A type system for checking information flows in distributed systems

Master thesis

Kasper Laursen

Kongens Lyngby 2015

DTU Compute
Department of Applied Mathematics and Computer Science

Master of Science in Engineering
Computer Science and Engineering

A type system for checking information flows in distributed systems
Master thesis

Kasper Laursen
s093078@student.dtu.dk

Technical University of Denmark
Department of Applied Mathematics and Computer Science

DTU Compute
Richard Petersens Plads
Building 324
2800 Kgs. Lyngby
DENMARK

Tel. +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary

As the aviation industry integrates an expanding number of software components in airplanes, it is becoming increasingly difficult to reason about the security of the software. This thesis addresses aspects of this issue in the context of an abstract model of avionics systems as a distributed system using synchronous communication channels. The security of a system can then be related to the information flows and whether these are allowed by a set of policies, thus, using well-established theory from decentralised label model (DLM).

The security properties are verified using a combined type system and Hoare logic, which certifies that the defined policies are not violated by the execution of the distributed system as a whole.

In addition, the thesis describes a concrete implementation of parts of the type system, including tests on a use case from the avionics industry. The testing reveals a false positive, in which a secure system is erroneously declared non-secure. Therefore, the type system needs further adjustments.

Preface

This master thesis was prepared at the department of Applied Mathematics and Computer Science at the Technical University of Denmark in fulfillment of the requirements for acquiring the Master of Science in Engineering degree in Computer Science and Engineering.

The original project plan and auto-evaluation of the project process as required by DTU are in Appendix C.

Prerequisites

The reader is expected to have a knowledge of semantics, inference systems, type systems and Hoare logic similar to what is presented in [NN07]. Furthermore, the reader is assumed to have some knowledge about security regarding information flows, integrity and confidentiality.

Acknowledgements

I would like to thank my supervisor Hanne Riis Nielson for our weekly meetings and support throughout the project. I would give a special thanks to Julie Lynggard for assistance with figure drawings. Furthermore a great thanks to Christian Gram Kalhauge, Tobias Bertelsen, Simon B. Laursen, Søren Dejgaard Schmidt, Søren Løvborg, Eddi Søgaard, and Daniel Agervig Krogh for proofreading and feedback.

Kongens Lyngby, July 11, 2015

[illegible]

Kasper Laursen

Contents

CHAPTER 1
Introduction

Airplanes have an increasing number of sensors and units that interact and exchange information [But07]. Some of the information flows between these units have different security levels, such as airplane controls and personal information. To ensure a safe flight it is therefore necessary to track which units can influence or read information to guarantee the integrity and confidentiality level of the information flow.

The fault-tolerance mechanisms used to develop airplanes have mostly dealt with hardware faults and general software bugs. Faults due to an intruder and malicious code have therefore not been considered in the software for airplanes [DD12].

The *U.S. Federal Aviation Administration* (FAA) certifies if an airplane is safe and operational also in terms of software. In a report by the *U.S. Government Accountability Office* (GAO) they warn FAA that they need a better approach to address cybersecurity [SL15; GAO15]. In recent news this is support by a security researcher that found vulnerabilities in the inflight entertainment system, claiming to be able to monitor traffic in the cockpit and even control the airplane [WIR15; Hur15]. FAA claims no vulnerability, so whether the researcher is speaking the truth (or perhaps exaggerating) is still an open question.

In the rest of the introduction we will see a case study concerning the avionics industry, how their problems can be solved, and the scientific contributions of this thesis.

1.1 Case study: Avionics industry

The *aviation electronics* (avionics) industry has developed safety-critical and reliable hardware and software for decades. In classical avionics each function was put in a separate avionics controller following a “federated architecture”, to ensure a high independence and reliability of each controller. In the mid-1990s, growth in communication and signal interfaces led to the weight, volume and power consumption of all the different controllers reaching airplane limits. Meanwhile, to keep maintaining all the controllers, the number of spare parts also kept growing, thereby increasing the ongoing cost of maintenance [But07].

The avionics industry therefore developed the *Integrated Modular Avionics* (IMA) architecture, where software with different security domains is integrated on shared commercial off-the-shelf components. This indeed lowered the weight, volume, power consumption, and hardware maintenance costs. Central to IMA is the idea of an operating system which can handle multiple applications separately (a separation kernel), guaranteeing that altering or adding applications has minimal or no effect on

Information type	Confidentiality	Integrity	Availability
Aircraft control information	Low	High	High
Airline operational information	High	Medium	Medium
Airline administrative information	High	Medium	Medium
Airline passenger information	High	High	Medium

Table 1.1: Security level of different information types [A81105, Table 3-4].

the other applications. IMA thus moved module costs from the separate hardware components to the development and certification of the separation kernel [But07].

The *Multiple Independent Levels of Security* (MILS) approach is a guideline on how to design, construct, integrate and evaluate secure systems. This approach decomposes components and locates the vulnerable parts, and suggests strict separation and information flow control, hence achieving the IMA safety and security requirements.

Aeronautical Radio, Incorporated (ARINC) is a major manufacturer of communication solutions for the aviation industry, among others. Among their many standards is ARINC report 811 [A81105], which provides security concepts for airborne networks. The ARINC report 811 also prescribes the decomposition of a system into several domains, with an actual example as shown in Figure 1.1 with four different domains, *Aircraft Control* (AC), *Airline Information Service* (AIS), *Passenger Information and Entertainment Service* (PIES), and *Passenger-owned Devices* (POD). These domains are only a minimal subset, where e.g. the Aircraft Control Domain could be split up into several domains, some for navigation and others for ground communication. ARINC report 811 also provides an "aircraft security process framework", which more or less is the *Common Criteria* (CC) [CC12a; CC12b; CC12c] in avionics terminology.

Although Figure 1.1 is a simplification, it still shows many of the different systems on an airplane which should be able to communicate across different security domains. For example, some systems in the AC domain should not be influenced by the passengers and yet there is a link between the PIES and AC through AIS. ARINC report 811 also suggests integrity, confidentiality and availability levels for different types of information, shown in Table 1.1. Aircraft control information could typically be a sensor, e.g. a GPS, where everyone is allowed to know the location (hence the confidentiality level is low), but on the other hand, no one must alter the GPS sensor reports (hence the integrity level must be high). Finally, it is always necessary to have the location (hence the availability level must also be high).

The primary example of a separation kernel used by the avionics industry is PikeOS by SYSGO, which fulfils and implements IMA and the MILS architecture [SYSGO]. The kernel is certified to the highest Safety Integrity Level, SIL 4.

The separate controllers of the federated architecture are therefore virtualised as applications on top of PikeOS. All these application can be part of different security domains, and the separation kernel then ensures that the information flow is secure across these domains at run time. The separation kernel hence specifies some common ground for defining policies, and either block or allow information flows.

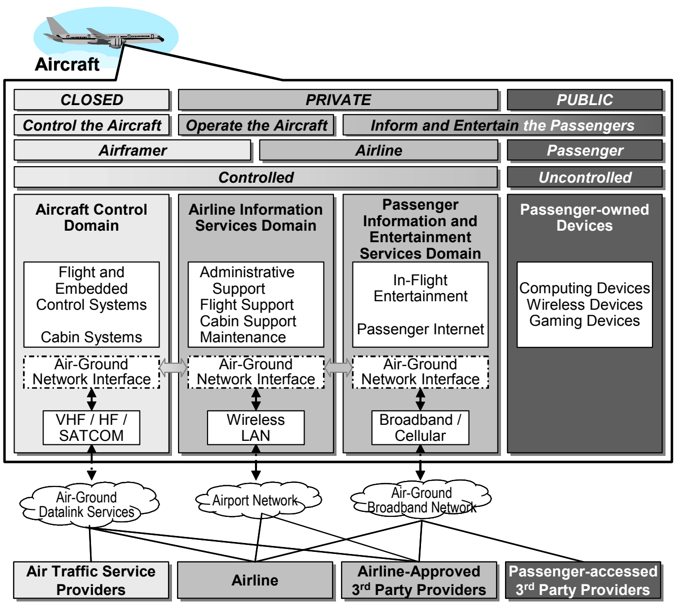
Aircraft
CLOSED
PRIVATE
PUBLIC
Control the Aircraft
Operate the Aircraft
Inform and Entertain the Passengers
Airframer
Airline
Passenger
Controlled
Uncontrolled
Aircraft Control Domain
Airline Information Services Domain
Passenger Information and Entertainment Services Domain
Passenger-owned Devices
Flight and Embedded Control Systems
Cabin Systems
Administrative Support
Flight Support
Cabin Support
Maintenance
In-Flight Entertainment
Passenger Internet
Computing Devices
Wireless Devices
Gaming Devices
Air-Ground Network Interface
Air-Ground Network Interface
Air-Ground Network Interface
VHF / HF / SATCOM
Wireless LAN
Broadband / Cellular
Air-Ground Datalink Services
Airport Network
Air-Ground Broadband Network
Air Traffic Service Providers
Airline
Airline-Approved 3rd Party Providers
Passenger-accessed 3rd Party Providers

Figure 1.1: Security domains according to ARINC report 811 [A81105].

Depending on the complexity of the system, the many security checks and policy lookups could take up much time and can therefore become a bottleneck in a real-time setting. Such a bottleneck can quickly have fatal consequences, e.g. if the main engine is not working properly and the separation kernel is spending time verifying the information flow from the engine sensor. This scenario could be even worse if the separation kernel is handling a queue of flows from non-operational applications such as the entertainment system and the flow from the engine is waiting to be verified.

Implementing an application to the separation kernel then raises questions about what to do if the information flow is not allowed by the policies. The execution could simply stop, which could result in a system not working as intended. Another approach is to keep a log of the illegal information flows for later investigation and possibly adjustment of policies, and continue running the application with the leaked information. In avionics, either approach could have fatal consequences.

1.2 Decentralised label model

The *Decentralised Label Model* (DLM) can be used to ensure confidentiality and integrity of data in a software system [ML97; ML00]. Inside the software system, *principals* are authorities representing the different software components and users, for whom information is owned, updated and affected by. Whenever there is an information flow from one set of variables to another there are corresponding *integrity labels* and *confidentiality labels*.

An integrity label is denoted $\{o \leftarrow \bar{s}\}$ where o is the owner principal and $\bar{s}$ is a set of principals. A variable x can therefore be associated with such a label denoted as an *influencer policy* $\{o \leftarrow \bar{s}\}\ x$, meaning that o is the owner of x, and o allows all the principals in $\bar{s}$ to influence x. The integrity label thus describes who can influence the information. Example:

$$\begin{aligned}
&\{o_1 \leftarrow s_1, s_2\}\ x;\\
&\{o_1 \leftarrow s_1, s_3\}\ y;\\
&\{o_1 \leftarrow s_1, s_2, s_3\}\ z;\\
&z := x + y;
\end{aligned}$$

As x can be influenced by s_1 and s_2, and y can be influenced by s_1 and s_3, the value of $x + y$ would altogether be influenced by s_1, s_2 and s_3. The resulting label for the value would then be $\{o_1 \leftarrow s_1, s_2, s_3\}$. The variable z may be influenced by s_1, s_2 and s_3 and the assignment $z := x + y$ does therefore follow the policies. As the labels use the same owner for all variables, the owner is irrelevant to this example.

Similarly, a confidentiality label is denoted $\{o \rightarrow \bar{s}\}$ where o is the owner, and $\bar{s}$ is a set of principals. A variable x can be associated with a label denoted as a *reader policy* $\{o \rightarrow \bar{s}\}\ x$, meaning that o is the owner of x, and o allows the principals $\bar{s}$ to read x. The confidentiality label therefore describes who can read the information. Example:

$$\begin{aligned}
&\{o_1 \rightarrow s_1, s_2\}\ x;\\
&\{o_1 \rightarrow s_1, s_3\}\ y;\\
&\{o_1 \rightarrow s_1\}\ z;\\
&z := x + y;
\end{aligned}$$

Here x may be read by s_1 and s_2, and y can be read by s_1 and s_3. The value of $x + y$ must then only be readable by s_1, in order not to violate the confidentiality of x and y. As z only allows s_1 as reader, the assignment $z := x + y$ follows the policies. Again, as the owner is the same for all variables, it is irrelevant to this example.

Just these two examples demonstrates a duality between influencer policies and reader policies. Integrity uses the union of the principals, while confidentiality uses the intersection. Formalising one part therefore reveals the other by swapping the operators. This is also one of realisations in [ML00]. Furthermore, the examples clearly shows that there is an information flow from x and y to z.

As an aside, note that a DLM label can have multiple owners e.g. $\{\{o_1 \rightarrow s_1, s_2\}; \{o_2 \rightarrow s_1, s_3\}\}$ where the effective reader set of the label here gives the singleton $\{s_1\}$, because that is the only principal the two owners agree on. Conversely an effective influencer set can be established for influencer policies.

In the two examples, it was easy to deduce that the assignment obeyed the policies, and static analysis of such a program could easily confirm this conclusion, thereby guaranteeing that the system is secure according to its policies, before it is even executed [VSI09].

DLM also provide a mechanism to *endorse* information (weakening its integrity) and *declassify* information (weakening its confidentiality), allowing information flows not otherwise permitted by the policies. Example:

```
{$o_1 \leftarrow s_1, s_2$} $x$;
{$o_1 \leftarrow s_1$} $y$;
{$o_1 \leftarrow s_1$} $z$;
endorse {$o_1 \leftarrow s_1, s_2$} to {$o_1 \leftarrow s_1$} in
  $z := x + y$;
end;
```

Here the label $\{o_1 \leftarrow s_1, s_2\}$ is endorsed to $\{o_1 \leftarrow s_1\}$. This allows x to influence z as they now have the same label. Conversely, a declassify statement exists allowing more to read the information.

Useful as endorse and declassify may be, their use should be strongly discouraged, as they weaken the security policies. If they are used, proper code review is therefore needed to ensure that there are no unintentional information leakages.

1.3 Gateway

In the separation kernel from the avionics case, it is sometimes impossible to establish the strict separation of resources suggested by MILS. An example is presented in [Mül+12] where a gateway is used to connect multiple domains, as shown in Figure 1.2 where p_1 wants to send information to c_1 and p_2 sends information to c_2, but they both have to use the same channel ch where m represents a multiplexer and d represents a demultiplexer. In this system we want a guarantee that the information from p_1 does not flow to c_2 and vice versa from p_2 to c_1.

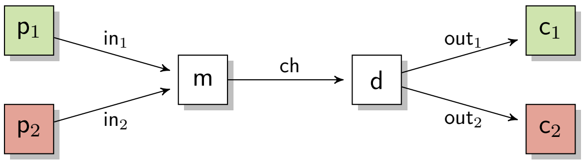

Figure 1.2: Gateway [NNL15].

