- ▷ for every agent i, the new observation function $\text{obs}'_i(s)$ is defined as $\text{obs}_i(s^I \cdot \pi_i(s))$.

For every agent i and every trace α, we have

$$\text{obs}_i(s^I \cdot \alpha) = \text{obs}_i(s^I \cdot \text{dtpurge}(\alpha, i, s^I)) = \text{obs}'_i(\tilde{s}^I \cdot \text{dtpurge}(\alpha, i, s^I))$$

and therefore the constructed system is observation equivalent to the original system.

Next, we have to show that the dynamic policy of the constructed system is transitively local state dependent. Let $i \in D$ and $s, t \in S$ with $\pi_i(s) = \pi_j(t)$. Then, there are $\alpha, \beta \in A^*$ with

$$\text{dtpurge}(\alpha, i, s^I) = \pi_i(s) = \pi_i(t) = \text{dtpurge}(\beta, i, s^I) \ .$$

Therefore, the states s and t are dt-similar for i and by the definition of dt-uniform, we have $i_s^{\leftarrowtail} = i_t^{\leftarrowtail}$. This implies directly that the policy of the constructed system is transitively local state dependent. From a similar argument, it follows that the systems have trace-equivalent policies, since the states $s^I \cdot \alpha$ and $s^I \cdot \text{dtpurge}(\alpha, i, s^I)$ are dt-similar for the agent i. □

Summarizing both results yields:

4.2.22 Theorem. *A system is dt-secure iff there exists an observation equivalent system that dynamic transitively securely constructed system from a trace-equivalent policy.*

4.3 Downgrading Over Time

As an intermediate definition, before defining dynamic intransitive noninterference, we consider a dynamic noninterference definition that allows agents to transmit information about earlier performed actions, but not to transmit information about actions of other agents. These new allowed effects can be interpreted as allowing agents to store information about their own actions and to delay the release until the policy allows to do so. We will call this noninterference definition dot-security.[2]

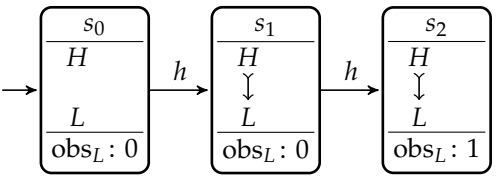

Figure 4.4. A dot-secure, but not dt-secure system

4.3.1 Example. The system in Figure 4.4 depicts a dot-secure system. Intuitively, L does not see the first h action. After performing this action, L does not know if the system is still in the initial state or if it has reached the state s_1. However, H has this information, since H is the only agent that performs any action and hence, it is always aware of the current state of the system. After the first action of H, the system is in a state where the information flow from H to L is not longer prohibited. Therefore, L is allowed to observe H's next action and also receive information about H's first action, since the second one downgrades the information that there was one of H's actions before the second h action. In summary, in this system with the given dynamic policy, the agent L is not allowed to distinguish traces ϵ and h, but it is allowed to distinguish h from hh.

On the other hand, if we apply the state-based unwinding conditions for transitive dynamic noninterference, we obtain $s_0 \sim_L s_1$ by ($\mathrm{LR}^{\mathsf{sdt}}$) and $s_1 \sim_L s_2$ by ($\mathrm{SC}^{\mathsf{sdt}}$). But since $\mathrm{obs}_L(s_1) = 0 \neq 1 = \mathrm{obs}_L(s_2)$, the system

[2] dot stands for downgrading over time

is not observation consistent for L and therefore insecure with respect to dt-security.

4.3.2 Definition (dot-security)**.** A system is *dot-secure* iff for every $u \in D$, $a \in A$, $s \in S$, and $\alpha \in A^*$ the following implication holds:

If $\mathrm{dom}(a) \not\rightarrow_s u$ and there are no $b \in A$ and $\beta, \gamma \in A^*$ with $\alpha = \beta b \gamma$, $\mathrm{dom}(a) = \mathrm{dom}(b)$, and $\mathrm{dom}(a) \rightarrowtail_{s \cdot a\beta} u$, then $\mathrm{obs}_u(s \cdot a\alpha) = \mathrm{obs}_u(s \cdot \alpha)$.

This definition says that if an agent $\mathrm{dom}(a)$ performs an action a in a state where it is not allowed to interfere with some agent u and later during the run $\mathrm{dom}(a)$ does not perform any other action in a state where it is allowed to interfere with u, then the two traces, one with and one without the action a, should be indistinguishable for u.

To decide whether a particular action is downgraded to some observer u by some actions occurring later in a trace, the function dotsrc collects these agents. Note that agents could only be added to the computed set of agents if they interfere in some state that is visited during the run described by the parameter α. So, there is no intransitivity involved, compared to the sources definition in the static case.

4.3.3 Definition (sources for downgrading over time (dotsrc))**.** For every $u \in D$, $s \in S$, $a \in A$, and $\alpha \in A^*$ define

$$\mathrm{dotsrc}(\epsilon, u, s) = \{u\} \ ,$$

$$\mathrm{dotsrc}(a\alpha, u, s) = \begin{cases} \mathrm{dotsrc}(\alpha, s, s \cdot a) \cup \{\mathrm{dom}(a)\} & \text{if } \mathrm{dom}(a) \rightarrowtail_s u \\ \mathrm{dotsrc}(\alpha, u, s \cdot a) & \text{otherwise} \ . \end{cases}$$

Based on the dotsrc definition, dot-security is expressed as follows: If an action a is not downgraded by agents involved in the trace after this action, then this action is not allowed to change the observations of the observing agent.

4.3.4 Lemma. *A system is dot-secure iff for every $u \in D$, $a \in A$, $s \in S$, and $\alpha \in A^*$ with $\mathrm{dom}(a) \notin \mathrm{dotsrc}(a\alpha, u, s)$, we have $\mathrm{obs}_u(s \cdot a\alpha) = \mathrm{obs}_u(s \cdot \alpha)$.*

Proof. First, we prove the implication from left to right. Assume that the system is dot-secure and let $u \in D$, $a \in A$, $s \in S$, and $\alpha \in A^*$ with

$\mathrm{dom}(a) \notin \mathrm{dotsrc}(a\alpha, u, s)$. By the definition of dotsrc, there are no $b \in A$ and $\beta, \gamma \in A^*$ with $\alpha = \beta b \gamma$, $\mathrm{dom}(a) = \mathrm{dom}(b)$, and $\mathrm{dom}(a) \rightarrowtail_{s \cdot a\beta} u$. Hence, by dot-security, we have $\mathrm{obs}_u(s \cdot \alpha) = \mathrm{obs}_u(s \cdot a\alpha)$.

For the other direction of the proof, assume that the system is *not* dot-secure. By the definition of dot-security, there are $u \in D$, $a \in A$, $s \in S$, and $\alpha \in A^*$ with $\mathrm{dom}(a) \not\rightarrowtail_s u$ and there are no $b \in A$ and $\beta, \gamma \in A^*$ with $\alpha = \beta b \gamma$ and $\mathrm{dom}(a) = \mathrm{dom}(b)$ and $\mathrm{dom}(a) \rightarrowtail_{s \cdot a\beta} u$ and $\mathrm{obs}_u(s \cdot a\alpha) \neq \mathrm{obs}_u(s \cdot \alpha)$. Hence, we have $\mathrm{dom}(a) \notin \mathrm{dotsrc}(a\alpha, u, s)$. □

Based on the dotsrc definition, we give a purge-based definition. Similar to the definition of dtpurge, we have to keep track of the state from which we take the policy. Again, if the purge operator removes an action, the argument of the purge operator which keeps the state remains the same. An action a is removed from the trace by dotpurge if it is not transmitted to the observing agent u, which is expressed as $\mathrm{dom}(a)$ is not in the set of agents, created by dotsrc.

4.3.5 Definition (dotpurge operator)**.** For every $u \in D$, $s \in S$, $a \in A$, and $\alpha \in A^*$ define

$$\mathrm{dotpurge}(\epsilon, u, s) = \epsilon$$

$$\mathrm{dotpurge}(a\alpha, u, s) = \begin{cases} a\ \mathrm{dotpurge}(\alpha, u, s \cdot a) & \text{if } \mathrm{dom}(a) \in \mathrm{dotsrc}(a\alpha, u, s) \\ \mathrm{dotpurge}(\alpha, u, s) & \text{otherwise .} \end{cases}$$

The dotpurge operator characterizes dot-security in the usual way. Similar to Theorem 4.2.5, we need to start purging from every arbitrary state.

4.3.6 Theorem. *A system is dot-secure iff for every $u \in D$, $s \in S$ and all $\alpha, \beta \in A^*$ with* $\mathrm{dotpurge}(\alpha, u, s) = \mathrm{dotpurge}(\beta, u, s)$*, we have* $\mathrm{obs}_u(s \cdot \alpha) = \mathrm{obs}_u(s \cdot \beta)$.

Proof. From left to right, we prove this claim by an induction on the combined length of α and β. Suppose the system is dot-secure. The claim obviously holds for the base case $\alpha = \beta = \epsilon$. Let $u \in D$ and $s \in S$ and let $\alpha, \beta \in A^*$ with $\mathrm{dotpurge}(\alpha, u, s) = \mathrm{dotpurge}(\beta, u, s)$. Assume as induction hypothesis that the implication on the right-hand side holds for all traces α'

and β' with smaller combined length and all possible states s. Let $\alpha = a\alpha'$ for some $a \in A$ and $\alpha' \in A^*$. We have the following two cases.
Case 1: $\mathrm{dom}(a) \notin \mathrm{dotsrc}(a\alpha', u, s)$.
Then we have

$$\mathrm{dotpurge}(\alpha', u, s) = \mathrm{dotpurge}(a\alpha, u, s) = \mathrm{dotpurge}(\beta, u, s) \ .$$

By induction hypothesis, we have $\mathrm{obs}_u(s \cdot \alpha') = \mathrm{obs}_u(s \cdot \beta)$. Since the system is dot-secure, by applying Lemma 4.3.4, we obtain $\mathrm{obs}_u(s \cdot a\alpha') = \mathrm{obs}_u(s \cdot \alpha')$.
Case 2: $\mathrm{dom}(a) \in \mathrm{dotsrc}(a\alpha', u, s)$.
Then we can assume that $\beta = b\beta'$ for some $b \in A$ and $\beta' \in A^*$ with $\mathrm{dom}(b) \in \mathrm{dotsrc}(b\beta', u, s)$. Otherwise we proceed with Case 1 with the roles of α and β swapped. It follows directly $a = b$ and $\mathrm{dotpurge}(\alpha', u, s \cdot a) = \mathrm{dotpurge}(\beta', u, s \cdot b)$. By applying the induction hypothesis on the traces α' and β' and on the state $s \cdot a$, we obtain $\mathrm{obs}_u(s \cdot a\alpha') = \mathrm{obs}_u(s \cdot b\beta')$.

For the other direction of the claim, we assume that the system is not dot-secure. By applying Lemma 4.3.4, there are $u \in D$, $s \in S$, $a \in A$, and $\alpha \in A^*$ with $\mathrm{dom}(a) \notin \mathrm{dotsrc}(a\alpha, u, s)$ and $\mathrm{obs}_u(s \cdot a\alpha) \neq \mathrm{obs}_u(s \cdot \alpha)$. Then we have $\mathrm{dotpurge}(a\alpha, u, s) = \mathrm{dotpurge}(\alpha, u, s)$ and still $\mathrm{obs}_u(s \cdot a\alpha) \neq \mathrm{obs}_u(s \cdot \alpha)$, which shows that the assertion on the right-hand side of this lemma does not hold. □

4.3.1 Unwinding for dot-security

We will characterize dot-security by an unwinding-like relation. However, since the definition is asymmetric, in general, this relation will not be an equivalence relation. Nevertheless, we call it an unwinding relation, even though it is not one in the sense of Section 2.6.3.

4.3.7 Definition (state-based downgrading over time unwinding)**.** A *state-based downgrading over time unwinding relation* for agents $u, v \in D$ is a relation $\precsim^{\mathsf{sdot}}_{u,v} \subseteq S \times S$ such that for every $a \in A$, $s, t \in S$ the following conditions are satisfied:

(LR$^{\mathsf{sdot}}$): If $\mathrm{dom}(a) = v$ and $v \not\rightsquigarrow_s u$, then $s \precsim^{\mathsf{sdot}}_{u,v} s \cdot a$.

(SC$^{\mathsf{sdot}}$): If $s \precsim^{\mathsf{sdot}}_{u,v} t$ and either $\mathrm{dom}(a) \neq v$ or $\mathrm{dom}(a) = v$ with $v \not\rightarrowtail_t u$, then $s \cdot a \precsim^{\mathsf{sdot}}_{u,v} t \cdot a$.

As usual, such an unwinding relation characterizes the corresponding security definition if it is observation consistent for the observing agent.

4.3.8 Theorem. *A system is dot-secure iff for every $u, v \in D$, there is a state-based downgrading over time unwinding relation $\precsim^{\mathsf{sdot}}_{u,v}$ that is observation consistent for u.*

Proof. First, we prove the implication from left to right. Suppose the system is dot-secure. Let $u, v \in D$. We say a trace $\alpha \in A^*$ has the property (†) in a state $t \in S$ iff

$$\begin{aligned} &\text{there are no } b \in A \text{ and } \beta, \gamma \in A^* \text{ with} \\ &\alpha = \beta b \gamma,\ v = \mathrm{dom}(a),\ \text{and } v \rightarrowtail_{t \cdot \beta} u\ . \end{aligned} \tag{†}$$

We define the binary relation $\precsim^{\mathsf{sdot}}_{u,v}$ for every $s, t \in S$ by

$$s \precsim^{\mathsf{sdot}}_{u,v} t \text{ iff for every } \alpha \in A^* \text{ which satisfies the property (†) in } t \text{ it holds } \mathrm{obs}_u(s \cdot \alpha) = \mathrm{obs}_u(t \cdot \alpha)\ .$$

We will show that the relation $\precsim^{\mathsf{sdot}}_{u,v}$ satisfies the properties (LR$^{\mathsf{sdot}}$) and (SC$^{\mathsf{sdot}}$), and that it is observation consistent for u.

For showing (LR$^{\mathsf{sdot}}$), let $s \in S$, $a \in A$ with $\mathrm{dom}(a) = v$ and $v \not\rightarrowtail_s u$. Let $\alpha \in A^*$ such that it satisfies the property (†) in $s \cdot a$. Since the system is dot-secure, we have $\mathrm{obs}_u(s \cdot \alpha) = \mathrm{obs}_u(s \cdot a\alpha)$.

For showing (SC$^{\mathsf{sdot}}$), let $s, t \in S$ with $s \precsim^{\mathsf{sdot}}_{u,v} t$ and assume that $s \cdot a \not\precsim^{\mathsf{sdot}}_{u,v} t \cdot a$ for some $a \in A$ with $\mathrm{dom}(a) \neq v$, or $\mathrm{dom}(a) = v$ with $v \not\rightarrowtail_t u$. Therefore, there exists some $\alpha \in A^*$ that satisfies the property (†) in $t \cdot a$, but $\mathrm{obs}_u(s \cdot a\alpha) \neq \mathrm{obs}_u(t \cdot a\alpha)$. But also $a\alpha$ satisfies (†) in t. This contradicts the assumption that $s \precsim^{\mathsf{sdot}}_{u,v} t$.

The observation consistency for u follows from setting $\alpha = \epsilon$.

For the other direction of this proof, assume that the system is *not* dot-secure.

Therefore, there are $u \in D$, $s \in S$, and $a \in A$ such that $\mathrm{dom}(a) \not\rightarrowtail_s u$, and there is $\alpha \in A^*$ satisfying property (†) in $s \cdot a$ and $\mathrm{obs}_u(s \cdot \alpha) \neq \mathrm{obs}_u(s \cdot a\alpha)$. Set $v = \mathrm{dom}(a)$ and let $\preceq^{\mathsf{sdot}}_{u,v}$ be a state-based downgrading over time unwinding relation for u and v. From ($\mathrm{LR}^{\mathsf{sdot}}$) it follows $s \preceq^{\mathsf{sdot}}_{u,v} s \cdot a$. Since α satisfies (†), from ($\mathrm{SC}^{\mathsf{sdot}}$) it follows $s \cdot \alpha \preceq^{\mathsf{sdot}}_{u,v} s \cdot a\alpha$. Hence, $\preceq^{\mathsf{sdot}}_{u,v}$ is not observation consistent for u. □

4.4 Dynamic Intransitive Noninterference

Dynamic intransitive noninterference is a generalization of intransitive noninterference to the setting with dynamic policies. Hence, we have to deal with different effects, those stemming from the intransitive behavior of downgrading and those stemming from the dynamic changes of policies and the delayed transmission of information like in the previous section. This combination makes the situation more complex than in all previously discussed settings. In this section, we discuss the idea of dynamic downgrading. Noninterference definitions for this setting will be provided in the next sections.

4.4.1 Dynamic Downgrading

In dynamic noninterference, the policies are distributed on the states. Since downgrading is about having a path from some agent that performs an action to some agent that makes some observation, we need to assemble the local policy to one large graph that captures each local policy and the dynamics of changing policies by actions.

The idea is that we have a graph where each vertex corresponds to an agent in a specific state. The edges are labeled with actions such that there is an edge from (v, s) to (u, t) iff $\mathrm{dom}(a) = v$ and $v \rightarrowtail_s u$ and $t = s \cdot a$. Hence, this graph contains both the edges of the local policies and the transitions of the underlying system.

More formally, the set of verticies of the graph is $V = D \times S$ and the set

of labeled edges is

$$E = \{((v,s),(u,t),a) \mid \mathrm{dom}(a) = v \text{ and } v \rightarrowtail_s u \text{ and } t = s \cdot a\} \ .$$

We say that an action a is downgraded by α to u starting in s if there is a path labeled with $a\alpha$ from $(\mathrm{dom}(a), s)$ to (u,t) for some state t. In the dynamic setting, we cannot simply take a subsequence as in the static case, since other downgraded actions can influence the policy and hence cannot be removed without possible effects on the downgrading.

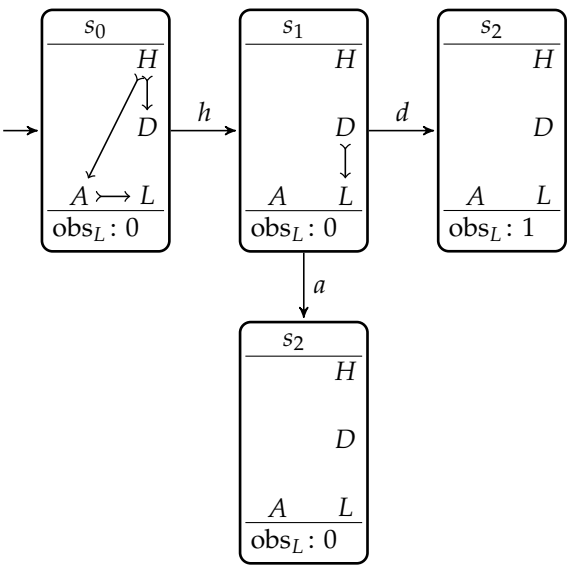

Figure 4.5. Dynamic intransitive downgrading

4.4.1 Example. Starting with the system of Figure 4.5, we generate the graph as described above. The result is depicted in Figure 4.6. As usual, self-loops are omitted. For example, if we are interested whether the action h is downgraded by the trace *had* starting in the initial state, then one can see that for every state $s \in S$, there is no path from (H, s_0) to (L, s). Hence, the action h is not downgraded to L via *had*. However, it is downgraded by *hd*,

since there is a path labeled with *hd* which ends in the vertex (L, s_2).

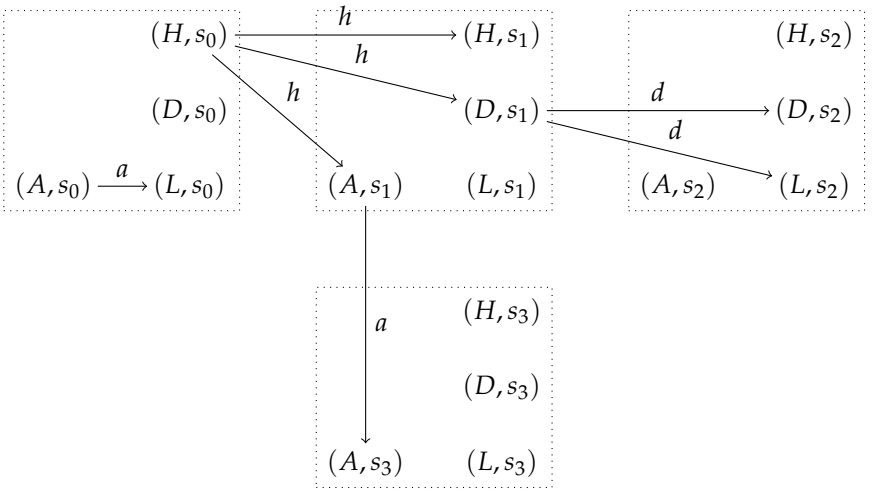

Figure 4.6. Dynamic intransitive downgrading graph

4.5 di-security

Dynamic intransitive noninterference combines both the downgrading over time behavior as explained in the previous section and the downgrading through agents as in the case of intransitive noninterference with a static policy, described in the previous chapter. The main idea is that an agent may transmit all information about previously performed actions if they have been transmitted to it earlier.

For formalizing this intuition, we need a function that finds all agents which are allowed to transmit information. Like in the static case, we provide a function called dsrc that collects all agents, which are allowed to interfere with some given agent. Obviously, in the dynamic case, this function has to keep track of the states in order to get the policy which applies.

As a formalization, we use Leslie's definition of sources [Les06].

4.5.1 Definition (dynamic sources (dsrc))**.** For every $u \in D$, $s \in S$, $a \in A$, and $\alpha \in A^*$ define

$$\mathrm{dsrc}(\epsilon, u, s) = \{u\}$$

$$\mathrm{dsrc}(a\alpha, u, s) = \begin{cases} \mathrm{dsrc}(\alpha, u, s \cdot a) \cup \{\mathrm{dom}(a)\} & \text{if } \mathrm{dom}(a) \\ & \rightarrowtail_s \mathrm{dsrc}(\alpha, u, s \cdot a) \\ \mathrm{dsrc}(\alpha, u, s \cdot a) & \text{otherwise .} \end{cases}$$

The set $\mathrm{dsrc}(a\alpha, u, s)$ is the set of all agents, to which the information that the action a has been performed, is transmitted by the sequence of actions α. Note that only information about performed actions is transmitted and not information about a particular action which has *not* been transformed. Intuitively, $\mathrm{dom}(a)$ is added by $\mathrm{dsrc}(a\alpha, u, s)$ to the set of agents collected so far if α describes a path from $\mathrm{dom}(a)$ to u in the local policies if one follows the path of α in the system.

Similar to the characterization of dot-security in Lemma 4.3.4, we provide a definition for dynamic intransitive noninterference based on the dsrc function. If the system would start in an arbitrary state and there is an action in a trace that is not downgraded to some agent u, then the performing of this action should not change u's observations.

4.5.2 Definition (di-security)**.** A system is *di-secure* iff for every $u \in D$, $s \in S$, $a \in A$, and $\alpha \in A^*$ the following implication holds:

$$\text{If } \mathrm{dom}(a) \notin \mathrm{dsrc}(a\alpha, u, s), \text{ then } \mathrm{obs}_u(s \cdot a\alpha) = \mathrm{obs}_u(s \cdot \alpha) \ .$$

We will provide several characterizations of di-security. The first one is based on an intransitive purge operator, adapted to the dynamic setting, which we will call dipurge. Applied to a trace, the dipurge operator proceeds from left to right and removes every action that is not downgraded by the remaining sequence of actions. Again, dipurge has to keep track of the state where the purge operation should be applied. However, if an action is removed, then the dipurge operator remains in the state, since the run of the system should behave in the same way as when this action had

not been performed. If the dipurge operator proceeded in the state that is reached after performing this action, some information about performing this action would get into the result of the purging.

This is the only difference between Leslie's [Les06] and our definition of an intransitive purge operator. In Leslie's definition, the transition is performed in any case, independently of whether the action has been purged or not. A comparison to the work of Leslie will be given in Section 4.8.

4.5.3 Definition (dipurge operator)**.** For every $u \in D$, $s \in S$, $a \in A$, and $\alpha \in A^*$ define

$$\begin{aligned} \mathrm{dipurge}(\epsilon, u, s) &= \epsilon \\ \mathrm{dipurge}(a\alpha, u, s) &= \begin{cases} a\ \mathrm{dipurge}(\alpha, u, s \cdot a) & \text{if } \mathrm{dom}(a) \in \mathrm{dsrc}(a\alpha, u, s) \\ \mathrm{dipurge}(\alpha, u, s) & \text{otherwise .} \end{cases} \end{aligned}$$

The corresponding characterization of di-security is as always. If two traces have the same dipurge values, then they have to lead to the same observation. However, in order to have a characterization of di-security, it is required that the purging has to be performed in any state of the system.

4.5.4 Theorem. *A system is di-secure iff for every $u \in D$, $s \in S$ and $\alpha, \beta \in A^*$ the following implication holds:*

$$\textit{If } \mathrm{dipurge}(\alpha, u, s) = \mathrm{dipurge}(\beta, u, s), \textit{ then } \mathrm{obs}_u(s \cdot \alpha) = \mathrm{obs}_u(s \cdot \beta) \ .$$

Proof. First, we prove this claim from left to right. Suppose that the system is di-secure and assume that the right-hand side of the implication does not hold. From all counter examples for the right-hand side, choose one where α and β are of minimal combined length for all possible choices of u and s. Hence, there are $u \in D$ and $s \in S$ with $\mathrm{dipurge}(\alpha, u, s) = \mathrm{dipurge}(\beta, u, s)$ and $\mathrm{obs}_u(s \cdot \alpha) \neq \mathrm{obs}_u(s \cdot \beta)$. Since at least one of the traces α or β is not the empty trace, we can assume that $\alpha = a\alpha'$ with $a \in A$ and $\alpha' \in A^*$.
Case 1: $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\alpha', u, s)$.
In this case, we have

$$\mathrm{dipurge}(a\alpha', u, s) = \mathrm{dipurge}(\alpha', u, s) = \mathrm{dipurge}(\beta, u, s) \ .$$

From di-security it follows: $\text{obs}_u(s \cdot \alpha') = \text{obs}_u(s \cdot a\alpha') \neq \text{obs}_u(s \cdot \beta)$. Hence α' and β are a counter example of smaller combined length, which contradicts our assumption.
Case 2: $\text{dom}(a) \in \text{dsrc}(a\alpha', u, s)$.
By the definition of the dipurge operator, we have $\text{dipurge}(a\alpha', u, s) = a\ \text{dipurge}(\alpha', u, s \cdot a)$. We can assume that $\beta = b\beta'$ with $b \in A$ and $\beta' \in A^*$ with $\text{dom}(b) \in \text{dsrc}(b\beta', u, s)$ since otherwise we would proceed with Case 1 with the roles of α and β swapped. Therefore, we have $\text{dipurge}(b\beta', u, s) = b\ \text{dipurge}(\beta', u, s \cdot b)$. It follows that $a = b$ and $\text{dipurge}(\alpha', u, s \cdot a) = \text{dipurge}(\beta', u, s \cdot a)$. But from $\text{obs}_u(s \cdot a\alpha') \neq \text{obs}_u(s \cdot a\beta')$ it follows that α' and β' are a counter example of smaller combined length.

This contradicts our initial assumption that the right-hand side of the claim is wrong.

For the other direction of the claim, assume that the right-hand side holds. We will show that the system is di-secure. Let $u \in D$, $s \in S$, $a \in A$, and $\alpha \in A^*$ with $\text{dom}(a) \notin \text{dsrc}(a\alpha, u, s)$. By the definition of dipurge, we have $\text{dipurge}(a\alpha, u, s) = \text{dipurge}(\alpha, u, s)$ and by the assumption that the right-hand side of the claim holds it follows $\text{obs}_u(s \cdot a\alpha) = \text{obs}_u(s \cdot \alpha)$. □

Next, we will analyze the monotonicity of the di-security definition. Under monotony, we understand that a secure system with respect to some policy remains secure under a less restrictive policy. In other words, an insecure system remains insecure under a more restrictive policy. Recall that a dynamic policy $(\rightarrowtail'_s)_{s \in S}$ is at least as restrictive as a policy $(\rightarrowtail_s)_{s \in S}$ if for every $u, v \in D$ and every $s \in S$, $u \rightarrowtail'_s v$ implies $u \rightarrowtail_s v$.

First we show that the definition of dsrc behaves in an expected way for a more restrictive policy.

4.5.5 Lemma. *Let $(\rightarrowtail_s)_{s \in S}$ be a dynamic policy and let $(\rightarrowtail'_s)_{s \in S}$ be at least as restrictive as the policy $(\rightarrowtail_s)_{s \in S}$. Denote with* dsrc *the source function with respect to $(\rightarrowtail_s)_{s \in S}$ and with* dsrc′ *the source function with respect to $(\rightarrowtail'_s)_{s \in S}$. Then we have for every $u \in D$, $s \in S$, and $\alpha \in A^*$ that* $\text{dsrc}(\alpha, u, s) \supseteq \text{dsrc}'(\alpha, u, s)$.

Proof. We prove this claim by an induction on the length of α and suppose as induction hypothesis that it holds for all traces of smaller length in all

states. For the inductive step, we distinguish the following three cases:
Case 1: $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\alpha, u, s)$.
Then we have

$$\begin{aligned}\mathrm{dsrc}(a\alpha, u, s) &= \mathrm{dsrc}(\alpha, u, s \cdot a) \\ &\supseteq \mathrm{dsrc}'(\alpha, u, s \cdot a) \ .\end{aligned}$$

From $\mathrm{dom}(a) \not\rightarrowtail_s \mathrm{dsrc}(\alpha, u, s \cdot a)$ it follows $\mathrm{dom}(a) \not\rightarrowtail_s \mathrm{dsrc}'(\alpha, u, s \cdot a)$ and by definition, we have $\mathrm{dsrc}'(\alpha, u, s \cdot a) = \mathrm{dsrc}'(a\alpha, u, s)$.
Case 2: $\mathrm{dom}(a) \in \mathrm{dsrc}(a\alpha, u, s)$ and $\mathrm{dom}(a) \notin \mathrm{dsrc}'(a\alpha, u, s)$.
In this case, we have

$$\begin{aligned}\mathrm{dsrc}(a\alpha, u, s) &= \{\mathrm{dom}(a)\} \cup \mathrm{dsrc}(\alpha, u, s \cdot a) \\ &\supseteq \{\mathrm{dom}(a)\} \cup \mathrm{dsrc}'(\alpha, u, s \cdot a) \\ &\supseteq \mathrm{dsrc}'(\alpha, u, s \cdot a) \\ &= \mathrm{dsrc}'(a\alpha, u, s) \ .\end{aligned}$$

Case 3: $\mathrm{dom}(a) \in \mathrm{dsrc}(a\alpha, u, s)$ and $\mathrm{dom}(a) \in \mathrm{dsrc}'(a\alpha, u, s)$.
Here, we have

$$\begin{aligned}\mathrm{dsrc}(a\alpha, u, s) &= \{\mathrm{dom}(a)\} \cup \mathrm{dsrc}(\alpha, u, s \cdot a) \\ &\supseteq \{\mathrm{dom}(a)\} \cup \mathrm{dsrc}'(\alpha, u, s \cdot a) \\ &= \mathrm{dsrc}'(a\alpha, u, s) \ .\end{aligned}$$

□

With this result of the dsrc function, we can simply conclude that di-security has the desired monotony property.

4.5.6 Corollary. *Let $(\rightarrowtail_s)_{s \in S}$ be a dynamic policy and let $(\rightarrowtail'_s)_{s \in S}$ be at least as restrictive as the policy $(\rightarrowtail_s)_{s \in S}$. If the system is di-secure with respect to $(\rightarrowtail'_s)_{s \in S}$, then it is di-secure with respect to $(\rightarrowtail_s)_{s \in S}$.*

Proof. Denote again with dsrc the dynamic source function with respect to $(\rightarrowtail_s)_{s \in S}$ and with dsrc' the dynamic source function with respect to $(\rightarrowtail'_s)_{s \in S}$. Let $u \in D$, $s \in S$, $a \in A$, and $\alpha \in A^*$. If $\mathrm{dom}(a) \notin \mathrm{dsrc}(\alpha, u, s)$,

then by Lemma 4.5.5, $\mathrm{dom}(a) \notin \mathrm{dsrc}'(\alpha, u, s)$ and hence, by di-security w.r.t. $(\rightarrowtail'_s)_{s \in S}$, we have $\mathrm{obs}_u(s^I \cdot a\alpha) = \mathrm{obs}_u(s^I \cdot \alpha)$. □

4.5.1 Unwinding for di-security

The second characterization of di-security is in terms of an unwinding-like definition. It defines a binary relation on the set of states, but in general, it is not an equivalence relation. However, as with dot-security, we will call it an unwinding relation nevertheless. In contrast to the previous state-based unwinding relations, we need to consider such a relation for every possible set of agents. Hence, the number of relations is exponential in the number of agents, but it is polynomial in the number of states.

4.5.7 Definition (state-based dynamic intransitive unwinding)**.** A *state-based dynamic intransitive unwinding relation* for a set of agents $D' \subseteq D$ is a relation $\precsim_{D'} \subseteq S \times S$ such that for every $s, t \in S$ and every $a \in A$, the following conditions hold:

(LR$^{\mathsf{sdi}}$)*:* $s \precsim_{\{u \in D \mid \mathrm{dom}(a) \not\rightarrowtail_s u\}} s \cdot a$.

(SC$^{\mathsf{sdi}}$)*:* If $s \precsim_{D'} t$, then $s \cdot b \precsim_{D''} t \cdot b$, where $D'' = D'$ if $\mathrm{dom}(b) \in D'$, and else $D'' = D' \cap \{u \mid \mathrm{dom}(b) \not\rightarrowtail_t u\}$.

Intuitively, $s \precsim_{D'} t$ expresses that there is a state $\tilde{s}$, an action a, and a trace α such that $s = \tilde{s} \cdot a\alpha$, $t = \tilde{s} \cdot \alpha$, and $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\alpha, u, \tilde{s})$ for *all* agents $u \in D'$.

A monotony property needed in the proof of the next theorem is: If two states are in relation for some set of agents D' then these states are in relation for every subset of D'. From the definition of $\precsim_{D'}$ it directly follows:

4.5.8 Proposition. *For every set $D' \subseteq D$ and all states $s, t \in States$ with $s \precsim_{D'} t$, we have for every $\tilde{D} \subseteq D'$ that $s \precsim_{\tilde{D}} t$.*

The next result states that the state-based dynamic intransitive unwinding indeed characterizes di-security.

4.5.9 Theorem. *A system is di-secure iff for every $D' \subseteq D$ there is a state-based dynamic intransitive unwinding relation $\precsim_{D'}$ that is observation consistent for every $u \in D'$.*

Proof. Suppose the system is di-secure. Define for all states $s, t \in S$ and every set of agents $D' \subseteq D$ the relation $s \precsim_D t$ by

$$s \precsim_{D'} t \text{ iff there are } \tilde{s} \in S,\ a \in A,\ \alpha \in A^* \text{ such that } s = \tilde{s} \cdot \alpha,\ t = \tilde{s} \cdot a\alpha \text{ and for all } u \in D' \text{ we have } \mathrm{dom}(a) \notin \mathrm{dsrc}(a\alpha, u, \tilde{s}) \ .$$

We will show that these relations satisfy the unwinding conditions ($\mathrm{LR}^{\mathsf{sdi}}$) and ($\mathrm{SC}^{\mathsf{sdi}}$) and that every relation $\precsim_{D'}$ is observation consistent for every $u \in D'$.

Observation consistency for every $u \in D'$ follows directly from di-security. The ($\mathrm{LR}^{\mathsf{sdi}}$) condition is satisfied by taking $\tilde{s} = s$ and $\alpha = \epsilon$. It remains to show the ($\mathrm{SC}^{\mathsf{sdi}}$) condition. Suppose that $s \precsim_{D'} t$ for some states s, t and some set of agents D'. By definition, there is some $\tilde{s} \in S$, $a \in A$, $\alpha \in A^*$ such that $s = \tilde{s} \cdot \alpha$, $t = \tilde{s} \cdot a\alpha$ and for all $u \in D'$ we have $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\alpha, u, \tilde{s})$. Let $b \in A$ and $u \in D'$. First, assume that $\mathrm{dom}(b) \in D'$. Then, we have $\mathrm{dom}(b) \notin \mathrm{dsrc}(a\alpha, \mathrm{dom}(b), \tilde{s})$. By the definition of the dsrc function, we have $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\alpha b, u, \tilde{s})$. Second, we assume that $\mathrm{dom}(b) \notin D'$ and $u \in D'$ with $\mathrm{dom}(b) \not\rightarrowtail_t u$. Therefore, we have $\mathrm{dsrc}(a\alpha b, u, \tilde{s}) = \mathrm{dsrc}(a\alpha, u, \tilde{s})$, which gives $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\alpha b, u, \tilde{s})$.

For the other direction of the proof, assume that the system is *not* di-secure. Therefore, there are $u \in D$, $s \in S$, $a \in A$, $\alpha \in A^*$ such that $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\alpha, u, s)$ and $\mathrm{obs}_u(s \cdot a\alpha) \neq \mathrm{obs}_u(s \cdot \alpha)$.

We will show that for any $\beta, \gamma \in A^*$ with $\beta\gamma = \alpha$, we have

$$s \cdot \beta \precsim_{\mathrm{dsrc}(\gamma, u, s \cdot a\beta)} s \cdot a\beta \ .$$

We will prove this claim by an induction on the length of β. For showing the base case, let $\beta = \epsilon$. Since $\mathrm{dom}(a) \not\rightarrowtail_v$ for all $v \in \mathrm{dsrc}(a\alpha, u, s)$, we have $\mathrm{dsrc}(a\alpha, u, s) = \mathrm{dsrc}(\alpha, u, s \cdot a)$ and $s \precsim_{\mathrm{dsrc}(\alpha, u, s \cdot a)} s \cdot a$.