Let z be the variable containing the value the demultiplexer is receiving though the channel ch. The upper part of the gateway says that z may only be influenced by p_1 and the lower part says that z only may be influenced by p_2 yielding the following two DLM policies

$$\{\mathsf{d} \leftarrow \mathsf{p}_1, \mathsf{m}, \mathsf{d}\}\ z \tag{1.1}$$
$$\{\mathsf{d} \leftarrow \mathsf{p}_2, \mathsf{m}, \mathsf{d}\}\ z \tag{1.2}$$

As the variable z will also be used to send the information to c_1 and c_2, the resulting label for z will be $\{\mathsf{d} \leftarrow \mathsf{p}_1, \mathsf{p}_2, \mathsf{m}, \mathsf{d}\}$ which does not suit c_1 as the value can then be influenced by p_2, and vice versa for c_2. The two policies are therefore dependent on the content they are handling.

Clearly m knows where the information comes from, and sends 1 or 2 to d along with the information to indicate the origin. Then d can use this as a trigger to endorse the proper label for z before it is sent to c_1 or c_2. This constellation though uses the endorsement from DLM and time consuming code review have to be done.

1.4 Contributions

In avionics, the most critical parts are checked using formal methods and verifications [DD12], for example using the Astrée static analyzer [Asta; Astb]. It is clear that some of the integrity and confidentiality issues from the avionics case can be directly solved using DLM, and the tools for static analysis can then be extended to verify DLM polices. The run-time checks performed by the separation kernel can therefore be minimised. This gives the overall system more available processing time, which is useful in a real-time setting. In this particular case, more applications can then be stacked in the separation kernel, leading to fewer controllers and lower power consumption.

In the gateway example, regular DLM is insufficient to eliminate time consuming (and possible erroneous) code review. In [NNL15] some of the features from DLM are presented and extended with "content dependent policies", which solves these problems, though the resulting type system is complicated and could potentially have some errors.

This thesis is mainly built upon the draft paper [NNd] where a more DLM-comparable notion of policies are used. Furthermore, the type system is simpler and thus more clear than [NNL15].

In [NNd], the first part only covers influencer policies, only listing ideas for the reader policies. As we will see later, the policies and type system will be fully described and proven here for both influencer and reader polices. Furthermore, the implementation of the type system raises a question on how to compare policies, and a neat result is derived on how to do this.

Overall this thesis therefore contributes to the correctness of the type system in [NNd] and details how it works and how some of the implementation challenges can be met.

1.5 Thesis overview

Chapter 2 contains the design of an abstract language for distributed systems which amounts to an instrumented semantics that enables monitoring of information flows, along with a rich language for defining security policies against which the information flows can be verified. Taken together, this leads to a formal definition of when a system is secure. In relation to the case study, the result can be used to model applications in a separation kernel and verify overall system security at run-time.

A type system is constructed in Chapter 3. Using the type system on a well defined distributed system (including policies), the distributed system can be verified for all executions of the system. In the end this leads to a correctness result ensuring the type system is sound according to the instrumented semantics.

Chapter 4 shows some of the difficulties that comes with implementing the type system. First, an analysis on how to simplify the comparison of the security policies. Second, a syntax-directed type system in the form of the actual type checker, to be proved sound according to the type system. Third, a short description of the software developed for this thesis.

Chapter 5 presents the gateway scenario and another fictive but realistic scenario from the avionics industry. Proper policies for the gateway scenario are described and then verified using the developed type checker.

All this amounts to a discussion in Chapter 6 of other, similar tools for verifying information flows, together with observations, further improvements, and extensions.

At last Chapter 7 presents the conclusion for this thesis.

Throughout the thesis, facts, lemmas and theorems are presented, with most proven formally. These proofs are not relevant for reading this thesis but can help in understanding some of the results. Most of the proofs have therefore been moved to the appendix, leaving just an annotation like for this paragraph. Some facts have not been proven, leaving only a proof idea. (proofs in Appendix A)

CHAPTER 2

Distributed systems

This chapter contains the formal definition of a distributed system and how to specify policies. This leads to a definition of when a system is secure according to its policies.

It begins by describing the syntax for distributed system and its statements along with the instrumented semantics which records the information flow throughout the system. Next, the syntax for the policies are formalised, enabling us to describe allowed information flows in the system. From this a full system can be defined, finally allowing us to verify that the system follows its policies, making it secure in relation to information flow.

In terms of security a *covert channel* is an information flow can be detected by observing the surroundings of a system, such as running time, power usage, noise and electromagnetic radiation [DD77]. As it is difficult to reason about these flows they will not be considered in this thesis.

2.1 Syntax

The system consists of a finite set of processes running in parallel

$$Sys \ ::= \ l_1 : S_1 \parallel \cdots \parallel l_n : S_n$$

Each process $\ell : S$ has a label ℓ and a top-level statement S. The label is an unique identifier $\ell \in \mathbf{Lab} = \{l_1, \ldots, l_n\}$ to distinguish each process, and the statement defines all the computational steps for the process.

A *security principal* is an authority that in the policies will be used to denote who owns, who can influence and who can read the information. A system has a finite set of security principals $\mathbf{Pr}$, and each process is statically assigned to a security principal in the mapping $\mathcal{S} : \mathbf{Lab} \to \mathbf{Pr}$. In addition, $\mathcal{S}$ is required to *not* be surjective, meaning that there must be at least one security principal not mapping to any process. This principal will be able to monitor all information flows in the system, and for that reason is typically denoted NSA. Security principals are generally denoted $o, q, s \in \mathbf{Pr}$, and a set of security principals are denoted $\bar{s} \subseteq \mathbf{Pr}$. An empty $\bar{s}$ is denoted ϵ.

Each process has its own set of variables $\mathbf{Var}_\ell$, emphasising the strict memory separation between processes. The disjoint union of all the variables for the processes is denoted $\mathbf{Var} = \biguplus_{l \in \mathbf{Lab}} \mathbf{Var}_l$.

Processes communicate only via synchronous channels from the set $\mathbf{Chan}$. These channels are polyadic, meaning that in one synchronous action multiple variables can be transferred; the arity of a channel is the number of variables the channel can transfer. An analogue to this is individual wires making up a full cable. Channels are typically denoted $ch \in \mathbf{Chan}$. The ith variable in a channel is denoted $\#_i$ and all channel variables are denoted $\mathbf{Var}_{\#} = \{\#_1, \ldots, \#_m\}$ where m is the maximal arity of all the channels **Chan**.

All the variables in the system for both processes and channels are denoted $\mathbf{Var}^+ = \mathbf{Var} \cup \mathbf{Var}_{\#}$. Single variables are denoted $x, y \in \mathbf{Var}$, variables or channel variables are denoted $u, z \in \mathbf{Var}^+$ and unspecified constants are denoted n. A set of variables is denoted $\bar{x} \subseteq \mathbf{Var}$ and $\bar{u} \subseteq \mathbf{Var}^+$, where the empty set is denoted ϵ.

The set of states $\mathbf{State} = \mathbf{Var} \to \mathbf{Val}$ denotes a mapping from process variables in **Var** to corresponding values in **Val** (e.g. $1, 2, \ldots$, or just v for arbitrary values). Including channel variables this is denoted $\mathbf{State}^+ = \mathbf{Var}^+ \to \mathbf{Val}$. A state is denoted $\sigma \in \mathbf{State}$ or $\sigma \in \mathbf{State}^+$. The value for x in σ is then $\sigma(x)$, and assigning x to a new value v in σ is denoted $\sigma[x \mapsto v]$. Multiple assignments are annotated using the following shorthand notation

$$\sigma[(x_i \mapsto v_i)_{i \leq k}] = \sigma[x_1 \mapsto v_1, x_2 \mapsto v_2, \ldots, x_k \mapsto v_k]$$

2.1.1 Statements S

The syntax for a statement S, arithmetic expressions a and boolean expressions b is:

$$\begin{array}{rcl} S & ::= & \mathsf{skip} \;\mid\; x := a \;\mid\; x :=^{\dagger} a \;\mid\; S_1; S_2 \;\mid\; \mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi} \\ & \mid & \mathsf{while}\ b\ \mathsf{do}\ S\ \mathsf{od} \;\mid\; ch!(a_1, \ldots, a_k) \;\mid\; ch!^{\dagger}(a_1, \ldots, a_k) \\ & \mid & ch?(x_1, \ldots, x_k) \;\mid\; ((S_1) \oplus (S_2)) \;\mid\; \lceil \bar{x} \rceil S \\ a & ::= & n \;\mid\; x \;\mid\; a_1\ op\ a_2 \\ b & ::= & \mathsf{true} \;\mid\; \mathsf{false} \;\mid\; \neg b \;\mid\; b_1 \wedge b_2 \;\mid\; b_1 \vee b_2 \;\mid\; a_1\ rel\ a_2 \end{array}$$

This makes up a simple "while" language greatly inspired by [NNL15; NNd]. The statement skip simply does nothing, $x := a$ assigns a to the variable x, and $S_1; S_2$ is the sequential composition of statements S_1 and S_2. In the conditional statement if b then S_1 else S_2 fi the condition b determines whether the execution continues on S_1 or S_2. For loops while b do S od as long the condition b holds, S is executed. The conditional and loop statement will be referred to as statements with tests.

Different from a simple "while" language is the inter-process communication which is achieved using the input and output statement. The output statement $ch!(a_1, \ldots, a_k)$ send the values $a_1, \ldots, a_k$ using the channel $ch \in \mathbf{Chan}$ (of arity k), and the input statement $ch?(x_1, \ldots, x_k)$ receives values from the channel ch, storing them in the variables $x_1, \ldots, x_k$. For arity $k = 1$, the parentheses can be left out, e.g. $ch!a$ and $ch?x$.

The assignment and output statements also have a corresponding *bypass* version denoted with †, which circumvents the policies. These statement is useful when the programmer intentionally want to leak information, such that the confidentiality and integrity of the information is preserved by other means than the software itself. As we have seen for DLM that is typically denoted endorse for integrity and declassify for confidentiality.

The $((S_1) \oplus (S_2))$ statement makes a non-deterministic choice between S_1 and S_2, only executing one of the nested statements.

The statement $\lceil \bar{x} \rceil S$ will in the semantics be used to keep track of implicit flows from the variables $\bar{x}$. This statement is therefore intended only to be used internally.

In arithmetic expressions, *op* is an arithmetic operator (e.g. $+, -, \ldots$). In boolean expressions, *rel* is a relational operator (e.g. $=, <, \leq, \ldots$). The boolean operators $\neg, \wedge, \vee$ are defined the usual way, where the boolean constants true is denoted true or tt and false is denoted false or ff. The set of all the free variables in an arithmetic or a boolean expression are denoted $\bar{a}$ and $\bar{b}$ respectively.

For arithmetics and boolean expressions the semantics is given by $[\![a]\!]\sigma$ and $[\![b]\!]\sigma$, which given an expression a or b and a state $\sigma \in \mathbf{State}$ returns the value from $\mathbf{Val}$ or a truth value true or false as appropriate. As an example let $\sigma(x) = 4$ and $\sigma(y) = 5$ then

$$[\![x + 1]\!]\sigma = [\![x]\!]\sigma + [\![1]\!]\sigma = \sigma(x) + 1 = 4 + 1 = 5$$
$$[\![x < y]\!]\sigma = [\![x]\!]\sigma < [\![y]\!]\sigma = \sigma(x) < \sigma(y) = 4 < 5 = \mathsf{true}$$

The rest of the semantics is presented in the next section.

2.2 Instrumented semantics

The semantics for system and processes is defined by instrumented operational semantics, describing how each individual transition makes up the whole computation. In general, a *transition relation* in the operational semantics has the form

$$\underbrace{\frac{\text{premises}}{\text{configuration} \rightarrow \text{configuration}}}_{judgement} \;\text{conditions}$$

If there are no premises, the transition relation is called an *axiom* and the line is omitted.

The idea of the semantics is to record the information flow for each computational step (*transition*) in a system. Take e.g. the process $l : y := x$ with the security principal $\mathcal{S}(l) = s$. Here there clearly are an information flow from the variable x to the variable y. Recall that the security principals $\mathbf{Pr}$ are used in the policies to denote who owns, who can influence and who can read information. All the security principals must therefore monitor this flow to verify whether y is allowed to influenced by x or if x is allowed to be read. In this example at least the security principal s observes the assignment of the value of variable x to the variable y, a fact recorded as the *flow* (x, s, y).

To capture this the semantics is split in two, with one part recording information flows on the system level and the other recording information flows on the process level. The *system flows* are information flows from one process variable to another, possibly across different processes via a channel. The process level flows, or *extended flows*, describe process-local flows from a local variable, or constant to a local variable or channel variable.

In the semantics for systems the configurations are denoted

$$\langle l_1 : S_1 \parallel \cdots \parallel l_n : S_n, \sigma\rangle$$

which is a finite system of n parallel processes with statements S_i, and with a state $\sigma \in \textbf{State}$ mapping all variables $\textbf{Var}$ to their current values. The special case where $S_i = \mathsf{skip}$ for all $i \leq n$ is called the *terminal configuration*.

The fact that all process variables are disjoint ensures that any inter-process information flow can only occur using channels. In other words, since a variable x always belongs to exactly one process, its value $\sigma(x)$ stays local to that process.

The *judgement* for a distributed system in the transition relation has the form

$$\langle l_1 : S_1 \parallel \cdots \parallel l_n : S_n, \sigma\rangle \xRightarrow[U,D]{F} \langle l_1 : S_1' \parallel \cdots \parallel l_n : S_n', \sigma'\rangle$$

In the transition relation the process $l_i : S_i$ becomes $l_i : S_i'$, and σ becomes σ'.

The transition records three sets $F \subseteq \textbf{Var} \times \textbf{Pr} \times \textbf{Var}$, $U \subseteq \textbf{Var}$, $D \subseteq \textbf{Var}$. A triple $(x, s, y) \in F$ denotes a *system flow*, which records that there is a flow from the variable x to y monitored by the security principal s. A variable $x \in U$ records that the variable x is *used* in the transition relation. A variable $y \in D$ records at that the variable y is *defined* in the transition relation.

As an example, take the system

$$l_1 : y := x; ch!y \parallel l_2 : ch?z$$

where the processes share a security principal $\mathcal{S}(l_1) = \mathcal{S}(l_2) = s$. The system flows are $\{(x, s, y), (x, \mathsf{NSA}, y)\}$ and $\{(y, s, z), (y, \mathsf{NSA}, z)\}$ where it can be seen that the special principal NSA also monitors the flow. Looking only at the first execution the variable x is used, and y is defined. The other execution where l_1 sends y to l_2 over ch to z, the variable y is used and z is defined. The fact that U only records all used variables, and D only records all (re)defined variables are formalised in Section 2.2.3.

The functions $\mathsf{fst}(F)$ and $\mathsf{trd}(F)$ denotes the first and third element of the flows

$$\mathsf{fst}(\bar{x} \times \bar{s} \times \bar{y}) = \bar{x} \qquad\qquad \mathsf{trd}(\bar{x} \times \bar{s} \times \bar{y}) = \bar{y}$$

Here $\mathsf{fst}(F) \subseteq U$, but not necessarily that $\mathsf{fst}(F) = U$. This is seen in the case where a test is passed, e.g. for this particular instance

$$l : \mathsf{if}\ x > 3\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi} \tag{2.1}$$

then $x \notin \mathsf{fst}(F)$ but x is clearly used in the condition and therefore $x \in U$, yielding $\mathsf{fst}(F) \neq U$. Of course there is an implicit flow from x to S_i, but this will first be recorded in the next transition relation. Implicit flows are covered more carefully in Section 2.2.2. Furthermore it is the case that $\mathsf{trd}(F) \subseteq D$, and again not necessarily that $\mathsf{trd}(F) = D$. This is seen in assignments of constants

$$l : x := 3 \tag{2.2}$$

where $x \notin \mathsf{trd}(F)$ but x is defined and therefore $x \in D$.