For the inductive step, let $\alpha = \beta b \gamma$ for some action b and traces β and γ such that

$$s \cdot \beta \precsim_{\mathrm{dsrc}(b\gamma, u, s \cdot a\beta)} s \cdot a\beta$$

holds. First, assume that $\mathrm{dom}(b) \in \mathrm{dsrc}(b\gamma, u, s \cdot a\beta)$. The condition ($\mathrm{SC}^{\mathsf{sdi}}$) gives

$$s \cdot \beta b \precsim_{\mathrm{dsrc}(b\gamma,u,s\cdot a\beta)} s \cdot a\beta \ .$$

By the definition of the sources, we have $\mathrm{dsrc}(b\gamma, u, s \cdot a\beta) = \mathrm{dsrc}(b\gamma, u, s \cdot a\beta) \cup \{\mathrm{dom}(b)\}$. Therefore, $\mathrm{dsrc}(b\gamma, u, s \cdot a\beta) \subseteq \mathrm{dsrc}(b\gamma, u, s \cdot a\beta)$, and by Proposition 4.5.8, we have

$$s \cdot \beta b \precsim_{\mathrm{dsrc}(\gamma,u,s\cdot a\beta b)} s \cdot a\beta \ .$$

Second, assume $\mathrm{dom}(b) \notin \mathrm{dsrc}(b\gamma, u, s \cdot a\beta)$. By the definition of dsrc, we have $\mathrm{dsrc}(b\gamma, u, s \cdot a\beta) = \mathrm{dsrc}(\gamma, u, s \cdot a\beta b)$. It also holds that

$$\mathrm{dsrc}(b\gamma, u, s \cdot a\beta) = \mathrm{dsrc}(b\gamma, u, s \cdot a\beta) \cap \{v \in D \mid \mathrm{dom}(b) \not\rightarrow_{s\cdot a\beta} v\} \ .$$

The combination of the last two equations results in

$$s \cdot \beta b \precsim_{\mathrm{dsrc}(\gamma,u,s\cdot a\beta b)} s \cdot a\beta b \ . \qquad \square$$

4.5.2 Intransitively Useless Edges and di-similarity

Similar to the dynamic transitive case, also in the intransitive case, we might have useless edges. However, the situation is more complex. For a policy without useless edges, it is no longer sufficient that every agent knows its incoming edges as we will see in the next example.

4.5.10 Example. As we have seen in the system of Figure 4.4, initially agent L does not know whether the system is still in the initial state or in the state s_1 and hence it does not know whether there is an edge from H to L or not. However, the edge from H to L in the state s_1 is clearly not useless, since removing it would turn a secure system into an insecure one.

Analogue to the definition of dt-similarity, we define a similarity relation on the states for the dynamic intransitive case.

4.5.11 Definition (di-similarity)**.** For an agent u let $\approx_u^{\mathsf{di}}$ be the smallest equivalence relation on S such that for every $s \in S$, $a \in A$, and $\alpha \in A^*$ with

$\mathrm{dom}(a) \notin \mathrm{dsrc}(a\alpha, u, s)$, we have

$$s \cdot a\alpha \approx_u^{\mathsf{di}} s \cdot \alpha \ .$$

We call states s, t with $s \approx_u^{\mathsf{di}} t$ *di-similar* for u.

Clearly, two states are di-similar for some u when it is required for u to have the same observations on these two states according to the definition of di-security. Hence, a system is di-secure if and only if for every agent u, the relation $\approx_u^{\mathsf{di}}$ is observation consistent for u.

We can now define intransitive useless edges. We treat an edge of a local policy as useless if its removal does not change the di-similarity relation of any agent.

4.5.12 Definition (intransitively useless edges)**.** Let e be an edge in a local policy of $(\rightarrowtail_s)_{s\in S}$ and let $(\rightarrowtail'_s)_{s\in S}$ be the policy obtained from $(\rightarrowtail_s)_{s\in S}$ by removing the edge e. For every $u \in D$, let $\approx_u^{\mathsf{di}}$ be the di-similar relation w.r.t. $(\rightarrowtail_s)_{s\in S}$ and let $\tilde{\approx}_u^{\mathsf{di}}$ be the di-similar relation w.r.t. $(\rightarrowtail'_s)_{s\in S}$. Then the edge e is *intransitively useless* iff for every $u \in D$ and for every $s, t \in S$ hold:

$$s \approx_u^{\mathsf{di}} t \text{ iff } s \tilde{\approx}_u^{\mathsf{di}} t \ .$$

An edge is intransitively useless if its removal does not forbid any previously allowed information flow. Note that the uselessness of an edge is independent of the observation function. Also, in a particular system with a static policy, there might be an edge whose removal would not affect security, however, in general, it depends on the observations of the agents.

The next example shows a system with an intransitively useless edge whose removal does not affect security.

4.5.13 Example. The system in Figure 4.7 depicts a *not* di-secure system. At a first glance, this system might look like a di-secure system. However, when we consider the trace $h_2h_1h_2$, then h_2 is not downgraded by the actions h_1h_2, but the traces $h_2h_1h_2$ and h_1h_2 lead to different observations.

We will show that the edge from H to L in state s_1 is useless. Without this edge, it is much easier to verify that this system is not secure, since the action h_2 performed in the state s_1 changes L's observation.

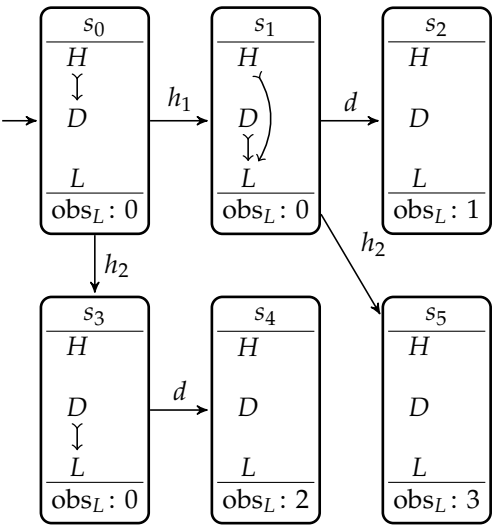

Figure 4.7. A system with an intransitively useless edge

Next, we will analyze formally why this edge is indeed intransitively useless by computing the di-similarity relation for L. One can easily verify that

$$
\begin{aligned}
s_0 \cdot h_2 h_1 &\approx^{\mathsf{di}}_L s_0 \cdot h_1 \ , & s_0 \cdot h_2 h_1 h_1 &\approx^{\mathsf{di}}_L s_0 \cdot h_2 h_1 \ , \\
s_0 \cdot h_2 h_1 h_1 &\approx^{\mathsf{di}}_L s_0 \cdot h_1 h_1 \ , & s_0 \cdot h_2 h_1 h_2 &\approx^{\mathsf{di}}_L s_0 \cdot h_2 h_1 \ , \\
s_0 \cdot h_2 h_1 h_2 &\approx^{\mathsf{di}}_L s_0 \cdot h_1 h_2 \ . &&
\end{aligned}
$$

By symmetry and transitivity of the $\approx^{\mathsf{di}}_L$ relation, we obtain

$$
s_0 \cdot h_1 \approx^{\mathsf{di}}_L s_0 \cdot h_1 h_1 \approx^{\mathsf{di}}_L s_0 \cdot h_1 h_2 \ .
$$

Hence, removing the edge from H to L in the state s_1 does not change the relation $\approx^{\mathsf{di}}_L$. Therefore, this edge is useless.

The next theorem shows that the removal of intransitively useless edges indeed does not affect security.

4.5.14 Theorem. *If a dynamic policy $(\rightarrowtail'_s)_{s\in S}$ is obtained from $(\rightarrowtail_s)_{s\in S}$ by removing intransitively useless edges, then the system is di-secure with respect to $(\rightarrowtail_s)_{s\in S}$ iff it is di-secure with respect to $(\rightarrowtail'_s)_{s\in S}$.*

Proof. Suppose the system is di-secure w.r.t. $(\rightarrowtail_s)_{s\in S}$ but not di-secure w.r.t. $(\rightarrowtail'_s)_{s\in S}$. Inductively, we can also assume that $(\rightarrowtail'_s)_{s\in S}$ arose from $(\rightarrowtail_s)_{s\in S}$ by removing a single edge e in one of the local policies. By the definition of di-security, we have $u \in D$, $s \in S$, $a \in A$, and $\alpha \in A^*$ with $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\alpha, u, s)$ (with respect to $(\rightarrowtail'_s)_{s\in S}$) and $\mathrm{obs}_u(s \cdot a\alpha) \neq \mathrm{obs}_u(s \cdot \alpha)$. But since the system is di-secure w.r.t. $(\rightarrowtail_s)_{s\in S}$, we have $\mathrm{dom}(a) \in \mathrm{dsrc}(a\alpha, u, s)$ (with respect to $(\rightarrowtail_s)_{s\in S}$). This contradicts the assumption that e is useless.

The other direction follows from the monotony of di-security as stated in Corollary 4.5.6, since $(\rightarrowtail_s)_{s\in S}$ is less restrictive than $(\rightarrowtail'_s)_{s\in S}$. □

4.5.3 Intransitive Uniformity

Like in the case of dynamic transitive noninterference, it is possible to give a reasonable definition of a uniform policy. A policy is uniform if every agent knows—in the sense that it is allowed to know—its incoming edges in each local policy. Unlike the transitive case, this is not equivalent to removing all useless edges. However, we will consider this special case of intransitive policies, since we can provide a sound and complete polynomial-size unwinding for it.

4.5.15 Definition (intransitive uniformity)**.** A dynamic policy $(\rightarrowtail_s)_{s\in S}$ is *intransitively uniform* if for every $u \in D$ and every $s, t \in S$ the following implication holds:

$$\text{If } s \approx_u^{\mathsf{di}} t, \text{ then } u_s^{\leftarrowtail} = u_t^{\leftarrowtail} \ .$$

In a system with an intransitively uniform policy, every agent knows its incoming edges of the local policy. In other words, if there is some reason why an agent is not allowed to distinguish two states, then this agent has the same incoming edges in the local policies of these two states. Next, we will give a state-based unwinding for systems with intransitively uniform policies.

4.5.16 Definition (state-based uniform dynamic intransitive unwinding)**.** A *state-based uniform dynamic intransitive unwinding relation* for agents v and u and a state $\tilde{s}$ is an equivalence relation $\sim^{\mathsf{sudi}}_{\tilde{s},v,u} \subseteq S \times S$ such that for every $s, t \in S$ and every $a \in A$ the following conditions hold:

(LR$^{\mathsf{sudi}}$)*:* If $v = \mathrm{dom}(a)$ and $v \not\rightarrow_{\tilde{s}} u$, then $\tilde{s} \sim^{\mathsf{sudi}}_{\tilde{s},v,u} \tilde{s} \cdot a$.

(SC$^{\mathsf{sudi}}$)*:* If $s \sim^{\mathsf{sudi}}_{\tilde{s},v,u} t$ and $a \in A$ with $v \not\rightarrow_{\tilde{s}} \mathrm{dom}(a)$, then $s \cdot a \sim^{\mathsf{sudi}}_{\tilde{s},v,u} t \cdot a$.

(PC$^{\mathsf{sudi}}$)*:* If $s \sim^{\mathsf{sudi}}_{\tilde{s},v,u} t$, then $u^{\leftarrowtail}_s = u^{\leftarrowtail}_t$.

The condition (PC$^{\mathsf{sudi}}$) of this definition is denoted as *policy consistency*. This condition requires that if two states have to be indistinguishable for some agent u, then u's incoming edges have to be the same in these states.

Similar to Theorem 3.4.9 in the case of a static policy, we can adopt the idea of a policy cut to systems with a uniform dynamic policy. However, the cut is only considered in a local policy for a single state. For this result, the following technical lemma is needed.

4.5.17 Lemma. *In a system with a dynamic intransitively uniform policy, we have for every $u \in D$, $s \in S$, $a \in A$, and $\beta, \gamma \in A^*$:*

$$\textit{If } \mathrm{dom}(a) \notin \mathrm{dsrc}(a\beta\gamma, u, s), \textit{ then } \mathrm{dsrc}(\gamma, u, s \cdot a\beta) = \mathrm{dsrc}(\gamma, u, s \cdot \beta)$$

and for every $v \in \mathrm{dsrc}(\gamma, u, s \cdot a\beta)$ and every prefix δ of β we have

$$s \cdot a\delta \approx^{\mathsf{di}}_v s \cdot \delta \ .$$

Proof. Let $u \in D$, $s \in S$, $a \in A$, and let $\beta, \gamma \in A^*$ with $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\beta\gamma, u, s)$. We will show this claim by an induction on γ. For the base case $\gamma = \epsilon$, we have

$$\mathrm{dsrc}(\epsilon, u, s \cdot \beta) = \{u\} = \mathrm{dsrc}(\epsilon, u, s \cdot a\beta) \ .$$

For every prefix δ of β, it follows from $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\beta\gamma, u, s)$ that $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\delta, u, s)$, and from the definition of di-similarity it follows that $s \cdot a\delta \approx^{\mathsf{di}}_u s \cdot \delta$.

As induction hypothesis, we consider a trace $\beta b \gamma$ with $b \in A$ and $\beta, \gamma \in A^*$ with

$$\mathrm{dsrc}(\gamma, u, s \cdot a\beta b) = \mathrm{dsrc}(\gamma, u, s \cdot \beta b) \ , \tag{I.H.}$$

and for every $v \in \mathrm{dsrc}(\gamma, u, s \cdot a\beta b)$ and every prefix δ of βb, we have

$$s \cdot a\delta \approx_v^{\mathsf{di}} s \cdot \delta \ .$$

Let be $v \in \mathrm{dsrc}(\gamma, u, s \cdot a\beta b)$. From the induction hypothesis, it follows $s \cdot a\beta \approx_v^{\mathsf{di}} s \cdot \beta$. By uniformity, we have $\mathrm{dom}(b) \rightarrowtail_{s \cdot a\beta} v$ if and only if $\mathrm{dom}(b) \rightarrowtail_{s \cdot \beta} v$. Therefore, we have $\mathrm{dsrc}(b\gamma, u, s \cdot \beta) = \mathrm{dsrc}(b\gamma, u, s \cdot a\beta)$. Let δ be a prefix of β and let $v \in \mathrm{dsrc}(b\gamma, u, s \cdot a\beta)$. If v is already in $\mathrm{dsrc}(\gamma, u, s \cdot \beta b)$, then by induction hypothesis it follows $s \cdot a\delta \approx_v^{\mathsf{di}} s \cdot \delta$. If this is not the case, then $v = \mathrm{dom}(b)$ and since $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\beta b\gamma, u, s)$, we have $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\delta, \mathrm{dom}(b), s)$. Otherwise $\mathrm{dom}(a)$ would be downgraded to u. By definition of di-similarity, we have $s \cdot a\delta \approx_{\mathrm{dom}(b)}^{\mathsf{di}} s \cdot \delta$. □

Now, we will prove the characterization of di-security in terms of policy cuts, given that the dynamic policy is intransitively uniform.

4.5.18 Lemma. *A system with a dynamic intransitively uniform policy is di-secure iff for all $u \in D$, $s \in S$, $a \in A$, and $\alpha \in A^*$ with $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\alpha, u, s)$ and $\mathrm{dom}(a) \not\rightarrowtail_s \mathrm{dom}(\mathrm{alph}(\alpha))$, we have $\mathrm{obs}_u(s \cdot \alpha) = \mathrm{obs}_u(s \cdot a\alpha)$.*

Proof. We will prove this lemma by contraposition. Suppose that the right-hand side of this lemma does not hold. Then it follows directly from the definition of di-security that the system is not di-secure.

For proving the other direction, assume that the system is *not* di-secure. By definition, there are $u \in D$, $s \in S$, $a \in A$, and $\alpha \in A^*$ with $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\alpha, u, s)$ and $\mathrm{obs}_u(s \cdot a\alpha) \neq \mathrm{obs}_u(s \cdot \alpha)$. Additionally, assume that α is of minimal length for all possible choices of u, s, and a.

For showing the property $\mathrm{dom}(a) \not\rightarrowtail_s \mathrm{dom}(\mathrm{alph}(\alpha))$, assume the converse. Therefore, α is of the form $\beta b \gamma$ for some $b \in A$ with $\mathrm{dom}(a) \rightarrowtail_s \mathrm{dom}(b)$ and some $\beta, \gamma \in A^*$.

From $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\beta\gamma, u, s)$ and Lemma 4.5.17 it follows $\mathrm{dsrc}(b\gamma, u, s \cdot a\beta) = \mathrm{dsrc}(b\gamma, u, s \cdot \beta)$. In the case of $\mathrm{dom}(b) \in \mathrm{dsrc}(b\gamma, u, s \cdot a\beta)$, we had

$\mathrm{dom}(a) \in \mathrm{dsrc}(a\beta b\gamma, u, s \cdot a)$, which would contradict our assumption. Therefore, we have $\mathrm{dom}(b) \in \mathrm{dsrc}(b\gamma, u, s \cdot a\beta)$, and by uniformity and Lemma 4.5.17, we also have $\mathrm{dom}(b) \in \mathrm{dsrc}(b\gamma, u, s \cdot \beta)$. Applying again Lemma 4.5.17, we have

$$\begin{aligned}\mathrm{dsrc}(b\gamma, u, s \cdot a\beta) &= \mathrm{dsrc}(\gamma, u, s \cdot a\beta b) \\ &= \mathrm{dsrc}(\gamma, u, s \cdot a\beta) \ ,\end{aligned}$$

and similarly

$$\begin{aligned}\mathrm{dsrc}(b\gamma, u, s \cdot \beta) &= \mathrm{dsrc}(\gamma, u, s \cdot \beta b) \\ &= \mathrm{dsrc}(\gamma, u, s \cdot \beta) \ .\end{aligned}$$

Therefore, we have $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\alpha, u, s) = \mathrm{dsrc}(a\beta\gamma, u, s)$. Since γ is of smaller length than α, it follows from $\mathrm{dom}(b) \notin \mathrm{dsrc}(b\gamma, u, s \cdot a\beta)$ that $\mathrm{obs}_u(s \cdot a\beta b\gamma) = \mathrm{obs}_u(s \cdot a\beta\gamma)$. Similarly from $\mathrm{dom}(b) \notin \mathrm{dsrc}(b\gamma, u, s \cdot \beta)$ it follows $\mathrm{obs}_u(s \cdot \beta b\gamma) = \mathrm{obs}_u(s \cdot \beta\gamma)$. Therefore, we have $\mathrm{obs}_u(s \cdot a\beta\gamma) \neq \mathrm{obs}_u(s \cdot \beta\gamma)$. This contradicts the minimality of α, since we would take the shorter trace $\beta\gamma$ instead. □

For intransitive uniformity, it is sufficient to consider only those traces where every action but the first one is downgraded.

4.5.19 Lemma. *Let be $u \in D$, $s \in S$, $a \in A$, and $\alpha \in A^*$. If the following two properties imply $u^{\leftarrow}_{s \cdot a\alpha} = u^{\leftarrow}_{s \cdot \alpha}$, then the system has an intransitively uniform policy:*

1. $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\alpha, u, s)$.
2. *If for every $b \in A$ and $\beta, \gamma \in A^*$ with $\alpha = \beta b\gamma$, then we have* $\mathrm{dom}(a) \not\rightarrowtail_s \mathrm{dom}(b)$.

Proof. For contradiction, assume that the claim of this lemma does not hold. Hence, there are $u \in D$, $s \in S$, $a \in A$, and $\alpha \in A^*$ with $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\alpha, u, s)$ and $u^{\leftarrow}_{s \cdot a\alpha} \neq u^{\leftarrow}_{s \cdot \alpha}$. Additionally, suppose that α is of smallest length for all possible choices of u, s, and a. By the conditions of this lemma, there are $b \in A$ and $\beta, \gamma \in A^*$ with $\alpha = \beta b\gamma$ and $\mathrm{dom}(a) \rightarrowtail_s \mathrm{dom}(b)$. From the definition of dsrc it follows that $\mathrm{dom}(b) \notin \mathrm{dsrc}(b\gamma, u, s \cdot a\beta)$.

Case 1: $u^{\leftarrow}_{s\cdot a\beta b\gamma} \neq u^{\leftarrow}_{s\cdot\beta\gamma}$.
This case is not possible since it contradicts the minimality of α.
Case 2: $u^{\leftarrow}_{s\cdot\beta b\gamma} \neq u^{\leftarrow}_{s\cdot\beta\gamma}$.
To obtain a contradiction, it is sufficient to show that $\mathrm{dom}(b)$ is not in the set $\mathrm{dsrc}(b\gamma, u, s^I \cdot \beta)$. For contradiction, assume that $\mathrm{dom}(b) \in \mathrm{dsrc}(b\gamma, u, s^I \cdot \beta)$. Let δ be a prefix of minimal length of $b\gamma$ such that there is a $v \in D$ with $\mathrm{dom}(b) \in \mathrm{dsrc}(\delta, v, s \cdot \beta)$, but $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\beta\delta, v, s)$. Such a v exists, since one could choose $v = u$ and $\delta = b\beta$. Then we can write δ as $\mu c\nu$ for some $c \in A$ and $\mu, \nu \in A^*$ with

$$\mathrm{dom}(b) \in \mathrm{dsrc}(\mu, \mathrm{dom}(c), s \cdot \beta) \text{ and } \mathrm{dom}(c) \rightarrowtail_{s\cdot\beta\mu} v \ .$$

Since $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\beta\delta, v, s)$, it follows $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\beta\mu, v, s)$. By the minimality of α, we have $u^{\leftarrow}_{s\cdot\beta\mu} = u^{\leftarrow}_{s\cdot a\beta\mu}$. In particular, we have $\mathrm{dom}(a) \rightarrowtail_{s\cdot a\beta\mu} u$. Hence, we have $\mathrm{dom}(b) \in \mathrm{dsrc}(\mu, \mathrm{dom}(c), s \cdot \beta)$. Since $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\beta\delta, v, s)$, we have $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\beta\mu, \mathrm{dom}(c), s)$. This is a contradiction to the minimality of δ.
Case 3: $u^{\leftarrow}_{s\cdot a\beta b\gamma} = u^{\leftarrow}_{s\cdot\beta\gamma}$ and $u^{\leftarrow}_{s\cdot\beta b\gamma} = u^{\leftarrow}_{s\cdot\beta\gamma}$.
From $u^{\leftarrow}_{s\cdot a\beta b\gamma} \neq u^{\leftarrow}_{s\cdot\beta b\gamma}$ it follows $u^{\leftarrow}_{s\cdot\beta b\gamma} \neq u^{\leftarrow}_{s\cdot\beta\gamma}$. Hence, we will show that $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\beta\gamma, u, s)$. Assume that this is not the case and we have $\mathrm{dom}(a) \in \mathrm{dsrc}(a\beta\gamma, u, s)$. Let δ be the prefix of minimal length of γ such that there is some $v \in D$ with $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\beta b\delta, v, s)$, but $\mathrm{dom}(a) \in \mathrm{dsrc}(a\beta\delta, v, s)$. Such a trace δ and an agent v exist, since one can take $\delta = \gamma$ and $v = u$. Similar to the previous case, we can split δ in $\delta = \mu c\nu$ with $c \in A$ and $\mu, \nu \in A^*$ such that $\mathrm{dom}(a) \in \mathrm{dsrc}(a\beta\mu, \mathrm{dom}(c), s)$ and $\mathrm{dom}(c) \rightarrowtail_{s\cdot a\beta\mu} v$. Since $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\beta b\delta, v, s)$ and $\mathrm{dom}(a) \rightarrowtail_s \mathrm{dom}(b)$, it follows $\mathrm{dom}(b) \notin \mathrm{dsrc}(b\delta, v, s \cdot a\beta)$. Since μ is a prefix of δ, we have $\mathrm{dom}(b) \notin \mathrm{dsrc}(b\mu, v, s \cdot a\beta)$. The minimality of α implies $v^{\leftarrow}_{s\cdot a\beta b\mu} = v^{\leftarrow}_{s\cdot a\beta\mu}$ and $\mathrm{dom}(c) \rightarrowtail_{s\cdot a\beta b\mu} v$. From $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\beta b\delta, v, s)$, it follows $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\beta b\mu, \mathrm{dom}(c), s)$.
We also have $\mathrm{dom}(a) \in \mathrm{dsrc}(a\beta\mu, \mathrm{dom}(c), s)$.
This contradicts the minimality of δ, since μ is shorter than δ. □

The next theorem shows that the existence of an unwinding is both necessary and sufficient for systems with intransitively uniform policies.

4.5.20 Theorem. *A dynamic policy of a system is intransitively uniform iff for every $u, v \in D$ and every $\tilde{s} \in S$, there is a state-based uniform dynamic intransitive unwinding relation $\sim^{\mathsf{sudi}}_{\tilde{s},v,u}$.*

Proof. For the proof from left to right, assume that the system has an intransitively uniform dynamic policy. Define for every $u, v \in D$ and every $\tilde{s} \in S$ the following relation for every $s, t \in S$:

$$s \sim^{\mathsf{sudi}}_{\tilde{s},v,u} t \text{ iff for every trace } \alpha \text{ that does no contain any } b \text{ with } v \rightarrowtail_{\tilde{s}} \mathrm{dom}(b), \text{ we have } u^{\leftarrow}_{s\cdot\alpha} = u^{\leftarrow}_{t\cdot\alpha} \ .$$

Clearly, $\sim^{\mathsf{sudi}}_{\tilde{s},v,u}$ is an equivalence relation and it satisfies ($\mathrm{PC}^{\mathsf{sudi}}$) since ϵ is a valid value for α.

For showing ($\mathrm{LR}^{\mathsf{sudi}}$), let $a \in A$, set $v = \mathrm{dom}(a)$ and let $\tilde{s} \in S$ with $v \not\rightarrowtail_{\tilde{s}} u$. Let $\alpha \in A^*$ such that there is no $b \in A$ which appears in α with $v \rightarrowtail_{\tilde{s}} \mathrm{dom}(b)$. Since we have $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\alpha, u, \tilde{s})$, it follows by the definition of uniformity that $u^{\leftarrow}_{s\cdot a\alpha} = u^{\leftarrow}_{s\cdot\alpha}$.

For showing ($\mathrm{SC}^{\mathsf{sudi}}$), let $s \sim^{\mathsf{sudi}}_{\tilde{s},v,u} t$ and $a \in A$ with $v \not\rightarrowtail_{\tilde{s}} \mathrm{dom}(a)$. Let α be a trace that does not contain any action b with $v \rightarrowtail_{\tilde{s}} \mathrm{dom}(b)$. Since $v \not\rightarrowtail_{\tilde{s}} \mathrm{dom}(a)$, also the trace $a\alpha$ has this property. Hence we have $u^{\leftarrow}_{s\cdot a\alpha} = u^{\leftarrow}_{t\cdot a\alpha}$. Therefore, we have $s \cdot a \sim^{\mathsf{sudi}}_{\tilde{s},v,u} t \cdot a$.

For proving the other direction, assume that for every state $\tilde{s}$ and every agent u and v there is a state-based uniform dynamic intransitive unwinding relation $\sim^{\mathsf{sudi}}_{\tilde{s},v,u}$. For contradiction, additionally, assume the system does not have an intransitive uniform dynamic policy. Due to Lemma 4.5.19, there are $u \in D$, $s \in S$, and $\alpha \in A^*$ with $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\alpha, u, s)$, α does not contain b with $\mathrm{dom}(a) \rightarrowtail_s \mathrm{dom}(b)$, and $u^{\leftarrow}_{s\cdot a\alpha} \neq u^{\leftarrow}_{s\cdot\alpha}$. Let $v = \mathrm{dom}(a)$ and let $\sim^{\mathsf{sudi}}_{s,v,u}$ be a state-based uniform dynamic intransitive unwinding relation. Since we have $\mathrm{dom}(a) \not\rightarrowtail_s u$, it follows from ($\mathrm{LR}^{\mathsf{sudi}}$) $s \sim^{\mathsf{sudi}}_{s,v,u} s \cdot a$. Then, since α does not contain b with $\mathrm{dom}(a) \rightarrowtail_s \mathrm{dom}(b)$, it follows with ($\mathrm{SC}^{\mathsf{sudi}}$) that $s \cdot \alpha \sim^{\mathsf{sudi}}_{s,v,u} s \cdot a\alpha$. Therefore, $u^{\leftarrow}_{s\cdot a\alpha} \neq u^{\leftarrow}_{s\cdot\alpha}$ contradicts the ($\mathrm{PC}^{\mathsf{sudi}}$) condition. □

The soundness and completeness of the unwinding conditions for systems with intransitive uniform policies is provided by the following theorem.

4.5.21 Theorem. *A system with an intransitive uniform policy is di-secure iff for every $u, v \in D$ and every $\tilde{s} \in S$, there is a state-based uniform dynamic intransitive unwinding relation $\sim^{\mathsf{sudi}}_{\tilde{s},v,u}$ that is observation consistent for u.*

Proof. For proving the direction from left to right, suppose that the system is di-secure. For every state $\tilde{s}$ and all agents $u, v \in D$ define a relation $\sim^{\mathsf{sudi}}_{\tilde{s},v,u}$ for every $s, t \in S$ by

$$s \sim^{\mathsf{sudi}}_{\tilde{s},v,u} t \text{ iff for every } \alpha \in A^* \text{ that contain no } b \in A \text{ with } v \rightarrowtail_{\tilde{s}} \mathrm{dom}(b), \\ \text{we have } \mathrm{obs}_u(s \cdot \alpha) = \mathrm{obs}_u(t \cdot \alpha) \ .$$

Clearly, this relation is an equivalence relation on the states and it is observation consistent for u by choosing $\alpha = \epsilon$.

For showing ($\mathrm{LR}^{\mathsf{sudi}}$), let $a \in A$ and suppose that $v = \mathrm{dom}(a)$ and $v \not\rightarrowtail_{\tilde{s}} u$. Let α be in A^* such that it does not contain an action b with $v \rightarrowtail_{\tilde{s}} \mathrm{dom}(b)$. Hence we have $v = \mathrm{dom}(a) \notin \mathrm{dsrc}(a\alpha, u, \tilde{s})$. Since the system is di-secure, we have $\mathrm{obs}_u(\tilde{s} \cdot a\alpha) = \mathrm{obs}_u(\tilde{s} \cdot \alpha)$ and hence $\tilde{s} \sim^{\mathsf{sudi}}_{\tilde{s},v,u} \tilde{s} \cdot a$.

For showing ($\mathrm{SC}^{\mathsf{sudi}}$), let $s, t \in S$ with $s \sim^{\mathsf{sudi}}_{\tilde{s},v,u} t$ and $a \in A$ with $v \not\rightarrowtail_{\tilde{s}} \mathrm{dom}(a)$. Let α be in A^* such that it does not contain an action b with $v \rightarrowtail_{\tilde{s}} \mathrm{dom}(b)$. Then also the trace $a\alpha$ has the property that it does not contain an action b with $v \rightarrowtail_{\tilde{s}} \mathrm{dom}(b)$. Therefore, we have $\mathrm{obs}_u(s \cdot a\alpha) = \mathrm{obs}_u(t \cdot a\alpha)$ and hence $s \cdot a \sim^{\mathsf{sudi}}_{\tilde{s},v,u} t \cdot a$.

For showing the other direction of the proof, assume that the system is *not* di-secure. Due to Lemma 4.5.18, there are $u \in D$, $s \in S$, $a \in A$, and $\alpha \in A^*$ with $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\alpha, u, s)$ and $\mathrm{dom}(a) \not\rightarrowtail_s \mathrm{dom}(\mathrm{alph}(\alpha))$ and $\mathrm{obs}_u(s \cdot \alpha) \neq \mathrm{obs}_u(s \cdot a\alpha)$. Set $v = \mathrm{dom}(a)$ and let $\sim^{\mathsf{sudi}}_{s,v,u}$ be an equivalence relation that satisfies ($\mathrm{LR}^{\mathsf{sudi}}$) and ($\mathrm{SC}^{\mathsf{sudi}}$). Since, clearly, $v \not\rightarrowtail_s u$, we have $s \sim^{\mathsf{sudi}}_{s,v,u} s \cdot a$ by ($\mathrm{LR}^{\mathsf{sudi}}$). By the property $\mathrm{dom}(a) \not\rightarrowtail_s \mathrm{dom}(\mathrm{alph}(\alpha))$ and the repeated application of ($\mathrm{SC}^{\mathsf{sudi}}$), we have $s \cdot \alpha \sim^{\mathsf{sudi}}_{s,v,t} s \cdot a\alpha$. From $\mathrm{obs}_u(s \cdot \alpha) \neq \mathrm{obs}_u(s \cdot a\alpha)$ it follows that $\sim^{\mathsf{sudi}}_{s,v,u}$ is not observation consistent for u. □

4.6 dta-security

As we have seen in the previous chapter about static noninterference, i-security has some disadvantages compared to ta-security, since i-security allows too much information flow in a complete asynchronous setting. The same problem appears in the dynamic setting, when all policies are the same in all states and we essentially have a global policy, and hence di-security and i-security are equivalent. Therefore, it is desirable to have a definition similar to ta-security for the dynamic setting which captures this issue. We will provide such a definition which we call dta-security by a trace-based unwinding, which can also be expressed by a dynamic version of the ta operator if the dynamic policy satisfies some consistency property. This consistency property of the policy is a weaker condition than the requirement of intransitively uniform policies. From this perspective dta-security is more restrictive than di-security.

However, di-security only allows transmission of information about *performed* actions. But it does not allow one to transmit information about *not performed* actions. We will see that dta-security will allow this kind of transmission of negative information. Hence, from this point of view dta-security is less restrictive than di-security. In Section 4.7, we will provide examples, which illustrate these issues and show that neither dta-security implies di-security nor di-security implies dta-security.

We will provide a dynamic ta operator, called dta which requires restrictions on the structure of the dynamic policy. We call these dynamic policies ta-consistent policies. Intuitively, dynamic policies are ta-consistent if every two agents know, in the sense of distributed knowledge, whether there is a local edge between them or not, after performing every particular trace.

Instead of defining dta-security by the use of a ta-like operator, we use a definition based on trace-based unwinding. We will see that this definition is more general than a definition by a ta-like operator since the latter requires some restriction on the structure of the dynamic policy.

4.6.1 Definition (trace-based unwinding for dta-security)**.** A *trace-based unwinding relation for dta-security* for an agent u is an equivalence relation $\sim_u^{\mathsf{tdta}} \subseteq A^* \times A^*$ that satisfies for every $a \in A$ and every $\alpha, \beta \in A^*$ the following conditions:

(LR$^{\mathsf{tdta}}$): If $\mathrm{dom}(a) \not\rightarrow_{s^I \cdot \alpha} u$, then $\alpha \sim_u^{\mathsf{tdta}} \alpha a$.

(SC$^{\mathsf{tdta}}$): If $\alpha \sim_u^{\mathsf{tdta}} \beta$ and $\alpha \sim_{\mathrm{dom}(a)}^{\mathsf{tdta}} \beta$, then $\alpha a \sim_u^{\mathsf{tdta}} \beta a$.

These unwinding conditions are very similar to those of the trace-based unwinding for ta-security. The only difference is that in the condition (LR$^{\mathsf{tdta}}$), the policy depends on the particular state where it is applied. This also shows that this is a very natural adaption of ta-security to the dynamic setting.

We define dta-secure by the existence of a trace-based unwinding instead of using a trace operator.

4.6.2 Definition (dta-security)**.** A system with a dynamic policy is *dta-secure* if for every agent u, there exists a trace-based unwinding relation $\sim_u^{\mathsf{tdta}}$ for dta-security that is observation consistent for u.

Next, we adapt the ta operator to the dynamic setting in a straightforward way. However, we will see that it does not lead to a reasonable security definition without further restrictions. The most obvious generalization of the ta operator is to apply it to the local policy in the current state.

4.6.3 Definition (dta operator)**.** For every agent $u \in D$, every action $a \in A$ and every trace $\alpha \in A^*$ define

$$\mathrm{dta}_u(\epsilon) = \epsilon$$

$$\mathrm{dta}_u(\alpha a) = \begin{cases} (\mathrm{dta}_u(\alpha), \mathrm{dta}_{\mathrm{dom}(a)}(\alpha), u) & \text{if } \mathrm{dom}(a) \rightarrowtail_{s^I \cdot \alpha} u \\ \mathrm{dta}_u(\alpha) & \text{otherwise .} \end{cases}$$

The next definition looks very similar to the definition of ta-security in the static case. But in the dynamic case, we do not use it as a security definition on its own and denote it as a consistency property.

4.6.4 Definition (dta-observation-consistency)**.** A system is dta-*observation-consistent* iff for every $u \in D$, $s \in S$, and $\alpha, \beta \in A^*$ the following condition holds:

$$\text{If } \mathrm{dta}_u(\alpha) = \mathrm{dta}_u(\beta), \text{ then } \mathrm{obs}_u(s^I \cdot \alpha) = \mathrm{obs}_u(s^I \cdot \beta) \ .$$

In the next example, we discuss why applying dta-observation-consistency does not lead to a reasonable security definition.

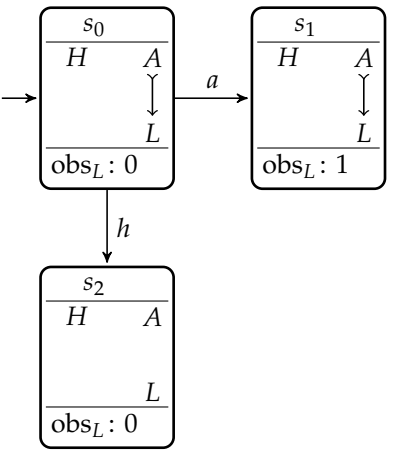

Figure 4.8. A dta-observation-consistent, but intuitively insecure system

4.6.5 Example. Let us have a closer look at the system in Figure 4.8. The local policies of the system never allow any information flow from agent H to any other agent. However, H's action h enforces a change of the local policy and forbids the previously allowed information flow from A to L. By performing the action h the allowed information flow from A to L is revoked. However, L can deduce, by observing 1, that H has not performed an action in the initial state. Hence, since L gains information about H's performed action, one should consider this system as insecure.

The only two (shortest) interesting traces in this system according to security are the traces a and ha. The values of the dta operator are

$$\begin{aligned} \text{dta}_L(a) &= (\epsilon, \epsilon, a) \text{ and} \\ \text{dta}_L(ha) &= \epsilon \ . \end{aligned}$$

Due to the different values of the dta operator of these two traces, and more generally, of any two traces which lead to different observations for L, the

system is dta-observation-consistent—a contradiction to our intuition.

This system is not only insecure according to our intuition, it is also not dta-secure. We have $\epsilon \sim_A^{\mathsf{tdta}} h$ and $\epsilon \sim_L^{\mathsf{tdta}} h$ by ($\mathrm{LR}^{\mathsf{tdta}}$) and, by ($\mathrm{SC}^{\mathsf{tdta}}$), $a \sim_L^{\mathsf{tdta}} ha$. Therefore the relation $\sim_L^{\mathsf{tdta}}$ is not observation consistent for L and hence the system is insecure.

We solve the problem of the discrepancy between dta-security and dta-observation-consistency by restricting the structure of the dynamic policy. The restriction is that the presence (or absence) of an edge between any two agents only depends on the dta values of these two agents. In the terminology of knowledge—here, we mean, as usual, the maximal allowed knowledge—every two agents always know, in the sense of distributed knowledge, whether there is an edge between them or not. We call this property of a dynamic policy dta-consistency.

4.6.6 Definition (dta-consistency)**.** A dynamic policy $(\rightarrowtail_s)_{s\in S}$ is dta-consistent iff for every $u, v \in D$ and every $\alpha, \beta \in A^*$ with $\mathrm{dta}_u(\alpha) = \mathrm{dta}_u(\beta)$ and $\mathrm{dta}_v(\alpha) = \mathrm{dta}_v(\beta)$, we have

$$u \rightarrowtail_{s^I \cdot \alpha} v \text{ iff } u \rightarrowtail_{s^I \cdot \beta} v \ .$$

4.6.7 Example. We continue with the system of Example 4.6.5. For the agents L and A, both agents have the same dta-value after the trace ϵ and the trace h, but after the first trace there is an edge from A to L and after the latter trace, this edge is not present. Hence, this system does not have a dta-consistent dynamic policy.

Next we show that in systems with a dta-consistent policy, dta-observation-consistency and dta-security are equivalent. The main part of this result comes from the next lemma.

4.6.8 Lemma. *Consider a system with a* dta*-consistent dynamic policy and let, for every $u \in D$, $\sim_u^{\mathsf{tdta}}$ be the smallest trace-based unwinding relation for dta-security. Then we have $\alpha \sim_u^{\mathsf{tdta}} \beta$ iff $\mathrm{dta}_u(\alpha) = \mathrm{dta}_u(\beta)$.*

Proof. First, we proof the implication from left to right. We can assume that the smallest trace-based unwinding relation is inductively created by applying the ($\mathrm{LR}^{\mathsf{tdta}}$) and ($\mathrm{SC}^{\mathsf{tdta}}$) conditions iteratively.

Case 1: Suppose $\alpha \sim_u^{\mathsf{tdta}} \beta$ is created from the ($\mathrm{LR}^{\mathsf{tdta}}$) condition.
Hence $\beta = \alpha a$ for some $a \in A$ with $\mathrm{dom}(a) \not\rightarrowtail_{s^I \cdot \alpha} a$. We have $\mathrm{dta}_u(\beta) = \mathrm{dta}_u(\alpha a) = \mathrm{dta}_u(\alpha)$ by the definition of the dta operator.
Case 2: Suppose α and β are constructed from the ($\mathrm{SC}^{\mathsf{tdta}}$) condition.
Then $\alpha = \alpha' a$ and $\beta = \beta' a$ with $\alpha' \sim_u^{\mathsf{tdta}} \beta'$ and $\alpha' \sim_{\mathrm{dom}(a)}^{\mathsf{tdta}} \beta'$. By induction hypothesis, we have $\mathrm{dta}_u(\alpha') = \mathrm{dta}_u(\beta')$ and $\mathrm{dta}_{\mathrm{dom}(a)}(\alpha') = \mathrm{dta}_{\mathrm{dom}(a)}(\beta')$. Since the policy is dta-consistent, we have $\mathrm{dom}(a) \rightarrowtail_{s^I \cdot \alpha'} u$ if and only if $\mathrm{dom}(a) \rightarrowtail_{s^I \cdot \beta'} u$. By definition of the dta operator, we have $\mathrm{dta}_u(\alpha) = \mathrm{dta}_u(\beta)$.

We proceed with the other direction of the proof. We prove this direction by an induction on the combined length of α and β. For the inductive step, let $\alpha = \alpha' a$ for some $a \in A$ and $\alpha' \in A^*$.
Case 1: $\mathrm{dom}(a) \not\rightarrowtail_{s^I \cdot \alpha'} u$.
Then we have $\mathrm{dta}_u(\alpha) = \mathrm{dta}_u(\alpha') = \mathrm{dta}_u(\beta)$. By induction hypothesis, we have $\alpha' \sim_u \beta$ and by applying the ($\mathrm{LR}^{\mathsf{tdta}}$) condition, we have $\alpha \sim_u^{\mathsf{tdta}} \beta$.
Case 2: $\mathrm{dom}(a) \rightarrowtail_{s^I \cdot \alpha'} u$.
In the case of $\beta = \beta' b$ with $\mathrm{dom}(b) \not\rightarrowtail_{s^I \cdot \beta'} u$, we could proceed with the first case with the roles of a and b swapped. Hence we assume that $\mathrm{dom}(b) \rightarrowtail_{s^I \cdot \beta'} u$. By the definition of the dta operator, we have $a = b$, $\mathrm{dta}_u(\alpha') = \mathrm{dta}_u(\beta')$, and $\mathrm{dta}_{\mathrm{dom}(a)}(\alpha') = \mathrm{dta}_{\mathrm{dom}(a)}(\beta')$. By induction, we have $\alpha' \sim_u^{\mathsf{tdta}} \beta'$ and $\alpha' \sim_{\mathrm{dom}(a)}^{\mathsf{tdta}} \beta'$ and by ($\mathrm{SC}^{\mathsf{tdta}}$) $\alpha' a \sim_u^{\mathsf{tdta}} \beta' b$. □

We summarize the connection between dta-consistency, dta-observation-consistency, and dta-security.

4.6.9 Corollary. *A system with a* dta*-consistent policy is* dta*-observation-consistent iff it is dta-secure.*

The next theorem shows the completeness of this approach by showing that by removing edges, every arbitrary dynamic policy can be translated into a dta-consistent dynamic policy without affecting dta-security. This shows that dta-consistency is the missing condition for achieving dta-security with the dta operator.

4.6.10 Theorem. *For every system M with states S and a dynamic policy $(\rightarrowtail_s)_{s \in S}$, there is an* dta*-consistent policy $(\rightarrowtail'_s)_{s \in S}$ such that M is dta-secure*

w.r.t. $(\rightarrowtail_s)_{s\in S}$ *iff* M' *is dta-secure w.r.t.* $(\rightarrowtail'_s)_{s\in S}$. *Moreover, the policy* $(\rightarrowtail'_s)_{s\in S}$ *is at least as restrictive as* $(\rightarrowtail_s)_{s\in S}$.

Proof. For every $u \in D$ let $\sim_u^{\mathsf{tdta}}$ be the smallest trace-based unwinding relation satisfying ($\mathrm{LR}^{\mathsf{tdta}}$) and ($\mathrm{SC}^{\mathsf{tdt}}$) w.r.t. the policy $(\rightarrowtail_s)_{s\in S}$. Define the policy $(\rightarrowtail'_s)_{s\in S}$ by

$$u \rightarrowtail'_{s^I\cdot\alpha} v \text{ iff for every } \beta \in [\alpha]_{\sim_u^{\mathsf{tdta}}} \cap [\alpha]_{\sim_v^{\mathsf{tdta}}} \text{ it holds } u \rightarrowtail_{s^I\cdot\beta} v \ .$$

From this definition, it is clear that the policy $(\rightarrowtail'_s)_{s\in S}$ is at least as restrictive as the policy $(\rightarrowtail_s)_{s\in S}$.

For every u let $\approx_u^{\mathsf{tdta}}$ be the smallest trace-based unwinding relation that satisfies ($\mathrm{LR}^{\mathsf{tdta}}$) and ($\mathrm{SC}^{\mathsf{tdta}}$) w.r.t. the policy $(\rightarrowtail'_s)_{s\in S}$. Let u be in D. We claim that $\sim_u^{\mathsf{tdta}} = \approx_u^{\mathsf{tdta}}$ holds. It is clear that $\sim_u^{\mathsf{tdta}} \subseteq \approx_u^{\mathsf{tdta}}$ holds, since $(\rightarrowtail'_s)_{s\in S}$ is at least as restrictive as $(\rightarrowtail_s)_{s\in S}$. By an induction on the inductive definition of $\approx_u^{\mathsf{tdta}}$, we will show for every $u \in D$ and every $\alpha, \beta \in A^*$ that $\alpha \approx_u^{\mathsf{tdta}} \beta$ implies $\alpha \sim_u^{\mathsf{tdta}} \beta$.

Let $\alpha = \alpha' a$ for some $a \in A$ and $\alpha' \in A^*$ and assume inductively that the claim holds for all shorter traces and all agents. Since $\sim_u^{\mathsf{tdta}}$ is the smallest unwinding relation, we can assume that $\sim_u^{\mathsf{tdta}}$ is iteratively constructed by applying the ($\mathrm{LR}^{\mathsf{tdta}}$) and ($\mathrm{SC}^{\mathsf{tdta}}$) conditions. Hence, it is sufficient to consider traces that are constructed by applying one of these conditions.
Case 1: α and β are constructed from the ($\mathrm{LR}^{\mathsf{tdta}}$) condition of $\approx_u^{\mathsf{tdta}}$.
Here we have $\alpha' = \beta$ and $\mathrm{dom}(a) \not\rightarrowtail'_{s^I\cdot\alpha'} u$. From $\alpha' a \approx_u^{\mathsf{tdta}} \alpha' \approx_u^{\mathsf{tdta}} \beta$ it follows by induction hypothesis $\alpha' \sim_u^{\mathsf{tdta}} \beta$. Since $\mathrm{dom}(a) \not\rightarrowtail'_{s^I\cdot\alpha'} u$, there is some $\gamma \in [\alpha']_{\sim_u^{\mathsf{tdta}}} \cap [\alpha']_{\sim^{\mathsf{tdta}}_{\mathrm{dom}(a)}}$ with $\mathrm{dom}(a) \not\rightarrowtail_{s^I\cdot\gamma} u$. From $\alpha' \sim_u^{\mathsf{tdta}} \gamma$ and $\alpha' \sim^{\mathsf{tdta}}_{\mathrm{dom}(a)} \gamma$ it follows by ($\mathrm{SC}^{\mathsf{tdta}}$) that $\alpha' a \sim_u^{\mathsf{tdta}} \gamma a$. From $\mathrm{dom}(a) \not\rightarrowtail_{s^I\cdot\gamma} u$ it follows by ($\mathrm{LR}^{\mathsf{tdta}}$) that $\gamma a \sim_u^{\mathsf{tdta}} \gamma$. By combination, we get $\alpha' a \sim_u^{\mathsf{tdta}} \alpha'$ and hence $\alpha \sim_u^{\mathsf{tdta}} \beta$.
Case 2: α and β are constructed from the ($\mathrm{SC}^{\mathsf{tdta}}$) condition of $\approx_u^{\mathsf{tdta}}$.
Note that the ($\mathrm{SC}^{\mathsf{tdta}}$) condition is independent of the policy. Hence we have $\alpha = \alpha' a$ and $\beta = \beta' a$ with $\alpha' \approx_u^{\mathsf{tdta}} \beta'$ and $\alpha' \approx^{\mathsf{tdta}}_{\mathrm{dom}(a)} \beta'$. By applying the induction hypothesis, we have $\alpha' \sim_u^{\mathsf{tdta}} \beta'$ and $\alpha' \sim^{\mathsf{tdta}}_{\mathrm{dom}(a)} \beta'$ and by the ($\mathrm{SC}^{\mathsf{tdta}}$) condition applied to $\sim_u^{\mathsf{tdta}}$, we have $\alpha' a \sim_u^{\mathsf{tdta}} \beta' a$. We have shown

that the system is dta-secure w.r.t. $(\rightarrowtail_s)_{s\in S}$ if and only if it is dta-secure w.r.t. $(\rightarrowtail'_s)_{s\in S}$.

It remains to show that the policy $(\rightarrowtail'_s)_{s\in S}$ is dta-consistent. First, we will show for every $u \in D$ and every $\alpha, \beta \in A^*$ that $\mathrm{dta}_u(\alpha) = \mathrm{dta}_u(\beta)$ (w.r.t. $(\rightarrowtail'_s)_{s\in S}$) implies $\alpha \approx_u^{\mathsf{tdta}} \beta$. We prove this by an induction on the combined length of α and β. Let $\alpha = \alpha' a$ and suppose that the claim holds for all traces of smaller combined length.
Case 1: $\mathrm{dom}(a) \not\rightarrowtail'_{s^I\cdot\alpha'} u$.
In this case, we have $\mathrm{dta}_u(\alpha' a) = \mathrm{dta}_u(\alpha')$. By induction hypothesis, we have $\alpha' \approx_u^{\mathsf{tdta}} \beta$ and by $(\mathrm{LR}^{\mathsf{tdta}})$ $\alpha' a \approx_u^{\mathsf{tdta}} \alpha' \approx_u^{\mathsf{tdta}} \beta$.
Case 2: $\mathrm{dom}(a) \rightarrowtail'_{s^I\cdot\alpha'} u$.
Then we can assume with the same argument as usual that $\beta = \beta' b$ holds for some $b \in A$ and $\beta' \in A^*$ with $\mathrm{dom}(b) \rightarrowtail'_{s^I\cdot\beta'} u$. Then we have $\mathrm{dta}_u(\alpha') = \mathrm{dta}_u(\beta')$, $\mathrm{dta}_{\mathrm{dom}(a)}(\alpha') = \mathrm{dta}_{\mathrm{dom}(a)}(\beta')$, and $a = b$. By induction hypothesis we have $\alpha' \sim_u^{\mathsf{tdta}} \beta'$ and $\alpha' \sim_{\mathrm{dom}(a)}^{\mathsf{tdta}} \beta'$ and by $(\mathrm{SC}^{\mathsf{tdta}})$, $\alpha' a \sim_u^{\mathsf{tdta}} \beta' a$.

Now we obtain the dta-consistency as follows: If $\mathrm{dta}_u(\alpha) = \mathrm{dta}_u(\beta)$ and $\mathrm{dta}_v(\alpha) = \mathrm{dta}_u(\beta)$ (w.r.t. $(\rightarrowtail'_s)_{s\in S}$), then we have $\alpha \in [\beta]_{\approx_u^{\mathsf{tdta}}} \cap [\beta]_{\approx_v^{\mathsf{tdta}}}$ and hence $u \rightarrowtail'_\alpha v$ iff $u \rightarrowtail'_\beta v$. □

4.6.1 Dynamic Intransitively Securely Constructed Systems

We provide more support for dta-consistent policies as they appear naturally in securely constructed systems in the dynamic intransitive case. We proceed with a definition of securely constructed systems. Similar to the dynamic transitive case, the edges of the dynamic polices depend only on the local states of the involved agents. Analogue to the static intransitive case, the information flow from an agent v to an agent u should only depend on v's and u's local state. Hence we require that the existence of a possible edge from v to u is uniquely determined by the local state of these two agents. Similar to the transitive case, we define local state dependent policies:

4.6.11 Definition (intransitively local state dependent policy)**.** For a system with local state spaces $S_0, \ldots S_{n-1}$ and global state space $S = S_0 \times \cdots \times S_{n-1}$, a dynamic policy $(\rightarrowtail_s)_{s\in S}$ is *intransitively local state dependent* if for every

$i, j \in D$ and every $s, t \in S$ with $\pi_j(s) = \pi_j(t)$ and $\pi_i(s) = \pi_i(t)$, we have

$$j \rightarrowtail_s i \text{ iff } j \rightarrowtail_t i \ .$$

Unlike in the transitive case, we require that the existence of an edge depends on both local states of i and of j, and not only on the local state of i. For the composition of these local transition functions to a global transition function, we require that the dynamic policy is intransitively local state dependent. With this restriction on the dynamic policy, the global transition function is defined by

$$(s_0, \ldots, s_{n-1}) \cdot a = (s'_0, \ldots, s'_{n-1})$$
$$\text{with } s'_i = \begin{cases} (s_{\mathrm{dom}(a)}, s_i) \cdot_i a & \text{if } \mathrm{dom}(a) \rightarrowtail_{(s_0,\ldots,s_{n-1})} i \\ s_i & \text{otherwise} \ . \end{cases}$$

As usual, we require that the observation function for each agent only depends on its local state. The next lemma provides the core argument why dynamic intransitively securely constructed systems are dta-secure.

4.6.12 Lemma. *In a dynamic intransitively securely constructed system, we have, for every $i \in D$ and every $\alpha, \beta \in A^*$ with $\mathrm{dta}_i(\alpha) = \mathrm{dta}_i(\beta)$, $\pi_i(s^I \cdot \alpha) = \pi_i(s^I \cdot \beta)$.*

Proof. Let $i \in D$. We prove this lemma by an induction on the combined length of α and β. Since the base case is clear, we proceed with the inductive step. Let $\alpha, \beta \in A^*$ with $\mathrm{dta}_i(\alpha) = \mathrm{dta}_i(\beta)$ and assume that for all traces with smaller combined length the claim holds. Since at least one of α and β is not the empty trace, we can assume, w.l.o.g., that $\alpha = \alpha' a$ for some $\alpha' \in A^*$ and $a \in A$.
Case 1: $\mathrm{dom}(a) \not\rightarrowtail_{s^I \cdot \alpha'} i$.
By the definition of intransitively securely constructed systems, we have $\pi_i(s^I \cdot \alpha' a) = \pi_i(s^I \cdot \alpha)$. From the definition of the dta operator it follows that $\mathrm{dta}_i(\alpha' a) = \mathrm{dta}_i(\alpha')$, and from $\mathrm{dta}_i(\alpha' a) = \mathrm{dta}_i(\beta)$, it follows by applying the the induction hypothesis that $\pi_i(s^I \cdot \alpha') = \pi_i(s^I \cdot \beta)$.
Case 2: $\mathrm{dom}(a) \rightarrowtail_{s^I \cdot \alpha'} i$.
We suppose that $\beta = \beta' b$ for some $\beta' \in A^*$ and $b \in A$ with $\mathrm{dom}(b) \rightarrowtail_{s^I \cdot \beta'} i$,

since otherwise we would proceed with Case 1 with the roles of α and β swapped. By the definition of the dta operator, we have $a = b$, $\mathrm{dta}_i(\alpha') = \mathrm{dta}_i(\beta')$, and $\mathrm{dta}_{\mathrm{dom}(a)}(\alpha') = \mathrm{dta}_{\mathrm{dom}(a)}(\beta')$. Applying the induction hypothesis gives $\pi_i(s^I \cdot \alpha') = \pi_i(s^I \cdot \beta')$ and $\pi_{\mathrm{dom}(a)}(s^I \cdot \alpha') = \pi_{\mathrm{dom}(b)}(s^I \cdot \beta')$. From the definition of the local transition function of an intransitively securely constructed system it follows

$$\begin{aligned}\pi_i(s^I \cdot \alpha' a) &= (\pi_{\mathrm{dom}(a)}(s^I \cdot \alpha'), \pi_u(s^I \cdot \alpha')) \cdot_i a \\ &= (\pi_{\mathrm{dom}(a)}(s^I \cdot \beta'), \pi_u(s^I \cdot \beta')) \cdot_i a \\ &= \pi_i(s^I \cdot \beta' b) \ . \end{aligned}$$ □

Since, by definition, every dynamic intransitively securely constructed system has an intransitively local state dependent policy, it follows from the previous lemma that such a system has a dta-consistent policy. Hence, for intransitively securely constructed systems, we do not need the requirement of a dta-consistent policy explicitly. The dependence of the observation functions on the local states follows immediately.

4.6.13 Corollary. *Every dynamic intransitively securely constructed system is dta-secure.*

The next result shows the completeness of expressing dta-secure systems as dynamic intransitively securely constructed systems.

4.6.14 Lemma. *For every dta-secure system, there exists an observation equivalent system which is dynamic intransitively securely constructed from a trace-equivalent policy.*

Proof. Let M be a dta-secure system with a dynamic policy $(\rightarrowtail_s)_{s \in S}$. According to Theorem 4.6.10, we can suppose that $(\rightarrowtail_s)_{s \in S}$ is dta-consistent. Let the agents be enumerated as $D = \{0, \ldots, n-1\}$. As the states of the constructed system, we take the dta-values of the corresponding agents. The new system is constructed as

- ▷ the local states of each agent i are $S_i = \{\mathrm{dta}_i(\alpha) \mid \alpha \in A^*\}$ and the global state space is as usual $S' = S_0 \times \cdots \times S_{n-1}$,

- ▷ the local initial state of each agent i is $s_i^I = \epsilon$,
- ▷ the edges of the dynamic polices $(\rightarrowtail'_s)_{s \in S'}$ are for every $i, j \in D$ and every $\alpha \in A^*$ defined by $j \rightarrowtail'_{s^I \cdot \alpha} i$ iff for every $\beta \in A^*$ with $\mathrm{dta}_i(\beta) = \mathrm{dta}_i(\alpha)$ it holds: $j \rightarrowtail_{s^I \cdot \beta} i$,
- ▷ for every agent i and every state s, we define the observation function $\mathrm{obs}'_i(s) = \mathrm{obs}_i(s^I \cdot \beta)$ for some $\beta \in A^*$ with $\beta = \pi_i(s)$.

Note that due to dta-observation-consistency, the observation function is well-defined. It also follows from dta-observation-consistency that the constructed system is observation equivalent to the system from which it is constructed.

Next, we show that the policy of the constructed system is intransitively local state dependent. Let $i, j \in D$ and $s, t \in S$ with $\pi_i(s) = \pi_i(t)$ and $\pi_j(t) = \pi_j(t)$. Then there are $\alpha, \beta \in A^*$ with

$$\begin{aligned} &\mathrm{dta}_i(\alpha) = \pi_i(s) = \pi_i(t) = \mathrm{dta}_i(\beta), \text{ and} \\ &\mathrm{dta}_j(\alpha) = \pi_j(s) = \pi_j(t) = \mathrm{dta}_j(\beta) \ . \end{aligned}$$

Because of the dta-consistency of the policy, the policy of the constructed system is intransitively local state dependent. Similarly, it follows that the systems have trace-equivalent policies. □

Summarizing the previous results give:

4.6.15 Theorem. *A system is dta-secure iff there exists an observation equivalent, dynamic intransitively securely constructed system.*

4.7 Relation between Dynamic Noninterference Definitions

We compare the different security definitions of this chapter with each other and explain how they are related to the security definitions of the previous chapter if the local policies are the same in all states.

Clearly, dt-security implies dot-security since the definitions are the same, except that in the definition of dot-security, there are further requirements on the trace α after the hidden action a. In Example 4.3.1, we showed that dot-security does not imply dt-security. That dot-security implies di-security follows directly from $\mathrm{dotsrc}(\alpha, u, s) \subseteq \mathrm{dsrc}(\alpha, u, s)$ for every $\alpha \in A^*$, $u \in D$, and $s \in S$. In contrast, di-security does not imply dot-security, since, applied to a system with a static policy, i-security does not imply t-security.

That dt-security implies dta-security can easily be seen by comparing the definition of a trace-based dynamic transitive unwinding with the definition of dta-security. Here, we have for every $u \in D$ and every $\alpha, \beta \in A^*$ that $\alpha \sim_u^{\mathsf{tdta}} \beta$ implies $\alpha \sim_u^{\mathsf{tdt}} \beta$. Again, in a system with a static policy, we have that ta-security does not imply t-security, and hence, dta-security does not imply dt-security.

The next example shows that neither dot-security nor di-security implies dta-security. The reason for that is essentially the same as in the static case: dot-security and di-security allow one to observe information about the ordering of actions which is not observable by any agent. According to dta-security this information about the order of actions is not allowed to be observed by any agent.

4.7.1 Example. We will show that the system depicted in Figure 4.9 is dot-secure, but L observes the order of the actions h_1 and h_2 in the case that there are not edges from H_1 and H_2 to L.

Computing the smallest state-based unwinding relations for dot-security gives:

$$
\begin{array}{ll}
s_0 \preceq^{\mathsf{sdot}}_{L,H_1} s_2 & s_0 \preceq^{\mathsf{sdot}}_{L,H_2} s_1 \\
s_1 \preceq^{\mathsf{sdot}}_{L,H_1} s_3 & s_2 \preceq^{\mathsf{sdot}}_{L,H_2} s_4 \\
s_1 \preceq^{\mathsf{sdot}}_{L,H_1} s_4 & s_2 \preceq^{\mathsf{sdot}}_{L,H_2} s_3 \\
 & s_2 \preceq^{\mathsf{sdot}}_{L,H_2} s_5 \\
 & s_2 \preceq^{\mathsf{sdot}}_{L,H_2} s_6
\end{array}
$$

Since for all pairs of states in this relation, the observation of L is 0, the system is dot-secure and hence di-secure. However, L should not be able

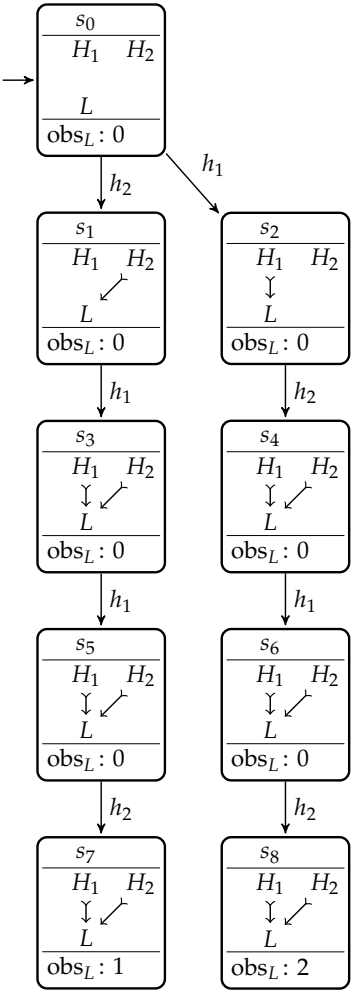

Figure 4.9. A dot-secure, but *not* dta-secure system

to observe, which of the agents H_1 and H_2 had performed the first action in the traces $h_1h_2h_1h_2$ and $h_2h_1h_1h_2$. This is captured by the definition of dta-security. Computing several elements of the unwinding relations $\sim_L^{\mathsf{tdta}}$, $\sim_{H_1}^{\mathsf{tdta}}$, and $\sim_{H_2}^{\mathsf{tdta}}$ is needed to show the equivalence of the traces $h_1h_2h_1h_2$ and $h_2h_1h_1h_2$ for L. We obtain:

$$\begin{aligned}
\epsilon &\sim_L^{\mathsf{tdta}} h_1 \\
\epsilon &\sim_L^{\mathsf{tdta}} h_2 \\
h_2 &\sim_L^{\mathsf{tdta}} h_1h_2 \\
h_1 &\sim_L^{\mathsf{tdta}} h_2h_1 \\
h_1h_2 &\sim_L^{\mathsf{tdta}} h_2h_1 \\
h_1h_2h_1 &\sim_L^{\mathsf{tdta}} h_2h_1h_1 \\
h_1h_2h_1h_2 &\sim_L^{\mathsf{tdta}} h_2h_1h_1h_2
\end{aligned}
\qquad
\begin{aligned}
\epsilon &\sim_{H_1}^{\mathsf{tdta}} h_2 \\
h_1 &\sim_{H_1}^{\mathsf{tdta}} h_2h_1 \\
h_1 &\sim_{H_1}^{\mathsf{tdta}} h_1h_2 \\
h_1h_2 &\sim_{H_2}^{\mathsf{tdta}} h_2h_1
\end{aligned}
\qquad
\begin{aligned}
\epsilon &\sim_{H_2}^{\mathsf{tdta}} h_1 \\
h_2 &\sim_{H_2}^{\mathsf{tdta}} h_1h_2 \\
h_2 &\sim_{H_2}^{\mathsf{tdta}} h_2h_1 \\
h_1h_2 &\sim_{H_2}^{\mathsf{tdta}} h_2h_1 \\
h_1h_2h_1 &\sim_{H_2}^{\mathsf{tdta}} h_2h_1h_2
\end{aligned}$$

From $\mathrm{obs}_L(s_0 \cdot h_1h_2h_1h_2) = 2 \neq 1 = \mathrm{obs}_L(s_0 \cdot h_2h_1h_1h_2)$ it follows that this system is not dta-secure.

The next example shows that dta-security does not imply di-security and hence not dot-security. This example shows that dta-security allows the transmission of information about not performed actions, which is not allowed by di-security.

4.7.2 Example. The system in Figure 4.10 is clearly *not* di-secure. We consider the traces hd and d since the dipurge values of these two traces are the same for L if one starts purging in s_0:

$$\mathrm{dipurge}(hd, L, s_0) = \mathrm{dipurge}(d, L, s_0) = d \ .$$

But the observations after these two traces are different for L.

For the trace-based equivalence relations of the definition of dta-security, we have that D is allowed to observe every action in every state. Hence the equivalence classes of the $\sim_D^{\mathsf{tdta}}$ are singleton sets and hence from the perspective of L, we cannot apply the ($\mathrm{SC}^{\mathsf{tdta}}$) condition with the d actions. We have $\epsilon \sim_L^{\mathsf{tdta}} h \sim_L^{\mathsf{tdta}} hd$, but it is not possible that the trace d is equivalent to any other trace that does not start with the action d. Therefore,

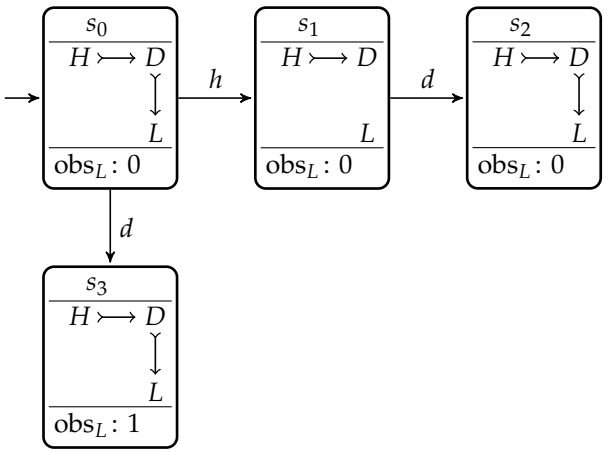

Figure 4.10. A dta-secure, but *not* di-secure system

the system is dta-secure.

Figure 4.11 summarizes the implications of the noninterference definitions for a setting with dynamic policies.

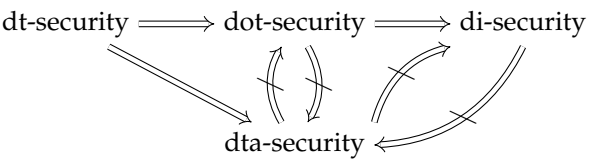

Figure 4.11. Relations between our dynamic noninterference definitions

If we have a system with a dynamic policy that has the same local policy in all states, we can interpret it as a system with a global policy and we can apply the security definitions of the previous chapter to it. From the definition of the dynamic notions of security it directly follows that t-security is equivalent to dt-security, i-security is equivalent to di-security, and ta-security is equivalent to dta-security. The most notable equivalence is that in the static case dot-security is also equivalent to t-security. This is

interesting, because the effects of intransitivity and downgrading disappear if one shifts from the dynamic to the static case.

4.8 Comparison with Leslie's Work

To the best of our knowledge, the first notion of intransitive noninterference for state-based systems with dynamic policies was introduced by Leslie [Les06]. In this section we will discuss her notion of security and compare it to ours. The system model used in Leslie's work is the one of Rushby [Rus92] and hence we can adapt her notion directly to our setting.

Instead of starting with the structurally easier case of transitive noninterference, Leslie adapted dynamic intransitive noninterference directly from the corresponding definition of static noninterference.

Leslie's definition of dynamic noninterference follows the same pattern as ours, however, there are slight, though important differences. Since we use the same definition for the dynamic sources function as in Leslie's work, we can directly state their dipurge definition, which we will denote here as Lpurge in order to have a clear distinction between their operator and our definition.

4.8.1 Definition (Leslie's dynamic intransitive purge operator [Les06])**.** For every $u \in D$, $s \in S$, $a \in A$, and $\alpha \in A^*$ define

$$\begin{aligned} \mathrm{Lpurge}(\epsilon, u, s) &= \epsilon \\ \mathrm{Lpurge}(a\alpha, u, s) &= \begin{cases} a\ \mathrm{Lpurge}(\alpha, u, s \cdot a) & \text{if } \mathrm{dom}(a) \in \mathrm{dsrc}(a\alpha, u, s) \\ \mathrm{Lpurge}(\alpha, u, s \cdot a) & \text{otherwise .} \end{cases} \end{aligned}$$

The only difference between the definition of Lpurge and dipurge is that in the case an action is purged, in this definition, the transition is performed and Lpurge proceeds in the state $s \cdot a$ instead of remaining in the state s, like the dipurge operator does. This is somehow counter intuitive, since removing an action from the purged trace means that the observation agent should not get any information about the fact that the action was in the trace. However, if we perform the transition, even in the case the action has

been purged, we still have some influence of the purged action in the result of the purged trace. Hence, the action is removed from the trace, but the fact that the action was in the trace can still have effects on the purging of the remaining actions in the trace. We will discuss this issue in one of the next examples after recalling Leslie's security definition. The corresponding security definition from [Les06], which we call L-security, is:

4.8.2 Definition (L-security [Les06])**.** A system is *L-secure* iff for every $u \in D$ and every $\alpha \in A^*$ it holds:

$$\mathrm{obs}_u(s^I \cdot \alpha) = \mathrm{obs}_u(s^I \cdot \mathrm{Lpurge}(\alpha, u, s^I)) \ .$$

Note that L-security only requires purging from the initial state rather than starting from an arbitrary state as required in di-security. First, we correct this issue by giving a stronger, persistent security definition based on Lpurge which requires this property.

4.8.3 Definition (L-per-security)**.** A system is *L-per-secure* iff for every $u \in D$, $\alpha \in A^*$ and every $s \in S$

$$\mathrm{obs}_u(s \cdot \alpha) = \mathrm{obs}_u(s \cdot \mathrm{Lpurge}(\alpha, u, s)) \ .$$

Clearly, from this definition it follows that L-per-security implies L-security. That L-per-security is indeed stricter then L-security is visible in the next example.

4.8.4 Example. In the system in Figure 4.12 the agent H has two actions, h_1 and h_2. The only interesting trace w.r.t. L-security is the trace h_1h_2d, which is purged to d and hence leads to the same observation, and the trace h_1d which is purged to ϵ and hence leads to the same observations, too. However, if one starts purging the trace h_2d in the state s_1 then the purged value is d which leads to different observations when starting from s_1.

With the next example, we show that L-per-security neither implies di-security nor dta-security.

4.8.5 Example. In this example, we explain why the system in Figure 4.13 is L-per-secure. We suppose that agent L has no actions. In this system, it is only necessary to consider purging from the initial state. Any trace starting

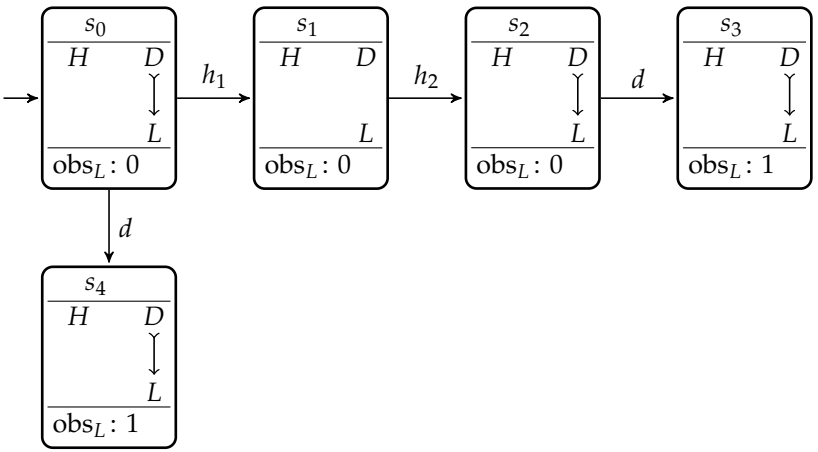

Figure 4.12. An L-secure, but *not* L-per-secure system

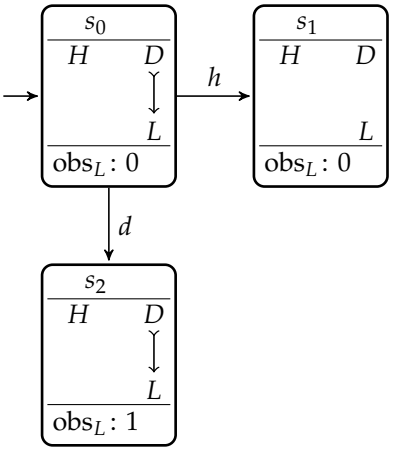

Figure 4.13. An L-per-secure, but neither di-secure nor dta-secure system

with an action h will be purged to ϵ, since after performing h, we are in the state s_1 and all remaining actions are purged and in the initial state also the action h is purged. For example for the trace hd, we have

$$\mathrm{Lpurge}(hd, L, s_0) = \mathrm{Lpurge}(d, L, s_1) = \epsilon \ .$$

But every trace starting with h or the empty trace ends in a state where L's observation is 0.

Conversely, every trace starting with an action d will be purged to a trace starting with d, since D is allowed to interfere with L in the initial state. But every trace starting with d ends in the state s_2 and therefore, on all of these traces, L has the same observation, namely 1.

As a consequence, the system is L-per-secure. However, if L observes 1, it knows that there was no action of H in the state s_0 which is some information about H. Intuitively, this contradicts the assumption that H is never allowed to interfere with any of the agents.

This system is *not* di-secure, since we have $\mathrm{dipurge}(hd, L, s_0) = d = \mathrm{dipurge}(d, L, s_0)$.

This system is *not* dta-secure, since we have by ($\mathrm{LR}^{\mathsf{tdta}}$) $\epsilon \sim_L^{\mathsf{tdta}} h$ and $\epsilon \sim_D^{\mathsf{tdta}} h$ and by ($\mathrm{SC}^{\mathsf{tdta}}$) $d \sim_L^{\mathsf{tdta}} hd$, and hence the system is not observation consistent for L.

Before we will show that di-security implies L-per-security and hence also L-security, we provide a technical lemma which states that the agents in the sources of a trace are exactly those whose actions are in the Lpurge value of the trace.

4.8.6 Lemma. *For every $u \in D$, $\alpha \in A^*$ and $s \in S$, we have*

$$\mathrm{dsrc}(\alpha, u, s) = \mathrm{dom}(\mathrm{alph}(\mathrm{Lpurge}(\alpha, u, s))) \ .$$

Proof. We prove this result by an induction on the length of α and suppose for the inductive step that $\alpha = a\alpha'$ holds for some $a \in A$ and $\alpha' \in A^*$. By induction hypothesis (I.H.), we assume that the claim holds for α' in all states.

Let s be a state. We distinguish these cases:
Case 1: $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\alpha', u, s)$.

Then we have

$$\begin{aligned}\mathrm{dom}(\mathrm{alph}(\mathrm{Lpurge}(a\alpha', u, s))) &= \mathrm{dom}(\mathrm{alph}(\mathrm{Lpurge}(\alpha', u, s \cdot a))) \\ &\overset{\text{(I.H.)}}{=} \mathrm{dsrc}(\alpha', u, s \cdot a) \\ &= \mathrm{dsrc}(a\alpha', u, s) \ .\end{aligned}$$

Case 2: $\mathrm{dom}(a) \in \mathrm{dsrc}(a\alpha', u, s)$.
Then we have

$$\begin{aligned}&\mathrm{dom}(\mathrm{alph}(\mathrm{Lpurge}(a\alpha', u, s))) \\ &\qquad = \mathrm{dom}(\mathrm{alph}(a\ \mathrm{Lpurge}(\alpha', u, s \cdot a))) \\ &\qquad = \mathrm{dom}(\{a\} \cup \mathrm{alph}(\mathrm{Lpurge}(\alpha', u, s \cdot a))) \\ &\qquad = \{\mathrm{dom}(a)\} \cup \mathrm{dom}(\mathrm{alph}(\mathrm{Lpurge}(\alpha', u, s \cdot a))) \\ &\qquad \overset{\text{(I.H.)}}{=} \{\mathrm{dom}(a)\} \cup \mathrm{dsrc}(\alpha', u, s \cdot a) \\ &\qquad = \mathrm{dsrc}(a\alpha', u, s) \ .\end{aligned}$$

□

With this result, we can now prove that di-security implies L-per-security.