Let $e \in \mathbf{Var}^+_\diamond = \mathbf{Var}^+ \cup \{\diamond\}$ denote an *extended variable*, and $E \subseteq \mathbf{Var}^+_\diamond \times \mathbf{Pr} \times \mathbf{Var}^+_\diamond$ denote a set of *extended flows*. It can record the same flows as system flows, but also from variables to channel variables and vice versa. The special element $\diamond$ will be used to ensure non-empty flows. If a variable is modified by constants then we must record that change and say that $\diamond \in \mathsf{fst}(E)$ (the variable is influenced in the process). Furthermore if a variable is used in a test it does not directly effect any values, but to record that the variable is read by the process we say that $\diamond \in \mathsf{trd}(E)$.

The judgement for a process in a transition relation has the form

$$\vdash_\ell \langle S; \sigma \rangle \xrightarrow[\alpha]{E} \langle S'; \sigma' \rangle$$

Here S and S' denotes the statements for the process ℓ, together with the states $\sigma', \sigma \in \mathbf{State}$, and the *action* α has the form

$$\alpha ::= \tau \quad | \quad ch!(v_1, \ldots, v_k) \quad | \quad ch?(v_1, \ldots, v_k)$$

The action α is given by the statement: If S is an output statement, the action is $ch!(v_1, \ldots, v_k)$, and similarly for input statements. Otherwise α is τ, denoting no inter-process communication.

The semantics ensures that flows inside the process from constants and variables to variables and channel variables are captured. The following process contains an example of each:

$$l : x := y;\ x := 4;\ ch!4;\ ch!x;\ ch?y;\ \mathsf{while}\ x > 4\ \mathsf{do\ skip\ od};\ \mathsf{while\ true\ do\ skip\ od};$$

Let $\mathcal{S}(l) = s$, the extended flows are then (y, s, x), $(\diamond, s, x)$, $(\diamond, s, \#_1)$, $(x, s, \#_1)$, $(\#_1, s, y)$, $(x, s, \diamond)$, $(\diamond, s, \diamond)$ plus the corresponding NSA flows.

From the extended flows E, the set of used variables can be extracted as $\mathsf{fst}(E) \cap \mathbf{Var}$, and likewise the set of defined variables as $\mathsf{trd}(E) \cap \mathbf{Var}$. The system flows for the local process are $E \cap (\mathbf{Var} \times \mathbf{Pr} \times \mathbf{Var})$.

2.2.1 Instrumented semantics for systems

The instrumented semantics for systems are shown in Rule Collection 2.1, which includes two transition relations. The first transition relation $[sys^\tau_{is}]$ contains the one transition of process l_i with the action τ. The system flows F, used U and defined D variables are then filtered from the extended flows E_i, by using the approach described just before, removing all the flows, used and defined variables containing $\diamond$.

The second transition relation $[sys^{ch}_{is}]$ contains transitions for two distinct processes l_i and l_j, where process l_i sends variables to l_j using a k-ary channel ch. The extended flows for process l_i and l_j would then be

$$E_i \supseteq \{(e_1, \mathcal{S}(l_i), \#_1), \ldots, (e_1, \mathcal{S}(l_i), \#_k)\}$$
$$E_j \supseteq \{(\#_1, \mathcal{S}(l_j), x_1), \ldots, (\#_k, \mathcal{S}(l_j), x_k)\}$$

If $s = \mathcal{S}(l_i) = \mathcal{S}(l_j)$, the pairwise combined flows of E_i and E_j

$$\{(e_1, s, x_1), \ldots, (e_k, s, x_k)\}$$

would tell that there is an information flow from the variables in process l_i to the variables in process l_j, observed by security principal s. The flows for this transition relation therefore uses the following combiner function, yielding the desired pairwise set:

$$E_1 \,\|\, E_2 = \left\{ (e,s,x) \;\middle|\; \begin{array}{r} (\exists i : (e,s,\#_i) \in E_1 \wedge (\#_i,s,x) \in E_2) \\ \vee\ ((e,s,x) \in E_2 \wedge e \in \mathbf{Var}) \end{array} \right\} \tag{2.3}$$

The combined flow $E_1 \,\|\, E_2$ could still contain $\diamond$, in the cases when a constant is sent instead of the value of a variable e.g. $ch!4$, and the appropriate intersection is used to remove invalid system flows in the transition relation. The other part of the combiner function ensures that all previously flows from process l_j is preserved.

$$[sys^{\tau}_{is}]\quad \frac{\vdash_{l_i} \langle S_i;\sigma\rangle \xrightarrow[\tau]{E_i} \langle S_i';\sigma'\rangle}{\langle l_1 : S_1 \,\|\, \cdots \,\|\, l_i : S_i \,\|\, \cdots \,\|\, l_n : S_n, \sigma\rangle \xRightarrow[U,D]{F} \langle l_1 : S_1 \,\|\, \cdots \,\|\, l_i : S_i' \,\|\, \cdots \,\|\, l_n : S_n, \sigma'\rangle}$$

$$\text{where}\quad \begin{array}{l} F = E_i \cap (\mathbf{Var} \times \mathbf{Pr} \times \mathbf{Var}) \\ U = \mathsf{fst}(E_i) \cap \mathbf{Var} \\ D = \mathsf{trd}(E_i) \cap \mathbf{Var} \end{array}$$

$$[sys^{ch}_{is}]\quad \frac{\vdash_{l_i} \langle S_i;\sigma\rangle \xrightarrow[ch!(v_1,\ldots,v_k)]{E_i} \langle S_i';\sigma'\rangle \qquad \vdash_{l_j} \langle S_j;\sigma'\rangle \xrightarrow[ch?(v_1,\ldots,v_k)]{E_j} \langle S_j';\sigma''\rangle}{\langle l_1 : S_1 \,\|\, \cdots \,\|\, l_i : S_i \,\|\, \cdots \,\|\, l_j : S_j \,\|\, \cdots \,\|\, l_n : S_n, \sigma\rangle \xRightarrow[U,D]{F} \langle l_1 : S_1 \,\|\, \cdots \,\|\, l_i : S_i' \,\|\, \cdots \,\|\, l_j : S_j' \,\|\, \cdots \,\|\, l_n : S_n, \sigma''\rangle}$$

$$\text{where}\quad \begin{array}{l} F = E_i \,\|\, E_j \cap (\mathbf{Var} \times \mathbf{Pr} \times \mathbf{Var}) \\ U = \mathsf{fst}(E_i \,\|\, E_j) \cap \mathbf{Var} \\ D = \mathsf{trd}(E_i \,\|\, E_j) \cap \mathbf{Var} \end{array}$$

$$\text{and } i \neq j$$

Rule Collection 2.1: Instrumented semantics for systems.

2.2.2 Instrumented semantics for processes

The instrumented semantics for processes is given in Rule Collection 2.2. The sets of flows are denoted using a simplified notation (juxtapositioning), such that $\{x\}$ is simply x, $\{\diamond\}$ is simply $\diamond$, and $\bar{a} \cup \{\diamond\}$ is denoted $\bar{a}\diamond$. As a reminder, $\bar{a}$ denotes the set of free variables in the arithmetic expression a and $\bar{b}$ denotes the set of free variables in the boolean expression b.

For all statements the information flow is observed by all security principals, unless the statement is one that bypasses the security restrictions. In statements with the bypass operation (†), the information flow should be observed by all security principals except the principal for the current process ℓ, i.e. $\mathbf{Pr} \setminus \mathcal{S}(\ell)$, or simply $\mathbf{Pr}_{\setminus \ell}$. The general form of the extended flows for the statements are therefore

$$e \times \mathbf{Pr} \times e' \qquad\qquad e \times \mathbf{Pr}_{\setminus \ell} \times e'$$

In $[ass_{is}]$, $[ass^{\dagger}_{is}]$, $[out_{is}]$, $[out^{\dagger}_{is}]$, $[in_{is}]$ there are *explicit flows*, where it is clear that the information is transferred from one variable to another. Furthermore there can also be *implicit flows*, where there indirectly is a data flow. Example:

$$x := 0; \mathsf{if}\ b\ \mathsf{then}\ x := 1\ \mathsf{else\ skip\ fi} \tag{2.4}$$

Depending on whether b is true or false, x becomes 1 or 0 respectively, and the value of x is therefore dependent on variable b, i.e. an implicit flow. The transitions rules for test statements ($[if^{\mathsf{tt}}_{is}]$, $[if^{\mathsf{ff}}_{is}]$, $[loop^{\mathsf{tt}}_{is}]$) gives rise to implicit flows into their inner statements. These flows are handled using the special statement $\lceil \bar{x} \rceil S$, where $\bar{x}$ is the set of implicit flow source variables. The rule $[loop^{\mathsf{ff}}_{is}]$ does therefore not use $\lceil \bar{x} \rceil S$ statement as the execution does not continue in the inner statement of the loop.

The implicit flows are appended to the extended flows in the following way

$$\lceil \bar{x} \rceil E = \{(y, s, e') \mid \exists (e, s, e') \in E : y \in \bar{x}e\} \tag{2.5}$$

where it is seen that flows from E are preserved, and flows from variables in $\bar{x}$ are added. The extended flows for the assignment inside the conditional branch in (2.4) will therefore be $\bar{b} \diamond \times \mathbf{Pr} \times x$.

As a reminder to ensure non-empty flows, $\diamond$ is added to first component of an extended flow when constants are used. In test statements $\diamond$ is added as a third component to reflect that variables are used. When there are no free variables in the test the first component of the extended flows also includes $\diamond$ (e.g. if the test only contains compositions of true and false).

2.2.3 Properties of the defined and used variables

The defined variables D in a system judgement only contains the variables that are (re)defined. All other variables $x \in \mathbf{Var} \setminus D$ stays the same in the transition relation. More formally this is specified as

Fact 2.1. *Assume* $\langle l_1 : S_1 \parallel \cdots \parallel l_n : S_n, \sigma \rangle \xRightarrow[U,D]{F} \langle l_1 : S'_1 \parallel \cdots \parallel l_n : S'_n, \sigma' \rangle$, *then* $\sigma(x) = \sigma'(x)$ *for all* $x \in \mathbf{Var} \setminus D$.

Proof idea. By induction on the instrumented semantic, using similar results for processes. $\square$

The used variables U in a system judgement only contains the variables that are used. All other variables $x \in \mathbf{Var} \setminus U$ are therefore not in any statement of S_i and can hence not effect any variable, e.g. x does not influence the variables in D. This is formalised using the same system twice with two different states σ_a and σ_b where only the used variables are equivalent in the states. The conclusion is then that all the defined and used variables $U \cup D$ are equivalent in σ'_a and σ'_b. More formally

$$[skip_{is}] \quad \vdash_\ell \langle \mathsf{skip};\sigma\rangle \xrightarrow[\tau]{E} \langle \mathsf{skip};\sigma\rangle \qquad \text{if } E = \diamond \times \mathbf{Pr} \times \diamond$$

$$[ass_{is}] \quad \vdash_\ell \langle x := a;\sigma\rangle \xrightarrow[\tau]{E} \langle \mathsf{skip};\sigma[x \mapsto [\![a]\!]\sigma]\rangle \qquad \text{if } E = \bar{a}\diamond \times \mathbf{Pr} \times x$$

$$[ass^{\dagger}_{is}] \quad \vdash_\ell \langle x :=^{\dagger} a;\sigma\rangle \xrightarrow[\tau]{E} \langle \mathsf{skip};\sigma[x \mapsto [\![a]\!]\sigma]\rangle \qquad \text{if } E = \bar{a}\diamond \times \mathbf{Pr}_{\backslash\ell} \times x$$

$$[comp^1_{is}] \quad \frac{\vdash_\ell \langle S_1;\sigma\rangle \xrightarrow[\alpha]{E} \langle S_1';\sigma'\rangle}{\vdash_\ell \langle S_1;S_2;\sigma\rangle \xrightarrow[\alpha]{E} \langle S_1';S_2;\sigma'\rangle} \text{ if } S_1' \neq \mathsf{skip} \qquad [comp^2_{is}] \quad \frac{\vdash_\ell \langle S_1;\sigma\rangle \xrightarrow[\alpha]{E} \langle \mathsf{skip};\sigma'\rangle}{\vdash_\ell \langle S_1;S_2;\sigma\rangle \xrightarrow[\alpha]{E} \langle S_2;\sigma'\rangle}$$

$$[if^{\mathsf{tt}}_{is}] \quad \vdash_\ell \langle \mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi};\sigma\rangle \xrightarrow[\tau]{E} \langle \lceil\bar{b}\rceil S_1;\sigma\rangle \qquad \begin{array}{l}\text{if } [\![b]\!]\sigma = \mathsf{true}\\ \text{and } E = \bar{b}\diamond \times \mathbf{Pr} \times \diamond\end{array}$$

$$[if^{\mathsf{ff}}_{is}] \quad \vdash_\ell \langle \mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi};\sigma\rangle \xrightarrow[\tau]{E} \langle \lceil\bar{b}\rceil S_2;\sigma\rangle \qquad \begin{array}{l}\text{if } [\![b]\!]\sigma = \mathsf{false}\\ \text{and } E = \bar{b}\diamond \times \mathbf{Pr} \times \diamond\end{array}$$

$$[loop^{\mathsf{tt}}_{is}] \quad \vdash_\ell \langle \mathsf{while}\ b\ \mathsf{do}\ S\ \mathsf{od};\sigma\rangle \xrightarrow[\tau]{E} \langle \lceil\bar{b}\rceil S;\mathsf{while}\ b\ \mathsf{do}\ S\ \mathsf{od};\sigma\rangle \qquad \begin{array}{l}\text{if } [\![b]\!]\sigma = \mathsf{true}\\ \text{and } E = \bar{b}\diamond \times \mathbf{Pr} \times \diamond\end{array}$$

$$[loop^{\mathsf{tt}}_{is}] \quad \vdash_\ell \langle \mathsf{while}\ b\ \mathsf{do}\ S\ \mathsf{od};\sigma\rangle \xrightarrow[\tau]{E} \langle \mathsf{skip};\sigma\rangle \qquad \begin{array}{l}\text{if } [\![b]\!]\sigma = \mathsf{false}\\ \text{and } E = \bar{b}\diamond \times \mathbf{Pr} \times \diamond\end{array}$$

$$[out_{is}] \quad \vdash_\ell \langle ch!(a_1,\dots,a_k);\sigma\rangle \xrightarrow[ch!(v_1,\dots,v_k)]{E} \langle \mathsf{skip};\sigma\rangle \qquad \begin{array}{l}\text{if } \forall i \leq k : v_i = [\![a_i]\!]\sigma\\ \text{and } E = \bigcup\limits_{i\leq k} \bar{a}_i \diamond \times \mathbf{Pr} \times \#_i\end{array}$$