4.8.7 Lemma. *Every di-secure system is L-per-secure.*

Proof. We prove this result by contraposition. Suppose we does not have a *not* L-per-secure system. Hence, there exist $u \in D$, $\alpha \in A^*$, and $s \in S$ with

$$\mathrm{obs}_u(s \cdot \alpha) \neq \mathrm{obs}_u(s \cdot \mathrm{Lpurge}(\alpha, u, s)) \ .$$

Additionally, we suppose that α is of minimal length for all possible choices of u and s . Since α is not the empty trace, we can assume that $\alpha = a\alpha'$ for some $a \in A$ and $\alpha' \in A^*$. From the minimality, it follows that the first action a is purged from the trace, i. e., $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\alpha', u, s)$ and hence $\mathrm{Lpurge}(a\alpha', u, s) = \mathrm{Lpurge}(\alpha', u, s \cdot a)$. Applying the observations to these purged traces after starting in the state s gives

$$\mathrm{obs}_u(s \cdot \mathrm{Lpurge}(\alpha', u, s \cdot a)) = \mathrm{obs}_u(s \cdot \mathrm{Lpurge}(a\alpha', u, s)) \ .$$

Since the trace α' is shorter than α, from the minimality of α, it follows

$$\mathrm{obs}_u(s \cdot a\alpha') = \mathrm{obs}_u(s \cdot a\ \mathrm{Lpurge}(\alpha', u, s \cdot a)) \ .$$

Because of the initial assumption, the values of the last two equations are different, which gives

$$\mathrm{obs}_u(s \cdot \mathrm{Lpurge}(\alpha', u, s \cdot a)) \neq \mathrm{obs}_u(s \cdot a\ \mathrm{Lpurge}(\alpha', u, s \cdot a)) \ .$$

It remains to show that $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\ \mathrm{Lpurge}(\alpha', u, s \cdot a), u, s)$. It follows from Lemma 4.8.6 that we have $a \notin \mathrm{alph}(\mathrm{Lpurge}(a\alpha', u, s))$, since $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\alpha', u, s)$. Further, if $\mathrm{dom}(a) \in \mathrm{dsrc}(a\ \mathrm{Lpurge}(\alpha', u, s \cdot a), u, s)$, then there is some $b \in \mathrm{alph}(\mathrm{Lpurge}(a\alpha', u, s))$ with $\mathrm{dom}(a) \rightarrowtail_s \mathrm{dom}(b)$. From Lemma 4.8.6 it follows that $\mathrm{dom}(b) \in \mathrm{dsrc}(a\alpha', u, s)$ and hence, $\mathrm{dom}(a) \in \mathrm{dsrc}(a\alpha', u, s)$, which is a contradiction to our initial assumption. Hence, the system is *not* di-secure. □

Next, we show that dta-security implies L-per-security, too.

4.8.8 Lemma. *Every dta-secure system is L-per-secure.*

Proof. We show this lemma by contraposition. Suppose that the system is not L-per-secure. Then there are $u \in D$, $s \in S$, and $\alpha \in A^*$ with

$$\mathrm{obs}_u(s \cdot \alpha) \neq \mathrm{obs}_u(s \cdot \mathrm{Lpurge}(\alpha, u, s)) \ .$$

Let $\delta \in A^*$ with $s = s^I \cdot \delta$. We will show the following claim: For every $\beta, \beta' \in A^*$ with $\alpha = \beta\beta'$ and for every $\gamma \in A^*$ with $\mathrm{Lpurge}(\alpha, u, s) = \gamma\ \mathrm{Lpurge}(\beta', u, s \cdot \beta)$ and for every $v \in \mathrm{dsrc}(\beta', u, s \cdot \beta)$, we have $\delta\beta \sim_v^{\mathsf{tdta}} \delta\gamma$.

We prove this claim by an induction on β. For the base case with $\beta = \epsilon$, we have that $\alpha = \beta'$ and hence $\gamma = \epsilon$ from which the claim is immediate. For the inductive step, let $\beta = \tilde{\beta}b$ for some action b. The induction hypothesis is: For some γ with $\mathrm{Lpurge}(\alpha, u, s) = \gamma\ \mathrm{Lpurge}(b\beta', u, s \cdot \tilde{\beta})$ and every $v \in \mathrm{dsrc}(b\beta', u, s \cdot \tilde{\beta})$, we have $\delta\tilde{\beta} \sim_v^{\mathsf{tdta}} \delta\gamma$.

We consider the following two cases:

Case 1: $\mathrm{dom}(a) \notin \mathrm{dsrc}(b\beta', u, s \cdot \tilde{\beta})$.

In this case, we have

$$\mathrm{dsrc}(b\beta', u, s \cdot \tilde{\beta}) = \mathrm{dsrc}(\beta', u, s \cdot \tilde{\beta} b)$$

and

$$\mathrm{Lpurge}(b\beta', u, s \cdot \tilde{\beta}) = \mathrm{Lpurge}(\beta', u, s \cdot \tilde{\beta} b) \ .$$

Hence, the new value of γ is the same as the previous value. And since $\mathrm{dom}(b) \not\rightarrow_{s \cdot \delta\tilde{\beta}} v$, by ($\mathrm{LR}^{\mathsf{tdta}}$), for every $v \in \mathrm{dsrc}(\beta', u, s \cdot \tilde{\beta} b)$, we have $\delta\tilde{\beta} b \sim_v^{\mathsf{tdta}} \delta\tilde{\beta}$, and combined with $\delta\tilde{\beta} \sim_v^{\mathsf{tdta}} \delta\gamma$ it follows the claim.
Case 2: $\mathrm{dom}(a) \in \mathrm{dsrc}(b\beta', u, s \cdot \tilde{\beta})$.
In this case, we have $\mathrm{dsrc}(b\beta', u, s \cdot \tilde{\beta}) = \{\mathrm{dom}(b)\} \cup \mathrm{dsrc}(\beta', u, s \cdot \tilde{\beta} b)$. Since

$$\begin{aligned} \mathrm{Lpurge}(\alpha, u, s) &= \gamma \ \mathrm{Lpurge}(b\beta', u, s \cdot \tilde{\beta}) \\ &= \gamma b \ \mathrm{Lpurge}(\beta', u, s \cdot \tilde{\beta} b) \ , \end{aligned}$$

the new value of γ is $\gamma' = \gamma b$. For every $v \in \mathrm{dsrc}(\beta', u, s \cdot \tilde{\beta} b)$, we have $\delta\tilde{\beta} \sim_v^{\mathsf{tdta}} \delta\gamma$ by induction hypothesis and additionally $\delta\tilde{\beta} \sim_{\mathrm{dom}(b)}^{\mathsf{tdta}} \delta\gamma$. From the condition ($\mathrm{SC}^{\mathsf{tdta}}$) it follows for every such v: $\delta\tilde{\beta} b \sim_v^{\mathsf{tdta}} \delta\gamma b = \delta\gamma'$.

From this claim it follows $\delta\alpha \sim_u^{\mathsf{tdta}} \delta \ \mathrm{Lpurge}(\alpha, u, s)$ with $\beta = \alpha$ and hence $\sim_u^{\mathsf{tdta}}$ is not observation consistent for u and the system is *not* dta-secure. □

Summarizing these results, we conclude that both di-security and dta-security imply L-per-security and hence L-security. All of these implications are strict. This is outlined in Figure 4.14.

We believe that both L-per-security and L-security are too weak. Additionally, we will point out a serious shortcoming of these two noninterference definitions.

The next example shows that the definition of L-security is not monotone according to restrictiveness of dynamic policies. The system in the next example is *not* L-secure, but becomes secure with a more restrictive policy. This is highly counterintuitive and an undesired property. As we have seen in Corollary 4.5.6, di-security has this monotonicity property. From

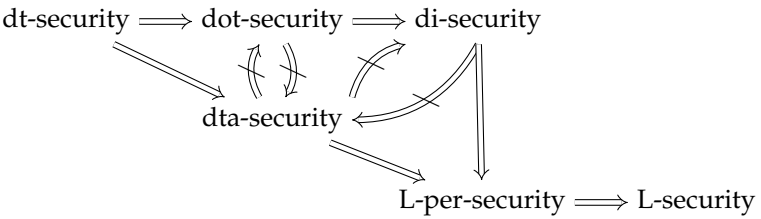

Figure 4.14. Relations between ours and Leslie's dynamic noninterference definitions

its definition, it follows clearly that dta-secure has this monotony property, too.

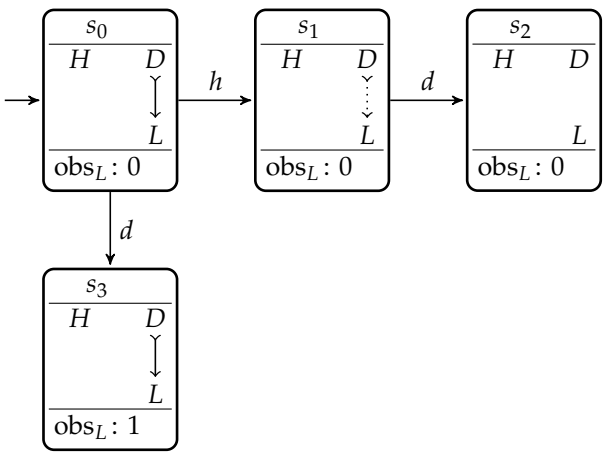

Figure 4.15. Non-monotony of L-security and L-per-security

4.8.9 Example. The system in Figure 4.15 has two different configurations of the dynamic policy indicated by the dotted policy edge in state s_1: in one configuration there is an edge from D to L and in the other, this edge is non-existing. First, consider the configuration where this edge exists. With respect to L-security and hence to L-per-security, this system is insecure,

since we have

$$\text{Lpurge}(hd, L, s_0) = \text{Lpurge}(d, L, s_1) = d \ ,$$

but $\text{obs}_L(s_0 \cdot hd) = 0 \neq 1 = \text{obs}_L(s_0 \cdot d)$.

However, if we remove the edge from D to L in state s_1, this system becomes both L-per-secure and L-secure. We have $\text{Lpurge}(hd, L, s_0) = \epsilon$ and $\text{obs}_L(s_0 \cdot hd) = 0 = \text{obs}_L(s_0)$.

This is highly counterintuitive since the policy without this edge is more restrictive than the policy with this edge.

The counterintuitive behavior of the L-security definition as outlined in Example 4.8.9 gives us more evidence that this is not a reasonable security definition and strengthens our proposals for alternative security definitions as provided in this chapter.

Chapter 5

Analyzing Information Flows - Algorithms and Complexity

Efficient algorithms for the verification of security properties and for the automated detection of security holes are fundamental for the application and acceptance of a theoretical framework. In this section, we analyze the complexity of the verification problem for finite-state systems for the security notions given in the previous chapters and present verification algorithms for those security notions for which we have a characterization by a polynomial-size unwinding.

We will show in Section 5.1 that on a system with a static policy, t-security, i-security, and ta-security can be verified in polynomial time. In Section 5.2, we will see that in systems with dynamic policies, both dt-security and dot-security can be verified in polynomial time, too. However, for the dynamic intransitive notion di-security, we show in Section 5.3 that the verification problem is NP-complete. In contrast, we will see in Section 5.5 that for those notions which include the observations in the allowed knowledge, to-security and ito-security, the verification problem is undecidable. We do not have analyzed the complexity of dta-security and leave it as an open problem for future research. At the end of this chapter, in Section 5.7, we provide a brief digression on the disjoint-set data structure which is extensively used in most of our algorithms.

Similar algorithms for static noninterference and undecidability results are submitted for publication in [EvdMSW13]. The NP-completeness result for di-security follows the same idea as the one in [ESW13].

5.1 Verification of t-security, i-security, and ta-security

We will show that t-security, i-security, and also ta-security can be verified in polynomial time. Indeed, they can be verified in nondeterministic logarithmic space. We will provide verification algorithms for these three security definitions by translating the state-based unwinding relations into algorithms. In the case that the verified system is insecure with respect to the corresponding security definition, the algorithm outputs two traces as a witness for the insecurity.

The common idea of these algorithms is to compute the smallest state-based unwinding relation and to verify observation consistency for the observing agent. We will provide a detailed high level description of these algorithms. We use the disjoint-set data structure for a highly efficient implementation of these algorithms. The details of this data structure will be given in Section 5.7.

More precisely, the algorithms start with singleton sets for each state of the system. By applying the unwinding conditions, the smallest equivalence relation on the states that satisfies all required unwinding condition will be computed. After each step, observation consistency is verified for the observing agent.

Except of the previously published algorithms in [EvdMSW13], to the best of our knowledge, no polynomial-time algorithm for the verification of t-security or ta-security has been published. For the verification of i-security, Pinsky [Pin95] claimed to provide a polynomial-time algorithm. However, it has been pointed out [Man01, EvdMSW13] that this result is wrong. In [HALL$^+$05], Hadj-Alouane et al. presented an exponential-time algorithm for the verification of i-security.

5.1.1 State-based Generalized Unwinding

We generalize the state-based unwinding for t-security and i-security in order to formulate a single algorithm for the verification and to avoid proving the correctness of the algorithms twice.

5.1.1 Definition (state-based generalized unwinding)**.** A *state-based generalized unwinding relation* for $u \in D$ and $L, X \subseteq D$ is an equivalence relation $\sim^{\mathsf{sg}}_{u,L,X} \subseteq S \times S$ that satisfies the following conditions for every $a \in A$, $s, t \in S$.

(LR$^{\mathsf{sg}}$): If $\mathrm{dom}(a) \in L$, then $s \sim^{\mathsf{sg}}_{u,L,X} s \cdot a$.

(SC$^{\mathsf{sg}}$): If $s \sim^{\mathsf{sg}}_{u,L,X} t$ and $\mathrm{dom}(a) \in X$, then $s \cdot a \sim^{\mathsf{sg}}_{u,L,X} t \cdot a$.

Instantiated with a policy, intuitively, L is the set of all agents which are not allowed to interfere with u. The set X are those agents which cannot transmit any information from any agent of L to u. The state-based generalized unwinding can be instantiated for t-security by $L = \{v \in D \mid v \not\rightarrow u\}$ and $X = D$ and for i-security for every agent v by $L = \{v\}$ and $X = \{w \in D \mid v \not\rightarrow w\}$. Hence, the state-based unwinding definition for t-security in Definition 3.2.9 and for i-security in Definition 3.4.11 are special cases of this generalized unwinding. The corresponding characterizations in terms of unwinding relations follow directly by taking u as the observing agent.

5.1.2 Verifying a State-based Generalized Unwinding

The first algorithm is the core of the algorithms for verifying t-security and i-security.

It computes the smallest state-based generalized unwinding relation for input parameters u, L, and X.

Algorithm 1 computes the the smallest equivalence relation that satisfies the conditions (LR$^{\mathsf{sg}}$) and (SC$^{\mathsf{sg}}$) of a state-based generalized unwinding. It starts with the identity relation. Then it increases the relation by iteratively applying the conditions (LR$^{\mathsf{sg}}$) and (SC$^{\mathsf{sg}}$) until they are satisfied for all states and actions. After every change of the relation computed so far, the algorithm checks if observation consistency holds. If this is not the case, the algorithm computes a witness, i. e., a pair of traces that proves the insecurity of the system, and interrupts it.

The equivalence relation is represented as a partition of the state space S, using the disjoint-set data structure. The list P maintains the pairs of states

Algorithm 1: generalized-unwinding(u, L, X)

Input: agent u, set of agents L, set of agents X

```
/* create a new partition                                          */
foreach s ∈ S do
    MAKE-SET(s) ;
let P be an empty list ;
let STORE be empty ;
/* left respect from L agents                                      */
foreach s ∈ S do
    foreach a ∈ A with dom(a) ∈ L do
        if FIND(s) ≠ FIND(s · a) then
            add ((s · a, s), (s, s), (a, ϵ)) to STORE;
            insert (s · a, s) into the list P ;
            UNION(s · a, s) ;
            if obs_u(s · a) ≠ obs_u(s) then
                return compute-witness(u, s · a, s, ϵ, ϵ) ;
/* step consistency from X agents                                  */
while P ≠ ∅ do
    take a pair (s, t) out of P ;
    foreach a ∈ A with dom(a) ∈ X do
        if FIND(s · a) ≠ FIND(t · a) then
            add ((s · a, t · a), (s, t), (a, a)) to STORE;
            insert (s · a, t · a) into the list P ;
            UNION(s · a, t · a) ;
            if obs_u(s · a) ≠ obs_u(t · a) then
                return compute-witness(u, s · a, t · a, ϵ, ϵ) ;
return false
```

Figure 5.1. Algorithm for computing a generalized unwinding

Procedure compute-witness(u, s, t, α, β)

```
1 if s = t then
2     return (u, s, α, β) ;
3 else
4     choose stored entry ((s, t), (s', t'), (a, b)) ;
5     compute-witness(u, s', t', aα, bβ) ;
```

Figure 5.2. Procedure for computing a witness for insecurity

that are merged in a union step. To compute a witness in the case of insecurity, the STORE data structure keeps track of all created pairs, as well as the pair of states and the pair of actions (or an action and the empty trace) from which it has been created.

The entries of the store data structure consist of three pairs of the form $(s,t),(s',t'),(a,b)$, where s,t,s',t' are states and a is an action and b is either an action or the empty trace. Such an entry is inserted into the store if a UNION operation is performed on the states s and t, where s is reached from s' by performing the action a and t is reached from t' by performing the action b (in the case of $b = \epsilon$, the state t is equal to t').

Since the union is only applied if $\text{FIND}(s) \neq \text{FIND}(t)$ and after the union, the FIND values of s and t are the same, the first pair of each stored entry is unique. Hence, the store data structure can be implemented as an array, indexed by the first pair of each entry.

If the union operation is applied to states with different observations for u, observation consistency for u is violated and the run of the algorithm terminates immediately with a call to the procedure compute-witness, which computes a witness for the insecurity.

To reference different values of the FIND function, we parameterize it with the number of unions done during the run of the algorithm. The function FIND_i, with $i \geq 0$, denotes the values of FIND after the ith application of UNION during the run of the algorithm. Note that only an application of the function UNION may change the value of the function FIND. We call i the union-number.

Similarly, the list P_i denotes the set of all pairs in P after the ith application of union. The set $P_{\leq i} = \bigcup_{j \leq i} P_j$ is the set of all pairs inserted into P up to the ith union-number, including those that have been removed. Since $P_{\leq i}$ is a set of pairs of states, we consider it as a binary relation on S.

Define for all states s and t the observation equivalence relation for agent u as

$$s \sim^{\mathrm{obs}_u} t \text{ iff } \mathrm{obs}_u(s) = \mathrm{obs}_u(t) \ .$$

To refer to the states of the construction of the equivalence relation, define for all union-numbers i and all states s and t

$$s \sim_i^{\textsc{Find}} t \text{ iff } \textsc{Find}_i(s) = \textsc{Find}_i(t) \ ,$$

and

$$\begin{aligned} s \sim_i^{\mathrm{step}} t \text{ iff } & s \sim_i^{\textsc{Find}} t \text{ and} \\ & \text{for all } a \in A \text{ with } \mathrm{dom}(a) \in X\text{: } s \cdot a \sim_i^{\textsc{Find}} t \cdot a \ . \end{aligned}$$

The relation $\sim_i^{\textsc{Find}}$ is the equivalence relation computed during the run of the algorithm after the ith union. The relation $\sim_i^{\mathrm{step}}$ is a subset of the the equivalence relation $\sim_i^{\textsc{Find}}$ that includes those states that are step consistent for all possible actions from A. Clearly, as mentioned, both relations are equivalence relations. Also note that the relations are monotone in the union-number, i. e., for every union-number i, and all states s and t, we have if $s \sim_i^{\textsc{Find}} t$, then $s \sim_{i+1}^{\textsc{Find}} t$ and if $s \sim_i^{\mathrm{step}} t$, then $s \sim_{i+1}^{\mathrm{step}} t$.

We want to show the correctness of Algorithm 1. The proof is split into the following lemmas.

5.1.2 Lemma. *Let i be a union-number. Then $\sim_i^{\textsc{Find}}$ is the smallest equivalence relation on S that includes $P_{\leq i}$.*

Proof. We prove this claim by an induction on the union-number i. For the base case, we have $i = 0$ and P_0 is empty. The smallest equivalence relation which includes the empty set is the identity relation, which represents the values of Find right before the first application of Union in the algorithm.

At each application of UNION, the pairs inserted into P are exactly those that are used by the union. Hence $\sim_i^{\text{FIND}}$ is the reflexive, symmetric, transitive closure of $P_{\leq i}$. Therefore, $\sim_i^{\text{FIND}}$ is the smallest equivalence relation on S that includes $P_{\leq i}$. □

5.1.3 Lemma. *If the algorithm terminates with a witness (u, s, α, β), then the traces α and β differ only in actions from agents of L, but lead to different observations of u starting in s, i. e.,* $\text{dom}(\text{alph}(\alpha))$ *and* $\text{dom}(\text{alph}(\beta))$ *are both subsets of $L \cup X$ and $\alpha{\restriction}_{X \setminus L} = \beta{\restriction}_{X \setminus L}$ and $\text{obs}_u(s \cdot \alpha) \neq \text{obs}_u(s \cdot \beta)$.*

Proof. The procedure compute-witness is called with two states as parameters having different observations for u, which are by the construction of the compute-witness procedure the states $s \cdot \alpha$ and $s \cdot \beta$. By the insertion of actions in STORE, there are actions of agents from L inserted asymmetrically and actions of agents from X are appended on both traces symmetrically, from which the claim follows. □

To complete the proof of the correctness of Algorithm 1, we show that the converse holds, too.

5.1.4 Lemma. *If the algorithm* generalized-unwinding*, run with input parameters u, L, and X, terminates with* `false`*, then there exists a state-based generalized unwinding relation $\sim_{u,L,X}^{\text{sg}}$ that is observation consistent for u.*

Proof. Let $u \in D$ and $L, X \subseteq D$. Consider a run of the algorithm generalized-unwinding terminating with `false`.

Let m be the last union-number. We will show inductively that the equivalence relation $\sim_m^{\text{FIND}}$ will satisfy the conditions (LR^{sg}) and (SC^{sg}) of the state-based generalized unwinding and also show that $\sim_m^{\text{FIND}}$ satisfies observation consistency for u.

Let i be the union-number right after the foreach-loop starting in line 5. Then the (LR^{sg}) condition holds, since for every $s \in S$ and every $a \in A$ with $\text{dom}(a) \in L$, we have $s \sim_i^{\text{FIND}} s \cdot a$.

For showing observation consistency of u, it is checked that the observations for u are equal in every two states which are merged by a union. Inductively, if the observations are equal for u on any two merged sets, the observations are then also equal in the resulting set. Therefore, for

every $i \leq m$, we have $\sim_i^{\text{FIND}} \subseteq \sim^{\text{obs}_u}$. Moreover, the relation $\sim_m^{\text{FIND}}$ satisfies observation consistency for u.

For showing (SC^{sg}), we observe that P_m is empty and that for every $s, t \in S$ with $(s,t) \in P_{\leq m}$, we have $s \sim_m^{\text{FIND}} t$ and $s \cdot a \sim_m^{\text{FIND}} t \cdot a$ for every $a \in A$ with $\text{dom}(a) \in X$. This is guaranteed by the foreach-loop in line 15 for every pair taken out of P in the line above. Therefore, we have $P_{\leq m} \subseteq \sim_m^{\text{step}}$. Since $\sim_m^{\text{step}}$ is an equivalence relation and since $\sim_m^{\text{FIND}}$ is the smallest equivalence relation that contains $P_{\leq m}$, we have $\sim_m^{\text{FIND}} \subseteq \sim_m^{\text{step}}$. Therefore, the relation $\sim_m^{\text{FIND}} = \sim_{u,L,X}^{\text{sg}}$ satisfies the (SC^{sg}) condition and is a state-based generalized unwinding relation for u, L, and X. □

The next lemma shows the correctness and the running time for the compute-witness procedure.

5.1.5 Lemma. *The procedure compute-witness computes a witness for insecurity in $O(|S| \cdot |A|)$.*

Proof. For showing the correctness of the procedure compute-witness, we will first show that the graph induced by the stored values forms a wood of directed rooted trees. Recall that for every stored entry of the form $e = ((s,t),(s',t'),(a,b))$ the projections onto its components are $\pi_0(e) = (s,t)$, $\pi_1(e) = (s',t')$ and $\pi_2(e) = (a,b)$. For every union-number i, the graph $G_i = (V_i, E_i)$ induced by the stored pairs of states is defined by

$$\begin{aligned} V_i = &\{\pi_0(e) \mid e \text{ is a stored entry up to the } i\text{th union-number}\} \\ &\cup \{\pi_1(e) \mid e \text{ is a stored entry up to the } i\text{th union-number}\} \ , \\ E_i = &\{(\pi_0(e), \pi_1(e)) \mid e \text{ is a stored entry up to the } i\text{th union-number}\} \ . \end{aligned}$$

We will show by an induction on the union-number i that the graph G_i is a wood and that each of its trees is a directed rooted tree. All edges are oriented towards the root and each root is of the form (s,s) for some state $s \in S$. We will also show that the following two inclusions hold for every union-number i:

$$P_{\leq i} \subseteq V_i \subseteq P_{\leq i} \cup \{(s,s) \mid s \in S\} \ .$$

In any iteration of the loop starting in line 5, only edges of the form $((s,t),(s',s'))$ with $s \neq t$ are inserted into the graph. Therefore, each connected component of each graph G_i after each iteration of this loop is a directed rooted tree, since only the root vertices are pairs with same entries.

In an iteration of the loop starting in line 15, an edge $e = ((s,t),(s',t'))$ is only inserted into G_i, if $\textsc{Find}_i(s) \neq \textsc{Find}_i(t)$. Therefore, $(s,t) \notin P_{\leq i} \cup \{(s'',s'') \mid s'' \in S\}$ and by induction hypothesis, we have $(s,t) \notin V_i$. Since $(s',t') \in P_{\leq i}$, the new edge e connects a new vertex with one from V_i. Hence, again, each connected component in the graph G_{i+1} is a directed rooted tree.

In the remainder of this proof, we analyze the complexity of this procedure. The procedure compute-witness is called with two states s and t with $(s,t) \in P_{\leq i}$. Then, it finds a path to the root (s',s') of the tree where (s,t) belongs to. This can be done within $O(|S|)$. Then a shortest path from s' to the initial state s^I can be found within $O(|S| \cdot |A|)$, which is an upper bound for the running time of the whole algorithm. □

Next, we analyze the running time of the algorithm generalized-unwinding.

5.1.6 Lemma. *The running time of the algorithm* generalized-unwinding *is bounded by* $O(|A| \cdot |S| \cdot \alpha(|S|))$.

Proof. The UNION operation is only applied to states in different sets and in this case, the number of sets in the partition is reduced by one. Hence the number of unions is bounded by $|S|$.

Also insertions into P are only done in combination with a union step. Hence the number of elements inserted into P during a run of the algorithm is bounded by $|S|$. The UNION and FIND operations have an amortized running time of $\alpha(|S|)$. The number of the iterations of the loops starting in line 5 and line 15 are bounded by $|A| \cdot |S|$. Hence the running time of the whole algorithm is in $O(|A| \cdot |S| \cdot \alpha(|S|))$. □

5.1.3 t-security

For the verification of t-security, Algorithm 1 can be easily used. The state-based generalized unwinding conditions can be parameterized such that

they are equivalent to the transitive unwinding conditions in Definition 3.2.9. This is achieved by setting $L(u) = \{v \in D \mid v \not\rightarrow u\}$ and $X = D$ iterated on every possible choices of $u \in D$. Hence the algorithm for verifying t-security has a loop running over all agents u and calls in each of its iterations Algorithm 1 with input parameters u, $L(u)$, and D.

Algorithm 2: verify-t-security

```
1 set RESULT = false ;
2 foreach u ∈ D do
3 |   RESULT = generalized-unwinding(u, {v ∈ D | v ↛ u}, D) ;
4 |   if RESULT then
5 |   |   return system is insecure with witness RESULT
6 return system is t-secure
```

Figure 5.3. Algorithm for verifying t-security

5.1.7 Lemma. *For a finite-state system, t-security can be verified in* $O(|D| \cdot |A| \cdot |S| \cdot \alpha(|S|))$ *by the algorithm* verify-t-security.

Proof. Algorithm 2 has a single loop which runs over all agents $u \in D$ and in each iteration the algorithm generalized-unwinding is called. Hence the running time is $|D|$ times the running time of the algorithm generalized-unwinding. □

5.1.4 i-security

Similar to the verification of t-security, i-security can be verified by instantiating the state-based generalized unwinding with appropriate sets for L and X. Here, we define for each $v \in D$ with $v \not\rightarrow u$ the sets $L(v) = \{v\}$ and $X(v) = \{w \in D \mid v \not\rightarrow w\}$. With these choices, the state-based generalized unwinding is equivalent to the state-based unwinding for i-security.

5.1.8 Lemma. *For a finite-state system, i-security can be verified in* $O(|D|^2 \cdot |A| \cdot |S| \cdot \alpha(|S|))$ *by the algorithm* verify-i-security.

Algorithm 3: verify-i-security

```
1 set RESULT = false ;
2 foreach u ∈ D do
3     foreach v ∈ D with v ↛ u do
4         RESULT = generalized-unwinding(u, {v}, {w ∈ D | v ↛ w}) ;
5         if RESULT then
6             return system is insecure with witness RESULT
7 return system is i-secure
```

Figure 5.4. Algorithm for verifying i-security

Proof. Algorithm 3 has two iterated loops which both run over a subset of D and in each iteration the algorithm generalized-unwinding is called. Hence the running time of this algorithm is $|D|^2$ times the running time of the algorithm generalized-unwinding. □

5.1.5 ta-security

ta-security can be verified in a similar way as t-security or i-security. However, we can not apply the algorithm for the verification of the generalized unwinding. Algorithm 4 verifies the conditions of the state-based unwinding for ta-security given in Definition 3.5.20. The main difference to Algorithm 1 is that the loop that computes the partition for the (LR^{sg}) condition is substituted by a loop that computes the ($SWAP^{sta}$) condition of Definition 3.5.20. As shown in Theorem 3.5.21, ta-security can be expressed as a combination of i-security and an additional unwinding. Since we have already shown how to verify i-security, it remains to provide an algorithm for the verification of the state-based unwinding conditions of Definition 3.5.20. Hence, for verifying ta-security, one needs to apply both Algorithm 3 and Algorithm 4.

5.1.9 Lemma. *On a finite-state system, ta-security can be verified in* $O(|D|^3 \cdot |A| \cdot |S| \cdot \alpha(|S|))$.

Algorithm 4: ta-security-verification

```
1  foreach u ∈ D do
2      foreach v ∈ D do
3          foreach w ∈ D with w ↛ v and v ↛ w and w ↛ u do
4              foreach s ∈ S do
5                  MAKE-SET(s) ;
6              let P be an empty list ;
7              let STORE be empty ;
               /* apply (SWAP^sta) condition                          */
8              foreach s ∈ S do
9                  foreach a ∈ A with dom(a) = v do
10                     foreach b ∈ A with dom(b) = w do
11                         if FIND(s · ab) ≠ FIND(s · ba) then
12                             add ((s · ab, s · ba), (s, s), (ab, ba)) to STORE;
13                             insert (s · ab, s · ba) into the list P ;
14                             UNION(s · ab, s · ba) ;
15                             if obs_u(s · ab) ≠ obs_u(s · ba) then
16                                 return
                                   compute-witness(u, s · ab, s · ba, ϵ, ϵ) ;
               /* apply (SC^sta) condition                            */
17             while P ≠ ∅ do
18                 take a pair (s, t) out of P ;
19                 foreach a ∈ A with v ↛ dom(a) or w ↛ dom(a) do
20                     if FIND(s · a) ≠ FIND(t · a) then
21                         add ((s · a, t · a), (s, t), (a, a)) to STORE;
22                         insert (s · a, t · a) into the list P ;
23                         UNION(s · a, t · a) ;
24                         if obs_u(s · a) ≠ obs_u(t · a) then
25                             return compute-witness(u, s · a, t · a, ϵ, ϵ) ;
26 return "ta-secure if di-secure"
```

Figure 5.5. An algorithm for the state-based ta-security unwinding

Proof. Algorithm 4 has a similar structure as Algorithm 1. However, each of the three outer loops iterates on a subset of D. This leads to the additional factor of $|D|^3$. The rest of the analysis of the running time is similar to that in the proof of Lemma 5.1.6. Since the running time of Algorithm 4 bounds the running time for verifying i-security, for our analysis, we do not need to take into account the latter. □

5.2 Verification of Dynamic Noninterference

Similar to those of the previous section, we provide verification algorithms for dt-security and dot-security. Due to the similar structure of the state-based unwinding conditions for dt-security to those for t-security, the verification algorithm is analogue. For the verification of dot-security, the verification algorithm differs, since the state-based unwinding relations for dot-security are in general not equivalence relations.

5.2.1 dt-security

As we saw in Theorem 4.2.8, dt-security is characterized by the existence of state-based unwinding relations for every agent. These unwinding relations lead to a verification algorithm similar to the one for t-security. In Figure 5.6, an algorithm for verifying dt-security is provided. The procedure compute-witness is the same as in the static case, since it is independent of the policy.

5.2.1 Lemma. *On a finite-state system, dt-security can be verified in* $O(|D| \cdot |A| \cdot |S| \cdot \alpha(|S|))$ *by the algorithm* verify-dt-security.

Proof. Algorithm 5 computes the smallest unwinding relation satisfying ($\mathrm{LR}^{\mathsf{sdt}}$) and ($\mathrm{SC}^{\mathsf{sdt}}$) and verifies that it is observation consistent for the observing agent. The only difference to the verification of t-security is that the policy is dynamic. However, the dynamic policy has no influence on the running time. □

Algorithm 5: verify-dt-security

```
   /* create a new partition                                              */
 1 foreach u ∈ D do
 2     foreach s ∈ S do
 3         MAKE-SET(s) ;
 4     let P be an empty list ;
 5     let STORE be empty ;
       /* apply (LR^sdt) condition                                        */
 6     foreach s ∈ S do
 7         foreach a ∈ A with dom(a) ↛_s u do
 8             if FIND(s) ≠ FIND(s · a) then
 9                 add ((s · a, s), (s, s), (a, ϵ)) to STORE;
10                 insert (s · a, s) into the list P ;
11                 UNION(s · a, s) ;
12                 if obs_u(s · a) ≠ obs_u(s) then
13                     return compute-witness(u, s · a, s, ϵ, ϵ) ;
       /* apply (SC^sdt) condition                                        */
14     while P ≠ ∅ do
15         take a pair (s, t) out of P ;
16         foreach a ∈ A do
17             if FIND(s · a) ≠ FIND(t · a) then
18                 add ((s · a, t · a), (s, t), (a, a)) to STORE;
19                 insert (s · a, t · a) into the list P ;
20                 UNION(s · a, t · a) ;
21                 if obs_u(s · a) ≠ obs_u(t · a) then
22                     return compute-witness(u, s · a, t · a, ϵ, ϵ) ;
23 return false
```

Figure 5.6. Algorithm for verifying dt-security

Algorithm 6: verify-dot-security

```
   /* create a new partition                                          */
1  foreach u ∈ D do
2      foreach v ∈ D do
3          let P be an empty list ;
4          let STORE be empty ;
           /* apply (LR^sdot) condition                               */
5          foreach s ∈ S do
6              foreach a ∈ A with dom(a) = v and v ⊬→_s u do
7                  if (s, s·a) not stored then
8                      add ((s, s·a), (s, s), (ϵ, a)) to STORE;
9                      insert (s, s·a) into the list P ;
10                     if obs_u(s) ≠ obs_u(s·a) then
11                         return compute-witness(u, s, s·a, ϵ, ϵ) ;
           /* apply (SC^sdot) condition                               */
12         while P ≠ ∅ do
13             take a pair (s, t) out of P ;
14             foreach a ∈ A with either dom(a) ≠ v or dom(a) = v with
               v ⊬→_t u do
15                 if (s·a, t·a) not stored then
16                     add ((s·a, t·a), (s, t), (a, a)) to STORE;
17                     insert (s·a, t·a) into the list P ;
18                     if obs_u(s·a) ≠ obs_u(t·a) then
19                         return compute-witness(u, s·a, t·a, ϵ, ϵ) ;
20 return false
```

Figure 5.7. Algorithm for verifying dot-security

5.2.2 dot-security

As we have already seen in this chapter, the existence of a characterization in terms of a state-based unwinding relation can be translated into a verification algorithm. This also holds for dot-security, since it can be characterized by the existence of a state-based downgrading over time unwinding, defined in Definition 4.3.7. However, these relations are not equivalence relations and hence cannot be expressed as partitions of the state space. This will be reflected in the complexity as we cannot use the highly efficient disjoint-set data structure as in the previous algorithms.

Nonetheless, we encode the conditions of this unwinding into a polynomial-time algorithm.

The algorithm verify-dot-security in Figure 5.7 computes for every pair of agents v, u the smallest unwinding relation $\precsim^{\mathsf{sdot}}_{u,v}$ and verifies after each new pair whether the created relation is observation consistent for u. The condition "(s,t) not stored" in line 7 and line 15 means that there is no triple in STORE, such that (s,t) is its first entry.

We provide here only a very coarse analysis of the running time.

5.2.2 Lemma. *On a finite-state system, dot-security can be verified in* $O(|D|^2 \cdot |A| \cdot |S|^2)$ *by the algorithm* verify-dot-security.

Proof. Algorithm 6 follows the same pattern as the previous algorithms, except that the visited states are not maintained by the disjoint-set data structure. This increases the running time, but does not affect the correctness of the algorithm.

The most important observation is that every possible pair of states is only inserted into P when a corresponding value is stored. Since values are never removed from the STORE, the stored values are uniquely determined by their first entry. Every possible pair of states is only inserted into P once. Hence the number of the iterations of the while-loop is bounded by $|S|^2$. □

5.3 Complexity of di-security

For dynamic intransitive noninterference, the verification problem has a higher complexity than for i-security, dt-security, or dot-security. Although we have seen that di-security can be characterized by the existence of a state-based unwinding, the number of needed unwinding relations is exponential in the number of the agents of the system. We will not provide an algorithm for the verification of di-security. Instead, we will show that the verification problem is NP-complete. We will show NP-hardness by a reduction from the 3-SAT problem which simplifies the proof compared to the reduction from 3-colorability problem as given in [ESW13].

First, we show that the verification problem is in NP.

5.3.1 Lemma. *Deciding whether a system is not di-secure is in NP.*

Proof. We will describe an algorithm which verifies that a system is not di-secure. The algorithm guesses a state s, an action a, an agent u, and a trace α and verifies that these satisfy

$$\mathrm{dom}(a) \notin \mathrm{dsrc}(a\alpha, u, s) \text{ and } \mathrm{obs}_u(s \cdot a\alpha) \neq \mathrm{obs}_u(s \cdot \alpha) \ . \qquad (*)$$

To show that this is an NP-algorithm, it suffices to show that the length of α is polynomially bounded by the size of the system. For that, let α be a trace of minimal length satisfying the condition $(*)$.

Let M^2 be the product system of a copy of the system M when the initial state is set to s and a copy of the system M when the initial state is set to $s \cdot a$. Clearly, the number of states of this system is $|S|^2$. If the length of α is greater than $|S|^2$, then the run of α in M^2 visits a state of M^2 twice. Therefore, α contains a loop. This loop can be removed from α without changing the reached state of M^2. We call β the reduced trace. Clearly, β and $a\beta$ lead to the same observations as α and $a\alpha$ do, respectively. Since β is a subsequence of α, we have $\mathrm{dom}(a) \notin \mathrm{dsrc}(a\beta, u, s)$. □

Next, we show the NP-hardness of the verification problem of di-security by a reduction from the 3-satisfiability problem (3-SAT). An instance of the 3-SAT problem is given by a conjunction of clauses, where each clause is a disjunction of exactly three literals. A literal is a boolean variable or its

negation. Such an instance is a positive instance if and only if it is satisfiable. It is well-known that this decidability problem is NP-complete [Coo71].

The following construction encodes a 3-SAT instance into a system equipped with agents and a dynamic policy. After the construction, we will show that the constructed system is *not* di-secure if and only if the used instance is a positive one.

Consider a 3-SAT instance given by variables $x_1, \ldots, x_n$ and clauses $c_1, \ldots, c_m$. Each clause c_i contains literals l_i^1, l_i^2, l_i^3. Each literal l_i^j is equal to some variable x_k or a negated variable $\neg x_k$.

First, we provide the formal construction of the system and will give a more intuitive description of the components afterwards.

For any variable x_i, the system has agents X_i and $\overline{X}_i$. The set of all these agents is defined as $\mathcal{X} = \{X_i, \overline{X}_i \mid 1 \leq i \leq n\}$. Additionally, we have agents L and H. Hence the set of all agents is $D = \mathcal{X} \cup \{L, H\}$. We identify every literal with an action. Therefore, for each variable x_i, there is an action x_i and one denoted by $\overline{x}_i$ corresponding to the literal $\neg x_i$. These actions can be performed by the agents X_i and $\overline{X}_i$, respectively. The agent H can perform an action h, the agent L has no actions. The states of the system are of the form

- (y, z, i) with $y \in \{\mathsf{right}, \mathsf{left}\}$, $z \in \{\mathsf{choice}, \mathsf{pos}, \mathsf{neg}\}$ and $i \in \{1, \ldots, n\}$,
- (y, clause, j) with $y \in \{\mathsf{right}, \mathsf{left}\}$ and $j \in \{1, \ldots, m\}$,
- (y, last) with $y \in \{\mathsf{right}, \mathsf{left}\}$.

The initial state is $s^I = (\mathsf{left}, \mathsf{choice}, 1)$.

The transitions of the action h are

$$(\mathsf{left}, \mathsf{choice}, 1) \cdot h = (\mathsf{right}, \mathsf{choice}, 1),$$

and for any $y \in \{\mathsf{right}, \mathsf{left}\}$, any $z \in \{\mathsf{pos}, \mathsf{neg}\}$ and any $1 \leq i < n$:

$$\begin{aligned}(y, z, i) \cdot h &= (y, \mathsf{choice}, i+1), \text{ and} \\ (y, z, n) \cdot h &= (y, \mathsf{clause}, 1)\ .\end{aligned}$$

The transitions of the actions x_i and $\overline{x}_i$ are for any $y \in \{\mathsf{right}, \mathsf{left}\}$ and any $1 \leq i \leq n$:

$$(y, \mathsf{choice}, i) \cdot x_i = (y, \mathsf{pos}, i),$$
$$(y, \mathsf{choice}, i) \cdot \overline{x}_i = (y, \mathsf{neg}, i),$$

and for every literal l_i^j contained in clause c_i, there is a transition

$$(y, \mathsf{clause}, i) \cdot l_i^j = (y, \mathsf{clause}, i+1) \qquad \text{if } i < n, \text{ and}$$
$$(y, \mathsf{clause}, i) \cdot l_i^j = (y, \mathsf{last}) \qquad \text{if } i = n \ .$$

All transitions not explicitly given loop in the corresponding state.

Only the agent L has non-constant observations. The observations of L are 0 on all states, except on the state $(\mathsf{left}, \mathsf{last})$, here, L observes 1.

The edges in the dynamic policy are given by: All agents of the form X_i and $\overline{X}_i$ interfere in all states with L. Additionally, for all $1 \leq i \leq n$, the agent H interferes with $\overline{X}_i$ in the state $(\mathsf{right}, \mathsf{pos}, i)$ and H interferes with X_i in the state $(\mathsf{right}, \mathsf{neg}, i)$. The agent H interferes with L in all states marked with left, except the initial state.

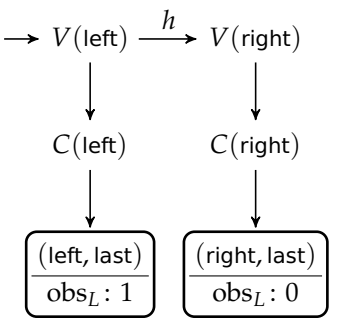

Figure 5.8. Coarse structure of the system $M(\varphi)$

The coarse structure of the constructed system is depicted in Figure 5.8. The system consists of subsystems $V(\mathsf{left})$ and $V(\mathsf{right})$. The latter one is depicted in Figure 5.9. The other one has the same structure, except for the

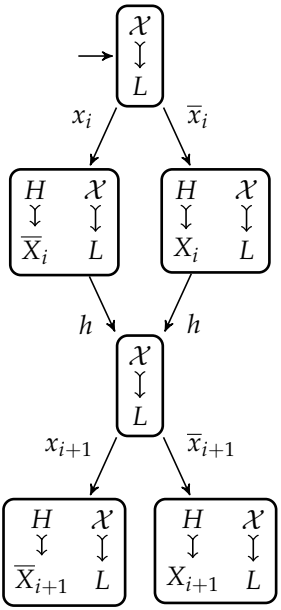

Figure 5.9. Structure of the subsystem $V(\mathsf{right})$

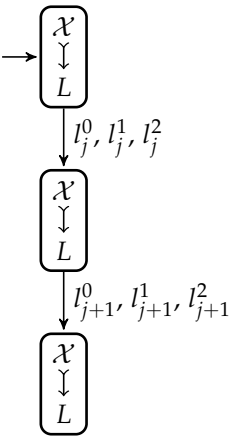

Figure 5.10. Structure of the subsystem $C(\mathsf{right})$

less restrictive policies as described above. The idea is that the choice of the actions x_i corresponds to an assignment of `true` to the respective variable and the choice of $\overline{x}_i$ corresponds to the assignment of `false` to it. In any of these cases, the agent that does not perform the action allowed to receive the following h action from H. This process is repeated for all variables occurring in the formula.

Afterwards, the system reaches the subsystems $C(\mathsf{left})$ or $C(\mathsf{right})$. Again, these are the same systems, except the less restrictive policies in the subsystem labeled with left. In this subsystem, each state corresponds to a particular clause of the formula. The actions that can be performed in a state corresponding to a clause c_i are exactly those that correspond to the literals of that formula. The last transitions of this subsystem lead to one of the states labeled with last. From these states, there are no non-looping transitions.

The main property of this system is that it contains a run $h\alpha$ from initial state to $(\mathsf{right}, \mathsf{last})$ such that h is not downgraded by α to L if and only if the 3-SAT instance is satisfiable. We call such a trace a *hiding* trace.

5.3.2 Definition (hiding trace)**.** A trace $h\alpha$ is *hiding* if $H \notin \mathrm{dsrc}(h\alpha, L, s^I)$ and $s^I \cdot h\alpha = (\mathsf{right}, \mathsf{last})$.

Intuitively, the subsystem $V(\mathsf{right})$ enforces to choose a variable assignment for all variables occurring in the formula. This is done by performing one of the actions x_i for `true` of $\overline{x}_i$ for `false`. In any of these cases, it is followed by an action h in a state where H is allowed to interfere with one of the agents X_i or $\overline{X}_i$, namely that one that has not performed an action in the previous state. This guarantees that if the agent that receives the action h performs an action later in a run, the performing of the initial action h will be revealed.

After leaving the subsystem $V(\mathsf{right})$, the run has to proceed into the subsystem $C(\mathsf{right})$. The intuition is that for each clause, one literal has to be chosen to proceed in the system. If a literal has been chosen that was not previously chosen in the subsystem $V(\mathsf{right})$, then, as explained, the h action is transmitted to L.

If the actions corresponding to the literals have been chosen in the way explained, then h is not transmitted to L and a hiding path $h\alpha$ exists.

Obviously, this is only possible if the formula is satisfiable.

5.3.3 Lemma. *Let φ be a 3-SAT formula. A system M constructed from φ has a hiding path iff φ is satisfiable.*

Proof. First, assume that the constructed system has a hiding path. Then, there is a sequence of actions α such that $H \notin \mathrm{dsrc}(h\alpha, L, s^I)$ and $s^I \cdot h\alpha = (\mathsf{right}, \mathsf{last})$. We choose α of minimal length from all possible choices with this property. The run of $h\alpha$ has no self-loops, since otherwise α would not be of minimal length. Since the run of α goes through the system $V(\mathsf{right})$, there is a prefix of α of the form $l_1 h l_2 h \cdots l_n h$, where l_i is one of the actions x_i or $\overline{x}_i$. Depending on that choice, define a truth assignment $f\colon \{x_1, \ldots, x_n\} \to \{\mathtt{true}, \mathtt{false}\}$ by

$$f(x_i) = \begin{cases} \mathtt{true} & \text{if } x_i = l_i \\ \mathtt{false} & \text{if } \overline{x}_i = l_i \ . \end{cases}$$

Assume that φ is not satisfied by this truth assignment. Therefore, there is a clause c_i such that all literals are mapped to $\mathtt{false}$ by f. Since the hiding path reaches the state $(\mathsf{right}, \mathsf{last})$, each action l corresponding to a literal of c_i that has been taken in the state $(\mathsf{right}, \mathsf{clause}, i)$. But since this action l has not been taken in the subsystem $V(\mathsf{right})$, there was an action h performed in a state s where $H \rightarrowtail_s \mathrm{dom}(l)$. Therefore, after performing the action l, the agent H is in the corresponding set of sources, which contradicts the property of a hiding path that $H \notin \mathrm{dsrc}(h\alpha, L, s^I)$. Hence, the formula φ is satisfiable.

Assume for the other direction of the proof that φ is satisfiable. Then, there is a satisfying truth assignment

$$f\colon \{x_1, \ldots, x_n\} \to \{\mathtt{true}, \mathtt{false}\} \ .$$

For any $1 \leq i \leq n$ set

$$l_i = \begin{cases} x_i & \text{if } f(x_i) = \mathtt{true} \\ \overline{x}_i & \text{if } f(x_i) = \mathtt{false} \ . \end{cases}$$

Since this truth assignment is satisfying, there is a sequence of literals $l_1^c, \ldots, l_m^c$ such that at least one of every clause is evaluated to `true`. Therefore, we have that $\{l_1^c, \ldots, l_m^c\} \subseteq \{l_1, \ldots, l_n\}$. Define the path α as

$$\alpha = l_1 h l_2 h \cdots l_n h l_1^c \cdots l_m^c \ .$$

By the construction of the system, we have $s^I \cdot h\alpha = (\mathsf{right}, \mathsf{last})$. Since every action l_i^c appeared earlier in the trace α, we have in all states s, where h is performed that $H \not\rightarrow_s \mathrm{dom}(l_i^c)$. Therefore the action h is not downgraded to L, i.e., $H \notin \mathrm{dsrc}(h\alpha, L, s^I)$. □

We will show that in systems arising from this construction, the existence of a hiding path is equivalent to being insecure with respect to the definition of di-security.

5.3.4 Lemma. *A system as constructed above is* not *di-secure iff it contains a hiding path.*

Proof. Assume that the constructed system is not di-secure. Since L is the only agent that has non-constant observations and H is the only agent that is not allowed to interfere with L in all states, there is a state s and a sequence of actions α such that $H \notin \mathrm{dsrc}(h\alpha, L, s)$ and $\mathrm{obs}_L(s \cdot h\alpha) \neq \mathrm{obs}_L(s \cdot \alpha)$. Thus, exactly one of the states $s \cdot h\alpha$ and $s \cdot \alpha$ has to be $(\mathsf{left}, \mathsf{last})$. Therefore, the state s is a state labeled with left. But in any of these states, except s^I, all agents, including H, are allowed to interfere with L. Therefore, the state s has to be s^I. Since the action h moves from the initial state to the right system, we have that $s^I \cdot \alpha$ has to be the state $(\mathsf{left}, \mathsf{last})$. Since the transitions are essentially identical in the left and the right subsystems, we have that $s^I \cdot h\alpha = (\mathsf{right}, \mathsf{last})$. Hence, $h\alpha$ is a hiding path.

For the other direction, assume that there is a hiding path $h\alpha$ in the constructed system. Since $s^I \cdot h\alpha = (\mathsf{right}, \mathsf{last})$, we have that $s^I \cdot \alpha = (\mathsf{left}, \mathsf{last})$. Therefore $\mathrm{obs}_L(s^I \cdot h\alpha) \neq \mathrm{obs}_L(s^I \cdot \alpha)$. From the existence of a hiding path, it follows that $H \notin \mathrm{dsrc}(h\alpha, L, s^I)$ and hence the system is not di-secure. □

From the lemmas above and the fact that the construction can be done in polynomial time, it follows that the verification of di-security is an NP-complete problem.

5.3.5 Theorem. *Deciding whether a system is not di-secure is NP-complete.*

5.4 Computing Information Flows

Besides the problem of verifying security and detecting security flaws, the computation of information flows is a further algorithmic problem of interest. Here, we consider systems with security domains (or agents) but without a policy. Then the question is: Between which security domains an information flow does exist in the system? This question can be rephrased: What is the most restrictive policy with respect to which the system is secure?

We will consider this problem of computing information flows only for the notions of t-security and i-security. We will see that we can reuse the algorithms used for verification. However, depending on the notion of security, we first have to clarify what is meant by a most restrictive policy.

5.4.1 t-security

For t-security the situation is quite simple. Every single edge in a policy only says something about the information flow between the two agents that are connected by the edge. Hence, we need an edge from an agent v to u if and only if v can influence u's observation in some way. The idea of computing a most restrictive policy is to verify for every possible edge $v \rightarrowtail u$ whether the system is t-secure with respect to a policy without this edge. Hence, we verify whether the system is secure for the policy $v \not\rightarrowtail u$ and $w \rightarrowtail u$ for all $w \neq v$. Then, if this system is secure, the edge is not needed in any policy. In the other case, the system is not secure without this edge and this edge is needed in every policy with respect to which the system is secure. Moreover, since in a most restrictive policy, every edge is uniquely defined as explained, such a policy is unique.

We formalize the intuition of a most restrictive policy as follows.

5.4.1 Definition (most restrictive t-security-preserving policy)**.** For a system equipped with agents D, a static policy $\rightarrowtail \subseteq D \times D$ is a *most restrictive*

t-security-preserving policy if the system is t-secure w.r.t. $\rightarrowtail$ and not t-secure w.r.t. $\rightarrowtail \setminus \{(v,u)\}$ for every $(v,u) \in \rightarrowtail$.

Such a most restrictive policy is found by Algorithm 7. The next lemma follows directly from the arguments explained above and the correctness of the generalized-unwinding procedure.

5.4.2 Lemma. *On a finite-state system, a most restrictive t-security-preserving policy can be found in* $O(|D|^2 \cdot |A| \cdot |S| \cdot \alpha(|S|))$ *by the algorithm* compute-transitive-information-flow

Algorithm 7: compute-transitive-information-flow

```
1 set ↣ = {(u,u) | u ∈ D} ;
2 foreach u ∈ D do
3     foreach v ∈ D with v ≠ u do
4         if generalized-unwinding(u, {v}, D) then
5             set ↣ = ↣ ∪ {(v,u)} ;
6 return ↣ ;
```

Figure 5.11. Algorithm for computing transitive information flows

5.4.2 i-security

The problem of defining a most restrictive policy is more complex in the context of an intransitive interpretation of a policy. Let us consider the policy $H \rightarrowtail D \rightarrowtail L$. There are two reasons why the edge $H \rightarrowtail D$ might be necessary in this policy. This can be either because D can observe any of H's actions or L can observe any of H's actions indirectly after some interaction of D. To keep this situation more simple, we will restrict ourselves to only a single observing agent and assume that all other agents have constant observations.

If we have only one observing agent, then a minimal policy will be acyclic and all edges are directed towards the observing agent, say u, which

forms a sink in the graph. However, a definition of a most restrictive policy is not immediate. Consider the two policies in Figure 5.12 with the only observing agent L. Clearly, every system that is i-secure w.r.t. the policy at the left-hand side is also i-secure w.r.t. the policy at the right-hand side, but the converse does not hold. Hence, it is reasonable to say that the left policy is more restrictive than the right one. On a more intuitive level, with the left policy, the information flow from H to L is conditioned by some interaction of D. The right policy allows unconditional information flow from H to L. We can conclude that the longer the paths are the more restrictive an intransitive policy is.

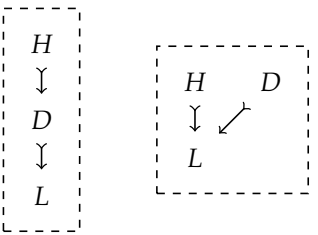

Figure 5.12. Two HDL-policies

The distance of an agent v to u in the policy $\rightarrowtail$ is denoted by $d_u^{\rightarrowtail}(v)$ and is the length of a shortest path from v to u in $\rightarrowtail$. As the length of a path we understand the number of edges in the path. If there is no path from v to u, we write $d_u^{\rightarrowtail}(v) = \infty$.

There might be different ways of how to define a partial order between policies according to restrictiveness. As explained above, a partial order induced by inclusion, as we had for t-security or in the previous chapters, does not work here. First, we decided to compare policies by the amount of agents that have no paths to the observing agent, then by the number of edges in the policy, and lastly by the sum of the shortest distances of each agent to the observing agent.

5.4.3 Definition (Restrictiveness between intransitive policies)**.** A policy $\rightarrowtail$ is *at least as restrictive as* a policy $\rightarrowtail'$ if

$$|\{v \in D \mid d_u^{\rightarrowtail}(v) = \infty\}| \geq |\{v \in D \mid d_u^{\rightarrowtail'}(v) = \infty\}| \ ,$$

and if both values are equal, then

$$| \rightarrowtail | \leq | \rightarrowtail' | \ ,$$

and if these values are equal, too, then

$$\sum_{v \in \{w \in D | d_u^{\rightarrowtail}(w) < \infty\}} d_u^{\rightarrowtail}(v) \geq \sum_{v \in \{w \in D | d_u^{\rightarrowtail'}(w) < \infty\}} d_u^{\rightarrowtail'}(v) \ .$$

We say that a policy is most restrictive if it is a maximal element according to this partial order between policies. A most restrictive policy for an observing agent u is computed by Algorithm 8. The idea is to compute the policy layer-wise, starting with the observing agent u in layer 0. Initially all other agents are isolated. Then it is tested whether there is an information flow from one of the isolated agents v through one agent w of the highest layer of the policy so far and all agents on a lower layer. In this case the edge $v \rightarrowtail w$ is needed.

Next, we show the correctness of the algorithm. That means that the system is secure w.r.t. the returned policy and that the policy is most restrictive according to the definition of restrictiveness above.

5.4.4 Lemma. *Let $\rightarrowtail$ be the policy constructed by Algorithm 8. Then*

1. *The system is i-secure w.r.t. $\rightarrowtail$.*
2. *If an edge $v \rightarrowtail w$ with $v \neq w$ is removed from $\rightarrowtail$, then the system is not i-secure anymore.*
3. *The distances from every agent to u are maximal.*

Proof. 1. We prove this claim inductively by insertions of agents into the policy graph. By that we mean to connect previously isolated agents with the remaining agents. With V_i, we denote the set of agents after the ith agent has been inserted. The first agent inserted is u, hence we have $V_1 = \{u\}$. The only line in the algorithm where agents are inserted into the policy is line 10. If an agent v is inserted in this line, in the following foreach-loop, it is checked whether there is a new forbidden information flow stemming from the insertion of v. Hence, after all iterations of this

Algorithm 8: compute-intransitive-information-flow

```
   Input: observing agent u
 1 set ↣ = {(v,v) | v ∈ D} ;
 2 foreach v ∈ D with v ≠ u do
 3 |  set layer(v) = ∞ ;
 4 set layer(u) = 0 ;
 5 for i = 1 to |D| do
 6 |  set X = {v ∈ D | layer(v) ≤ i − 2} ;
 7 |  foreach v ∈ D with layer(v) = ∞ do
 8 |  |  foreach w ∈ D with layer(w) = i − 1 do
 9 |  |  |  if generalized-unwinding( u, {v}, X ∪ {w} ) then
10 |  |  |  |  set ↣ = ↣ ∪ {(v,w)} ;
11 |  |  |  |  set layer(v) = i ;
12 |  |  |  |  foreach x ∈ D with layer(x) > 1 do
13 |  |  |  |  |  if generalized-unwinding( u, {x},
   |  |  |  |  |  {y ∈ D | x ↛ y and layer(y) < ∞}) then
14 |  |  |  |  |  |  set ↣ = ↣ ∪ {(x,v)} ;
15 return ↣ ;
```

Figure 5.13. Algorithm for computing intransitive information flow for a single observer

innermost foreach-loop, the system, restricted to the non-isolated agents, is again secure.

Note that if in some iteration of the for-loop starting in line 5, no agent is inserted in the policy graph, in no further iteration of that loop any agent will be inserted. Moreover, in the last iteration of this loop, no agent will be inserted and hence, from all remaining isolated agents, no information flow to u is possible.

2. An edge $v \rightarrowtail w$ with $v \neq w$ is only inserted into the constructed policy if there is some set of agents W such that generalized-unwinding(v, $\{w\}$, W) does not return `false`. Hence, the system without the edge $v \rightarrowtail w$ is

insecure.

3. We prove this claim inductively by inserting agents into the layers. Clearly $d_u^{\rightarrowtail}(v) = \texttt{layer}(v)$ where $\rightarrowtail$ is the policy returned by the algorithm. By induction, every agent inserted into the policy has maximal distance to u. When a new agent is inserted, then this was necessary to preserve security and hence, the distance to u of this agent is maximal again. The edges inserted in line 14 do not reduce the distance of any inserted vertex to u. □

5.5 Beyond Decidability

In contrast to the previous result, the verification problems for the notions of to-security and ito-security are undecidable. The main difference between these two notions and all other notions in this thesis is that the allowed information depends on the observations of the agents. This is the deeper reason why one can build systems in which the length of a witness for insecurity cannot be bounded by the size of the system.

5.5.1 Undecidability of to-security

We will show that the verification problem for to-security is undecidable. The proof is a reduction from Post's Correspondence Problem [Pos46] introduced in the next paragraph. It is well-known that this problem is undecidable.

Post's Correspondence Problem (PCP) An instance of PCP consists of two finite sequences $x_1, \ldots, x_n$ and $y_1, \ldots, y_n$ of words over some alphabet Σ with at least two symbols. Such an instance is a positive instance if and only if there exists a sequence of indices $i_1, \ldots, i_k$ with $1 \leq i_j \leq n$ for all $1 \leq j \leq k$ such that

$$x_{i_1} \ldots x_{i_k} = y_{i_1} \ldots y_{i_k} \ .$$

We will give a reduction from PCP to to-insecurity, which shows that the verification problem for to-security is undecidable. This problem remains undecidable for a fixed policy with three agents. This improves the result in [EvdMSW11] where this result was only shown for at least four agents. Clearly, to-security is decidable for a policy with two agents, since the policy is transitive and hence, to-security and t-security coincide.

5.5.1 Theorem. *It is undecidable whether a system is to-secure, even for a fixed policy with three agents.*

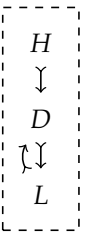

Figure 5.14. The modified HDL policy in the undecidability result of to-security

Proof. To prove the correctness of this reduction, we encode a PCP instance into a system M with agents H, D, L and a policy with edges $H \rightarrowtail D$, $D \rightarrowtail L$, and $L \rightarrowtail D$, as depicted in Figure 5.14, such that the instance is a positive instance if and only if the system is *not* to-secure. For this reduction consider a PCP instance given by some alphabet Σ and sequences of words $x_1, \ldots, x_n$ and words $y_1, \ldots, y_n$. With X and Y we denote the set of these words, respectively.

Intuitively, in the constructed system, the agent L guesses words over Σ. The agent H chooses whether these words will be compared with words from X or with words from Y. It also guesses when a possible sequence of actions guessed by L ends Additionally, it also guesses the corresponding index of the word, such that the sequence of actions guessed by L is exactly the word with H's guessed index from either the set X or Y. The agent D observes the guessed indices of H. It has a single action that ends this procedure and downgrades all of its observations to L. If the sequence of symbols guessed by L and the indices guessed by H match to a sequence

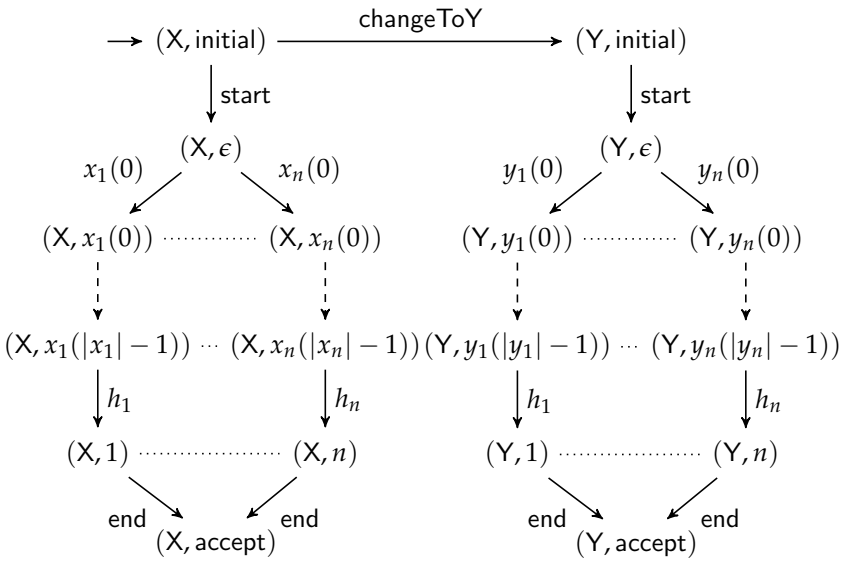

Figure 5.15. System constructed from a PCP instance

of words of X or of Y, depending on the initial choice of H, the agent L observes this choice of H.

The proof can be sketched as follows: If we have a positive instance, then there is a sequence of symbols such that these symbols are a sequence of words from both X and Y with the same sequence of indices. In this case, L has the same allowed information but observes whether this sequence was compared to X or to Y. Therefore the system is insecure.

If the instance is not positive, then there are no two guesses with the same sequence of symbols and the same indices matching both X and Y. One of these guesses cannot be expressed as a sequence of words from X or Y. The agent D can distinguish these cases and hence, the allowed information of L differs.

We proceed with the formal description of the constructed system, starting with the state space. With X and Y we denote just symbols that indicate whether the sequences of symbols are compared to words of X or

to words of Y, respectively. An overview of the structure of the system is illustrated in Figure 5.15—however, several edges are missing for clarity.

There are the following kinds of states:

1. $(\mathsf{X}, \mathsf{initial})$ and $(\mathsf{Y}, \mathsf{initial})$,
2. $(\mathsf{X}, \mathsf{accept})$ and $(\mathsf{Y}, \mathsf{accept})$,
3. reject
4. (X, γ) where γ is a prefix of some word from X, and (Y, γ) where γ is a prefix of some word from Y,
5. (X, j) and (Y, j) for every $1 \leq j \leq n$.

The initial state of the system is $s^I = (\mathsf{X}, \mathsf{initial})$.

The following actions can be performed:
Agent H has an action $\mathsf{changeToY}$ and actions h_i for guessing the index i for every $1 \leq i \leq n$. Agent D has a single action end. The actions of agent L are identified with the symbols of Σ, and in addition, it has an action start.

The transition function is defined by:

- $(\mathsf{X}, \mathsf{initial}) \cdot \mathsf{changeToY} = (\mathsf{Y}, \mathsf{initial})$,
- for every $z \in \{\mathsf{X}, \mathsf{Y}\}$ by

$$(z, \mathsf{initial}) \cdot \mathsf{start} = (z, \epsilon) \ ,$$

- for every $a \in \Sigma$ and every prefix γ of a word from z by

$$(z, \gamma) \cdot a = \begin{cases} (z, \gamma a) & \text{if } \gamma a \text{ is a prefix of a word from } z \\ \mathsf{reject} & \text{otherwise} \ , \end{cases}$$

- for every $1 \leq i \leq n$ and every prefix γ of a word from z by

$$(z, \gamma) \cdot h_i = \begin{cases} (z, i) & \text{if } \gamma \text{ is the } i\text{th word from the sequence } z \\ \mathsf{reject} & \text{otherwise} \ , \end{cases}$$

▷ for every $a \in \Sigma$ and $1 \leq i \leq n$ by

$$(z,i) \cdot a = \begin{cases} (z,a) & \text{if } a \text{ is a prefix of a word from } z \\ \mathsf{reject} & \text{otherwise} \end{cases},$$

▷ for every $1 \leq i \leq n$ by

$$(z,i) \cdot \mathsf{end} = \mathsf{accept} \ .$$

All not mentioned transitions lead to the state reject.

The observations of agent H are the same on all states. The observations of D are:

for every $z \in \{\mathsf{X},\mathsf{Y}\}$ and for every $1 \leq i \leq n$

$$\begin{aligned} \mathrm{obs}_D((z,i)) &= i \ , \\ \mathrm{obs}_D(\mathsf{reject}) &= \uparrow \ , \end{aligned}$$

and for all other states s

$$\mathrm{obs}_D(s) = \curvearrowright \ .$$

The symbol $\uparrow$ stands for a failure and $\curvearrowright$ for an ongoing run. The observations of L are

$$\begin{aligned} \mathrm{obs}_L((\mathsf{X},\mathsf{accept})) &= X \ , \\ \mathrm{obs}_L((\mathsf{Y},\mathsf{accept})) &= Y \ , \end{aligned}$$

and for all other states s

$$\mathrm{obs}_L(s) = \bot \ .$$

Intuitively, with the action changeToY, the agent H decides whether the following action sequence is compared to words from X or from Y. If this decision has been made, agent L starts the run with the start action. Once

performed, agent L proceeds with actions corresponding to symbols of the given alphabet Σ. However, these sequences have to match words in X or Y, depending on H's initial choice. Otherwise the system reaches the reject state. After L has performed a sequence of symbols matching a word, H guesses the index of the word. Again, if this guess is wrong, the system goes into the reject state. After such a guess of H, the agent L can either proceed with the next word or the agent D can perform the action end to signal the end of such a sequence of words and indices. In the latter case, the system reaches a state marked with accept and agent L observes whether the sequences of actions has been compared to words in X or in Y. The action end downgrades the sequence of H's index-guesses to L, since it was the observation of D. Hence, L obtains the sequence of performed actions corresponding to symbols from Σ and the sequence of matching indices. With this information, L can reconstruct whether this was a sequence of words from X or from Y, except it is possible that it is a sequence of words from both X and Y with the same indices. But the latter case is only possible if it is a positive instance. In that case, L can deduce whether the changeToY action has been performed in the initial state, which is not allowed by the to operator.

We will proceed now with a formal proof of Theorem 5.5.1. First, assume that the PCP instance is a positive one. Then there exists a sequence of indices $i_1, \ldots, i_k$ such that $x_{i_1} \cdots x_{i_k} = y_{i_1} \cdots y_{i_k}$. As explained, the corresponding traces of the system are

$$\alpha = \mathsf{start}\, x_{i_1}\, h_{i_1} \cdots x_{i_k}\, h_{i_k}\, \mathsf{end}$$

and

$$\beta = \mathsf{changeToY}\, \mathsf{start}\, y_{i_1}\, h_{i_1} \cdots y_{i_k}\, h_{i_k}\, \mathsf{end}\ .$$

We will show that these two traces are a witness for the to-insecurity of the system by applying Lemma 3.6.4. For agent L, we have for the tpurge values:

$$\mathrm{tpurge}_L(\alpha) = \mathsf{start}\, x_{i_1} \cdots x_{i_k}\, \mathsf{end} = \mathsf{start}\, y_{i_1} \cdots y_{i_k}\, \mathsf{end} = \mathrm{tpurge}_L(\beta)\ .$$

For agent D the tview values are:

$$\text{tview}_D(\alpha) = \curvearrowright i_1 \curvearrowright i_2 \cdots \curvearrowright i_k \,\mathsf{end} = \text{tview}_D(\beta) \ .$$

However, L's observations are different after these two traces:

$$\text{obs}_L(s^I \cdot \alpha) = X \neq Y = \text{obs}_L(s^I \cdot \beta) \ .$$

Hence the system is to-insecure.

For the other direction of the proof assume that the system is to-insecure. Again by Lemma 3.6.4, there are traces α and β such that $\text{tpurge}_L(\alpha) = \text{tpurge}_L(\beta)$, $\text{tview}_D(\alpha) = \text{tview}_D(\beta)$, and $\text{obs}_L(s^I \cdot \alpha) \neq \text{obs}_L(s^I \cdot \beta)$.

Since the observations of L after α and β are different, in at least one of these traces the action end has to appear and leads the system to reach either $(\mathsf{X}, \mathsf{accept})$ or $(\mathsf{Y}, \mathsf{accept})$. Since the end action appears in the tview value of D, it has to be performed in both traces. Note that after performing end, the system reaches either one of the states marked with accept or the state marked with reject, and hence, no further actions have any effects on the system. Therefore, we can, w.l.o.g., assume that both traces end with the action end.

Since D's tview values of both traces are the same, in none of these traces the system has reached the reject state before the end action where D's observation is $\uparrow$. And since one of these traces reaches an accept state, D has not $\uparrow$ in its tview. Hence, for both traces D's tview value has the form $\curvearrowright i_1 \cdots \curvearrowright i_k \,\mathsf{end}$. But from this tview value, it follows that the end action has been performed in either the state (X, i_k) or (Y, i_k). Hence the reached state is $(\mathsf{X}, \mathsf{accept})$ or $(\mathsf{Y}, \mathsf{accept})$, respectively. Since the observations of L are different after performing α and β, exactly one of the traces reaches $(\mathsf{X}, \mathsf{accept})$ and the other reaches $(\mathsf{Y}, \mathsf{accept})$. Hence exactly one trace starts with the action changeToY.

Since the sequences reach different accept states, the tpurge values of these traces w.r.t. L have to be

$$\mathsf{start}\, x_{i_1} \cdots x_{i_k}\, \mathsf{end}$$

and

$$\mathsf{start}\, y_{i_1} \cdots y_{i_k}\, \mathsf{end}\ ,$$

and both sequences have to be equal. This constitutes an evidence that this PCP instance is a positive one. □

5.5.2 Undecidability of ito-security

As for to-security, the verification problem for ito-security is also undecidable. We will show that by a reduction from to-security to ito-security.

First, we will provide the construction of the reduction. For a given system M, we construct a system M' such that the system M is to-secure if and only if the system M' is ito-secure.

Recall that the difference between the ito and the to operator is that for an agent $\mathrm{dom}(a)$ with $\mathrm{dom}(a) \rightarrowtail u$, but $\mathrm{dom}(a) \neq u$, the ito operator allows u to get the sequence $\mathrm{view}_{\mathrm{dom}(a)}(\alpha a)$. While the to operator allows u only to receive the sequence $\mathrm{view}_{\mathrm{dom}(a)}(\alpha)$. Note that nonetheless, the to operator transmits the action a after the sequence α. Therefore, the only additional information that is transmitted is the observation after the action a. This is also only important if the action a is the last action of $\mathrm{dom}(a)$, since otherwise the observation after the action a is eventually transmitted by the next action performed by $\mathrm{dom}(a)$.

The reduction handles this by introducing new actions, so-called final actions. Such an action corresponds to the last action of each agent performed in a trace. After such an action, the observation of that agent changes to the uninformative information $\perp$ and will not be changed anywise. The additional information that is transmitted by the ito operator is artificially removed. Therefore, the reduced system has to keep track of the agents that have already performed one of their final actions.

More precisely, a system M with states S, initial state s^I, actions A, transition function step, observation function obs, agents D, and the corresponding dom function is transformed into a system M' with states S', initial state $\tilde{s}^I$, actions A', transition function step$'$, same agents D, and dom$'$ function as follows.

A new set of additional actions $A^{\mathsf{f}} = \{a^{\mathsf{f}} \mid a \in A\}$ with $\mathrm{dom}'(a^{\mathsf{f}}) = \mathrm{dom}(a)$, denoted as final actions, is introduced. The states and actions of the created systems are:

$$\begin{aligned} S' &= S \times \mathcal{P}(D) \ , \\ \tilde{s}^I &= (s^I, \emptyset) \ , \\ A' &= A \cup A^{\mathsf{f}} \ . \end{aligned}$$

The observation function of the new system is defined for every $s \in S$ and $U \subseteq D$ by

$$\mathrm{obs}'_u(s, U) = \begin{cases} \mathrm{obs}_u(s) & \text{if } u \notin U \\ \bot & \text{otherwise} \ . \end{cases}$$

The transition function is in both systems denoted with $\cdot$, since the intended system is clear from the structure of the state space. For any $s \in S$, any action $a \in A$, and any $U \subseteq D$, the transition function is defined as:

$$(s, U) \cdot a = \begin{cases} (s \cdot a, U) & \text{if } \mathrm{dom}'(a) \notin U \text{ and } a \in A \\ (s \cdot b, U \cup \{\mathrm{dom}'(a)\}) & \text{if } \mathrm{dom}'(a) \notin U \text{ and } a \in A^{\mathsf{f}} \\ & \qquad \text{with } a = b^{\mathsf{f}} \\ (s, U) & \text{if } \mathrm{dom}'(a) \in U \ . \end{cases}$$

For clarity, for functions like view, tview, etc., we use primed functions if the function refers to the system M', and the usual function name if it refers to the original system M.