$$[out^{\dagger}_{is}] \quad \vdash_\ell \langle ch!^{\dagger}(a_1,\dots,a_k);\sigma\rangle \xrightarrow[ch!(v_1,\dots,v_k)]{E} \langle \mathsf{skip};\sigma\rangle \qquad \begin{array}{l}\text{if } \forall i \leq k : v_i = [\![a_i]\!]\sigma\\ \text{and } E = \bigcup\limits_{i\leq k} \bar{a}_i \diamond \times \mathbf{Pr}_{\backslash\ell} \times \#_i\end{array}$$

$$[in_{is}] \quad \vdash_\ell \langle ch?(x_1,\dots,x_k);\sigma\rangle \xrightarrow[ch?(v_1,\dots,v_k)]{E} \langle \mathsf{skip};\sigma[(x_i \mapsto v_i)_{i\leq k}]\rangle$$
$$\text{if } E = \bigcup_{i\leq k} \#_i \times \mathbf{Pr} \times x_i$$

$$[cho_{is}] \quad \frac{\vdash_\ell \langle S_i;\sigma\rangle \xrightarrow[\alpha]{E} \langle S_i';\sigma'\rangle}{\vdash_\ell \langle (((S_1) \oplus (S_2));\sigma\rangle \xrightarrow[\alpha]{E} \langle S_i';\sigma'\rangle} \text{ for } i = 1,2$$

$$[impl^1_{is}] \quad \frac{\vdash_\ell \langle S;\sigma\rangle \xrightarrow[\alpha]{E} \langle S';\sigma'\rangle}{\vdash_\ell \langle \lceil\bar{x}\rceil S;\sigma\rangle \xrightarrow[\alpha]{\lceil\bar{x}\rceil E} \langle \lceil\bar{x}\rceil S';\sigma'\rangle} \text{ if } S' \neq \mathsf{skip} \qquad [impl^2_{is}] \quad \frac{\vdash_\ell \langle S;\sigma\rangle \xrightarrow[\alpha]{E} \langle \mathsf{skip};\sigma'\rangle}{\vdash_\ell \langle \lceil\bar{x}\rceil S;\sigma\rangle \xrightarrow[\alpha]{\lceil\bar{x}\rceil E} \langle \mathsf{skip};\sigma'\rangle}$$

Rule Collection 2.2: Instrumented semantics for processes.

Fact 2.2. *Assume* $\langle l_1 : S_1 \parallel \cdots \parallel l_n : S_n, \sigma_a\rangle \xrightarrow[U,D]{F} \langle l_1 : S'_1 \parallel \cdots \parallel l_n : S'_n, \sigma'_a\rangle$ *and* $\sigma_a(x) = \sigma_b(x)$ *for all* $x \in U$*. Then there exist* σ'_b *such that*

$$\langle l_1 : S_1 \parallel \cdots \parallel l_n : S_n, \sigma_b\rangle \xrightarrow[U,D]{F} \langle l_1 : S'_1 \parallel \cdots \parallel l_n : S'_n, \sigma'_b\rangle$$

and furthermore $\sigma'_a(y) = \sigma'_b(y)$ *for all* $y \in U \cup D$*.*

Proof idea. By induction on the instrumented semantic, using similar results for processes and Fact 2.1. □

2.3 Policies

Policies are used to describe the security restrictions for a system. They therefore includes information about who owns the information and who is allowed to influence or read it.

2.3.1 Syntax

The syntax for the policies $P \in \mathbf{Pol}$ is

$$\begin{array}{rcl} P & ::= & \{\bar{u} : \bar{o} \leftarrow \bar{s}\} \quad | \quad \{\bar{u} : \bar{o} \rightarrow \bar{s}\} \quad | \quad (\varphi \Rightarrow P) \quad | \quad P^1 \bullet P^2 \\ \varphi & ::= & \mathsf{true} \quad | \quad \mathsf{false} \quad | \quad \neg\varphi \quad | \quad \varphi_1 \land \varphi_2 \quad | \quad \varphi_1 \lor \varphi_2 \\ & | & a_1^+ \; rel \; a_2^+ \quad | \quad (\exists \bar{u} : (\varphi)) \\ a^+ & ::= & n \quad | \quad x \quad | \quad \# \quad | \quad a_1^+ \; op \; a_2^+ \end{array}$$

Here $\bar{u} \subseteq \mathbf{Var}^+$ is a set of variables or channel variables, and $\bar{o}, \bar{s} \subseteq \mathbf{Pr}$ is sets of security principals. The policy $\{\bar{u} : \bar{o} \leftarrow \bar{s}\}$ is an *influencer policy*, specifying the variables, owners and influencers. The owners, specified as $\bar{o}$, allows the security principals $\bar{s}$ to influence the values of the variables in $\bar{u}$. In the policy $\{x, y : s_1 \leftarrow s_1, s_2\}$, the principal s_1 is the owner of the variables x and y, and s_1 allows s_1 and s_2 to influence the value of x and y. The influencer policies therefore concerns the integrity of the information.

The policy $\{\bar{u} : \bar{o} \rightarrow \bar{s}\}$ is a *reader policy*, which specifies variables, owners and readers. The owners $\bar{o}$ allows the security principals $\bar{s}$ to read the values of the variables in $\bar{u}$. In the policy $\{x, y : s_1 \rightarrow s_1, s_2\}$, the principal s_1 is the owner of x, y and specifies that only s_1 and s_2 can read the value of x and y. The reader policies therefore concerns the confidentiality of the information.

If the sets $\bar{u}$, $\bar{o}$ and $\bar{s}$ from influencer and reader policies are empty these are denoted with ϵ, e.g. if $\bar{o}$ allows that no one may read the variables $\bar{x}$ this is denoted $\{\bar{x} : \bar{o} \rightarrow \epsilon\}$. Conversely, $\star$ can be used in $\bar{u}$, $\bar{o}$ and $\bar{s}$ to describe the full set. For the variables or channel variables $\star = \mathbf{Var}^+$, and for security principals $\star = \mathbf{Pr}$. For the variables local to a process ℓ the full set is denoted $\star_\ell = \mathbf{Var}_\ell$, and for all channel variables likewise $\star_\# = \mathbf{Var}_\#$. The policy $\{\bar{x} : \bar{o} \rightarrow \star\}$ therefore denotes that $\bar{o}$ allows all security principals to read the variables in $\bar{x}$.

The policy $(\varphi \Rightarrow P)$ is a *conditional policy* and only if the predicate φ holds, P is imposed. Just as $\bar{b}$ denotes the free variables in a boolean expression, $\bar{\varphi}$ denotes the free variables in a predicate. The operators *rel* and *op* are the same as in boolean and arithmetic expressions. The predicate φ is an extension of the boolean predicate where the arithmetic expressions are replaced with an extended arithmetic expressions a^+ which also includes channel variables $\# \in \mathbf{Var}_{\#}$. Moreover it is extended with the existential quantifier $(\exists \bar{u} : (\varphi))$ denoting that there exists values for the variables $\bar{u} = u_1,..,u_n$ such that φ holds. The variables $\bar{u}$ are therefore bound to φ, and are thus no longer part of the free variables of φ. Substituting the variable x in φ with an arithmetic expression a is denoted $\varphi[a/x]$. All the free occurrences of the variable x in φ is then replaced with the arithmetic expression a.

At last the policy $P^1 \bullet P^2$ is the combination of policy P^1 and P^2, and expresses that both policies should be considered.

Satisfaction of predicates

In a state $\sigma \in \mathbf{State}^+$ the satisfaction of φ is denoted $\sigma \models \varphi$ (σ models φ). All the free variables of φ is looked up in σ, and the condition can then be evaluated. If $\sigma(x)$ is not defined and $x \in \bar{\varphi}$ then $\sigma \not\models \varphi$.

This leads to the following fact, that new mappings in σ, reflect substitutions in φ and vice versa.

Fact 2.3. *$\sigma[x \mapsto [\![a]\!]\sigma] \models \varphi$ iff $\sigma \models \varphi[a/x]$.* *(informal proof in Appendix A.1)*

2.3.2 Variables and owners

The *free variables* for policies $\mathsf{fv}(P)$ are defined as

$$\begin{aligned}
\mathsf{fv}(\{\bar{u} : \bar{o} \leftarrow \bar{s}\}) &= \bar{u} \\
\mathsf{fv}(\{\bar{u} : \bar{o} \rightarrow \bar{s}\}) &= \bar{u} \\
\mathsf{fv}((\varphi \Rightarrow P)) &= \bar{\varphi} \cup \mathsf{fv}(P) \\
\mathsf{fv}(P^1 \bullet P^2) &= \mathsf{fv}(P^1) \cup \mathsf{fv}(P^2)
\end{aligned}$$

The *influencer variables* $\mathsf{iv}(P)$ are all the variables that are used in influencer policies in P and defined as

$$\begin{aligned}
\mathsf{iv}(\{\bar{u} : \bar{o} \leftarrow \bar{s}\}) &= \bar{u} \\
\mathsf{iv}(\{\bar{u} : \bar{o} \rightarrow \bar{s}\}) &= \epsilon \\
\mathsf{iv}((\varphi \Rightarrow P)) &= \mathsf{iv}(P) \\
\mathsf{iv}(P^1 \bullet P^2) &= \mathsf{iv}(P^1) \cup \mathsf{iv}(P^2)
\end{aligned}$$

and likewise the *reader variables* $\mathsf{rv}(P)$ for the reader policies

$$\begin{aligned}
\mathsf{rv}(\{\bar{u} : \bar{o} \leftarrow \bar{s}\}) &= \epsilon \\
\mathsf{rv}(\{\bar{u} : \bar{o} \rightarrow \bar{s}\}) &= \bar{u} \\
\mathsf{rv}((\varphi \Rightarrow P)) &= \mathsf{rv}(P) \\
\mathsf{rv}(P^1 \bullet P^2) &= \mathsf{rv}(P^1) \cup \mathsf{rv}(P^2)
\end{aligned}$$

Furthermore the *influencer owners* $\mathsf{io}(P, y)$ are the owners from influencer policies for the variable y and defined as

$$\begin{aligned}
\mathsf{io}(\{\bar{u} : \bar{o} \leftarrow \bar{s}\}, y) &= \begin{cases} \bar{o} & \text{if } y \in \bar{u} \\ \epsilon & \text{otherwise} \end{cases} \\
\mathsf{io}(\{\bar{u} : \bar{o} \rightarrow \bar{s}\}, y) &= \epsilon \\
\mathsf{io}((\varphi \Rightarrow P)\,, y) &= \mathsf{io}(P, y) \\
\mathsf{io}(P^1 \bullet P^2, y) &= \mathsf{io}(P^1, y) \cup \mathsf{io}(P^2, y)
\end{aligned}$$

and likewise for *reader owners* $\mathsf{ro}(P, y)$

$$\begin{aligned}
\mathsf{ro}(\{\bar{u} : \bar{o} \leftarrow \bar{s}\}, y) &= \epsilon \\
\mathsf{ro}(\{\bar{u} : \bar{o} \rightarrow \bar{s}\}, y) &= \begin{cases} \bar{o} & \text{if } y \in \bar{u} \\ \epsilon & \text{otherwise} \end{cases} \\
\mathsf{ro}((\varphi \Rightarrow P)\,, y) &= \mathsf{ro}(P, y) \\
\mathsf{ro}(P^1 \bullet P^2, y) &= \mathsf{ro}(P^1, y) \cup \mathsf{ro}(P^2, y)
\end{aligned}$$

All these definitions are used in Section 2.3.7 to describe proper policies for the system.

2.3.3 Influencers and readers

For a policy P and a state $\sigma \in \mathbf{State}^+$ the set $\mathsf{Infl}\,(P, \sigma, q, y)$ denotes the set of security principals who can influence the variable $y \in \mathbf{Var}^+$, where $q \in \mathbf{Pr}$ is an owner for y. Likewise the set $\mathsf{Read}\,(P, \sigma, q, y)$ the set of security principals who can read the variable y where q is owns y. These sets are denoted respectively as the *influencer set* and the *reader set* or shortly as *influencers* and *readers*. The influencer set is defined as

$$\mathsf{Infl}\,(\{\bar{u} : \bar{o} \leftarrow \bar{s}\}, \sigma, q, y) = \begin{cases} \bar{s} & \text{if } y \in \bar{u} \wedge q \in \bar{o} \\ \epsilon & \text{otherwise} \end{cases} \tag{2.6}$$

$$\mathsf{Infl}\,(\{\bar{u} : \bar{o} \rightarrow \bar{s}\}, \sigma, q, y) = \epsilon \tag{2.7}$$

$$\mathsf{Infl}\,((\varphi \Rightarrow P)\,, \sigma, q, y) = \begin{cases} \mathsf{Infl}\,(P, \sigma, q, y) & \text{if } \sigma \models \varphi \\ \epsilon & \text{otherwise} \end{cases} \tag{2.8}$$

$$\mathsf{Infl}\left(P^1 \bullet P^2, \sigma, q, y\right) = \mathsf{Infl}\left(P^1, \sigma, q, y\right) \cup \mathsf{Infl}\left(P^2, \sigma, q, y\right) \tag{2.9}$$

Similar is the definition for the reader set, which is the dual of the influencer set. Here intersection is used instead of union and the full set of security principals **Pr** instead of ϵ

$$\mathsf{Read}\left(\{\bar{u} : \bar{o} \leftarrow \bar{s}\}, \sigma, q, y\right) = \mathbf{Pr} \tag{2.10}$$

$$\mathsf{Read}\left(\{\bar{u} : \bar{o} \rightarrow \bar{s}\}, \sigma, q, y\right) = \begin{cases} \bar{s} & \text{if } y \in \bar{u} \land q \in \bar{o} \\ \mathbf{Pr} & \text{otherwise} \end{cases} \tag{2.11}$$

$$\mathsf{Read}\left((\varphi \Rightarrow P), \sigma, q, y\right) = \begin{cases} \mathsf{Read}\left(P, \sigma, q, y\right) & \text{if } \sigma \models \varphi \\ \mathbf{Pr} & \text{otherwise} \end{cases} \tag{2.12}$$

$$\mathsf{Read}\left(P^1 \bullet P^2, \sigma, q, y\right) = \mathsf{Read}\left(P^1, \sigma, q, y\right) \cap \mathsf{Read}\left(P^2, \sigma, q, y\right) \tag{2.13}$$

Consider the following influencer policies

$$P = \{s_1 : s_1, s_2 \leftarrow y\} \bullet \{s_1 : s_1, s_3 \leftarrow y\}$$

The influencers for the variable y according to s_1 is

$$\mathsf{Infl}\left(P, \sigma, s_1, y\right) = \{s_1, s_2\} \cup \{s_1, s_3\} = \{s_1, s_2, s_3\}$$

Despite that the first policy only allows s_1 and s_2 to influence y s_3 is also allowed as an influencer. It is therefore only necessary to an influencer in one influencer policy to count as an influencer for the whole policy. Consider the following reader policies

$$P = \{s_1 : s_1, s_2 \rightarrow y\} \bullet \{s_1 : s_1, s_3 \rightarrow y\}$$

The readers for the variable y according to s_1 is then

$$\mathsf{Read}\left(P, \sigma, s_1, y\right) = \{s_1, s_2\} \cap \{s_1, s_3\} = \{s_1\}$$

Conversely all reader policies in a policy must agree on who may read the information and are thereby considered a reader.