Before proving the correctness of this construction, we need a function that transforms traces of one system into traces of the other system. For transforming traces of the original system into traces of the system constructed by the reduction, we define a function

$$\mathrm{convert}\colon D \times A^* \to A'^* \ .$$

Applied to an agent u and a trace α, it replaces in α for each agent $v \neq u$

with $v \rightarrowtail u$ the last action a with $\mathrm{dom}(a) = v$ by the action a^{f}.

For transforming traces of the reduced system back into traces of the original system, we define a function convertback: $A'^* \to A^*$ that transforms final actions into the corresponding non-final actions. If an agent has no final action in a trace, none of its actions will be removed. For any action $a \in A'$, let $\overline{a} = a$ if $a \in A$ and $\overline{a} = b$ if $a \in A^{\mathsf{f}}$ with $a = b^{\mathsf{f}}$. More precisely, the function convertback is inductively defined by

$$\mathrm{convertback}(\epsilon) = \epsilon$$

$$\mathrm{convertback}(\alpha a) = \begin{cases} \mathrm{convertback}(\alpha) & \text{if there is a final action} \\ & \text{of } \mathrm{dom}'(a) \text{ in } \alpha \\ \mathrm{convertback}(\alpha)\,\overline{a} & \text{if there is no final action} \\ & \text{of } \mathrm{dom}'(a) \text{ in } \alpha \ . \end{cases}$$

The next lemma collects some properties of the function convertback.

5.5.2 Lemma. *For every $u \in D$ and every $\alpha \in A'^*$ that does not contain a final action of u, we have*

1. $\mathrm{obs}_u(s^I \cdot \mathrm{convertback}(\alpha)) = \mathrm{obs}'_u(\tilde{s}^I \cdot \alpha)$, *and*
2. $\mathrm{view}_u(\mathrm{convertback}(\alpha)) = \mathrm{view}'_u(\alpha)$.

Proof. 1. By the construction of the system M', if an agent has performed one of its final actions, all further actions of this agent will be ignored, i. e., the corresponding transitions loop. Therefore, any trace $\alpha \in A'^*$ leads in M' to essentially the same state as the trace $\mathrm{convertback}(\alpha)$ in M. The precise meaning of that is: If $\tilde{s}^I \cdot \alpha = (s, U)$ for some $s \in S$ and $U \subseteq D$, then $s = s^I \cdot \mathrm{convertback}(\alpha)$. If $\alpha \in A'^*$ does not contain any final action of u, then $s^I \cdot \alpha$ is some state (s, U) with $u \notin U$ and therefore $\mathrm{obs}'_u(s, U) = \mathrm{obs}_u(s)$. This gives

$$\mathrm{obs}_u(s^I \cdot \mathrm{convertback}(\alpha)) = \mathrm{obs}'_u(\tilde{s}^I \cdot \alpha) \ .$$

2. This claim is an extension of the previous one to the view_u function. We will prove it by an induction on α.

The claim clearly holds for the base case $\alpha = \epsilon$. Suppose that $\alpha = \alpha' a$ for some $a \in A'$ and $\alpha' \in A'^*$ and assume that α does not contain a final action of u and that for α' the induction hypothesis holds. Hence, we have as induction hypothesis:

$$\text{view}_u(\text{convertback}(\alpha')) = \text{view}'_u(\alpha) \ . \tag{I.H.}$$

Consider the following cases:
Case 1: $\text{dom}'(a) = u$.
Then a is not a final action and α' contains no final action of u, so $\text{convertback}(\alpha' a) = \text{convertback}(\alpha')\, a$ and

$$\begin{aligned}
\text{view}'_u(\alpha' a) &= \text{view}'_u(\alpha')\, a\, \text{obs}'_u(\tilde{s}_0^I \cdot \alpha' a) \\
&\overset{\text{(I.H.)}}{=} \text{view}_u(\text{convertback}(\alpha'))\, a\, \text{obs}'_u(\tilde{s}_0^I \cdot \alpha' a) \\
&= \text{view}_u(\text{convertback}(\alpha'))\, a\, \text{obs}_u(s^I \cdot \text{convertback}(\alpha' a)) \\
&= \text{view}_u(\text{convertback}(\alpha'))\, a\, \text{obs}_u(s^I \cdot \text{convertback}(\alpha') a) \\
&= \text{view}_u(\text{convertback}(\alpha') a) \\
&= \text{view}_u(\text{convertback}(\alpha' a)) \ .
\end{aligned}$$

Case 2: $\text{dom}'(a) \neq u$ and there is no final action of $\text{dom}'(a)$ in α'.
Then $\text{convertback}(\alpha' a) = \text{convertback}(\alpha')\, \overline{a}$, therefore,

$$\begin{aligned}
\text{view}'_u(\alpha' a) &= \text{view}'_u(\alpha') \hat{\circ} \text{obs}'_u(\tilde{s}_0^I \cdot \alpha' a) \\
&\overset{\text{(I.H.)}}{=} \text{view}_u(\text{convertback}(\alpha')) \,\hat{\circ}\, \text{obs}'_u(\tilde{s}_0^I \cdot \alpha' a) \\
&= \text{view}_u(\text{convertback}(\alpha')) \,\hat{\circ}\, \text{obs}_u(s^I \cdot \text{convertback}(\alpha' a)) \\
&= \text{view}_u(\text{convertback}(\alpha')) \,\hat{\circ}\, \text{obs}_u(s^I \cdot \text{convertback}(\alpha') a) \\
&= \text{view}_u(\text{convertback}(\alpha') a) \\
&= \text{view}_u(\text{convertback}(\alpha' a)) \ .
\end{aligned}$$

Case 3: $\text{dom}'(a) \neq u$ and there is a final action of $\text{dom}'(a)$ in α'.

Then convertback$(\alpha' a) =$ convertback(α'), therefore,

$$\begin{aligned}
\text{view}'_u(\alpha' a) &= \text{view}'_u(\alpha') \,\hat{\circ}\, \text{obs}'_u(\tilde{s}_0^I \cdot \alpha' a) \\
&\overset{\text{(I.H.)}}{=} \text{view}_u(\text{convertback}(\alpha')) \,\hat{\circ}\, \text{obs}'_u(\tilde{s}_0^I \cdot \alpha' a) \\
&= \text{view}_u(\text{convertback}(\alpha')) \,\hat{\circ}\, \text{obs}_u(s^I \cdot \text{convertback}(\alpha' a)) \\
&= \text{view}_u(\text{convertback}(\alpha')) \,\hat{\circ}\, \text{obs}_u(s^I \cdot \text{convertback}(\alpha')) \\
&= \text{view}_u(\text{convertback}(\alpha')) \\
&= \text{view}_u(\text{convertback}(\alpha' a)) \ .
\end{aligned}$$

This shows the inductive step and therefore, we have

$$\text{view}_u(\text{convertback}(a\alpha')) = \text{view}'_u(a\alpha') \ . \qquad \square$$

The correctness of the reduction provided before the previous lemma is stated in the next lemma.

5.5.3 Lemma. *A system M is to-secure iff the system M', constructed as described above, is ito-secure.*

Proof. First, we show the implication from left to right. For that, we will show that for every $u \in D$ and every $\alpha, \beta \in A'^*$ that do not contain a final action of u, with

$$\text{tpurge}'_u(\alpha) = \text{tpurge}'_u(\beta)$$

and

$$\text{ftview}'_v(\alpha) = \text{ftview}'_v(\beta) \text{ for all } v \neq u, v \rightarrowtail u \ ,$$

we have

$$\text{tpurge}_u(\text{convertback}(\alpha)) = \text{tpurge}_u(\text{convertback}(\beta))$$

and

$$\text{tview}_v(\text{convertback}(\alpha)) = \text{tview}_v(\text{convertback}(\beta)) \text{ for all } v \neq u, v \rightarrowtail u \ .$$

We prove the claim by an induction on the combined length of α and β. This claim clearly holds for the base case $\alpha = \beta = \epsilon$.

Consider $\alpha = \alpha' a$ and β, neither containing a final action of agent u, such that $\text{tpurge}'_u(\alpha' a) = \text{tpurge}'_u(\beta)$ and $\text{ftview}'_v(\alpha' a) = \text{ftview}'_v(\beta)$ for all $v \neq u$ with $v \rightarrowtail u$ hold. As induction hypothesis, suppose that the implication holds for all traces of smaller combined length.

We consider the following cases:
Case 1: $\text{dom}'(a) \not\rightarrowtail u$.
This gives

$$\text{tpurge}'_u(\alpha') = \text{tpurge}'_u(\alpha' a) = \text{tpurge}'_u(\beta)$$

and

$$\text{ftview}'_v(\alpha') = \text{ftview}'_v(\alpha' a) = \text{ftview}'_v(\beta)$$

for all $v \neq u$, $v \rightarrowtail u$. By applying the induction hypothesis, we obtain

$$\text{tpurge}_u(\text{convertback}(\alpha')) = \text{tpurge}_u(\text{convertback}(\beta))$$

and

$$\text{tview}_v(\text{convertback}(\alpha')) = \text{tview}_v(\text{convertback}(\beta))$$

for every $v \neq u$ with $v \rightarrowtail u$.

If there is a final action of $\text{dom}'(a)$ in α', then $\text{convertback}(\alpha' a) = \text{convertback}(\alpha')$ and the claim for the pair $\alpha' a$ and β follows directly by applying the induction hypothesis.

From now on, we assume that there is no final action of $\text{dom}'(a)$ in α'. Hence, since $\text{dom}(\overline{a}) = \text{dom}'(a) \not\rightarrowtail u$, we have

$$\begin{aligned}\text{tpurge}_u(\text{convertback}(\alpha' a)) &= \text{tpurge}_u(\text{convertback}(\alpha')\,\overline{a}) \\ &= \text{tpurge}_u(\text{convertback}(\alpha')) \\ &= \text{tpurge}_u(\text{convertback}(\beta))\ ,\end{aligned}$$

and for all $v \neq u$, $v \rightarrowtail u$, we have:

$$\begin{aligned}\mathrm{tview}_v(\mathrm{convertback}(\alpha' a)) &= \mathrm{tview}_v(\mathrm{convertback}(\alpha')\,\overline{a}) \\ &= \mathrm{tview}_v(\mathrm{convertback}(\alpha')) \\ &= \mathrm{tview}_v(\mathrm{convertback}(\beta))\ .\end{aligned}$$

Case 2: $\mathrm{dom}'(a) \rightarrowtail u$.
In this case, we have $\mathrm{tpurge}'_u(\alpha')\,a = \mathrm{tpurge}'_u(\alpha' a) = \mathrm{tpurge}'_u(\beta)$, and hence, $\beta = \beta' b$ for some action b. If $\mathrm{dom}'(b) \not\rightarrowtail u$, then we may swap the roles of α and β and apply the previous case. Hence, without loss of generality, $\mathrm{dom}'(b) \rightarrowtail u$, and we have $\mathrm{tpurge}'_u(\beta) = \mathrm{tpurge}'_u(\beta')\,b$. It follows that $a = b$ and $\mathrm{tpurge}'_u(\alpha') = \mathrm{tpurge}'_u(\beta')$. For every $v \neq u$ with $v \rightarrowtail u$ and $v \neq \mathrm{dom}(a)$, we have

$$\begin{aligned}\mathrm{ftview}'_v(\alpha') &= \mathrm{ftview}'_v(\alpha' a) \\ &= \mathrm{ftview}'_v(\beta' a) \\ &= \mathrm{ftview}'_v(\beta')\ .\end{aligned}$$

If $v = \mathrm{dom}'(a)$, we have

$$\begin{aligned}\mathrm{view}'_v(\alpha' a) &= \mathrm{ftview}'_v(\alpha' a) \\ &= \mathrm{ftview}'_v(\beta' a) \\ &= \mathrm{view}'_v(\beta' a)\ .\end{aligned}$$

This implies $\mathrm{view}'_v(\alpha') = \mathrm{view}'_v(\beta')$ and therefore, we have $\mathrm{ftview}'_v(\alpha') = \mathrm{ftview}'_v(\beta')$.

Thus, by the induction hypothesis,

$$\mathrm{tpurge}_u(\mathrm{convertback}(\alpha')) = \mathrm{tpurge}_u(\mathrm{convertback}(\beta'))$$

and

$$\mathrm{tview}_v(\mathrm{convertback}(\alpha')) = \mathrm{tview}_v(\mathrm{convertback}(\beta'))$$

for every $v \neq u$ with $v \rightarrowtail u$.

If there is a final action of $\mathrm{dom}'(a)$ in α' then, by $\mathrm{tpurge}'_u(\alpha') = \mathrm{tpurge}'_u(\beta')$, there is also a final action of $\mathrm{dom}'(a)$ in β', and hence, we have both $\mathrm{convertback}(\alpha' a) = \mathrm{convertback}(\alpha')$ and $\mathrm{convertback}(\beta' a) = \mathrm{convertback}(\beta')$. The desired conclusion directly follows from the inductive conclusion above. Alternately, if there is no final action of $\mathrm{dom}'(a)$ in α', then there is no final action of $\mathrm{dom}'(a)$ in β'. In this case, since $\mathrm{dom}(\overline{a}) = \mathrm{dom}'(a) \rightarrowtail u$, we have

$$\begin{aligned}
\mathrm{tpurge}_u(\mathrm{convertback}(\alpha' a)) &= \mathrm{tpurge}_u(\mathrm{convertback}(\alpha')\,\overline{a}) \\
&= \mathrm{tpurge}_u(\mathrm{convertback}(\alpha'))\,\overline{a} \\
&= \mathrm{tpurge}_u(\mathrm{convertback}(\beta'))\,\overline{a} \\
&= \mathrm{tpurge}_u(\mathrm{convertback}(\beta')\,\overline{a}) \\
&= \mathrm{tpurge}_u(\mathrm{convertback}(\beta' a)) \ .
\end{aligned}$$

In the case $\mathrm{dom}'(a) \neq v$, for every $v \neq u$ with $v \rightarrowtail u$, we have

$$\begin{aligned}
\mathrm{tview}_v(\mathrm{convertback}(\alpha' a)) &= \mathrm{tview}_v(\mathrm{convertback}(\alpha')\,\overline{a}) \\
&= \mathrm{tview}_v(\mathrm{convertback}(\alpha')) \\
&= \mathrm{tview}_v(\mathrm{convertback}(\beta')) \\
&= \mathrm{tview}_v(\mathrm{convertback}(\beta')\,\overline{a}) \\
&= \mathrm{tview}_v(\mathrm{convertback}(\beta' a)) \ .
\end{aligned}$$

In the case $\mathrm{dom}'(a) = v$ we argue as follows. If a is not a final action, then we have

$$\begin{aligned}
\mathrm{view}'_{\mathrm{dom}'(a)}(\alpha' a) &= \mathrm{ftview}'_{\mathrm{dom}'(a)}(\alpha' a) \\
&= \mathrm{ftview}'_{\mathrm{dom}'(a)}(\beta' a) \\
&= \mathrm{view}'_{\mathrm{dom}'(a)}(\beta' a) \ .
\end{aligned}$$

Since there is no final action of $\mathrm{dom}'(a)$ in α' or β', and since a is not a final action, there is no final action of $\mathrm{dom}'(a)$ in $\alpha' a$ or $\beta' a$, and therefore,

$$\mathrm{view}_{\mathrm{dom}(a)}(\mathrm{convertback}(\alpha' a)) = \mathrm{view}_{\mathrm{dom}(a)}(\mathrm{convertback}(\beta' a)) \ ,$$

using the equivalence proved above. From this equation, it follows that

$$\text{tview}_{\text{dom}(a)}(\text{convertback}(\alpha' a)) = \text{tview}_{\text{dom}(a)}(\text{convertback}(\beta' a)) \ .$$

In the case that a is a final action, we have

$$\begin{aligned} \text{view}'_{\text{dom}'(a)}(\alpha')\, a\, \bot &= \text{ftview}'_{\text{dom}'(a)}(\alpha' a) \\ &= \text{ftview}'_{\text{dom}'(a)}(\beta' a) \\ &= \text{view}'_{\text{dom}(a)'}(\beta')\, a\, \bot \ . \end{aligned}$$

Therefore $\text{view}'_{\text{dom}'(a)}(\alpha') = \text{view}'_{\text{dom}'(a)}(\beta')$.

Since there is no final action of $\text{dom}'(a)$ in α' or β', it follows, using the equivalence proved above, that

$$\text{view}_{\text{dom}(a)}(\text{convertback}(\alpha')) = \text{view}_{\text{dom}(a)}(\text{convertback}(\beta')) \ .$$

Thus,

$$\begin{aligned} \text{tview}_{\text{dom}(a)}(\text{convertback}(\alpha' a)) &= \text{tview}_{\text{dom}(a)}(\text{convertback}(\alpha')\, \overline{a}) \\ &= \text{view}_{\text{dom}(a)}(\text{convertback}(\alpha'))\, \overline{a} \\ &= \text{view}_{\text{dom}(a)}(\text{convertback}(\beta'))\, \overline{a} \\ &= \text{tview}_{\text{dom}(a)}(\text{convertback}(\beta' a)) \ . \end{aligned}$$

This completes the proof of the claim.

For completing the proof of the direction from left to right, suppose that M is to-secure, but assume that M' is not ito-secure. By Lemma 3.6.8, there exist $\alpha, \beta \in A'^*$ and an agent u such that

$$\text{tpurge}'_u(\alpha) = \text{tpurge}'_u(\beta) \ ,$$

$$\text{ftview}'_v(\alpha) = \text{ftview}'_v(\beta) \text{ for every } v \neq u \text{ with } v \rightarrowtail u \ ,$$

and

$$\mathrm{obs}'_u(\tilde{s}_0^I \cdot \alpha) \neq \mathrm{obs}'_u(\tilde{s}_0^I \cdot \beta) \ .$$

It follows from $\mathrm{tpurge}'_u(\alpha) = \mathrm{tpurge}'_u(\beta)$ that α contains a final action of agent u if and only if β contains a final action of agent u. But if both contain such a final action, then

$$\mathrm{obs}'_u(\tilde{s}_0^I \cdot \alpha) = \bot = \mathrm{obs}'_u(\tilde{s}_0^I \cdot \beta) \ ,$$

which contradicts the assumption. Thus neither α nor β contain a final action of u. Therefore, we have

$$\mathrm{tpurge}_u(\mathrm{convertback}(\alpha)) = \mathrm{tpurge}_u(\mathrm{convertback}(\beta))$$

and

$$\mathrm{ftview}_v(\mathrm{convertback}(\alpha)) = \mathrm{ftview}_v(\mathrm{convertback}(\beta))$$

for all agents $v \neq u$ with $v \rightarrowtail u$. We also have

$$\mathrm{obs}_u(s^I \cdot \mathrm{convertback}(\alpha)) \neq \mathrm{obs}_u(s^I \cdot \mathrm{convertback}(\beta)) \ .$$

By the characterization of to-security of Lemma 3.6.4, this implies that M is not to-secure.

For the other direction of the proof, we use the function convert as defined above.

We observe for all $u \in D$ that if γ, γ' are prefixes of α and $\mathrm{convert}_u(\alpha)$, respectively, of the same length, then for all $U \subseteq D$, if $\tilde{s}^I \cdot \gamma' = (s, U)$, then $s = s^I \cdot \gamma$. Therefore, since $\mathrm{convert}_u(\alpha)$ contains no final action of agent u, we have $\mathrm{obs}_u(s^I \cdot \mathrm{convert}_u(\alpha)) = \mathrm{obs}_u(s^I \cdot \alpha)$. Moreover, if γ, γ' are prefixes of the same length of α and $\mathrm{convert}_u(\alpha)$, respectively, and γ does not contain the rightmost action of agent v in α (if any), then $\mathrm{view}_v(\gamma) = \mathrm{view}_v(\gamma')$.

We show for all $u \in D$ and all $\alpha, \alpha' \in A^*$ that if $\mathrm{tpurge}_u(\alpha) = \mathrm{tpurge}_u(\alpha')$ and $\mathrm{tview}_v(\alpha) = \mathrm{tview}_v(\alpha')$ for all $v \neq u$, $v \rightarrowtail u$, then

$$\mathrm{tpurge}'_u(\mathrm{convert}_u(\alpha)) = \mathrm{tpurge}'_u(\mathrm{convert}_u(\alpha'))$$

and

$$\mathrm{ftview}'_v(\mathrm{convert}_u(\alpha)) = \mathrm{ftview}'_v(\mathrm{convert}_u(\alpha'))$$

for every $v \neq u$ with $v \rightarrowtail u$.

By an argument similar to that for the opposite direction, it then follows that if M' is ito-secure then M is to-secure.

We observe that for any agent u, the functions tpurge_u and $\mathrm{convert}_u$ commute. This shows for all $\alpha, \alpha' \in A^*$ that if $\mathrm{tpurge}_u(\alpha) = \mathrm{tpurge}_u(\alpha')$, then $\mathrm{tpurge}'_u(\mathrm{convert}_u(\alpha)) = \mathrm{tpurge}'_u(\mathrm{convert}_u(\alpha'))$.

To complete the argument, we assume $\mathrm{tpurge}_u(\alpha) = \mathrm{tpurge}_u(\alpha')$ and for every agent $v \neq u$ with $v \rightarrowtail u$, we have $\mathrm{tview}_v(\alpha) = \mathrm{tview}_v(\alpha')$, and we will show that $\mathrm{ftview}'_v(\mathrm{convert}_u(\alpha)) = \mathrm{ftview}'_v(\mathrm{convert}_u(\alpha'))$ holds.

From $\mathrm{tpurge}_u(\alpha) = \mathrm{tpurge}_u(\alpha')$ it follows that the sequences of actions of agent v in α and α' are the same. In particular, if neither sequence contains an action of agent v, then the claim is trivial. Suppose that a is the last action of agent v in both α and α'. Then we may write $\alpha = \alpha_1 a \alpha_2$ and $\alpha' = \alpha'_1 a \alpha'_2$, where α_2, α'_2 contain no actions of agent v. Since $\mathrm{tview}_v(\alpha) = \mathrm{tview}_v(\alpha')$, we have $\mathrm{view}_v(\alpha_1) = \mathrm{view}_v(\alpha'_1)$. Also $\mathrm{convert}'_u(\alpha) = \gamma_1 a^{\mathsf{f}} \gamma_2$ and $\mathrm{convert}'_u(\alpha') = \gamma'_1 a^{\mathsf{f}} \gamma'_2$ hold, where γ_1, γ'_1 are of the same length as α_1, α'_1, respectively, and γ_2, γ'_2 contain no actions of agent v. Thus, using the observation above, we obtain

$$\begin{aligned}
\mathrm{ftview}'_v(\mathrm{convert}_u(\alpha)) &= \mathrm{ftview}'_v(\gamma_1 a^{\mathsf{f}} \gamma_2) \\
&= \mathrm{view}'_v(\gamma_1)\, a^{\mathsf{f}} \perp \\
&= \mathrm{view}'_v(\alpha_1)\, a^{\mathsf{f}} \perp \\
&= \mathrm{view}'_v(\alpha'_1)\, a^{\mathsf{f}} \perp \\
&= \mathrm{view}'_v(\gamma'_1)\, a^{\mathsf{f}} \perp \\
&= \mathrm{ftview}'_v(\mathrm{convert}_u(\alpha'))\ . \qquad \square
\end{aligned}$$

From the construction, it is clear that this system constructed by a computable function. Hence, from the previous lemma and the undecidability result for to-security it follows:

5.5.4 Theorem. *It is undecidable whether a system is ito-secure, even for a fixed policy with three agents.*

5.6 Beyond Deterministic Finite-State Machines

In this thesis, we mainly consider *deterministic* machines. However, several results can be adapted to nondeterministic finite-state machines or semantically richer machine models, for example pushdown systems. In particular, the trace-based security definitions are independent of the underlying machine model. However, reasonable security definitions for nondeterministic systems are not as obvious as they might seem. There is a vast bulk of literature dealing with noninterference-like properties for nondeterministic systems. Several of them are listed in Section 7.2.

In this section, we discuss only the complexity of the verification problem for some richer machine models, including nondeterministic finite-state machines. Essentially, nearly every input-enabled machine or automaton model can be used to model noninterference properties in a similar way as we do in this work. We claim that the complexity of the verification problem for noninterference properties like t-security is polynomially equivalent to the complexity of the equivalence problem of the underlying automata model: The problem of deciding whether two given automata accept the same language.

Since this is not a precise statement, we cannot provide a proof for it. Instead, we will give a general construction pattern of how such reductions work without restricting ourselves to a particular automata model.

We require that the automata have a complete transition relation, i. e., for every configuration and every element of the input alphabet, there is a transition that can be applied.

Hardness We will give a reduction that transforms two automata into a single system which is t-secure w.r.t. the HL policy if and only if these two automata accept the same language.

Let M_0 and M_1 be two automata of the same automata model. Let $S_0^{M_0}$ and $S_0^{M_1}$ be the sets of initial states, respectively. Let A be the common

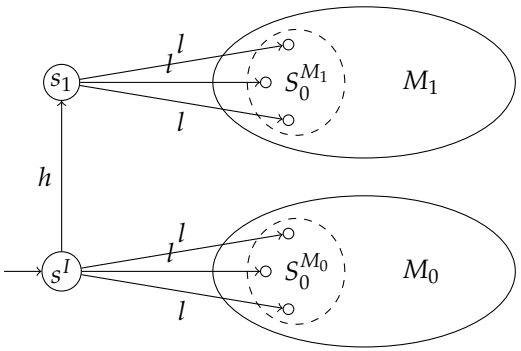

Figure 5.16. Reduction from automata equivalence to t-security

input alphabet. Let h and l be two new symbols, not contained in A. The constructed system M, as depicted in Figure 5.16, contains two new states s^I and s_1 and all states of M_0 and M_1. We assume that the set of states of M_0 is disjoint to the set of states of M_1. The actions of M are $A \cup \{h, l\}$. In addition to all transitions of M_0 and M_1, the constructed system M has a transition from s^I to s_1 labeled with h and transitions from s^I to all initial states of M_0 labeled with l and from s_1 to all initial states of M_1 also labeled with l. The state s^I is the only initial state of the system M. All transitions not mentioned, corresponding to the actions h and l, loop. The agents are H and L with $\mathrm{dom}(h) = H$ and $\mathrm{dom}(a) = L$ for all $a \in A \cup \{l\}$. For the policy we require that there is no edge from L to H. Agent H has constant observations and agent L observes whether a state is accepting or not. Such a system can only be insecure if L observes if the transition from s^I to s_1 labeled with h has been taken or not. This is the case if and only if there is a word that is accepted by exactly one of the two systems of M_0 and M_1.

Note that the system M is deterministic if the systems M_0 and M_1 are deterministic and, in particular, M's set of initial states has only one element.

Completeness For the other direction, we transform a system with the HL policy into two automata such that these automata accept the same

language if and only if this system is t-secure.

Let M be some system. Consider the policy with agents H and L such that $L \not\rightarrowtail H$. We assume w.l.o.g. that H has constant observations and L has only two possible observations on the set of configurations, say 0 and 1.

We construct two automata M_0 and M_1 as follows. The automaton M_0 is the same as the system M. The automaton M_1 is the same as M, but all transitions labeled with actions of H are removed and for preserving the completeness of the transition function, loops, which do not change the configuration of the system, are inserted. The accepting configurations of these two systems are exactly those in which L has observation 1.

Intuitively, the automaton M_1 purges the input word and ignores all inputs from H. Hence, for any word α, these two automata have the same acceptance behavior if and only if α and $\mathrm{tpurge}_L(\alpha)$ lead to the same observation for L in the original system M.

Related Results The complexity of t-security adapted to nondeterministic finite-state systems was analyzed in [vdMZ07]. They showed that the verification problem is PSPACE-complete. This is also the complexity one would expect from this construction pattern, since it is the complexity of the equivalence problem for nondeterministic finite automata [Kle56].

In [DRS05], D'Souza et al. showed that several of Mantel's basic security predicates (BSPs) [Man00, Man03], which can also be used to express t-security-like properties on nondeterministic finite-state systems, can be verified in PSPACE.

In [DHK+08], the verification problem for BSPs was analyzed on pushdown systems and it was shown that it is undecidable. One would also obtain an undecidability result from the construction above, since the equivalence problem for pushdown automata is undecidable.

Barthe et al. [BDR11] presented a technique for analyzing information flows by self-composition in the context of language-based information flow security, which has some analogy to this construction.

5.7 Excursus: The Disjoint-Set Data Structure

Several of the algorithms in this chapter made extensive use of a disjoint-set data structure [Tar75] for maintaining a collection of pairwise disjoint sets. They used this data structure to maintain partitions of the state space, derived from the state-based unwinding conditions. We provide an excursus on this data structure and its associated algorithms.

The main feature of this data structure is that it allows the union of pair-wise disjoint sets. The definitions of the data structure and the corresponding algorithms are summarized in Figure 5.17. Each set in the the maintained collection is identified by a representative which is an element of this set. Such a set is created by the function MAKE-SET(x) which creates a singleton set containing x. In this case, x is the representative of this set. It is required that x is not an element of any other set. For every element x, the representative of the set, which x belongs to, can be requested by FIND(x).

To merge two disjoint sets, the function UNION(x, y) creates the union of the set containing x and of the set containing y. The new representative of this union is either the previous representative of the set containing x or the one of the set containing y. The choice depends on an internal ranking function. The previous sets containing x and y are removed from the data structure and substituted by the union, to have again pairwise disjoint sets.

In the disjoint-set data structure, each set is represented as a rooted tree. The root of a tree is the representative and all edges are directed towards the root. To be highly efficient, the data structure uses two heuristics, called *union by rank* and *path compression*.

The idea of the union by rank heuristic is to keep track of the size of the trees and if the union operation is applied to let the the smaller tree become a subtree of the larger tree. However, the rank is not the exact size of the tree, instead, it is an upper bound for the height of the tree.

The path compression heuristic is used to reduce the length of paths within the trees. If the FIND(x) method is called, the path of x and all entries on this path to the root of the tree are reduced to one (if x is not the root itself). This not only reduces the length of the paths of these entries to the root, but also it reduces the distance of all subtrees of any of these entries to the root.

A precise analysis of the running times of these procedures can be found in [Tar75].

We will give now the mentioned procedures in pseudo code, following the presentation in [CLRS01]. Let X be the set that should be maintained. For every entry $x \in X$, there is an array $\textsc{rank}[x]$ that stores an integer value, which represents the rank of the value x. The value $\textsc{parent}[x]$ is a pointer to the successor of x in its tree. The procedure for the union operation has a sub procedure, called Link, which effectively performs the union of two sets.

Function Make-Set(x)

```
1 set PARENT[x] = x ;
2 set RANK[x] = 0 ;
```

Function Union(x,y)

```
1 LINK(FIND(x), FIND(y)) ;
```

Function Link(x, y)

```
1 if RANK[x] > RANK[y] then
2 |  set PARENT[y] = x ;
3 else
4 |  set PARENT[x] = y ;
5 |  if RANK[x] = RANK[y] then
6 |  |  set RANK[y] = RANK[y] + 1
```

Function Find-Set(x)

```
1 if x ≠ PARENT[x] then
2 |  set PARENT[x] = FIND(PARENT[x]) ;
3 return PARENT[x] ;
```

Figure 5.17. Procedures of the disjoint-set data structure

For defining the running times of these procedures, we need the Ackermann function and its inverse. The Ackermann function $A\colon \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

is inductively defined by

$$\begin{aligned} A(0,m) &= m+1 \\ A(n+1,0) &= A(n,1) \\ A(n+1,m+1) &= A(n,A(n+1,m)) \end{aligned}$$

The generalized inverse of the Ackermann function is defined as

$$\alpha(n) = \min\{k \mid A(k,1) \geq n\} \ .$$

The running time of these procedures can now be expressed with help of the generalized inverse of the Ackermann function.

5.7.1 Theorem. *A sequence of m* Make-Set, Union *and* Find *operations on a set with n elements can be performed in $O(m \cdot \alpha(n))$.*

The interpretation of this theorem is that each of this operations has nearly constant time running times on an amortized analysis, since the value $\alpha(n)$ is less than 5 for all practical issues.

Chapter 6

Application to Access Control

Historically, the development of noninterference and access control systems share a strong a connection. Detection of covert channels in access control systems was one of the earliest applications of noninterference and the initial motivation for developing this general theory of information flow security. Already in the seminal work of Goguen and Meseguer [GM82], their theory was motivated by applications to access control systems. Later, the observation that intransitive effects of information flow arise in access control systems very naturally, led to the development of the theory of intransitive noninterference [HY87]. Also their generalizations [Rus92] and corrections [vdM07] were motivated by applications to access control systems. However, these results only analyzed information flow in access control systems with a fixed or static access control mechanism, also known as mandatory access control. For analyzing information flows in these systems, Rushby [Rus92] and van der Meyden [vdM07] showed that intransitive noninterference with static polices is an appropriate theory. In systems with dynamic rights—rights which may change during the run of the access control system—a rigorous theory for formulating and analyzing information flows was missing. In this chapter, we close this gap—at least partially—by demonstrating that the theory of dynamic intransitive noninterference, developed in the previous chapter, can successfully be applied to an access control system with a distributed and dynamic access control mechanism.

We will give a brief introduction to access control in general in Section 6.1 and discuss information flow in mandatory access control systems in Section 6.2. We introduce discretionary access control systems in Section 6.3 combined with an introduction to a distributed dynamic access

control system called Flume [KT09]. In the remaining Section 6.4, we translate the Flume system into our system model and analyze it carefully with regard to dynamic noninterference.

6.1 Introduction to Access Control

In general, systems that enforce confidentiality and integrity properties are called access control systems. Access control systems (or systems equipped with an access control mechanism), are used to protect data from unauthorized access or manipulation. The permission to access data is called authorization.

In an access control system, a usual distinction is between the active parts that can read or write data, and those parts which should be protected from unintended access or modification. The former could be users or processes and are called *subjects*, the latter are mostly files, variables, or some part of the memory and are called *objects*. Additionally, there is a fixed number of access controls or rights that a subject may have on an object. The earliest formalization goes back to Lampson [Lam74] and has been refined by Graham and Denning [GD72].

The simplest structure of an access control system is an access control matrix M, where for every subject s and every object o each cell $M[s, o]$ of the matrix contains the rights that the subject s has on the object o. The rights are taken from some finite set of rights, mostly including the classic rights of *read* and *write* access.

Mandatory Access Control (MAC) In mandatory access control systems, the policy is controlled by some administrator and is globally fixed. The users or processes (subjects) have neither the ability to change the policy nor to pass any rights to any to subject.

Historically, MAC systems have been introduced in the context of military database systems. But recently, modern operating systems also implement MAC, as for instance SELinux and AppArmor in Linux and Mandatory Integrity Control in Microsoft Windows Vista and its successors.

Discretionary Access Control (DAC) Unlike mandatory access control, in a discretionary access control system, the users or processes (or in general subjects) have the ability to modify the policy under some restrictions.

The most famous example is the UNIX file mode where access to files is represented by write, read, and execute bits for each of user, group, and others. Additionally, files are owned by some user and some group. Any of the owners can modify the right of each file and hence can grant rights to other users or remove a specific right from them, as long as it relates to its own file. It is also possible to changes the owner of a file, which also possibly changes the rights of a specific user with respect to the file.

6.1.1 The Bell-LaPadula Model

The most influential work on the mechanism for enforcing confidential properties is from Bell and LaPadula [BL73, BL76]. This framework, also denoted as *Bell-LaPadula model*, defines confidentiality polices for access control systems. It is also known as multilevel security (MLS). We present this access control model not only because of its historical influence and importance, we also use it as an example to introduce basic concepts, used later in this chapter. However, here we only present a very reduced and simplified version of it and refer to the original work for a complete presentation.

The Bell-LaPadula model requires a multilevel security policy. Historically, such a policy was defined in a military style. In the context of subjects, their security levels are denoted as *security clearances*, a linearly ordered set of security levels, for example depicted in Figure 6.1. In the context of objects this is denoted as *security classification*. Intuitively the security clearance of a subject must be as high as the security classification of an object in order to have access to it. That means, in general, information is allowed to go upwards, but never allowed to go downwards.

In addition to the security clearances or classifications, there are also security categories, which are partially ordered sets with a minimum and maximum, which usually form a lattice. In Figure 6.2, the diagram depicts the relation between them, as they are partially ordered by inclusion.

A security level then consists of both the security clearance or clas-

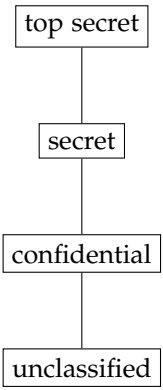

Figure 6.1. Totally ordered security clearances

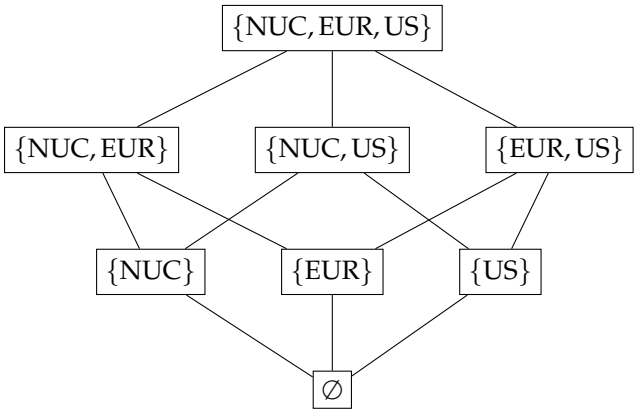

Figure 6.2. Partially ordered security categories

sification and the security category, for example a security level can be $(\text{confidential}, \{\text{NUC}, \text{EUR}\})$. This partial order of the security categories also defines a partial order on the security level. A security level $l = (C, L)$ is at least as high as $l' = (C', L')$ for security clearances C and C' and security categories L and L' if $C \geq C'$ and $L \subseteq L'$

Read and write operations in the Bell LaPadula Model are only allowed according to the security level, captured by the following two properties:

simple security property (ss-property): A subject at a given security level l is only allowed to read an object with security level l' with $l' \leq l$ (no read-up).

star property ($\star$-property): A subject at a given security level l' is only allowed to write to an object with security l' with $l' \geq l$ (no write-down).

These security levels provide only an upper bound for the subjects' access to objects. Additionally there is an access control matrix which provides which right a particular subject has to a particular object. Both the security levels and the access control matrix are state-dependent and may change during the run of the system. To capture that a particular subject has only those rights that are allowed by the access control matrix, there is a third property:

discretionary security property (ds-property): Every right that a particular subject has to a particular object is allowed by the access control matrix of the corresponding state.

A system is then said to be secure (as a Bell LaPadula model) if it satisfies the ss-property, the $\star$-property, and the ds-property.

Often in this context, a fourth property is mentioned, namely the *principle of tranquility*. This property states that the security levels are fixed and do not change during a run of a system. It can be seen as an option and pushes the system in the direction of a MAC system.

Biba Model

While the Bell-LaPadula model is used to satisfy confidentiality properties, the Biba model [Bib77] addresses integrity policies. Integrity policies ad-

dress the problem of integrity or correctness of data, rather than disclosure. The concept of integrity is dual to confidentiality. Only for completeness, we mention the properties here:

- ▷ A subject can read an object only if its integrity level is at most that of the object.
- ▷ A subject can write an object only if its integrity level is at least that of the object.

Capability Systems

One class of discretionary access control systems is the class of capability systems, which is also used in our case-study of the Flume system. Capabilities are communicable, unforgeable tokens of authority and usually refer to a right on an object. A subject can only access an object if it has the capability that allows it to do so. Capability can be used for both, protecting confidentiality and integrity. Usually, capability systems have some mechanism to exchange capabilities among the subjects. However, there are often restrictions on the communication between subjects, for example, a subject itself may need the capability to send capabilities to other subjects. The main difference to many other discretionary access control systems is that the capabilities are not owned by a specific subject. Instead, capabilities are held by subjects and may be passed to other subjects. Again, it is required that a reference monitor or the operating system enforces the restriction given by the design of the capability system in order to achieve security.

A common question in this context is "is it possible for a particular subject to gain a particular access right at a particular object?". The decidability and complexity of this question depends on the expressiveness of the underlying framework for modeling the access control system. It ranges from linear time in *take-grant* systems [LM82] to undecidable for *protection systems* in [HRU76].

However, we will not address this question. Instead, we will analyze an access control system with respect to illegal information flows. This problem is orthogonal to this question and is often not considered in the

classic literature on discretionary access control systems and capability systems.

6.2 Information Flow in MAC Systems

A definition of security domains and of a policy as a relation between them is fundamental for analyzing information flow or noninterference in a system. In general, definitions of access control systems provide neither of them. To apply our theory of noninterference, we need to interpret access control systems and to extract definitions of domains or agents and a policy from the access control mechanism.

For defining security domains, the two possibly most obvious ways are either to take each subject and each object as a security domain, or to take only the subjects. Rushby and van der Meyden favored the latter approach in [Rus92, vdM07]. The subjects are the agents or security domains and the objects are defined separately. The implicit idea of the information flow interpretation is that subjects are allowed to interact only with each other by writing to and reading from objects. This interpretation leads directly to a policy which says that an agent u is allowed to interfere with an agent v if and only if there exists an object to which u has write access and v has read access. This interpretation has been formalized as "reference monitor conditions" and van der Meyden [vdM07] has shown that for an access control system satisfying these conditions is equivalent to ta-security.

6.3 Information Flow in DAC Systems

An appropriate formalization of DAC systems as information flow systems is often more complex than in the case of MAC systems or even not possible. DAC systems usually have some mechanism for changes of access control rights during runtime, which is, in an information flow interpretation, a dynamic policy. However, in our interpretation of systems with dynamic policies, the changes of policies are performed by actions and therefore can be uniquely assigned to a particular agent which is responsible for the policy change. This is not the case in all DAC systems. In some systems the

policy changes come from some global mechanism, they can generally be performed unconditionally by any agent, or it is completely not specified who is responsible for the policy changes. To analyze information flow security, we always need any action and any changes to a policy to have some responsibility to some agent. If this technical restriction is given, then our noninterference definitions provide a tool for detecting undesired information flows which may stem from communication forbidden by some implicit policy or from policy changes which are not safe. As in the information flow analysis of MAC systems, for DAC systems we also need an interpretation of security domains and a policy. Here the situation might be more complex since DAC systems often have communication methods which lie outside the communication through the objects. These are necessary to provide a mechanism for policy changes which often involves some direct communication or interaction between the subjects. An example of DAC systems which have these desired requirements for an information flow analysis are capability systems. In these systems, capabilities are passed around by agents and policy changes are a direct consequence of actions and the exchange of capabilities. That is why we use a capability system to strengthen the usefulness of the developed theory of dynamic noninterference and use it to show the absence of security flaws in the conceptual design of this access control system.

6.3.1 Distributed Information Flow Control

In this section, we introduce the distributed information flow control system Flume following the presentation of Krohn and Tromer [KT09]. This system has been successfully implemented as an extension to the Linux kernel [KYB+07] and provides a mechanism to control the access of processes to files or system resources. The subjects in the Flume system are processes and technically the objects are also implemented as processes, but they cannot perform any actions by themselves. At this state of the description of the Flume system, we keep the terminology of processes to have a clear distinction to our translation of this system in our framework where we will call the active components agents.

A distributed information flow control system is an access control system

without a global access control mechanism. This distributed access control mechanism organizes the rights of each subject by possessing of capabilities.

First, we explain the capability-based approach of the Flume system. In Flume, capabilities are derived from *tags*. Tags are assigned to objects and are opaque tokens which are used to track data when it flows through the systems. They are taken from some set $\mathcal{T}$ and are associated with one of the categories secrecy or integrity. A tag does not have any intended meaning—one might see it as a randomly generated string, or in cryptographic terminology, as a nonce. Any object gets from its generation a new, unique tag. Any subset of $\mathcal{T}$ is denoted as a label. To any process p, two labels are assigned Sec_p and Int_p, the *secrecy* and the *integrity set* of p, respectively. Intuitively, the set Sec_p is the collection of tags whose assigned objects p is allowed to read from and Int_p is the collection of tags whose assigned objects p is allowed to write to. These labels are state dependent and may change during a run of the system.

The rights of a process to modify its own secrecy or integrity set are denoted as *capabilities* that are derived from tags. For each tag t, there are two capabilities t^+ and t^-. The capability t^+ allows the process that possesses it to add the tag t to its secrecy or integrity set and t^- allows to remove this tag from one of these sets. If a process does not possess the corresponding capability, it is not allowed to add or remove the tag respectively. To refer to these sets of capabilities, we introduce some notation. Any of the following sets relate to a single state during some run of the system, but for brevity, we omit a parameterization of these sets by a state at this point of the description of the system. The set of all possible capabilities is denoted as $\mathcal{O} = \mathcal{T} \times \{+, -\}$. The set of all capabilities owned by a process p is $O_p \subseteq \mathcal{O}$. One says a process p has *dual privilege* on a tag t if it owns both t^+ and t^-. The set of all those tags is denoted with $D_p = \{t \in \mathcal{T} \mid t^+ \in O_p \text{ and } t^- \in O_p\}$. For any set of capabilities $O \subseteq \mathcal{T}$ define $O^+ = \{t \in \mathcal{T} \mid t^+ \in O\}$ and $O^- = \{t \in \mathcal{T} \mid t^- \in O\}$.

One very special property of the Flume system is the use of global capabilities. The set of global capabilities $\hat{O}$ can contain arbitrary capabilities and every process is allowed to treat these capabilities as its own. However, it is assumed that processes cannot enumerate the whole set $\hat{O}$. We assume that a process can only get a capability from $\hat{O}$ if it knows the name of

the associated tag which may also come from another capability derived from the same tag. If a process does not know the name of a particular tag, we assume that this process cannot gain any corresponding capability even if it is in the global set of capabilities. As it is worked out in detail in [KT09], processes have the possibility to guess tag names, but since the probability is small for a sufficiently large set of possible tags, we neglect the possibility of guessing tags in this work to keep the analysis of this system more simple. The insertion of tags into $\hat{O}$ is only possible by tag allocation. Every process has the possibility to create or allocate new tags. If a process creates a new tag t, it owns the tag's new capabilities t^+ and t^- and in the same step as the creation, it also decides which of these two capabilities are inserted into the set of global capabilities. After this creation step, it is not possible to insert any capability into $\hat{O}$. Since every process is allowed to own all global capabilities, we define its effective set of capabilities as $\overline{O}_p = O_p \cup \hat{O}$. Similarly, the effective set of dual privileges is $\overline{D}_p = \{t \in \mathcal{T} \mid t^+ \in \overline{O}_p \text{ and } t^- \in \overline{O}_p\}$.

In the Flume system, processes can communicate with each other by reading or writing to objects and by sending messages directly. These messages can also contain capabilities. Every process can send any set of its capabilities to any other process with which it is allowed to communicate (restrictions on the communication are explained later).

The labels Sec_p and Int_p of each process p are fundamental for the security of the Flume system. These sets can only be changed explicitly by particular actions of the process itself. Any process can only modify its own secrecy and integrity set, but it is only allowed to do so if the process owns the capability that allows it to do so. Such label changes are called *safe label changes*. Formally, if L is one of the labels Sec_p or Int_p and if L' is the new label after the change, then this change from L to L' is safe if and only if

$$L' \setminus L \subseteq (\overline{O}_p)^+ \quad \text{and} \quad L \setminus L' \subseteq (\overline{O}_p)^- \ .$$

This condition just says that if p inserts a tag t into L, then it has to own the capability t^+ and if it removes a tag t from L, then it has to own the capability t^-. For instance, the capability t^+ is the right to access an object tagged with t, and the capability t^- is the right to downgrade information

of an object tagged with t.

The relation of the secrecy and integrity sets of the process induces a policy between them. Classically, the policy between processes p and q induced by this kind of secrecy and integrity sets is

$$p \rightarrowtail q \quad \text{iff} \quad Sec_p \subseteq Sec_q \text{ and } Int_q \subseteq Int_p \ .$$

With $p \rightarrowtail q$, we mean that p may communicate with q for example by sending messages. The condition $Sec_p \subseteq Sec_q$ says that q has read access to all the objects p has read access to. Hence p cannot have access to any information that q is not allowed to have access to and hence, it is safe that p is allowed to communicate with q. The other condition $Int_q \subseteq Int_p$ says that p may write to all objects that q has write access to. Hence q cannot modify any data that p is not allowed to have write access to and again, it is safe that p is allowed to communicate with q.

The Flume system relaxes these rules by tags that could be in the secrecy or integrity sets without changing these sets permanently. If a process p has dual privileges on a tag, i. e., $t \in \overline{D}_p$ then p can add and remove the tag t from one of its labels and transform the label into the previous state afterwards. In Flume, these hypothetical label changes are used, what slightly modifies the conditions for edges in a policy. This condition is called a *safe message exchange*. A message from p to q is safe if and only if

$$Sec_p \setminus \overline{D}_p \subseteq Sec_q \cup \overline{D}_q \quad \text{and} \quad Int_q \setminus \overline{D}_q \subseteq Int_p \cup \overline{D}_p \ .$$

6.4 Information Flow Analysis of Flume

We proceed with a formalization and a translation of the Flume system in our framework. This includes a careful translation of the processes into security domains and a translation of the safe message exchange condition of the Flume system into a dynamic policy. Furthermore, we need a description of the state space including observations of the agents and a transition function which incorporates the actions of the Flume system.

6.4.1 System Model

First, we define the security domains of a system. A property of the interpretation of policies in this thesis is that if an agent u performs an action, then every agent with which u is allowed to interfere might learn every action that u performs. A situation where u has two actions, say a_v and a_w and an agent v is allowed to see the action a_v and another agent w is allowed to receive action a_w but not vice versa, cannot be expressed directly since any policy has an edge from u to v and from u to w and hence v may also observe a_w and w may observe a_v. To overcome this problem of the modeling of the policy, one can split the agent u into two agents, (u, v) and (u, w) as depicted in Figure 6.3. The agent (u, v) then is allowed to interfere with v and (u, w) is allowed to interfere with w and the actions are assigned to each of (u, v) and (u, w), respectively. The two parts of u then are connected by a complete (directed) graph. Due to intransitivity, the action a_v might influence w, but only if the action a_w has been performed afterwards.

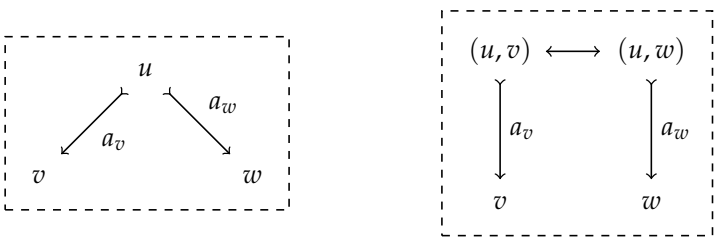

Figure 6.3. Splitting of agents

The Flume system only allows communication between single processes. To express this property in our policy, we use the exact construction as above and "split" each process into several security domains. Let P be the set of all processes. Every process is split into $|P|$ security domains, each for a possible interaction with any other process and one for internal actions. The set of all security domains is then $D = P \times P$. Every domain of the form (p, q) belongs to the process p and we denote the set $\{(p, q) \mid q \in P\}$ as the components of process p. Intuitively, a security domain (p, q) is the

output interface of process p which might—depending on the dynamic policy—send messages to process q. We denote the security domain (p, p) as the internal part of the process p. This domain is never allowed to send messages to any other process.

We assume that every process p has its own state space S_p. The need of a global state space would only come from the use of the global capability set. We will explain later how we can overcome this issue and how we can express it as a part of the local capability sets.

6.4.2 Objects

According to Krohn's thesis [Kro08], objects are modeled as processes with an empty ownership set, immutable secrecy and integrity set, fixed at creation. If a process p writes to an object o, then there is an information flow from p to o which can easily be expressed in our framework by an action of p's output interface. If a process p reads an object o, then there is an information flow from o to p. Since in our framework, information can only flow by performing actions and the information flows from the domain that performs the action, the read action of p has to be performed by o. To avoid that if p reads object o, all other processes which have read access are informed about this read action, we split the object o into $|P|$ components, each for a possible read action of every single process.

The components of each object are $\{(o, p) \mid p \in P\}$. If a process has write access to o, then the policy allows the domain (q, o) to interfere with all components of o. If a process p has read access, then the policy allows only the domain (o, p) to interfere with all components of p.

In the Flume system, the objects are modeled as processes where the read and write actions are send-message actions. Then the internal component of an object has no action and hence cannot change any policy.

From now on, with P we denote all processes including those which represent objects. We will not distinguish further between subjects and objects or different kinds of processes.

6.4.3 Tags, Capabilities, and Policies

In this thesis, we will only analyze the information flow by considering secrecy sets. Due to the duality of integrity, an analysis using integrity sets or both kinds of sets is expected to be analogue.

To refer to different states in a run, we parameterize the previously introduced tag and capability sets with the state. For example, the set $Sec_p(s)$ denotes p's secrecy set in the state s.

With a dynamic (or state-dependent) secrecy set, we define the induced policy for all $p, q, q' \in P$ and all $s \in S$ by:

$$(p,q) \rightarrowtail_s (p,q') \text{ and}$$
$$(p,q) \rightarrowtail_s (q,q') \text{ iff } Sec_p(s) \setminus \overline{D}_p(s) \subseteq Sec_q(s) \cup \overline{D}_q(s)$$

In the second case, we also write $p \rightarrowtail_s q$. The edges of the first line can be seen as static since they exist in all states. This part of the policy relates to the fact that we have split the process p into its components as described above. All security domains, which relate to the same process, i. e., those which have the same first entry in their pair, form a clique. The second line of this policy definition are the policy edges induced by Flume's safe message condition restricted to secrecy sets. This line says that the outgoing interface (p,q) of p for possibly sending messages to q is allowed to interfere with q if and only if the safe message condition is satisfied. Note that (p,q) is always allowed to interfere with every or with no component of q. From this structure of the policy, it directly follows that every component of a single process always receives the same actions. Inductively, the traces only consist of their own and the received actions of each component of a single process and are the same for each component. Hence, we can refer to traces of processes rather than to traces of agents. Therefore, it is sufficient to apply any trace-based operator to traces of a process rather to traces of one of its components.

6.4.4 Policy Locality and Global Capability Awareness

Before modeling the actions of the Flume system, we show that it is possible to substitute the set of global capabilities by local capability sets for each agent without affecting the safe message condition and hence without affecting the policy.

For that, we show that in the Flume system the policy can equivalently be defined as $\hat{O}_p \subseteq \hat{O}$ instead of $\hat{O}$ for each agents. The set $\hat{O}_p$ is the set of those capabilities from $\hat{O}$ which the process p is aware of. If the process has seen a tag t or one of the derived capabilities t^+ or t^- then we assume that the process knows the name of the tag t and hence can request any of the capability from $\hat{O}$ corresponding to t. If a process p has received a capability t^+ or t^-, then p is aware of all capabilities relating to t which are in $\hat{O}$, hence in every state we have $\{t^+, t^-\} \cap \hat{O}(s) \subseteq \hat{O}_p(s)$. Due to the general assumption of perfect recall of the agents, we suppose that the agents never forget any seen information about the capabilities and hence $\hat{O}_p$ is monotonically increasing during every run of the system.

The set of tags where an agent has dual privileges for, based on the capabilities it is aware of, is defined for every state s and every process p by

$$\tilde{D}_p(s) = \{t \in \mathcal{T} \mid t^+, t^- \in O_p(s) \cup \hat{O}_p(s)\} \ .$$

The next result shows that the safe message condition can be equivalently expressed with this set of tags instead of the original definition.

6.4.1 Lemma. *For every process p and q and every state s, we have*

$$Sec_p(s) \setminus \overline{D}_p(s) \subseteq Sec_q(s) \cup \overline{D}_q(s) \text{ iff } Sec_p(s) \setminus \tilde{D}_p(s) \subseteq Sec_q(s) \cup \tilde{D}_q(s) \ .$$

Proof. First, we proof the implication from left to right. Assume that $Sec_p(s) \setminus \tilde{D}_p(s) \not\subseteq Sec_p(s) \setminus \overline{D}_p(s)$. Then, there is some tag $t \in Sec_p(s)$ with $t \in \overline{D}_p(s)$, but $t \notin \tilde{D}_p(s)$. This is not possible, since p is aware of all capabilities in $\hat{O}(s)$ which correspond to one of its tags. Hence we have $Sec_p(s) \setminus \tilde{D}_p(s) = Sec_p(s) \setminus \overline{D}_p(s)$. Assume that still $Sec_p(s) \setminus \tilde{D}_p(s) \not\subseteq Sec_q(s) \cup \tilde{D}_q(s)$ holds. Since $\overline{D}_p(s) \setminus \tilde{D}_p(s)$ are tags t with t^+ and t^- in $\hat{O}(s)$, the set $Sec_p(s) \setminus \tilde{D}_p(s)$ must contain tags with t^+ and t^- in $\hat{O}(s)$ but not

in $\hat{O}_p(s)$. Again this is not possible, since p is aware of the capabilities corresponding to tags of its secrecy set.

The other direction of the proof simply follows from

$$Sec_p(s) \setminus \overline{D}_p(s) \subseteq Sec_p(s) \setminus \tilde{D}_p(s) \subseteq Sec_q(s) \cup \tilde{D}_q(s) \subseteq Sec_q(s) \cup \overline{D}_q(s) \ .$$

□

The benefit of this result for our interpretation of the Flume system is that we do not have to reason about a global state space which could possibly be the reason for a covert channel. Instead, we have only local state spaces for each process, which prevent these kinds of information channels.

6.4.5 Actions and Transitions

We will translate the actions of the Flume system into our framework and will explain the behavior of the corresponding transitions. To indicate the ownership of each action, we prefix every action with the name of the process. We provide a brief description for each action and analyze their effects to the policy. We implicitly assume that the structure of the state space represents the traces and hence no state is visited twice. The following actions are possible in our interpretation of the Flume system:

p.create_tag_c With this action, process p creates a new tag. The value c can be either None, Remove, or Add and indicates whether none, the capability t^-, or the capability t^+ is granted to the set of global capabilities, where t is the created tag. This action is an internal action of the process p, hence $\mathrm{dom}(\mathsf{p.create\_tag\_c}) = (p, p)$. Let

$$s \cdot \mathsf{p.add\_tag\_c} = s'.$$

A new tag t has been created and p owns the corresponding capabilities: $O_p(s') = O_p(s) \cup \{t^+, t^-\}$. Depending on the value of c, the set $\hat{O}$ changes and also the set $\hat{O}_p$. However, since no other agent is aware of any of these privileges, the sets $\hat{O}_q$ do not change for agents $q \neq p$. In the case of $\mathsf{c} = \mathsf{None}$, we have that $\hat{O}$ does not change, i. e., $\hat{O}(s') = \hat{O}(s)$.

In the case of $\mathsf{c} = \mathsf{Remove}$, we have $\hat{O}(s') = \hat{O}(s) \cup \{t^-\}$ and in the case of $\mathsf{c} = \mathsf{Add}$, we have $\hat{O}(s') = \hat{O}(s) \cup \{t^+\}$. Only the local state space of p is affected by this action. No local policy is changed by this action.

p.add_tag_t With this action, process p adds the tag t to its secrecy set if it owns the corresponding capability t^+. Again, this is an internal action of the process p: $\mathrm{dom}(\mathsf{p.add\_tag\_t}) = (p, p)$. For a transition of the form

$$s \cdot \mathsf{p.add\_tag\_t} = s' \ ,$$

the state s' might contain some information about the success of the attempt to add the tag t to the secrecy set. We have $Sec_p(s') = Sec_p(s) \cup \{t\}$ if $t^+ \in \overline{O}_p(s)$ and $Sec_p(s') = Sec_p(s)$ otherwise. Only the local state space of p is affected by this action. This action might remove outgoing edges from some domain (p, q) and might insert incoming edges to all (p, q).

p.remove_tag_t This action is dual to the p.create_tag_c action and removes the tag t from p's secrecy set if p owns the capability t^-. Again, this is an internal action with $\mathrm{dom}(\mathsf{p.remove\_tag\_t}) = (p, p)$. For a transition

$$s \cdot \mathsf{p.remove\_tag\_t} = s' \ ,$$

the state s' might include some information about the success of removing the tag t. We have $Sec_p(s') = Sec_p(s) \setminus \{t\}$ if $t^- \in \overline{O}_p(s)$ and $Sec_p(s') = Sec_p(s)$ otherwise. Only the local state space of p is affected by this action. This action might remove outgoing edges from (p, q) for some q and might insert incoming edges to all (p, q).

p.send_message_m_to_q With this action, process p sends a message with content m which is addressed to q. This action can only be performed by the corresponding interface for outgoing messages of p to q, i. e., $\mathrm{dom}(\mathsf{p.send\_message\_m\_to\_q}) = (p, q)$. The delivery of this message to q depends on weather the message is safe. In our formulation, q only receives this message if $p \rightarrowtail_s q$ holds, where s is the state in which the message was sent. Hence, a transition on q's local state space is only performed if $p \rightarrowtail_s q$, assuming this was performed in the global state s.

This action affects p's local state space and possibly q's local state space, but only if the policy allows interference. This action has no effect to the policies under the assumption that the message does not contain information about tag names. Otherwise it may change $\hat{O}_q$. However, this would not affect security, since p is allowed to interfere with q.

p.send_cap_c_to_q With this action, p can send q one of its capabilities c if it is a safe message. Similar to the p.send_message_m_to_q action, we have $\mathrm{dom}(\mathsf{p.send\_cap\_c\_to\_q}) = (p, q)$. This action has no effect on q if it is performed in a state s with $p \not\rightarrowtail_s q$ or if $c \notin \overline{O}_p(s)$. If this action is a safe message, then we have for the corresponding transition

$$s \cdot \mathsf{p.send\_cap\_c\_to\_q} = s'$$

that it holds $O_q(s') = O_q(s) \cup \{c\}$.

This action affects p's local state space and possibly q's local state space, depending if it is a safe message. This action might remove outgoing edges from some (q, q') and might insert incoming edges to all (q, q').

p.drop_cap_c With this action, agent p can remove the capability c from its set $O_p(s)$. This is again an internal action of p: $\mathrm{dom}(\mathsf{p.drop\_cap\_c}) = (p, p)$. For a transition of the form

$$s \cdot \mathsf{p.drop\_cap\_c} = s' \ ,$$

we have $O_p(s') = O_p(s) \setminus \{c\}$. Only the local state space of p is affected by this action. This action might add outgoing edges from some (p, q) and might insert incoming edges to all (p, q).

Compared to the presentation in [KT09], there are several slight differences. Most notably is that we do not consider the possibility of forking processes. To formulate the creation of processes, we need to have a non fixed set of security domains which is not part of our framework so far. Another change is that we split the sending of messages and the sending of capabilities into different actions. In the original paper, this can be done by a single action. However, we do not believe that this would affect the security of the system.

6.4.6 Security of the Flume System

In this section, we will analyze the security of the Flume system. Due to the dynamically changing policies of the Flume system, it is clear that we need a security definition which supports dynamic policies. It is also reasonable that we have to take intransitive effects into account, since information may flow through several domains. These assumptions restrict us to only two of our security definitions: di-security and dta-security. Next, we will explain why di-security is not an appropriate security for this setting and then we will analyze the security of Flume with respect to dta-security.

The next example shows that the Flume system is not di-secure.

6.4.2 Example. The system in Figure 6.4 depicts a part of a state space that may exist in an instance of the Flume system. The systems consists of processes u, v, w, and x, which are split into their corresponding parts—except x since it does not perform any action and one may see it as its internal part. In this setting, we are only interested in the observations of agent x and all observations of other agents are ignored. We suppose that we have tags, just denoted as u, v, w, and x and each agent has the tag which corresponds to its own name in its secrecy set. Hence each agent needs the corresponding tag of another agent in order to interfere with it. Initially, we suppose that u is allowed to interfere with w, w is allowed to interfere with v, and v is allowed to interfere with x. To show that this is a possible state of the Flume system, the tag sets of the agents are listed in Figure 6.5. The set of global capabilities $\hat{O}$ is assumed to be empty and hence the sets $\overline{D}_y$ and $\tilde{D}_y$ are the same for all agents $y \in D$.

Initially, it is supposed that the agent w has the tag u and the capability u^+, but v and x do not. Due to the safe message condition, agent v is allowed to send a message to x in the initial state. Throughout this example, the observation of x switches from 0 to 1 if it receives a message. Another possible action in the initial state is an action w, which sends the capability u^+ to v. After that, in state s_1, agent v can still send a message to x safely. However, if v adds the tag u to its secrecy set, the safe message condition w.r.t. x is no longer valid and the edge from v to x disappears, and in state s_2 all messages sent are blocked by the system. In that case, x does not even notice that a message has been sent.

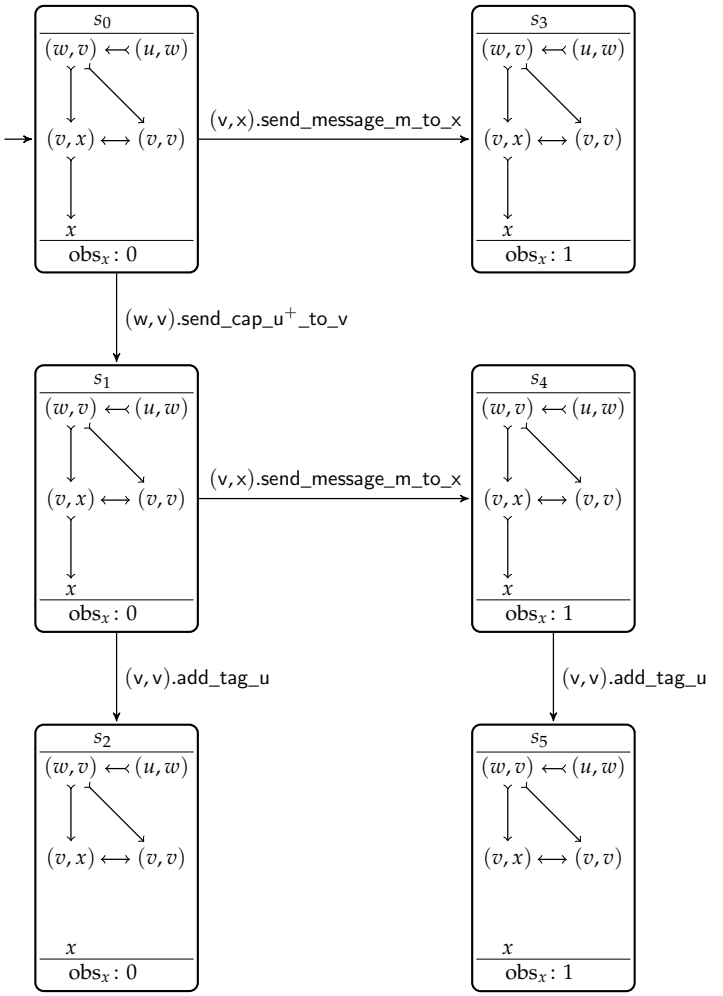

Figure 6.4. A part of a Flume state space

state	Sec_w	O_w	Sec_u	O_u	Sec_v	O_v	Sec_x	O_x
s_0	w, u	u^+, u^-	u	$-$	v, w	w^+, w^-	x, v	$-$
s_1	w, u	u^+, u^-	u	$-$	v, w	w^+, w^-, u^+	x, v	$-$
s_2	w, u	u^+, u^-	u	$-$	v, w, u	w^+, w^-, u^+	x, v	$-$
s_3	w, u	u^+, u^-	u	$-$	v, w	w^+, w^-	x, v	$-$
s_4	w, u	u^+, u^-	u	$-$	v, w	w^+, w^-, u^+	x, v	$-$
s_5	w, u	u^+, u^-	u	$-$	v, w, u	w^+, w^-, u^+	x, v	$-$

Figure 6.5. The tags and capabilities of the agents

The states s_0 and s_3 are only in this example to make this scenario plausible and to show that we have not constructed a setting which cannot be instantiated. For analyzing security, we start in state s_1 and consider the following trace, consisting of two actions,

$$(\mathsf{v},\mathsf{v}).\mathsf{add\_tag\_u}\ (\mathsf{v},\mathsf{x}).\mathsf{send\_message\_m\_to\_x}\ ,$$

and the trace

$$(\mathsf{v},\mathsf{x}).\mathsf{send\_message\_m\_to\_x}$$

consisting of only a single action.

First, we apply di-security to this system and use the characterization of Theorem 4.5.4. Both traces above have the same dipurge value:

$$\begin{aligned}\mathrm{dipurge}((\mathsf{v},\mathsf{v}).\mathsf{add\_tag\_u}\ (\mathsf{v},\mathsf{x}).\mathsf{send\_message\_m\_to\_x}, s_1, x)\\ = \mathrm{dipurge}((\mathsf{v},\mathsf{x}).\mathsf{send\_message\_m\_to\_x}, s_1, x)\\ = (\mathsf{v},\mathsf{x}).\mathsf{send\_message\_m\_to\_x}\ .\end{aligned}$$

Since both traces lead to different observations for x, the system is insecure with respect to di-security.

Example 6.4.2 shows that the Flume system is insecure with respect to di-security. There are two possible conclusions from this result. Either the Flume system (or our modeling of it) is indeed insecure, or the security definition of di-security is too restrictive. We will argue for the latter case. The main reason is that it is hard to explain why the system of Example 6.4.2 should be intuitively insecure. It seems to be reasonable that if x receives a message from v, then x also gains the information that v has not performed the (v,v).add_tag_u action, since otherwise, x has not received any message from v. Therefore, x obtains information about things that have *not* happened. Transmission of this kind of negative information is not supported by the notion of di-security. This is the reason why we believe that this is a shortcoming of di-security rather than a flaw in the Flume system.

As we already have seen, this shortcoming has been corrected by the notion of dta-security. We will show that the Flume system (or more precisely, our interpretation of the Flume system) is dta-secure by showing that it satisfies the pattern of an intransitively securely constructed system.

In a securely constructed system, it is supposed that every agent has its own state space. However, if we let every component of a process have the same copy of a local state space and the local transition functions are exactly the same on each copy, then we have the same effect, we have with a single local state space for each process, since from the construction of the policy, a component of a process p is either interfering with all or with none of some other process's components. Hence, from now on, it is sufficient to talk about the local state space of a process rather than of an agent.

As usual, we require that the observation function of each agent/process depends only on its local state space. We have already seen, for the security of the system, it is not necessary to specify it more precisely.

We have also seen that the actions of the Flume system only affect the local state spaces of the agents, and that the tag and capability sets only depend on the local state space of the corresponding agents. Hence, the dynamic policy is intransitively local-state dependent. The global transition function follows the construction pattern of the securely constructed system, since local transitions are only performed if the local policy in that state allows it. Hence by Corollary 4.6.13, the Flume system is dta-secure.

6.4.7 Krohn and Tromer's Noninterference Result

In their work [KT09], Krohn and Tromer provided a noninterference result for the Flume system. However, their analysis does not take the dynamics of the policy into account. They only examine *export protection tags*. An export protection tag t is a tag with the property that $t^+ \in \hat{O}$, but $t^- \notin \hat{O}$. They show that a process p with such an export protection tag in its secrecy set cannot interfere with any process q which does not have this tag under the assumption that p does not own the capability t^- and q does not own any of the capabilities t^+, t^- initially. Such a result only shows that data from a file, protected with an export protection tag, cannot be leaked to an untrustworthy process. However, this is a very limited result because protection of data by such tags should not be the only security requirement of such a complex and powerful access control mechanism like the Flume system provides. Moreover, their result completely ignores possible information flows which could stem from policy changes.

6.4.8 Summary

We analyzed the security of the Flume system by applying our dynamic noninterference security definition to it and improved the previous security analysis of the authors. We have seen that our definition of di-security is too strong, at least for this setting of a distributed access control system. However, the notion of dta-security seems to be appropriate and we believe that this is the right definition for analyzing information flows in access control systems with dynamic security requirement. We did this analysis of the Flume system to show that the developed theory of dynamic noninterference has its applications and to provide evidence that this theory has the right definitions.

Chapter 7

Other Frameworks for Noninterference

The term *noninterference* is loosely used for expressing the absence of causal dependencies and information flows. Several syntactically and semantically different frameworks have been developed to analyze noninterference-like properties.

In this chapter, we only recall the most influential works and categorize them in the following sections. Due to the intended simplicity of the basic idea of noninterference, essentially every framework in that systems can be expressed can be used to formulate noninterference properties. However, due to the very restrictive nature of plain vanilla noninterference, many relaxations and variations have been developed. The different expressiveness of diverse frameworks leads to many noninterference properties while their relation between each other is not obvious. Many of these properties use particular properties of the framework and cannot be directly adapted to other frameworks.

For several frameworks such as process algebras and language-based security, good surveys exist and we will cover these topics very briefly. In contrast, we examine other frameworks, in particular computational noninterference, in more detail, even though it does not reflect the amount of work on this field.

7.1 Deterministic Systems

The most classical works on noninterference are based on deterministic systems. Starting with Goguen and Meseguer's work [GM82], systems consist on state spaces with deterministic transitions which are input enabled, i. e., every action can be performed in every state. Also all follow up works, namely [HY87, Rus92, vdM07], used in this thesis as a basis, require deterministic state-based systems. However, van der Meyden [vdM07] changed the used system from action-observed to state-observed. In an action-observed system, observation results from performance of a particular action, while in a state-observed system, the observation function is defined on the set of states. However, this is only a minor change to the system model as these system models can be translated into each other [vdMZ10]. Besides the extensions of noninterference in the directions of intransitive and dynamic noninterference, Zhang gives a very general approach of conditional noninterference [Zha11].

In [TW08], the authors use a weaker interpretation of noninterference, called incident insensitive noninterference. In their semantics of a policy $H \not\rightarrow L$, the agent L is allowed to observe *if* H has performed an action but not *which* of its actions. We will see in Section 7.6 that this interpretation of policies is closer to the semantics of information flows in language-based security.

7.2 Nondeterministic Systems

A natural generalization is to extend the setting to nondeterministic systems. Nondeterministic systems usually arise from an underspecification or from some probabilistic behavior which is needed in many systems with security critical components. Another source of nondeterminism is the scheduling of concurrent processes if the scheduler is not modeled explicitly. Nondeterminism is defined as that for a particular state s and a particular action a, the transition relation leads to possibly more than one state if one performs the action a in the state s.

Here, we survey the most influential works for state-based and trace-

based frameworks—except those works based mainly on process algebras, which are reviewed in the next section. For nondeterministic transition systems, several semantic models have been developed, which makes a comparison of the different notions of noninterference difficult. However, van der Meyden and Zhang provide a comparison of several of them in [vdMZ10].

In a nondeterministic setting, a simple purge-based definition, like the one of Goguen and Meseguer [GM82] is not sufficient. Due to the nondeterministic behavior, an action sequence describes several runs which can lead to different observations. The general idea is that if an agent H is not allowed to interfere with an agent L, then H's behavior should not influence L's possible observations. Hence, the nondeterministic behavior of the system can be essential for its security since it brings uncertainty into the system which disguises H's behavior.

The first notion of noninterference in this context is Sutherland's *nondeducibility* [Sut86] using a very general approach with a possible world semantics. Wittbold and Johnson [WJ90] extended this definition to synchronous systems and analyzed strategic properties called *nondeducibility on strategies*. McCullough [McC88] introduced the notion of *generalized noninterference*, but at the same time, they pointed out that this notion is not composable. Their intuition of noninterference in nondeterministic systems was:

> The input of a high–level signal may not alter the possible future sequences of low–level events.

To achieve composability, they introduced *restrictiveness* [McC88, McC90], which is an unwinding-like definition for nondeterministic systems. A weaker notion, which preserves composability, is Johnson and Thayer's *forward correctability* [JT88]. McLean introduced *separability* [McL94], which requires that for every trace of high events and every trace of low events, every possible interleaving of them is a valid trace of the system. The notion of *non-inference*, introduced by O'Halloran [O'H90], requires that from L's point of view it is always possible that H has not interacted with the system. Zakinthinos and Lee present a framework based on *Low Level Equivalent Sets* which are sets of traces that have the same view for agent L.

Mantel [Man00, Man03] proposes a framework of basic security predicates (BSP), which are basic building blocks for other security properties. They claim that all (transitive) noninterference properties in nondeterministic state-based systems can be expressed as a conjunction of some of the BSPs. Mantel also provides an extension to intransitive policies in [Man01].

In a more recent work, Engelhardt et al. [EvdMZ12] adapted the definition of ta-security to nondeterministic systems and showed that it is necessary to value coalitions. In [MZ08], van der Meyden and Zhang worked out the role of a scheduler with respect to information flow. A desired property for specifying systems is *refinement*. A system M refines a system M' if M contains less behavior then M', or it is more concrete than M'. The system M can also be seen as an implementation of M'. As pointed out in [Jac88] and [McL94], in general, noninterference is not preserved under refinement. That means that a system might be secure, but a refinement of it is not. In the literature, this property is referred to as *refinement paradox*. The deeper reason for this is that noninterference in nondeterministic systems requires that, from its observations, L considers every behavior of H possible. Hence, L never knows if different observations for the same sequence of its actions stem from the interaction of H or the nondeterminism of the system. But if in a refined system some nondeterministic choices have been removed from the system, then L may deduce that some observations are influenced by H.

7.3 Process Algebras

Process algebras (or process calculi) are popular frameworks for specifying noninterference properties. They are designed to reason about distributed systems interacting via the exchange of messages or by shared actions. However, the underlying semantic model is a labeled transition system and hence, there is a strong connection to nondeterministic state-based systems.

The two most common process algebras are the calculus of *Communicating Sequential Processes (CSP)* invented by Hoare [Hoa78] and the *Calculus of communicating systems (CCS)* developed by Milner [Mil89].