It can here be seen that the influencers and readers represents some of the same ideas from DLM as shown in Section 1.2, and that they are dual of each others.

2.3.4 Preorder

For the influencer and reader sets we can now define the ordering $\sqsubseteq$ between two policies

$$\begin{aligned} P^1 \sqsubseteq P^2 \text{ iff } & \forall \sigma, q, y : \mathsf{Infl}\left(P^1, \sigma, q, y\right) \subseteq \mathsf{Infl}\left(P^2, \sigma, q, y\right) \land \\ & \forall \sigma, q, y : \mathsf{Read}\left(P^1, \sigma, q, y\right) \supseteq \mathsf{Read}\left(P^2, \sigma, q, y\right) \end{aligned} \tag{2.14}$$

Fact 2.4. *The ordering $\sqsubseteq$ is a preorder, because it is reflexive and transitive.*
(proven by looking at influencers and readers in Appendix A.2)

A partial order is a preorder which is also antisymmetric. This is not the case for the preorder $\sqsubseteq$. Having $P^1 = \{\epsilon : \bar{o} \leftarrow \bar{s}\}$ and $P^2 = \{\bar{u} : \epsilon \leftarrow \bar{s}\}$, it holds that

$P^1 \sqsubseteq P^2$ and $P^2 \sqsubseteq P^1$, but the policies are different e.g. $P^1 \neq P^2$. $\sqsubseteq$ is therefore not antisymmetric.

For the preorder an *equivalence relation* $\equiv$ is established instead

$$P^1 \equiv P^2 \text{ iff } P^1 \sqsubseteq P^2 \wedge P^2 \sqsubseteq P^1 \tag{2.15}$$

By using the example from before, the policies $P^1 = \{\epsilon : \bar{o} \leftarrow \bar{s}\}$ and $P^2 = \{\bar{u} : \epsilon \leftarrow \bar{s}\}$, are equivalent i.e. $P^1 \equiv P^2$, because the influencer sets are the same for both policies, and the reader set are the same for both policies.

2.3.5 Bottom and top policy

The bottom element of **Pol** is denoted $\{\bot\}$ and has the property that $\{\bot\} \sqsubseteq P$ for all $P \in \mathbf{Pol}$. The top element of **Pol** is denoted $\{\top\}$ and has the property that $P \sqsubseteq \{\top\}$ for all $P \in \mathbf{Pol}$.

Due to (2.14) the policy $\{\bot\}$ should represent a policy where no one is allowed as influencers and all security principals are allowed as readers. Consider therefore following

$$\{\epsilon : \epsilon \leftarrow \epsilon\} \bullet \{\star : \star \rightarrow \star\} \sqsubseteq P \tag{2.16}$$

On the left-hand side the influencers are always ϵ and the readers are $\star$; the order is therefore true for all policies P, and the left-hand side is equivalent to $\{\bot\}$. In fact using the equivalence relation it can be seen that there are multiple ways of denoting the bottom element. Let $\epsilon \subseteq \bar{u} \subseteq \star$, $\epsilon \subseteq \bar{o} \subseteq \star$, $\epsilon \subseteq \bar{s} \subseteq \star$ then

$$\begin{align}
\{\bot\} &\equiv \{\epsilon : \bar{o} \leftarrow \bar{s}\} \equiv \{\epsilon : \bar{o} \rightarrow \bar{s}\} \tag{2.17}\\
&\equiv \{\bar{u} : \epsilon \leftarrow \bar{s}\} \equiv \{\bar{u} : \epsilon \rightarrow \bar{s}\} \tag{2.18}\\
&\equiv \{\bar{u} : \bar{o} \leftarrow \epsilon\} \tag{2.19}\\
&\equiv \{\star : \star \rightarrow \star\} \tag{2.20}
\end{align}$$

At last it is seen that $\{\bot\} \not\equiv \{\bar{u} : \bar{o} \rightarrow \epsilon\}$ if $\bar{u}, \bar{o} \neq \epsilon$, which makes sense looking at the reader policy for $\{\top\}$ defined below. The last inequality together with all the equivalences are proven in Appendix A.3, by looking at the influencer set $\mathsf{Infl}\,(P, \sigma, q, y)$ and reader set $\mathsf{Read}\,(P, \sigma, q, y)$ for all possible values for σ, q and y.

Conversely, for $\{\top\}$, all security principals should be allowed to influence and no security principal should be allowed to read. Consider therefore following

$$P \sqsubseteq \{\star : \star \leftarrow \star\} \bullet \{\star : \star \rightarrow \epsilon\} \tag{2.21}$$

On the right-hand side the influencers are $\star$ and the readers are ϵ, which makes the right-hand side equivalent to $\{\top\}$. There is no other way of writing the top element because non of the elements in (2.21) can be changed and still have the same property for all P.

From here on $\{\bot\}$ will simply be denoted $\{\}$ and $\{\top\}$ will explicitly be denoted $\{\star : \star \leftarrow \star\} \bullet \{\star : \star \rightarrow \epsilon\}$.

2.3.6 Bounded join-semilattice

Definition 2.1. Let L be a set with a partial ordering $\preceq$, and $\sqcup$ be an upper bound operator with respect to $\preceq$; i.e. having $l_1, l_2 \in L$ then $l_1 \preceq l_1 \sqcup l_2$ and $l_2 \preceq l_1 \sqcup l_2$. If for all $l_1, l_2 \in L$ there exists a least upper bound $l_3 \in L$ such that $l_1 \sqcup l_2 \preceq l_3$, then $(L, \sqcup, \preceq)$ is a *join-semilattice.* A join-semilattice is called *bounded* if it contains an identity element $\perp \in L$, i.e. the least element, such that $l \sqcup \perp = l$ for all $l \in L$.

Fact 2.5. $(\mathbf{Pol}, \bullet, \sqsubseteq_\equiv)$ *is a bounded join-semilattice.*

Proof. The ordering $\sqsubseteq$ is a preorder, and using $\equiv$ as the equivalence relation for $\sqsubseteq$ (denoted $\sqsubseteq_\equiv$) yields a partial ordering, because it then is reflexive, transitive and antisymmetric. The operator $\bullet$ is an upper bound operator on **Pol** seen by the following

$$P^1 \sqsubseteq P^1 \bullet P^2 \qquad\qquad P^2 \sqsubseteq P^1 \bullet P^2 \tag{2.22}$$

verified by the influencer and reader sets for $i \in 1, 2$, and all σ, q and y:

$$\begin{aligned} \mathsf{Infl}\left(P^i, \sigma, q, y\right) &\subseteq \mathsf{Infl}\left(P^1, \sigma, q, y\right) \cup \mathsf{Infl}\left(P^2, \sigma, q, y\right) \\ \mathsf{Read}\left(P^i, \sigma, q, y\right) &\supseteq \mathsf{Read}\left(P^1, \sigma, q, y\right) \cap \mathsf{Read}\left(P^2, \sigma, q, y\right) \end{aligned}$$

Furthermore if $P^1 \sqsubseteq P^3$ and $P^2 \sqsubseteq P^3$ then it follows for all σ, q and y that

$$\begin{aligned} &\mathsf{Infl}\left(P^1, \sigma, q, y\right) \subseteq \mathsf{Infl}\left(P^3, \sigma, q, y\right) \wedge \mathsf{Infl}\left(P^2, \sigma, q, y\right) \subseteq \mathsf{Infl}\left(P^3, \sigma, q, y\right) \\ &\quad\Leftrightarrow \mathsf{Infl}\left(P^1, \sigma, q, y\right) \cup \mathsf{Infl}\left(P^2, \sigma, q, y\right) \subseteq \mathsf{Infl}\left(P^3, \sigma, q, y\right) \\ &\mathsf{Read}\left(P^1, \sigma, q, y\right) \supseteq \mathsf{Read}\left(P^3, \sigma, q, y\right) \wedge \mathsf{Read}\left(P^2, \sigma, q, y\right) \supseteq \mathsf{Read}\left(P^3, \sigma, q, y\right) \\ &\quad\Leftrightarrow \mathsf{Read}\left(P^1, \sigma, q, y\right) \cap \mathsf{Read}\left(P^2, \sigma, q, y\right) \supseteq \mathsf{Read}\left(P^3, \sigma, q, y\right) \end{aligned}$$

which gives that

$$P_1 \bullet P_2 \sqsubseteq P_3 \quad \text{iff} \quad P_1 \sqsubseteq P_3 \wedge P_2 \sqsubseteq P_3 \tag{2.23}$$

verifying that $\bullet$ is a least upper bound operator, and that $(\mathbf{Pol}, \bullet, \sqsubseteq_\equiv)$ is a join-semilattice.

By definition of the $\bullet$ operator the policies are closed $\forall P^1, P^2 \in \mathbf{Pol} : P^1 \bullet P^2 \in \mathbf{Pol}$. Furthermore for $\bullet$ the policies are idempotent, commutative and associative:

$$P \bullet P \equiv P \tag{2.24}$$

$$P^1 \bullet P^2 \equiv P^2 \bullet P^1 \tag{2.25}$$

$$(P^1 \bullet P^2) \bullet P^3 \equiv P^1 \bullet (P^2 \bullet P^3) \tag{2.26}$$

Appendix A.4 shows these equivalences, exploiting that the influencer and reader sets uses basic set operations $\cup$ and $\cap$, which also are idempotent, commutative and associative.

The policy $\{\}$ is the bottom element of **Pol** and the top element is $\{\star : \star \leftarrow \star\} \bullet \{\star : \star \rightarrow \epsilon\}$. We therefore have that

$$\{\} \bullet P \equiv P \tag{2.27}$$

$$\{\star : \star \leftarrow \star\} \bullet \{\star : \star \rightarrow \epsilon\} \bullet P \equiv \{\star : \star \leftarrow \star\} \bullet \{\star : \star \rightarrow \epsilon\} \tag{2.28}$$

making the join-semilattice bounded. □

Equivalences

Influencer and reader policies underly some distributive laws, as seen in the following equivalences:

$$\{\bar{u}_1\bar{u}_2 : \bar{o} \leftarrow \bar{s}\} \equiv \{\bar{u}_1 : \bar{o} \leftarrow \bar{s}\} \bullet \{\bar{u}_2 : \bar{o} \leftarrow \bar{s}\} \tag{2.29}$$
$$\{\bar{u}_1\bar{u}_2 : \bar{o} \rightarrow \bar{s}\} \equiv \{\bar{u}_1 : \bar{o} \rightarrow \bar{s}\} \bullet \{\bar{u}_2 : \bar{o} \rightarrow \bar{s}\} \tag{2.30}$$
$$\{\bar{u} : \bar{o}_1\bar{o}_2 \leftarrow \bar{s}\} \equiv \{\bar{u} : \bar{o}_1 \leftarrow \bar{s}\} \bullet \{\bar{u} : \bar{o}_2 \leftarrow \bar{s}\} \tag{2.31}$$
$$\{\bar{u} : \bar{o}_1\bar{o}_2 \rightarrow \bar{s}\} \equiv \{\bar{u} : \bar{o}_1 \rightarrow \bar{s}\} \bullet \{\bar{u} : \bar{o}_2 \rightarrow \bar{s}\} \tag{2.32}$$
$$\{\bar{u} : \bar{o} \leftarrow \bar{s}_1\bar{s}_2\} \equiv \{\bar{u} : \bar{o} \leftarrow \bar{s}_1\} \bullet \{\bar{u} : \bar{o} \leftarrow \bar{s}_2\} \tag{2.33}$$

The missing distribution law for reader policies does not hold $\{\bar{u} : \bar{o} \rightarrow \bar{s}_1\bar{s}_2\} \not\equiv \{\bar{u} : \bar{o} \rightarrow \bar{s}_1\} \bullet \{\bar{u} : \bar{o} \rightarrow \bar{s}_2\}$. This is seen in the reader set where there for one $y \in \bar{u}$, $q \in \bar{o}$ and all s_1, s_2

$$\begin{aligned}
&\mathsf{Read}\left(\{\bar{u} : \bar{o} \rightarrow \bar{s}_1\bar{s}_2\}, \sigma, q, y\right) = \bar{s}_1\bar{s}_2\\
&\mathsf{Read}\left(\{\bar{u} : \bar{o} \rightarrow \bar{s}_1\} \bullet \{\bar{u} : \bar{o} \rightarrow \bar{s}_2\}, \sigma, q, y\right)\\
&\qquad = \mathsf{Read}\left(\{\bar{u} : \bar{o} \rightarrow \bar{s}_1\}, \sigma, q, y\right) \cap \mathsf{Read}\left(\{\bar{u} : \bar{o} \rightarrow \bar{s}_2\}, \sigma, q, y\right)\\
&\qquad = \bar{s}_1 \cap \bar{s}_2
\end{aligned}$$

Indeed $\bar{s}_1\bar{s}_2 \supseteq \bar{s}_1 \cap \bar{s}_2$ but $\bar{s}_1 \cap \bar{s}_2 \not\supseteq \bar{s}_1\bar{s}_2$ for all s_1, s_2, this equivalence does not hold.

For conditional policies there are the following equivalencies

$$\left(\varphi \Rightarrow P^1 \bullet P^2\right) \equiv \left(\varphi \Rightarrow P^1\right) \bullet \left(\varphi \Rightarrow P^2\right) \tag{2.34}$$
$$(\varphi_1 \wedge \varphi_2 \Rightarrow P) \equiv (\varphi_1 \Rightarrow (\varphi_2 \Rightarrow P)) \tag{2.35}$$
$$(\varphi \Rightarrow \{\}) \equiv \{\} \tag{2.36}$$
$$(\mathsf{true} \Rightarrow P) \equiv P \tag{2.37}$$
$$(\mathsf{false} \Rightarrow P) \equiv \{\} \tag{2.38}$$

Correctness of all the equivalences are shown in Appendix A.5.

2.3.7 Localised policies and system policies

All the policies for a distributed system $Sys = l_1 : S_1 \parallel \cdots \parallel l_n : S_n$ which contains m channels is denoted

$$P_{Sys} = P_{l_1} \bullet \cdots \bullet P_{l_n} \bullet P_{ch_1} \bullet \cdots \bullet P_{ch_m}$$

where P_ℓ is a *process policy* for the process with label ℓ and P_{ch} is a *channel policy* for the channel named ch. All the process policies in the system are denoted

$$P_\bullet = P_{l_1} \bullet \cdots \bullet P_{l_n}$$

All the process policies P_ℓ and channel policies P_{ch} must be *localised*.

Definition 2.2. A process policy P_ℓ is *localised* if

- $\mathsf{fv}(P_\ell) \subseteq \mathbf{Var}_\ell$,
- $(\forall u \in \mathsf{iv}(P_\ell) : (\mathcal{S}(\ell) \in \mathsf{io}(P_\ell, u)))$, and
- $(\forall u \in \mathsf{rv}(P_\ell) : (\mathcal{S}(\ell) \in \mathsf{ro}(P_\ell, u)))$.

First, the process policies may only contain variables from their own domain, such that P_ℓ only contains variables from $\mathbf{Var}_\ell$. For the system $l_1 : x := 4 \parallel l_2 : y := 2$ where $\mathcal{S}(l_i) = s_i$ for $i \in 1, 2$ the policy $(y > 4 \Rightarrow \{x : s_1 \leftarrow s_1\})$ is not localised to l_1 because it uses the variable y which belongs to l_2.

Intuitively it makes sense that the process owns its own variables, here formalised in the requirement that it must be explicitly defined. For all variables in the process policy P_ℓ which are used in influencer policies, i.e. $\mathsf{iv}(P_\ell)$, the security domain for the process $\mathcal{S}(\ell)$ must be in the influencer owners for that variable, i.e. $\mathcal{S}(\ell) \in \mathsf{io}(P_\ell, u)$. Similar rules apply to the reader policies in the process policy, before they can be considered localised.

Definition 2.3. A channel policy P_{ch} is *localised* if $\mathsf{fv}(P_{ch}) \subseteq \mathbf{Var}_\#$.

A channel has no security principal like the processes and a localised channel policy must therefore only contain channel variables.

2.4 Fully defined system

To summarise, a *fully defined system* is the tuple $(Sys, \mathcal{S}, \mathsf{NSA}, P_{Sys})$ where Sys is a system of processes

$$Sys = l_1 : S_1 \parallel \cdots \parallel l_n : S_n$$

which have m channels of arity (at most) k. The security principals

$$\mathcal{S}(\ell) \in \mathbf{Pr}$$

should be defined for all $\ell \in \mathbf{Lab} = \{l_1, \ldots, l_n\}$, with a special security principal

$$\mathsf{NSA} \neq \mathcal{S}(\ell)$$

for all ℓ, making $\mathcal{S}$ non-surjective.

With these definitions, the following sets can be derived:

$$\begin{aligned}
\mathbf{Lab} &= \{l_1, \ldots, l_n\} \\
\mathbf{Var} &= \biguplus_{l \in \mathbf{Lab}} \mathbf{Var}_l \\
\mathbf{Var}_\# &= \{\#_1, \cdots, \#_k\} \\
\mathbf{Var}^+ &= \mathbf{Var} \cup \mathbf{Var}_\# \\
\mathbf{Pr} &= \{\mathcal{S}(l_1)\} \cup \cdots \cup \{\mathcal{S}(l_n)\} \cup \mathsf{NSA} = \star
\end{aligned}$$

Finally, the system must specify the policies for all processes and channels

$$P_{Sys} = P_{l_1} \bullet \cdots \bullet P_{l_n} \bullet P_{ch_1} \bullet \cdots \bullet P_{ch_m} \tag{2.39}$$

where all the process policies and channel policies must be localised. If there are no policies defined for the channel, process or system it amounts to $\{\}$.

2.5 Security predicates

Given all processes policies for a system $P_\bullet = P_{l_1} \bullet \cdots \bullet P_{l_n}$ the *security predicate for systems* takes the form

$$\begin{aligned}
&\mathsf{sec}_{P_\bullet}(\sigma; U, F, D; \sigma') = \\
&\qquad \forall (u, s, u') \in F : \mathsf{Infl}\,(P_\bullet, \sigma, s, u) \subseteq \mathsf{Infl}\,(P_\bullet, \sigma', s, u') \wedge \\
&\qquad\qquad \mathsf{Read}\,(P_\bullet, \sigma, s, u) \supseteq \mathsf{Read}\,(P_\bullet, \sigma', s, u') \\
&\qquad \forall s : \forall u \in \mathbf{Var} \setminus D : \mathsf{Infl}\,(P_\bullet, \sigma, s, u) \subseteq \mathsf{Infl}\,(P_\bullet, \sigma', s, u) \wedge \\
&\qquad\qquad \mathsf{Read}\,(P_\bullet, \sigma, s, u) \supseteq \mathsf{Read}\,(P_\bullet, \sigma', s, u)
\end{aligned}$$

A secure system is considered secure if none of the policies are violated in the single transition relation. Here σ and σ' represents the states before and after the flow F takes place, and U and D are the used and defined variables in the state. The system flow F records the information flow from one set of variables $\bar{u}$ to another set of variables $\bar{u}'$ by a set of observers $\bar{s}$. The security predicate for systems ensures that there are more or the same influencers for the variables in $\bar{u}'$ than $\bar{u}$, to ensure integrity. Conversely, there should be less or the same readers for the variables in $\bar{u}'$ than $\bar{u}$, to ensure confidentiality.

The other part of the security predicate for systems ensures that all previously existing variables preserve their integrity and confidentiality during the transition from one state to another. The description of the security predicate leads to the following definition of a secure system.

Definition 2.4. The transition relation $\langle l_1 : S_1 \parallel \cdots l_n : S_n, \sigma \rangle \xRightarrow[U,D]{F} \langle l_1 : S'_1 \parallel \cdots \parallel l_n : S'_n, \sigma' \rangle$ is *secure* if $\mathsf{sec}_{P_\bullet}(\sigma; U, F, D; \sigma')$ holds.

A similar result can be shown for processes. Given two policies P, $P' \in \mathbf{Pol}$, two states $\sigma, \sigma' \in \mathbf{State}^+$ and an extended flow $E \in \mathbf{Var}^+_\diamond \times \mathbf{Pr} \times \mathbf{Var}^+_\diamond$, the *security predicate for processes* is defined by

$$\begin{aligned}
&\mathsf{sec}(P, \sigma; E; \sigma', P') = \\
&\qquad \forall (u, s, u') \in E \cap (\mathbf{Var}^+ \times \mathbf{Pr} \times \mathbf{Var}^+) : \mathsf{Infl}\,(P, \sigma, s, u) \subseteq \mathsf{Infl}\,(P', \sigma', s, u') \wedge \\
&\qquad\qquad \mathsf{Read}\,(P, \sigma, s, u) \supseteq \mathsf{Read}\,(P', \sigma', s, u') \\
&\qquad \forall s : \forall u \in \mathbf{Var} \setminus \mathsf{trd}(E) : \mathsf{Infl}\,(P, \sigma, s, u) \subseteq \mathsf{Infl}\,(P', \sigma', s, u) \wedge \\
&\qquad\qquad \mathsf{Read}\,(P, \sigma, s, u) \supseteq \mathsf{Read}\,(P', \sigma', s, u)
\end{aligned}$$

The two parts are similar to the security predicate for systems. The extended flows are filtered such that they only describes flows from variables and channel variables. The policy P then describes the policy which should be enforced before the flow and P' is the policy that should be enforced after the flow. The other part of the security predicate for processes ensures that all the previously existing variables preserve their integrity and confidentiality, just as in the security predicate for systems.

Definition 2.5. The transition relation $\vdash_\ell \langle S;\sigma\rangle \xrightarrow[\alpha]{E} \langle S';\sigma'\rangle$ is *secure* if

- $\alpha = \tau$ and $\mathsf{sec}(P_\ell, \sigma; E; \sigma', P_\ell)$ holds, or
- $\alpha = ch!(v_1, \ldots, v_k)$ and $\mathsf{sec}(P_\ell, \sigma; E; \sigma'[(\#_i \mapsto v_i)_{i\leq k}], P_\ell \bullet P_{ch})$ holds, or
- $\alpha = ch?(v_1, \ldots, v_k)$ then $\mathsf{sec}(P_\ell \bullet P_{ch}, \sigma[(\#_i \mapsto v_i)_{i\leq k}]; E; \sigma', P_\ell)$ holds.

It is now seen for the output statement that the policy P_ℓ is enforced before the flow, and the policy $P_\ell \bullet P_{ch}$ is enforced after the flow, such that the integrity and confidentiality is preserved in the information flow from the variables to the channel variables, and vice versa for the input statement.

The fully defined system enables runtime verification of whether the system and process is secure, using the two security predicates $\mathsf{sec}_{P_\bullet}(\sigma; U, F, D; \sigma')$ and $\mathsf{sec}(P, \sigma; E; \sigma', P')$. Implementing such a system then raises questions about what to do if the security predicates do not hold. The execution could simply stop, which could result in a system not working as intended. Another approach is to log the wrong information flow and continue the running the system with the leaked information. A further investigation can then be done and thereby make adjustment of policies or the program. Chapter 3 describes a type system where all possible flows for the system can be examined before run time. On the basis of that result, a distributed system can then be declared secure according to its policies.

2.6 Examples

2.6.1 Assignments

Consider the fully defined system $(Sys, \mathcal{S}, \mathsf{NSA}, P_{Sys})$ where

$$Sys = l : x := 4; y := x \tag{2.40}$$

and $\mathcal{S}(l) = s$. Given the policy

$$P_{Sys} = P_\bullet = P_l = \{xy : s \leftarrow s\} \tag{2.41}$$

where s allows x and y to be influenced by s. Because $\mathcal{S}(l) = s$, there is no problem.

All the flows in the system can now be determined. Using $[ass_{is}]$ and $[comp^2_{is}]$ we have

$$\frac{\vdash_\ell \langle x := 4; \sigma\rangle \xrightarrow[\tau]{\bar{4}\diamond\times\mathbf{Pr}\times x} \langle \mathsf{skip}; \sigma[x \mapsto [\![4]\!]\sigma]\rangle}{\langle l : x := 4; y := x, \sigma\rangle \xRightarrow[\epsilon,\{x\}]{\epsilon} \langle l : y := x, \sigma[x \mapsto [\![4]\!]\sigma]\rangle} \tag{2.42}$$

Let $\sigma' = \sigma[x \mapsto [\![4]\!]\sigma]$. Using $[ass_{is}]$ and $[comp^1_{is}]$, we have

$$\frac{\vdash_\ell \langle y := x; \sigma' \rangle \xrightarrow[\tau]{\bar{x}\diamond \times \mathbf{Pr} \times y} \langle \mathsf{skip}; \sigma'[y \mapsto [\![x]\!]\sigma'] \rangle}{\langle l : y := x, \sigma' \rangle \overset{x \times \mathbf{Pr} \times y}{\underset{\{x\},\{y\}}{\Longrightarrow}} \langle l : \mathsf{skip}, \sigma'[y \mapsto [\![x]\!]\sigma'] \rangle} \tag{2.43}$$

Let $\sigma'' = \sigma'[y \mapsto [\![x]\!]\sigma']$. The only system flows are therefore $(x \times \mathbf{Pr} \times y)$. The security predicate gives

$$\begin{aligned}
&\mathsf{sec}_{\{xy:s\leftarrow s\}}(\sigma; \epsilon, \epsilon, \{x\}; \sigma') \\
&\quad = \forall s' \in \{s, \mathsf{NSA}\} : \forall u \in \mathbf{Var} \setminus \{x\} : \\
&\qquad\quad \mathsf{Infl}\,(\{xy : s \leftarrow s\}, \sigma, s', u) \subseteq \mathsf{Infl}\,(\{xy : s \leftarrow s\}, \sigma', s', u) \wedge \\
&\qquad\quad \mathsf{Read}\,(\{xy : s \leftarrow s\}, \sigma, s', u) \supseteq \mathsf{Read}\,(\{xy : s \leftarrow s\}, \sigma', s', u)
\end{aligned}$$

Since the system flow in the first execution is ϵ, the first part of the security predicate is left out. The rest of the predicate is true as the only difference is σ and σ' which does not change the influencers and readers for this simple policy. For the other execution

$$\begin{aligned}
&\mathsf{sec}_{\{xy:s\leftarrow s\}}(\sigma'; \{x\}, x \times \mathbf{Pr} \times y, \{y\}; \sigma'') \\
&\quad = \forall (x, s', y) \in F : \\
&\qquad\quad \mathsf{Infl}\,(\{xy : s \leftarrow s\}, \sigma', s', x) \subseteq \mathsf{Infl}\,(\{xy : s \leftarrow s\}, \sigma'', s', y) \\
&\qquad\quad \mathsf{Read}\,(\{xy : s \leftarrow s\}, \sigma', s', x) \subseteq \mathsf{Read}\,(\{xy : s \leftarrow s\}, \sigma'', s', y) \\
&\qquad \forall s' \in \{s, \mathsf{NSA}\} : \forall u \in \mathbf{Var} \setminus \{y\} : \\
&\qquad\quad \mathsf{Infl}\,(\{xy : s \leftarrow s\}, \sigma', s', u) \subseteq \mathsf{Infl}\,(\{xy : s \leftarrow s\}, \sigma'', s', u) \wedge \\
&\qquad\quad \mathsf{Read}\,(\{xy : s \leftarrow s\}, \sigma', s', u) \supseteq \mathsf{Read}\,(\{xy : s \leftarrow s\}, \sigma'', s', u)
\end{aligned}$$

The first part is true as x and y have the same policy. The second part is also true using same augmenting as before. All the system flows are hence secure according to the policy in (2.41) as expected.

Next, consider another policy

$$P_{Sys} = P_l = \{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\}) \tag{2.44}$$

Here x may be influenced by s but y may only be influenced by s if $x > 5$. The first statement in (2.40) truly ensures that $x = 4$ thereby making $x > 5$ false. The flow from x to y is therefore illegal and the system is not secure according to this policy. Again this can be seen in the security predicate

$$\begin{aligned}
&\mathsf{sec}_{\{x:s\leftarrow s\}\bullet(x>5\Rightarrow\{y:s\leftarrow s\})}(\sigma; \epsilon, \epsilon, \{x\}; \sigma') \\
&\quad = \forall s' \in \{s, \mathsf{NSA}\} : \forall u \in \mathbf{Var} \setminus \{x\} : \\
&\qquad\quad \mathsf{Infl}\,(\{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\}), \sigma, s', u) \\
&\qquad\qquad \subseteq \mathsf{Infl}\,(\{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\}), \sigma', s', u) \\
&\qquad\quad \mathsf{Read}\,(\{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\}), \sigma, s', u) \\
&\qquad\qquad \supseteq \mathsf{Read}\,(\{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\}), \sigma', s', u)
\end{aligned}$$

Just as before the first part of the security predicate is left out. The relation can be false when $u = y$ and $s' = s$ which for influencers gives

$$\begin{aligned}
&\mathsf{Infl}\left(\{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\}), \sigma, s, y\right) \subseteq \\
&\quad \mathsf{Infl}\left(\{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\}), \sigma', s, y\right) \\
&\quad = \epsilon \cup \mathsf{Infl}\left((x > 5 \Rightarrow \{y : s \leftarrow s\}), \sigma, s, y\right) \subseteq \epsilon \mathsf{Infl}\left((x > 5 \Rightarrow \{y : s \leftarrow s\}), \sigma', s, y\right) \\
&\quad = \begin{cases} s & \text{if } \sigma \models x > 5 \\ \epsilon & \text{otherwise} \end{cases} \subseteq \begin{cases} s & \text{if } \sigma' \models x > 5 \\ \epsilon & \text{otherwise} \end{cases} \\
&\quad = \begin{cases} s & \text{if } \sigma \models x > 5 \\ \epsilon & \text{otherwise} \end{cases} \subseteq \epsilon
\end{aligned}$$

We have not specified the value for $\sigma(x)$ and we therefore have that $\sigma \not\models x > 5$. This also makes sense as y may not be influenced unless x is explicitly defined in the system. Hence the security predicate does not hold.