An introductory survey to noninterference in CSP is [Rya01] and to

noninterference in CCS is [FG01]. We will briefly explain the basic adaption of the Goguen and Meseguer's definition of noninterference to CSP following the presentation in [RS01]. The main differences between Goguen and Meseguer's setting and process algebra are that the latter is nondeterministic, not input-total, and that there are no observations of agents. Instead of observations, depending on the internal state, events of the systems might be accepted or rejected. Usually, the events or external actions are partitioned into the security domain. For brevity, we consider the simple high/low setting, where high (H) is not allowed to interfere with low (L). Intuitively, H must not influence L possible traces nor the set of possible actions that L can perform after some trace. Formally, a process P satisfies noninterference if for all $tr, tr' \in \text{traces}(P)$ it holds:

$$\text{If } tr\restriction_L = tr'\restriction_L, \text{ then } \mathcal{SF}(P/tr)\restriction_L = \mathcal{SF}(P/tr')\restriction_L \ .$$

The set traces(P) is the set of all possible traces of the process P. The term $tr\restriction_L$ is the restriction of the trace tr to the events of L. Hence, it is essentially the same as applying the classical purge function w.r.t. L to the trace tr. The process P/tr denotes the process P after the trace tr. The stable failure set $\mathcal{SF}(Q)$ of a process Q is the set of all pairs (t, X) where t is a trace of the process Q and X is a subset of all events which are refused by Q after the trace t. An event is refused by a process Q after a trace t if it is not a possible event of Q after t. The restriction of L should be understood as the restriction of the trace t to L and the intersection of X with L, i. e., the term $(t, X)\restriction_L$ is $(t\restriction_X, X \cap L)$.

In [Ros95], Roscoe argued for a security definition which required that the system is deterministic from L's view and showed that security is preserved under refinement.

7.4 Computational Noninterference

In the last decade, formal frameworks based on interacting Turing machines or equivalent models have become popular in the analysis of cryptographic protocols. These frameworks take the limitations of the attacker's computational power into account. They are used to reason on cryptographic

protocols and primitives whose security relies on assumptions about the computational resources of the parties and the attacker that are involved in a concrete setting. The "striking feature" [CCH+11] of these frameworks is their ability to handle and preserve security under composition. For example, in a protocol where some encryption is used one would assume that the attacker can only make polynomial-time computations with respect to some security parameter, which can be roughly seen as the key length. Such assumption would rule out the possibility of an exhaustive key search for decrypting some message. Nonetheless, an overall theory of complexity has not been settled for these frameworks.

The first work, which defines noninterference in a computational setting, is from Backes and Pfitzmann [BP04] using a framework from Pfitzmann and Waidner [PW01]. Instead of Turing machines, they use scheduled machines, which communicate through ports that are connected by input and output buffers.

The common idea of all definitions in the context of computational noninterference can be described as follows: Given a system, its security domains are the communication interfaces (tapes) or the users that interact with the system. The assumption is that the users can only communicate with the system and possibly with some environment, but not with each other directly. Then, a user H noninterferes with some user L if it is not possible for H to communicate any information that it got from the environment to L.

In general, it is assumed that the users are machines, too. Formally, there is some machine E, called the environment, which chooses a bit b randomly and provides this bit as an input to H. Then the goal of H and L (in the sense that H tries to interfere with L) is to transfer some information about this bit from H to L. After a computational bounded run of all machines, the user L is required to output its guess of the bit b to the environment. Then, in this context, H noninterferes with L if the probability of L's output only differs in a negligible amount from the probability of H's input. Otherwise, there is some interference from H to L.

For the overall security of a system, this property has to hold for all possible pairs H and L of agents with $H \not\rightarrow L$. Note that this notion corresponds to transitive noninterference, since paths of security domain

are not considered.

The work of Halevi et al. [HKN05] and the work of Canetti and Vald [CV12] follow similar idea. Both use, with some modifications, the framework of Universally Composable Security (UC) introduced by Canetti [Can00]. Instead of noninterference, they denote their security property as *confinement*, referring to Lampson's work [Lam73].

To the best of our knowledge, the only work in the context of computational noninterference, which deals with intransitive polices, is the work of Backes and Pfitzmann [BP03]. In this work, two different notions of intransitive noninterference are provided. The first definition is called *blocking non-interference*. The idea is to take out a set of users forming a cut between two agents H and L and to demand that there is no information flow from H to L in a similar sense to the definition of the transitive case as described above.

This is quite similar to our characterization of i-security by policy cuts. However, in Backes and Pfitzmann's work, it is required to consider every possible cut for any pair of agents, instead of only taking the successors of H.

The idea of their second definition is that if L can guess H's input bit with some non-negligible probability, then there is some cut of users between H and L, which can also guess this bit. This is somehow related to the notion of to-security, since only observable information can be transmitted.

The theorems which were shown in all of these works are that if a system satisfies the corresponding noninterference definition, then any system, which is indistinguishable to the environment, satisfies noninterference, too. However, this is, despite some technical results or examples, the only result which has been shown in this area of computational noninterference.

This is surprising, because these frameworks are developed to have built-in compositional properties and in none of these works the composition of systems and policies is considered. It even seems to be straightforward that the composition of two systems, each satisfying a policy, satisfies the composed policy (with some possible renaming or identification of different security domains), or at least the transitive closure of the composed policy.

The main technique in the context of computational frameworks is simulation. The idea is that two systems are equivalent if they are indis-

tinguishable to the environment. Usually one of these systems, called the ideal functionality, provides some desired functionality in a secure, but possibly not realizable way. The other system is some real system, which is designed to have this functionality. Then this real system is secure if it is indistinguishable to the ideal system.

This approach of simulation-based security has not been adapted to noninterference, in the sense of defining an ideal functionality for an arbitrary system and a given policy. Intuitively, the ideal functionality should enforce noninterference by splitting the system into different parts and allows communication between these parts only according to the policy. A common property of all these frameworks is that the communication between machines can only be done by predefined communication channels, which rules out any covert channels since these machines have no common resources.

However, both syntactically and semantically, these frameworks significantly differ from ours. The main differences are that in these frameworks only the input/output behavior of the systems is considered, rather than the sequences of all actions. Additionally, these frameworks are probabilistic or at least nondeterministic.

7.5 Probabilistic and Quantitative Information Flow

In many real systems, the existence of covert channels cannot be completely avoided. Hence it might be sufficient that very little information is leaked through a covert channel or that the information is only leaked with low probability.

McLean [McL90] extended Sutherland's nondeducibility to a probabilistic setting. Gray [Gra92] generalized this work and compared it with an information-theoretic approach. In [Gra90], Gray adopted McCullough's restrictiveness to probabilistic systems.

Instead of defining information flow or noninterference by a change of the distribution or a conditional probability, a classic way is to quantify the channel capacity, mostly, in terms of Shannon's information theory. Quantifying information flows provides the possibility to limit the transmission of

information without forbidding it at all. It is concerned with the amount of information which may flow from one part of a system to another, rather than simply whether it flows. We refer to Mu [Mu08] for a recent survey on this topic. One of the earliest works using this approach is Millen [Mil87], where the input and output channels of a system are modeled as random variables. According to their work, information flows from X to Y if the mutual information between the input of X and the output of Y is not zero. Other measures besides Shannon's entropy and mutual information are proposed in [Smi09] for quantifying the information flow in the execution of probabilistic programs.

7.6 Language-based Information Flow Security

Language-based information flow security is about formulating information flow or noninterference properties in programming languages. Despite the common idea of noninterference, language-based information flow security has very little in common with trace-based noninterference mainly considered in this thesis.

The basic idea of language-based information flow security is to attach security domains to variables and possibly to the input and output channels of a program. This is done on a programming language level. Such a programming language is denoted as *security-typed language*. A policy, mostly forming a lattice, gives restriction from which variable information may flow to which other variable. This induces an access control structure and the aim is to preserve confidentiality and secrecy of particular values. Usually this is achieved by an enforcement mechanism which over-approximates security or noninterference in order to avoid a precise security analysis, which is in general an undecidable problem. The security is usually enforced at compile-time by type-checking and therefore has nearly no run-time overhead.

The challenging part is to avoid information leaks coming from different program paths taken at run-time. Hence, branches and loops have to be carefully considered.

A common distinction is between explicit and implicit flows. An explicit

flow is understood as the assignment of a variable marked as high to a value of a variable marked as low, as in Figure 7.1. We will always assume that `h` is a high and `l` is a low variable and information flows from high to low is forbidden.

```
h := l
```

Figure 7.1. An explicit information flow

```
if (h) then
  l := 0
else
  l := 1
```

Figure 7.2. An implicit information flow

Implicit information flows are not as obvious. In the example in Figure 7.2, the implicit information flow stems from a branch on a high variable, which possibly influences the value of a low variable. Other implicit information flows typically may arise from loops in a similar way.

The first work that addresses the problem of information flow is from Denning and Denning [DD77]. From that on, a vast body of work has been done on the topic of language-based information flow security.

Sabelfeld and Myers [SM03] provide a survey consisting of more than a hundred works. A more introductory work on language-based information flow security is [Smi07]. A more recent work from Sabelfeld and Sands [SS09] has a stronger focus on downgrading and information release.

We only recall the most important directions of research in this area and give a brief introduction to the basic concepts. We summarize the basics of information flow security following the presentation in [SM03].

Information flow security in language-based systems is mostly defined in two parts. The first part is a security-type system, which is a collection of typing rules with the aim to enforce noninterference. The second part is a semantic-based definition which defines what noninterference is, mostly defined as a relation between the input and output values of a program or the corresponding memories assigned to the security domains. Then the soundness of a security-type system holds if any program following the typing rules is noninterference secure according to the semantic-based

definition. This is the typical result usually shown in works on language-based noninterference.

First we will have a closer look at an example of a security-typed system. We recall the example of a simple security-typed imperative while-language from [SM03], which is equivalent to the type-system given by Volpano et al. [VIS96].

$$\vdash exp : high \qquad \frac{h \notin Vars(exp)}{\vdash exp : low} \tag{E1-2}$$

$$[pc] \vdash \mathsf{skip} \qquad [pc] \vdash h := exp \qquad \frac{\vdash exp : low}{[low] \vdash l := exp} \tag{C1-3}$$

$$\frac{[pc] \vdash C_1 \qquad [pc] \vdash C_2}{[pc] \vdash C_1; C_2} \qquad \frac{\vdash exp : pc \qquad [pc] \vdash C}{[pc] \vdash \mathsf{while}\ exp\ \mathsf{do}\ C} \tag{C4-5}$$

$$\frac{\vdash exp : pc \qquad [pc] \vdash C_1 \qquad [pc] \vdash C_2}{[pc] \vdash \mathsf{if}\ exp\ \mathsf{then}\ C_1 \mathsf{else}\ C_2} \qquad \frac{[high] \vdash C}{[low] \vdash C} \tag{C6-7}$$

Figure 7.3. A simple security-type system

To track implicit information flow correctly, a program-counter label (pc) has been introduced. This label collects the label of the current context. In the example of Figure 7.2, the label of pc is high after entering the if-branch. An assignment is then considered secure, only if it is at least as restrictive as pc. In this example, this is not the case since it is an assignment of a low variable. Hence, this code snippet has to be rejected as insecure. Note that this also would be treated as insecure if in both cases of the branch the value of the low variable is set to the same value. This over-approximation is needed for an efficient static analysis. The main advantage of a static analysis is that it guarantees security for all possible runs. Alternatively, a run-time analysis would only provide security for a single execution path.

The typing rules are summarized in Figure 7.3. For simplicity, only two security domains, namely *low* and *high*, are used. According to each security domain, there are variables h and l containing the initial values of *high* and *low*, respectively.

The statement $\vdash exp : \tau$ means that the expression *exp* has the type τ as security type. A statement $[\tau] : C$ means that the program C can be typed in the security context τ. The types of the variables containing the initial values are l : *low* and h : *high*. Here, *pc* denotes the security context of the program label which can be either *high* or *low* and the latter is also its initial value.

In this simple setting, the security context and the security types can only be *low* and *high*. The rules (E1-2) state that every arbitrary expression can be typed as *high* and every expression that does not contain any variable of type *high* can be typed as *low*.

The typing rule (C1) says that the skip command can by typed in any context. Also any assignment of the h variable is allowed in any context (rule (C2)). The rule (C3) states that the assignment to the low variable l is only allowed if the assigned expression has the security type *low*. The rules (C4-6) are compositional rules. For the loop and the branch rule, the the security type of the condition (*exp*) has to match the types of the commands executed within the loop or at each branch to avoid information leakage from the condition. The last rule (C7) is a subsumption rule that allows every command in high context to be also typed in low context. This is needed to reset the program counter to *low* after it was *high*.

For verifying that a type system, as presented here, enforces noninterference, a suitable definition of noninterference is needed. The intuition of such a definition is that different high inputs have no effect on the outputs of low. For any two input states s_1 and s_2, which have the same low input, the output states s_1' and s_2' are indistinguishable for low after runs of the program starting in s_1 and s_2, respectively. The used indistinguishability relation depends on a used security definition. The simplest one is just to require the same outputs on all of the low variables.

The first work which provides a complete secure type system is from Volpano et al. [VIS96]. A lot of work has been done to adapt these ideas to the syntactical richness of modern programming languages. To mention only a few, these include functions, procedures, objects, exceptions, threads, and network communication.

Secure type-systems have been successfully implemented as extensions of existing programming languages. The JFlow language [Mye99] is an ex-

tension of the Java language for tracking information flow and its successor Jif has been implemented in the Jif compiler. The functional language ML has been extended to Flow Caml in [PS03].

The use of secure type-systems, as described so far, can only rule out covert channels stemming from the control path taken in a run and the leakage of information from a high to a low context. However, covert channels, lying outside of this consideration, may still exist. In general, everything a low observer can observe about high's inputs may result in a covert channel. A possible information channel is termination. If a run of a program terminates depending on high's input value, then low gains information about this value. In the example in Figure 7.4, the program does not terminate if the high variable is equal to 1 and terminates immediately in all other cases.

```
l = 0
while h == 1 do
  skip
l = 1
```

Figure 7.4. A program with a termination channel

However, this requires the assumption that termination or non-termination of a program is observable. This assumption is quite strong, since the problem whether a program terminates for a fixed input is undecidable. This assumption implies that low can "decide" an undecidable problem by its observation. In [AHSS08], it has been argued that termination channels can leak more than one bit if the program has side-effects.

However, in practice, it is reasonable that low can distinguish runs taking a long time from those taking a short time. In that case, an observer may also approximately distinguish terminating from non-terminating runs by observing whether the run terminates within a sufficiently large interval of time.

This leads to the class of timing channels. A timing channel is an information channel where low can deduce some information about inputs from the running time of a program. Hence, low can distinguish

very time-consuming computations from those that terminate immediately. Agat [Aga00] provides mechanisms for closing timing channels.

To address intransitive policies, in language-based noninterference, the technique of downgrading or declassification has been introduced. Sabelfeld and Sands [SS09] discuss different dimensions of declassification. Their four dimensions are about *what* information is released, *who* releases the information, *where* in a system the information is released, and *when* information is released.

A common technique for information release in security type-systems is an explicit declassification statement introduced into the language, which changes the security labels. Zdancewic and Myers [ZM01, Zda03] introduced robust declassification, a notion of downgrading which focuses on the integrity of a downgrading decision. In [ML97], Myers and Liskov introduced a decentralized label approach, in which each variable has a security label. Dynamic changes of the label has been considered by Zheng and Myers in [ZM04]. The label of a variable can change during a run of the program which indirectly changes the induced policy. This approach has been incorporated in an extension of the JIF language. In [BS06], Broberg and Sands introduced Flow Locks to handle dynamic policies. These locks are conditions whether an information flow from a particular high to a low variable is allowed.

Askarov and Sabelfeld provide a knowledge-based definition of information flow and information release with support of some cryptographic primitives in [AS07]. In [AC12], Askarov and Chong define information flows in terms of learning of an adversary for supporting dynamic policies.

Chapter 8

Conclusion

In this thesis, we have discussed several noninterference definitions for state-based systems. We started with the very simple and restrictive notion of Goguen and Meseguer's t-security and studied generalizations and weakenings of it in two orthogonal directions: intransitive and dynamic noninterference. Intransitive noninterference allows one to describe information flows through several agents. It takes the transmission of information via several security domains into account and allows one to specify more complex security requirements. Dynamic noninterference allows one to specify state-dependent policies and allows dynamic changes in the policy. Both generalizations extend noninterference with conditions on the information flow. We have seen in this thesis that the situation is more involved if these two extensions are combined. This results in different reasonable security definitions for dynamic intransitive noninterference.

For a better understanding of the theory of noninterference, we developed new techniques. With information sets, we got an elegant and simple way of characterizing intransitive information flows. The decomposition of systems into securely constructed systems provided new insights in the intended structure of secure systems. A main focus of this thesis was the characterization of security definitions in terms of a sound and complete unwinding. We showed that it is worth to reason about both trace-based and state-based unwinding. A trace-based unwinding is a powerful tool that provides a very simple and direct definition of intransitive noninterference in both the static and the dynamic setting. Characterizations in terms of a state-based unwinding are mostly more complex, since the state space may not contain enough information about traces, as needed for completeness. We solved this problem by increasing the number of necessary relations.

In particular, state-based unwinding relations are needed for verification algorithms.

We applied several of our techniques to the classic notion of transitive noninterference. Afterwards, we performed a deeper analysis of intransitive noninterference, including the notions of i-security and ta-security. We obtained new insights in the relation between them. For both, we provided the first sound and complete characterization in terms of a state-based unwinding. We also provided characterizations for the notions of to-security and ito-security as needed for undecidability results.

In this thesis, we provided the first rigorous definitions of dynamic noninterference. These include definitions for both a transitive and an intransitive interpretation of a policy as well as a new notion between them. The notion of dt-security is a very restrictive definition of dynamic noninterference, which does not allow any intransitive effects. The security definition of dot-secure allows the transmission of previously performed actions, and di-security is a generalization into a fully dynamic intransitive noninterference definition. For all of these noninterference definitions, we provided several characterizations, using our developed techniques when they seemed to be appropriate. These include sound and complete characterizations in terms of a state-based unwinding for all of these notions. For dynamic intransitive noninterference, we provided an alternative definition, dta-security, which has advantages according to the transmission of information about not performed actions and the applicability to asynchronous systems.

For those security definitions, we have a characterization in a polynomial-size unwinding, we provided efficient polynomial-time algorithms which computed the unwinding relations. These noninterference definitions were t-security, i-security, ta-security, dt-security, and dot-security. In contrast, we showed that the verification problem for di-security is NP-complete. Beyond these complexity results, we showed that the verification problem for to-security and ito-security is undecidable. Furthermore, we gave a general construction pattern which can be used to prove for several automata models that the verification problem for t-security in the corresponding system model lies in the same complexity class as the language-equivalence problem for the automata model.

In a case study, we analyzed the distributed dynamic access control system Flume with respect to dynamic intransitive noninterference. We proved this system secure according to dta-security using the technique of securely constructed systems. It strengthened our proposition of new dynamic noninterference definitions and showed their applicability.

In Chapter 7, we gave an overview about other semantic models for formulating and analyzing noninterference properties. Due to the vast amount of work in this area, we only outlined the main directions.

The research presented in this thesis leads to new open questions. Mainly, in the quite new area of dynamic noninterference, there are several directions for future projects. A general problem in the context of noninterference is the large amount of theoretical works compared to only a few applications of it. In particular, we would like to see more applications of dynamic noninterference to obtain more evidences that our framework is an adequate formalization of the desired security requirements. An analysis of further discretionary access control systems, as well as an adaption to role-based access control systems, could be a first step.

For representing dynamic information flow requirements, we used dynamic policies defined on the states of the system. A clearer separation between the system and the policy might be a desired property. One approach in this direction is to define the local policies on the set of traces instead of on the set of states. However, for the verification problem a finite representation of the dynamic policy is necessary, which again possibly leads to finite-state systems.

Another way of defining dynamic noninterference is by a static policy and a dynamic assignment of actions to agents. It would be interesting to see a formalization of this idea for a state-based setting and how it relates to our framework.

Due to the successful application of dta-security to a dynamic access control system, a further investigation of this notion is desired. In particular, if possible, we would like to have a characterization in terms of a state-based unwinding in order to derive an efficient verification algorithm for this notion.

From an algorithmic perspective, the computation of information flow and minimal policies is worth deeper investigation. We provided a starting

point with algorithms for t-security and i-security and left all other notions of noninterference open.

Bibliography

[AC12] Aslan Askarov and Stephen Chong. Learning is change in knowledge: Knowledge-based security for dynamic policies. In Stephen Chong, editor, *CSF*, pages 308–322. IEEE, 2012.

[Aga00] Johan Agat. Transforming out timing leaks. In Mark N. Wegman and Thomas W. Reps, editors, *POPL*, pages 40–53. ACM, 2000.

[AHSS08] Aslan Askarov, Sebastian Hunt, Andrei Sabelfeld, and David Sands. Termination-insensitive noninterference leaks more than just a bit. In Sushil Jajodia and Javier Lopez, editors, *Computer Security - ESORICS 2008*, volume 5283 of *Lecture Notes in Computer Science*, pages 333–348. Springer Berlin Heidelberg, 2008.

[AS07] Aslan Askarov and Andrei Sabelfeld. Gradual release: Unifying declassification, encryption and key release policies. In *IEEE Symposium on Security and Privacy*, pages 207–221. IEEE Computer Society, 2007.

[BDG+10] Michael Backes, Markus Dürmuth, Sebastian Gerling, Manfred Pinkal, and Caroline Sporleder. Acoustic side-channel attacks on printers. In *USENIX Security Symposium*, pages 307–322. USENIX Association, 2010.

[BDR11] Gilles Barthe, Pedro R. D'Argenio, and Tamara Rezk. Secure information flow by self-composition. *Mathematical Structures in Computer Science*, 21(6):1207–1252, 2011.

[Bib77] Kenneth J. Biba. Integrity considerations for secure computer systems. Technical Report MTR-3153, The Mitre Corporation, April 1977.

[Bis03] Matt Bishop. *Computer Security: Art and Science*. Addison-Wesley Professional, Boston, MA, USA, 2003.

[BL73] David E. Bell and Leonard J. LaPadula. Secure computer systems: Mathematical foundations. Technical Report AD0770768, MITRE CORP BEDFORD MA, NOV 1973.

[BL76] David E. Bell and Leonard J. LaPadula. Secure computer system: unified exposition and multics interpretation. Technical Report ESD-TR-75-306, Mitre Corporation, Bedford, M.A., March 1976.

[BP03] Michael Backes and Birgit Pfitzmann. Intransitive noninterference for cryptographic purpose. In *IEEE Symposium on Security and Privacy*, pages 140–. IEEE Computer Society, 2003.

[BP04] Michael Backes and Birgit Pfitzmann. Computational probabilistic noninterference. *International Journal of Information Security*, 3(1):42–60, 2004.

[BS06] Niklas Broberg and David Sands. Flow locks: Towards a core calculus for dynamic flow policies. In Peter Sestoft, editor, *ESOP*, volume 3924 of *Lecture Notes in Computer Science*, pages 180–196. Springer, 2006.

[Can00] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. *IACR Cryptology ePrint Archive*, page 67, 2000.

[CCH+11] Ran Canetti, Suresh Chari, Shai Halevi, Birgit Pfitzmann, Arnab Roy, Michael Steiner, and Wietse Venema. Composable security analysis of os services. In Javier Lopez and Gene Tsudik, editors, *Applied Cryptography and Network Security*, volume 6715 of *Lecture Notes in Computer Science*, pages 431–448. Springer Berlin Heidelberg, 2011.

[CLRS01] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001.

[CMR07] Franck Cassez, John Mullins, and Olivier H. Roux. Synthesis of non-interferent systems. In *4th Int. Conf. on Mathematical Methods, Models and Architectures for Computer Network Security (MMM-ACNS'07)*, volume 1, pages 307–321, Saint Petersburg, Russie, Fédération De, 2007. Springer.

[com12] Common criteria for information technology security evaluation version 3.1 revision 4, September 2012. URL: `http://www.commoncriteriaportal.org`.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Symposium on Theory of Computing*, pages 151–158. ACM, 1971.

[Cor12] Symantec Corporation. 2012 norton cybercrime report, September 2012. URL: `http://www.norton.com/2012cybercrimereport`.

[CS10] Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.

[CV12] Ran Canetti and Margarita Vald. Universally composable security with local adversaries. *IACR Cryptology ePrint Archive*, page 117, 2012.

[DD77] Dorothy E. Denning and Peter J. Denning. Certification of programs for secure information flow. *Communications of the ACM*, 20(7):504–513, 1977.

[DHK+08] Deepak D'Souza, Raveendra Holla, Janardhan Kulkarni, Raghavendra K. Ramesh, and Barbara Sprick. On the decidability of model-checking information flow properties. In R. Sekar and Arun K. Pujari, editors, *ICISS*, volume 5352 of *Lecture Notes in Computer Science*, pages 26–40. Springer, 2008.

[DRS05] Deepak D'Souza, K. R. Raghavendra, and Barbara Sprick. An automata based approach for verifying information flow properties. *Electronic Notes in Theoretical Computer Science*, 135(1):39–58, 2005.

[ESW12] Sebastian Eggert, Henning Schnoor, and Thomas Wilke. Dynamic noninterference: Consistent policies, characterizations and verification. *CoRR*, abs/1208.5580, 2012.

[ESW13] Sebastian Eggert, Henning Schnoor, and Thomas Wilke. Noninterference with local policies. In Krishnendu Chatterjee and Jiri Sgall, editors, *MFCS*, volume 8087 of *Lecture Notes in Computer Science*, pages 337–348. Springer, 2013.

[EvdMSW11] Sebastian Eggert, Ron van der Meyden, Henning Schnoor, and Thomas Wilke. The complexity of intransitive noninterference. In *IEEE Symposium on Security and Privacy*, pages 196–211. IEEE Computer Society, 2011.

[EvdMSW13] Sebastian Eggert, Ron van der Meyden, Henning Schnoor, and Thomas Wilke. Complexity and unwinding for intransitive noninterference. Submitted for publication in the journal Information and Computation, 51 pages, 2013. URL: `http://arxiv.org/abs/1308.1204`.

[EvdMZ12] Kai Engelhardt, Ron van der Meyden, and Chenyi Zhang. Intransitive noninterference in nondeterministic systems. In *ACM Conference on Computer and Communications Security*, pages 869–880, 2012.

[FG01] Riccardo Focardi and Roberto Gorrieri. Classification of security properties (Part I: information flow). In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design, FOSAD 2000, Bertinoro, Italy, September 2000*, volume 2171 of *LNCS*, pages 331–396. Springer, 2001.

[FHMV95] Ronald Fagin, Joseph. Y. Halpern, Yoram Moses, and

Moshe. Y. Vardi. *Reasoning About Knowledge*. MIT-Press, 1995.

[GD72] G. Scott Graham and Peter J. Denning. Protection: Principles and practice. In *Proceedings of the May 16-18, 1972, Spring Joint Computer Conference*, AFIPS '72 (Spring), pages 417–429, New York, NY, USA, 1972. ACM.

[GM82] Joseph A. Goguen and José Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pages 11–20, 1982.

[GM84] Joseph A. Goguen and José Meseguer. Unwinding and inference control. In *IEEE Symposium on Security and Privacy*, 1984.

[Gra90] James W. Gray III. Probabilistic interference. In *Research in Security and Privacy, 1990. Proceedings., 1990 IEEE Computer Society Symposium on*, pages 170–179, 1990.

[Gra92] James W. Gray III. Toward a mathematical foundation for information. *Journal of Computer Security*, 1(3-4):255–294, 1992.

[HALL+05] N.B. Hadj-Alouane, S. Lafrance, Feng Lin, J. Mullins, and M.M. Yeddes. On the verification of intransitive noninterference in multilevel security. *IEEE Trans. on Systems, Man and Cybernetics, Part B*, 35(5):948– 958, Oct. 2005.

[HKN05] Shai Halevi, Paul A. Karger, and Dalit Naor. Enforcing confinement in distributed storage and a cryptographic model for access control. *IACR Cryptology ePrint Archive*, 2005:169, 2005.

[Hoa78] C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*, 21(8):666–677, August 1978.

[HRU76] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, August 1976.

[HY87] J. Thomas Haigh and William D. Young. Extending the noninterference version of MLS for SAT. *IEEE Transactions on Software Engineering*, SE-13(2):141–150, Feb 1987.

[Jac88] Jeremy Jacob. Security specifications. In *IEEE Symposium on Security and Privacy*, pages 14–23. IEEE Computer Society, 1988.

[JT88] Dale M. Johnson and F. Javier Thayer. Security and the composition of machines. In *IEEE Computer Security Foundations Workshop*, pages 72–89. MITRE Corporation Press, 1988.

[Ker83] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX, 1883.

[Kle56] Stephen C. Kleene. Representation of events in nerve nets and finite automata. *Automata Studies*, 1956.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer Berlin Heidelberg, 1996.

[KR02] Calvin Ko and Timothy Redmond. Noninterference and intrusion detection. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 177–187, 2002.

[Kro08] Maxwell Krohn. *Information Flow Control for Secure Web Sites*. PhD thesis, September 2008.

[KT09] Maxwell Krohn and Eran Tromer. Noninterference for a practical difc-based operating system. In *IEEE Symposium onSecurity and Privacy*, pages 61–76, 2009.

[KYB+07] Maxwell Krohn, Alexander Yip, Micah Brodsky, Natan Cliffer, M. Frans Kaashoek, Eddie Kohler, and Robert Morris. Information flow control for standard os abstractions. *SIGOPS - Operating Systems Review*, 41(6):321–334, October 2007.

[Lam73] Butler W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, October 1973.

[Lam74] Butler W. Lampson. Protection. *SIGOPS - Operating Systems Review*, 8(1):18–24, January 1974.

[Les06] Rebekah Leslie. Dynamic intransitive noninterference. In *Proc. IEEE International Symposium on Secure Software Engineering*, 2006.

[LM82] Abe Lockman and Naftaly H. Minsky. Unidirectional transport of rights and take-grant control. *IEEE Transactions on Software Engineering*, 8(6):597–604, 1982.

[LM03] Stéphane Lafrance and John Mullins. An information flow method to detect denial of service vulnerabilities. *J.UCS Journal of Universal Computer Science*, 9(11):1350–, 2003.

[Man00] Heiko Mantel. Possibilistic definitions of security – an assembly kit. In *Proc. IEEE Computer Security Foundations Workshop*, pages 185–199, July 2000.

[Man01] Heiko Mantel. Information flow control and applications - bridging a gap. In *FME*, pages 153–172, 2001.

[Man03] Heiko Mantel. *A Uniform Framework for the Formal Specification and Verification of Information Flow Security*. PhD thesis, Universität des Saarlandes, 2003.

[McC88] Daryl McCullough. Noninterference and the composability of security properties. In *Proc. IEEE Symposium on Security and Privacy*, pages 177–186, 1988.

[McC90] Daryl McCullough. A hookup theorem for multilevel security. *IEEE Transactions on Software Engineering*, 16(6):563–568, 1990.

[McL90] John McLean. Security models and information flow. In *IEEE Computer Society Symposium on Research in Security and Privacy*, pages 180–187, 1990.

[McL94] John McLean. A general theory of composition for trace sets closed under selective interleaving functions. In *Proc. IEEE Symposium on Security and Privacy*, pages 79–93, May 1994.

[Mil87] Jonathan K. Millen. Covert channel capacity. In *IEEE Symposium on Security and Privacy*, pages 60–66. IEEE Computer Society, 1987.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[ML97] Andrew C. Myers and Barbara Liskov. A decentralized model for information flow control. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, SOSP '97, pages 129–142, New York, NY, USA, 1997. ACM.

[MMB+12] Toby Murray, Daniel Matichuk, Matthew Brassil, Peter Gammie, and Gerwin Klein. Noninterference for operating system kernels. In Chris Hawblitzel and Dale Miller, editors, *Certified Programs and Proofs*, volume 7679 of *Lecture Notes in Computer Science*, pages 126–142. Springer Berlin Heidelberg, 2012.

[Mu08] Chunyan Mu. Quantitative information flow for security: a survey. Technical Report TR-08-06, Department of Computer Science, King's College London, September 2008. URL: `http://www.dcs.kcl.ac.uk/technical-reports/papers/TR-08-06.pdf`.

[MVCT11] Philip Marquardt, Arunabh Verma, Henry Carter, and Patrick Traynor. (sp)iphone: decoding vibrations from nearby keyboards using mobile phone accelerometers. In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM Conference on Computer and Communications Security*, pages 551–562. ACM, 2011.

[MWTG00] W. Martin, P. White, F.S. Taylor, and A. Goldberg. Formal construction of the mathematically analyzed separation kernel. In IEEE Computer Society Press, editor, *Proc. 15th IEEE Conf. on Automated Software Engineering*, 2000.

[Mye99] Andrew C. Myers. Jflow: Practical mostly-static information flow control. In Andrew W. Appel and Alex Aiken, editors, *POPL*, pages 228–241. ACM, 1999.

[MZ08] Ron van der Meyden and Chenyi Zhang. Information flow in systems with schedulers. In *Computer Security Foundations Symposium, 2008. CSF '08. IEEE 21st*, pages 301–312, 2008.

[O'H90] Colin O'Halloran. A calculus of information flow. In *ESORICS*, pages 147–159. AFCET, 1990.

[Pin95] S. Pinsky. Absorbing covers and intransitive non-interference. In *Proc. IEEE Symp. on Security and Privacy*, pages 102–113, 1995.

[Pos46] Emil L. Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52, 1946.

[PS03] François Pottier and Vincent Simonet. Information flow inference for ml. *ACM Trans. Program. Lang. Syst.*, 25(1):117–158, 2003.

[PW01] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy*, pages 184–200, 2001.

[RG99] A. W. Roscoe and M. H. Goldsmith. What is intransitive noninterference? In *IEEE Computer Security Foundations Workshop*, pages 228–238, 1999.

[Ros95] A.W. Roscoe. CSP and determinism in security modelling. In *Proc. IEEE Symp. on Security and Privacy*, pages 114–221, 1995.

[RS01] P.Y.A. Ryan and S.A. Schneider. Process algebra and noninterference. *Journal of Computer Security*, 9(1/2):75–103, 2001.

[Rus81] John Rushby. Design and verification of secure systems. In *Proc. 8th Symposium on Operating Systems Principles*, pages

12–21, Asilomar CA, Dec 1981. (ACM Operating Systems Review, Vol 15, No. 1).

[Rus92] John Rushby. Noninterference, transitivity, and channel-control security policies. Technical Report CSL-92-02, SRI International, 1992. URL: http://www.csl.sri.com/papers/csl-92-2/.

[Rya01] Peter Y. A. Ryan. Mathematical models of computer security. In R. Focardi and R. Gorrieri, editors, *Foundations of Security Analysis and Design, FOSAD 2000, Bertinoro, Italy, September 2000*, volume 2171 of *LNCS*, pages 1–62. Springer, 2001.

[SM03] Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *Selected Areas in Communications, IEEE Journal on*, 21(1):5 – 19, jan 2003.

[Smi07] Geoffrey Smith. Principles of secure information flow analysis. In Mihai Christodorescu, Somesh Jha, Douglas Maughan, Dawn Song, and Cliff Wang, editors, *Malware Detection*, volume 27 of *Advances in Information Security*, pages 291–307. Springer US, 2007.

[Smi09] Geoffrey Smith. On the foundations of quantitative information flow. In Luca Alfaro, editor, *Foundations of Software Science and Computational Structures*, volume 5504 of *Lecture Notes in Computer Science*, pages 288–302. Springer Berlin Heidelberg, 2009.

[SRS+00] Gerhard Schellhorn, Wolfgang Reif, Axel Schairer, Paul A. Karger, Vernon Austel, and David C. Toll. Verification of a formal security model for multiapplicative smart cards. In Frédéric Cuppens, Yves Deswarte, Dieter Gollmann, and Michael Waidner, editors, *ESORICS*, volume 1895 of *Lecture Notes in Computer Science*, pages 17–36. Springer, 2000.

[SS09] Andrei Sabelfeld and David Sands. Declassification: Dimensions and principles. *Journal of Computer Security*, 17(5):517–548, 2009.

[Sut86] David Sutherland. A model of information. In *Proc. 9th National Computer Security Conf.*, pages 175–183, 1986.

[Tar75] Robert E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, 1975.

[TW08] Michael Carl Tschantz and Jeannette M. Wing. Extracting conditional confidentiality policies. In Antonio Cerone and Stefan Gruner, editors, *SEFM*, pages 107–116. IEEE Computer Society, 2008.

[US 85] US Department of Defense. *Department of Defense Trusted Computer System Evaluation Criteria*, December 1985. DOD 5200.28-STD (supersedes CSC-STD-001-83).

[usl11] United states code, 2006 edition, supplement 5, title 44. Added Pub. L. 107-347, title III, §301(b)(1), Dec. 17, 2002, 116 Stat. 2947, 2011.

[vdM07] Ron van der Meyden. What, indeed, is intransitive noninterference? In Joachim Biskup and Javier Lopez, editors, *ESORICS*, volume 4734 of *Lecture Notes in Computer Science*, pages 235–250. Springer, 2007.

[vdMZ07] Ron van der Meyden and Chenyi Zhang. Algorithmic verification of noninterference properties. *Electronic Notes in Theoretical Computer Science*, 168:61–75, 2007.

[vdMZ10] Ron van der Meyden and Chenyi Zhang. A comparison of semantic models for noninterference. *Theoretical Computer Science - Journal*, 411(47):4123–4147, 2010.

[VIS96] Dennis M. Volpano, Cynthia E. Irvine, and Geoffrey Smith. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(2/3):167–188, 1996.

[vO04] David von Oheimb. Information flow control revisited: Noninfluence = noninterference + nonleakage. In *ESORICS*, pages 225–243, 2004.

[WJ90] J. Todd Wittbold and Dale M. Johnson. Information flow in nondeterministic systems. In *IEEE Symposium on Security and Privacy*, pages 144–161, 1990.

[YLBHA09] Moez Yeddes, Feng Lin, and Nejib Ben Hadj-Alouane. Modifying security policies for the satisfaction of intransitive non-interference. *IEEE Transactions on Automatic Control*, 54(8):1961–1966, 2009.

[ZAM12] Danfeng Zhang, Aslan Askarov, and Andrew C. Myers. Language-based control and mitigation of timing channels. In Jan Vitek, Haibo Lin, and Frank Tip, editors, *PLDI*, pages 99–110. ACM, 2012.

[Zha11] Chenyi Zhang. Conditional information flow policies and unwinding relations. In Roberto Bruni and Vladimiro Sassone, editors, *TGC*, volume 7173 of *Lecture Notes in Computer Science*, pages 227–241. Springer, 2011.

[ZM04] Lantian Zheng and Andrew C. Myers. Dynamic security labels and noninterference (extended abstract). In *Formal Aspects in Security and Trust*, pages 27–40, 2004.