For completeness let us look at the security predicate for the second execution.

$$\begin{aligned}
&\mathsf{sec}_{\{x:s\leftarrow s\}\bullet(x>5\Rightarrow\{y:s\leftarrow s\})}(\sigma'; \{x\}, x \times \mathbf{Pr} \times y, \{y\}; \sigma'') \\
&\quad = \forall (x, s', y) \in F : \\
&\qquad \mathsf{Infl}\left(\{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\}), \sigma', s', x\right) \\
&\qquad\quad \subseteq \mathsf{Infl}\left(\{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\}), \sigma'', s', y\right) \\
&\qquad \mathsf{Read}\left(\{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\}), \sigma', s', x\right) \\
&\qquad\quad \subseteq \mathsf{Read}\left(\{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\}), \sigma'', s', y\right) \\
&\quad \forall s' \in \{s, \mathsf{NSA}\} : \forall u \in \mathbf{Var} \setminus \{y\} : \\
&\qquad \mathsf{Infl}\left(\{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\}), \sigma', s', u\right) \\
&\qquad\quad \subseteq \mathsf{Infl}\left(\{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\}), \sigma'', s', u\right) \wedge \\
&\qquad \mathsf{Read}\left(\{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\}), \sigma', s', u\right) \\
&\qquad\quad \supseteq \mathsf{Read}\left(\{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\}), \sigma'', s', u\right)
\end{aligned}$$

For the first part the influencers for x is clearly s on the left-hand side. On the right-hand side y becomes ϵ as $\sigma'' \not\models x > 5$ because $\sigma''(x) = 4$. The security predicate does not hold for this execution either.

The distributed system $(Sys, \mathcal{S}, \mathsf{NSA}, P_{Sys})$ is therefore not secure according to the system policies in (2.44).

2.6.2 Variable influenced by itself

Take the following system

$$l : x := x - 4 \qquad \mathcal{S}(l) = s \qquad P_\bullet = P_l = \{x : s_2 \leftarrow s_2\} \qquad \mathbf{Pr} = \{s, s_2, \mathsf{NSA}\}$$

where $s \neq s_2$. The policy says that x is owned by s_2 and may only be influenced by s_2. Intuitively this system should not be secure, because clearly x is influenced by $\mathcal{S}(l) = s$

in the assignment. But the policy is not localised, and the system is not defined properly. Looking at the flows gives $(x \times \mathbf{Pr} \times x) = \{(x, s, x), (x, s_2, x), (x, \mathsf{NSA}, x)\}$.

$$\frac{\vdash_l \langle x := x - 4; \sigma\rangle \xrightarrow[\tau]{x \diamond \times \mathbf{Pr} \times x} \langle \mathsf{skip}; \sigma[x \mapsto [\![x-4]\!]\sigma]\rangle}{\langle l : x := x - 4, \sigma\rangle \xRightarrow[\{x\},\{x\}]{x \times \mathbf{Pr} \times x} \langle l : \mathsf{skip}, \sigma[x \mapsto [\![x-4]\!]\sigma]\rangle}$$

Now let $\sigma' = \sigma[x \mapsto [\![x-4]\!]\sigma]$ and let $\mathbf{Var} = \{x\}$. The security predicate concerning integrity for the system is then

$$\begin{aligned} &\mathsf{sec}_{P_\bullet}(\sigma; \{x\}, (x \times \mathbf{Pr} \times x), \{x\}; \sigma') \\ &\quad = \forall (u, s, u') \in (x \times \mathbf{Pr} \times x) : \mathsf{Infl}\,(P_\bullet, \sigma, s, u) \subseteq \mathsf{Infl}\,(P_\bullet, \sigma', s, u') \\ &\qquad\quad \forall s : \forall u \in \mathbf{Var} \setminus \{x\} : \mathsf{Infl}\,(P_\bullet, \sigma, s, u) \subseteq \mathsf{Infl}\,(P_\bullet, \sigma', s, u) \end{aligned}$$

making the first part is true. The second part

$$\begin{aligned} \mathsf{Infl}\,(\{x : s_2 \leftarrow s_2\}, \sigma, s, x) &\subseteq \mathsf{Infl}\,(\{x : s_2 \leftarrow s_2\}, \sigma', s, x) && = \epsilon \subseteq \epsilon \\ \mathsf{Infl}\,(\{x : s_2 \leftarrow s_2\}, \sigma, s_2, x) &\subseteq \mathsf{Infl}\,(\{x : s_2 \leftarrow s_2\}, \sigma', s_2, x) && = s_2 \subseteq s_2 \\ \mathsf{Infl}\,(\{x : s_2 \leftarrow s_2\}, \sigma, \mathsf{NSA}, x) &\subseteq \mathsf{Infl}\,(\{x : s_2 \leftarrow s_2\}, \sigma', \mathsf{NSA}, x) && = \epsilon \subseteq \epsilon \end{aligned}$$

is also true as there are no variables $u \in \mathbf{Var} \setminus \{x\} = \epsilon$.

The system is therefore secure but the policies are not defined properly.

2.6.3 Multiple uses of same channel

When having multiple processes it is allowed to use the same channel such that

$$l_1 : ch!x \parallel l_2 : ch?y \parallel l_3 : ch?z$$

This systems will though never reach the terminal configuration, as one of the processes will wait for receiving input, but never receive anything. This is of course also allowed the other way around with multiple senders and one receiver

$$l_1 : ch!x \parallel l_2 : ch!y \parallel l_3 : ch?z$$

2.6.4 Bypass

Consider the following system:

$$l_1 : x :=^\dagger y; ch!x \parallel l_2 : ch?u; v :=^\dagger u \tag{2.45}$$

which contains three executions, first one bypass assignment, then one channel transfer (input and output statement), and last a bypass assignment. It is assumed that $\mathbf{Pr} = \{s_1, s_2, s_3\}$ and $\mathcal{S}(l_i) = s_i$ for $i = 1, 2$, leaving s_3 as the special principal. The flows can be determined using the instrumented semantics for systems and processes.

It is assumed that $\sigma(y)$ is defined. In the first execution using $\sigma_1 = \sigma[x \mapsto [\![y]\!]\sigma]$ the semantics gives

$$\frac{\dfrac{\vdash_{l_1} \langle x :=^{\dagger} y; \sigma\rangle \xrightarrow[\tau]{\{y,\diamond\}\times\{s_2,s_3\}\times\{x\}} \langle \mathsf{skip}; \sigma[x \mapsto [\![y]\!]\sigma]\rangle}{\vdash_{l_1} \langle x :=^{\dagger} y; ch!x; \sigma\rangle \xrightarrow[\tau]{\{y,\diamond\}\times\{s_2,s_3\}\times\{x\}} \langle ch!x; \sigma_1\rangle}}{\langle l_1 : x :=^{\dagger} y; ch!x \parallel l_2 : ch?u; v :=^{\dagger} u, \sigma\rangle \xRightarrow[\{y\},\{x\}]{\{y\}\times\{s_2,s_3\}\times\{x\}} \langle l_1 : ch!x \parallel l_2 : ch?u; v :=^{\dagger} u, \sigma_1\rangle}$$

In the second execution let $n = [\![x]\!]\sigma_1$ and $\sigma_2 = \sigma_1[u \mapsto n]$, which yields

$$\frac{\vdash_{l_1} \langle ch!x; \sigma_1\rangle \xrightarrow[ch!n]{\{x\}\times\{s_1,s_2,s_3\}\times\{\#_1\}} \langle \mathsf{skip}; \sigma_1\rangle \quad \dfrac{\vdash_{l_2} \langle ch?u; \sigma_1\rangle \xrightarrow[ch?n]{\{\#_1\}\times\{s_1,s_2,s_3\}\times\{u\}} \langle \mathsf{skip}; \sigma_1[u \mapsto n]\rangle}{\vdash_{l_2} \langle ch?u; v :=^{\dagger} u; \sigma_1\rangle \xrightarrow[ch?n]{\{\#_1\}\times\{s_1,s_2,s_3\}\times\{u\}} \langle v :=^{\dagger} u; \sigma_2\rangle}}{\langle l_1 : ch!x \parallel l_2 : ch?u; v :=^{\dagger} u, \sigma_1\rangle \xRightarrow[\{x\},\{u\}]{\{x\}\times\{s_1,s_2,s_3\}\times\{u\}} \langle l_1 : \mathsf{skip} \parallel l_2 : v :=^{\dagger} u, \sigma_2\rangle}$$

In the last and third execution let $\sigma_3 = \sigma_2[v \mapsto [\![u]\!]\sigma_2]$, which gives:

$$\frac{\vdash_{l_2} \langle l_2 : v :=^{\dagger} u; \sigma_2\rangle \xrightarrow[\tau]{\{u,\diamond\}\times\{s_1,s_3\}\times\{v\}} \langle \mathsf{skip}; \sigma_2[v \mapsto [\![u]\!]\sigma_2]\rangle}{\langle l_1 : \mathsf{skip} \parallel l_2 : v :=^{\dagger} u, \sigma_2\rangle \xRightarrow[\{u\},\{v\}]{\{u\}\times\{s_1,s_3\}\times\{v\}} \langle l_1 : \mathsf{skip} \parallel l_2 : \mathsf{skip}, \sigma_3\rangle}$$

In Table 2.1 the overall flows for the system is expanded for each of the security principal.

Assume we have a policy such that nobody may influence x. Because of the bypass this is *ok* for s_1 as there will not be observed any flow to x. The process only bypasses the policies for its own security principal, and s_2 will therefore observe this flow. Hence with such a policy the system will not be secure.

Execution	s_1	s_2	s_3
1		(y, s_2, x)	(y, s_3, x)
2	(x, s_1, u)	(x, s_2, u)	(x, s_3, u)
3	(u, s_1, v)		(u, s_3, v)

Table 2.1: System flows for (2.45) distributed on security domains.

CHAPTER 3 Type system

This chapter describes an inference system, combining a type system and a Hoare logic. The system can statically check if a fully defined system complies its policies before actually running the system. The programmer can therefore modify the policies and system until the type system yields a secure system. For the type system, a correctness result is then presented, mapping each execution from the type system to a legal execution in the instrumented semantics, according to the specified policies.

The type system springs from a classical Hoare logic, where the essential is the *Hoare triple*

$$\{\phi\}S\{\phi'\}$$

where ϕ is the *precondition*, ϕ' is the *postcondition* (collectively known as *assertions*), with both formulated using predicate logic. A triple $\{\phi\}S\{\phi'\}$ specifies that if the precondition ϕ holds in the initial state, and if the execution of the statement S terminates, then the postcondition ϕ' will hold in the state at which S halts [NN07].

The provability of the Hoare triple is denoted

$$env \vdash \{\phi\}S\{\phi'\}$$

where *env* is the *type environments*. A *typing rule* has the general form

$$\underbrace{\frac{\text{premises}}{env \vdash \{\phi\}S\{\phi'\}}}_{typing\ judgement} \text{ side conditions}$$

When there are no premises, the typing rule is called an *axiom* and the line is omitted. The *type environments* are used in the side conditions and the premises containing declarations, labels, etc.

The system is assumed to be fully defined, as described in Section 2.4. A typing judgement for distributed systems takes the form

$$\vdash \{\phi_1 \& \cdots \& \phi_n\}\ l_1 : S_1 \parallel \cdots \parallel l_n : S_n \tag{3.1}$$

where ϕ_i are the precondition for process l_i in the form of a predicate only containing variables from $\mathbf{Var}_{l_i}$. There is only one typing rule for a system

$$[sys_{ts}] \quad \frac{\epsilon \vdash_{l_1} \{\phi_1\}S_1\{\mathsf{true}\} \quad \cdots \quad \epsilon \vdash_{l_n} \{\phi_n\}S_n\{\mathsf{true}\}}{\vdash \{\phi_1 \& \cdots \& \phi_n\}\ l_1 : S_1 \parallel \cdots \parallel l_n : S_n} \tag{3.2}$$

All the individual processes should therefore end with true as their postcondition. For this reason, the typing judgement for systems does not have any postcondition, since this would be on the form of $\{\mathsf{true}\,\&\,\cdots\,\&\,\mathsf{true}\}$.

From (3.2) it is seen that the typing judgement for processes takes the form

$$\bar{y} \vdash_{\ell} \{\phi\}S\{\phi'\} \tag{3.3}$$

where $\{\phi\}S\{\phi'\}$ is a classical Hoare triple. The label ℓ denotes the process in question and $\bar{y}$ the set of variables that indirectly influences the statement S through implicit flows. The label ℓ and set of variables $\bar{y}$ are therefore type environments, for the judgement.

All typing rules and axioms for processes are specified in Rule Collection 3.1 and will be explained in more details in Section 3.4. First the notation for the assertions, side conditions and substitutions are described.

3.1 Assertions

The predicate logic for preconditions and postconditions is restricted to

$$\begin{array}{rl} \phi ::= & \mathsf{true} \quad | \quad \mathsf{false} \quad | \quad \neg\phi \quad | \quad \phi_1 \wedge \phi_2 \quad | \quad \phi_1 \vee \phi_2 \\ & | \quad a_1^+ \ rel \ a_2^+ \quad | \quad (\exists \bar{u} : (\phi)) \end{array}$$

making it syntactically equivalent to the policy conditions φ from Section 2.3.1. The assertions can therefore be used as conditions for policies e.g. $(\phi \Rightarrow P)$.

3.2 Substitutions in policies

Let a be an arithmetic operation and z a variable or channel variable, then $P[a/z]$ is defined as

$$\begin{aligned} \{\bar{u} : \bar{o} \leftarrow \bar{s}\}[a/z] &= \{\bar{u} : \bar{o} \leftarrow \bar{s}\} \\ \{\bar{u} : \bar{o} \rightarrow \bar{s}\}[a/z] &= \{\bar{u} : \bar{o} \rightarrow \bar{s}\} \\ (\varphi \Rightarrow P)[a/z] &= (\varphi[a/z] \Rightarrow P[a/z]) \\ (P^1 \bullet P^2)[a/z] &= P^1[a/z] \bullet P^2[a/z] \end{aligned}$$

This substitution therefore only has an effect in the conditions for the policies, and the notion of $\varphi[a/z]$ is already covered in Section 2.3.1, meaning that all occurrences of z in φ are replaced with a.

To determine the influencers and readers recall from (2.8) and (2.12) that $\sigma \models \varphi$ should hold for conditional policies. A substitution of the variables in the policies $P[a/z]$ can therefore be reflected in a change to the state σ.

Fact 3.1. $\mathsf{Infl}\,(P[a/x]\,, \sigma, q, z) = \mathsf{Infl}\,(P, \sigma[x \mapsto [\![a]\!]\sigma], q, z)$
(proven by structural induction on P in Appendix A.6)

$[skip_{ts}]$ $\bar{y} \vdash_\ell \{\phi\}\mathsf{skip}\{\phi'\}$ if $(\phi \Rightarrow \phi')$

$[ass_{ts}]$ $\bar{y} \vdash_\ell \{\phi\}x := a\{\phi'\}$ if $(\phi \Rightarrow \phi'[a/x]) \wedge (\phi \Rightarrow P_\ell\langle \bar{a}\bar{y}/x\rangle) \sqsubseteq P_\ell[a/x]$

$[ass^\dagger_{ts}]$ $\bar{y} \vdash_\ell \{\phi\}x :=^\dagger a\{\phi'\}$ if $(\phi \Rightarrow \phi'[a/x]) \wedge$
$(\phi \Rightarrow P_\ell\langle \bar{a}\bar{y}/x\rangle) \sqsubseteq P_\ell[a/x] \bullet \{x : \mathcal{S}(\ell) \leftarrow \star\} \bullet \{x : \mathcal{S}(\ell) \rightarrow \epsilon\}$

$[comb_{ts}]$ $$\frac{\bar{y} \vdash_\ell \{\phi\}S_1\{\phi''\} \quad \bar{y} \vdash_\ell \{\phi''\}S_2\{\phi'\}}{\bar{y} \vdash_\ell \{\phi\}S_1; S_2\{\phi'\}}$$

$[if_{ts}]$ $$\frac{\bar{y}\bar{b} \vdash_\ell \{\phi \wedge b\}S_1\{\phi'\} \quad \bar{y}\bar{b} \vdash_\ell \{\phi \wedge \neg b\}S_2\{\phi'\}}{\bar{y} \vdash_\ell \{\phi\}\mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}\{\phi'\}}$$

$[loop_{ts}]$ $$\frac{\bar{y}\bar{b} \vdash_\ell \{\phi \wedge b\}S\{\phi\}}{\bar{y} \vdash_\ell \{\phi\}\mathsf{while}\ b\ \mathsf{do}\ S\ \mathsf{od}\{\phi \wedge \neg b\}}$$

$[out_{ts}]$ $\bar{y} \vdash_\ell \{\phi\}ch!(a_1, \ldots, a_k)\{\phi'\}$ if $(\phi \Rightarrow \phi') \wedge$
$(\phi \Rightarrow P_\ell\langle \bar{a}_i\bar{y}/\#_i\rangle_{i\leq k}) \sqsubseteq P_{ch}[a_i/\#_i]_{i\leq k} \bullet \{\star_\ell : \star \leftarrow \star\} \bullet \{\star_\ell : \star \rightarrow \epsilon\}$

$[out^\dagger_{ts}]$ $\bar{y} \vdash_\ell \{\phi\}ch!^\dagger(a_1, \ldots, a_k)\{\phi'\}$ if $(\phi \Rightarrow \phi') \wedge$
$(\phi \Rightarrow P_\ell\langle \bar{a}_i\bar{y}/\#_i\rangle_{i\leq k}) \sqsubseteq P_{ch}[a_i/\#_i]_{i\leq k} \bullet \{\star_\# : \mathcal{S}(\ell) \leftarrow \star\} \bullet \{\star_\# : \mathcal{S}(\ell) \rightarrow \epsilon\}$
$\bullet \{\star_\ell : \star \leftarrow \star\} \bullet \{\star_\ell : \star \rightarrow \epsilon\}$

$[in_{ts}]$ $\bar{y} \vdash_\ell \{\phi\}ch?(x_1, \ldots, x_k)\{\phi'\}$ if $(((\exists x_1, \ldots, x_k : (\phi))) \Rightarrow \phi') \wedge$
$(\phi \Rightarrow P_\ell\langle \bar{y}/x_i\rangle_{i\leq k}) \bullet P_{ch}\langle \#_i/x_i\rangle_{i\leq k} \sqsubseteq P_\ell[\#_i/x_i]_{i\leq k} \bullet \{\star_\# : \star \leftarrow \star\}$
$\bullet \{\star_\# : \star \rightarrow \epsilon\}$

$[cho_{ts}]$ $$\frac{\bar{y} \vdash_\ell \{\phi\}S_1\{\phi'\} \quad \bar{y} \vdash_\ell \{\phi\}S_2\{\phi'\}}{\bar{y} \vdash_\ell \{\phi\}((S_1) \oplus (S_2))\{\phi'\}}$$

$[impl_{ts}]$ $$\frac{\bar{y}\bar{x} \vdash_\ell \{\phi\}S\{\phi'\}}{\bar{y} \vdash_\ell \{\phi\}\lceil \bar{x} \rceil S\{\phi'\}}$$

$[cons_{ts}]$ $$\frac{\bar{y} \vdash_\ell \{\psi\}S\{\psi'\}}{\bar{y} \vdash_\ell \{\phi\}S\{\phi'\}} \quad \text{if } (\phi \Rightarrow \psi) \wedge (\psi' \Rightarrow \phi')$$

Rule Collection 3.1: Type system for processes.

Fact 3.2. $\mathsf{Read}\left(P[a/x], \sigma, q, z\right) = \mathsf{Read}\left(P, \sigma[x \mapsto [\![a]\!]\sigma], q, z\right)$
(proven by structural induction on P in Appendix A.6)

The other substitution rule uses a set of variables or channel variables $\bar{y}$, and a variable or channel variable z, and are defined by

$$\begin{aligned}
\{\bar{u} : \bar{o} \leftarrow \bar{s}\}\langle\bar{y}/z\rangle &= \begin{cases} \{\bar{u}z : \bar{o} \leftarrow \bar{s}\} & \text{if } \bar{u} \cap \bar{y} \neq \epsilon \\ \{\bar{u} : \bar{o} \leftarrow \bar{s}\} & \text{otherwise} \end{cases} \\
\{\bar{u} : \bar{o} \rightarrow \bar{s}\}\langle\bar{y}/z\rangle &= \begin{cases} \{\bar{u}z : \bar{o} \rightarrow \bar{s}\} & \text{if } \bar{u} \cap \bar{y} \neq \epsilon \\ \{\bar{u} : \bar{o} \rightarrow \bar{s}\} & \text{otherwise} \end{cases} \\
(\varphi \Rightarrow P)\langle\bar{y}/z\rangle &= (\varphi \Rightarrow P\langle\bar{y}/z\rangle) \\
(P^1 \bullet P^2)\langle\bar{y}/z\rangle &= P^1\langle\bar{y}/z\rangle \bullet P^2\langle\bar{y}/z\rangle
\end{aligned}$$

The substitution has an effect on the owners, influencer and readers of the variable z in the policies. For $P\langle\bar{y}/z\rangle$ we can say that z is extended with policies for $\bar{y}$ in P. This results in the following facts, where the influencers are simply the influencers of all policies concerning $\bar{y} \cup \{z\}$, and the readers are the intersection of all the readers for the variables $\bar{y} \cup \{z\}$.

Fact 3.3. $\mathsf{Infl}\left(P\langle\bar{y}/x\rangle, \sigma, q, z\right) = \begin{cases} \bigcup_{u \in x\bar{y}} \mathsf{Infl}\left(P, \sigma, q, u\right) & \textit{if } x = z \\ \mathsf{Infl}\left(P, \sigma, q, z\right) & \textit{otherwise} \end{cases}$
(proven by structural induction on P in Appendix A.7)

Fact 3.4. $\mathsf{Read}\left(P\langle\bar{y}/x\rangle, \sigma, q, z\right) = \begin{cases} \bigcap_{u \in x\bar{y}} \mathsf{Read}\left(P, \sigma, q, u\right) & \textit{if } x = z \\ \mathsf{Read}\left(P, \sigma, q, z\right) & \textit{otherwise} \end{cases}$
(proven by structural induction on P in Appendix A.7)

At last, a shorthand notation is used for making multiple substitutions at once

$$\begin{aligned}
P_{ch}[a_i/u_i]_{i \leq k} &= (\cdots((P_{ch}[a_1/u_1])[a_2/u_2])\cdots)[a_k/u_k] \\
P_{ch}\langle\bar{x}_i/u_i\rangle_{i \leq k} &= (\cdots((P_{ch}\langle\bar{x}_1/u_1\rangle)\langle\bar{x}_2/u_2\rangle)\cdots)\langle\bar{x}_k/u_k\rangle
\end{aligned}$$

which is used in $[out_{ts}]$ and $[in_{ts}]$.

3.3 Side conditions

The side conditions in the type system are an extension of the assertions

$$\begin{aligned}
c \ ::= \ & \mathsf{true} \ | \ \mathsf{false} \ | \ \neg c \ | \ c_1 \wedge c_2 \ | \ c_1 \vee c_2 \\
| \ & a_1^+ \ rel \ a_2^+ \ | \ (\exists \bar{u} : (c)) \ | \ c_1 \Rightarrow c_2 \ | \ P^1 \sqsubseteq P^2
\end{aligned}$$

Different from the assertions is the implication $c_1 \Rightarrow c_2$, defined the usual way (shorthand for $\neg c_1 \vee c_2$). Intuitively it means that c_1 is stronger than c_2.

The side conditions are also extended with the preorder $\sqsubseteq$ as defined in (2.14), determining whether or not $P^1 \sqsubseteq P^2$. It is difficult to determine $P^1 \sqsubseteq P^2$ due to the complexity of the policies and the fact that the influencers and readers must be determined for all inputs. The result would therefore not be simply true or false but interdependent on the policy conditions, variables, channel variables, and security principals for the two policies. In Section 4.2 it is fully covered how to determine these interdependencies.

3.4 Type system for processes

Most of the rules in the type system look similar to a standard type system [NN07], and most of the axioms have extended side conditions which ensures that the information flow is secure. Each rule will be described here.

The $[skip_{ts}]$ rule is

$$\bar{y} \vdash_\ell \{\phi\}\mathsf{skip}\{\phi'\} \quad \text{if } (\phi \Rightarrow \phi')$$

It is clear that skip does nothing and always terminates, thus if the postcondition ϕ holds in the initial state before skip is executed, and ϕ' is constructed such that $\phi \Rightarrow \phi'$, then ϕ' will hold in the state where skip halts.

The $[ass_{ts}]$ rule is

$$\bar{y} \vdash_\ell \{\phi\}x := a\{\phi'\} \quad \text{if } \underbrace{(\phi \Rightarrow \phi'[a/x])}_{(1)} \wedge \underbrace{\overbrace{(\phi \Rightarrow P_\ell\langle \bar{a}\bar{y}/x\rangle)}^{(a)} \sqsubseteq \overbrace{P_\ell[a/x]}^{(b)}}_{(2)}$$

Part (1) is similar to the skip statement, but because the assignment updates the value for x, these changes needs to reflect the postcondition ϕ'. If ϕ holds, and if the execution of $a := x$ terminates (which it always does), and ϕ' is constructed in such a way that $\phi \Rightarrow \phi'[a/x]$, then ϕ' will hold in the state where $x := a$ halts.

Looking back at the security predicate for processes $\mathsf{sec}(P, \sigma; E; \sigma', P')$, the policies P should apply in the state σ before the flow, and the policy P' should apply to the system in the state σ' after the flow. In the type system there is no direct record of the state of the system, which is instead reflected in the postcondition. The changes made by $x := a$ are thus made directly in policies such that they reflect the changes between σ and σ'. Part (2) of the side condition for $[ass_{ts}]$ therefore concerns the policies. There needs to be a relation between the policies in the state before and after the assignment $x := a$, such that the policies concerning x are not violated by the update of x. The relation is two modified versions of the local policy P_ℓ, described here.

(a) Considerer $(\phi \Rightarrow P_\ell\langle \bar{a}\bar{y}/x\rangle) \sqsubseteq P_\ell$. In the assignment $x := a$ there is an information flow from the free variables inside a (denoted $\bar{a}$) to x. The local policy should therefore ensure that all the policies concerning the variables $\bar{a}$ are transferred to x (i.e. extending x with the policies for $\bar{y}$ in P_ℓ). Now the original policy P_ℓ should allow more or the same influencers for integrity, and fewer or the same readers for confidentiality than $(\phi \Rightarrow P_\ell\langle \bar{a}\bar{y}/x\rangle)$. The variables implicitly affecting the statement $\bar{y}$ are also included here, all yielding $P_\ell\langle \bar{a}\bar{y}/x\rangle$.

ϕ is the precondition holding before the execution of the assignment. The condition could therefore contain information regarding x, and the policy is therefore bound to this.

(b) Considerer $P_\ell \sqsubseteq P_\ell[a/x]$. The local policy P_ℓ could contain conditional policies where the condition φ contains x, i.e. $x \in \bar{\varphi}$. In the statement $x := a$ it is clear that $x = a$, and all the occurrences of x can therefore be replaced by a. For symmetry one could argue that this part also should be bound by ϕ', but that is not necessary, because the first part (1) ensures that the postcondition is implied by the precondition.

We now have $(\phi \Rightarrow P_\ell\langle \bar{a}\bar{y}/x\rangle) \sqsubseteq P_\ell$ and $P_\ell \sqsubseteq P_\ell[a/x]$ which combined gives (2). The side condition in $[ass_{ts}]$ ensures that when $x := a$ terminates no policies are violated due to the update of x by a.

The $[ass^\dagger_{ts}]$ rule is similar to $[ass_{ts}]$, only with a change in (2) of the side conditions

$$\underbrace{(\phi \Rightarrow \phi'[a/x])}_{(1)} \wedge \underbrace{\overbrace{(\phi \Rightarrow P_\ell\langle \bar{a}\bar{y}/x\rangle)}^{(a)} \sqsubseteq \overbrace{P_\ell[a/x]}^{(b)} \bullet \overbrace{\{x : \mathcal{S}(\ell) \leftarrow \star\} \bullet \{x : \mathcal{S}(\ell) \rightarrow \epsilon\}}^{(c)}}_{(2)}$$

(c) In the bypass assignment statement, the policies regarding x should be ignored. To model this, the policy $\{x : \mathcal{S}(\ell) \leftarrow \star\} \bullet \{x : \mathcal{S}(\ell) \rightarrow \epsilon\}$ is added. Looking at the right-hand side of the relation the influencers would then be $\star$ and the readers ϵ for the variable x from the process ℓ. The influencers for x would then yield $\cdot \subseteq \star$ and readers $\cdot \supseteq \epsilon$ which is always true.

The $[comb_{ts}]$ rule is simply applying the relevant typing rules for each of the statements S_1 and S_2.

$$\frac{\bar{y} \vdash_\ell \{\phi\}S_1\{\phi''\} \quad \bar{y} \vdash_\ell \{\phi''\}S_2\{\phi'\}}{\bar{y} \vdash_\ell \{\phi\}S_1; S_2\{\phi'\}}$$

If ϕ holds before the execution of $S_1; S_2$ then if the execution terminates, then ϕ' holds in the final state. Furthermore if S_1 terminates the postcondition ϕ'' for S_1 holds, and serves as precondition for S_2. If S_2 terminates then ϕ' holds in the final state for the execution of $S_1; S_2$.

The $[if_{ts}]$ typing rule for conditional branching if b then S_1 else S_2 fi is

$$\frac{\bar{y}\bar{b} \vdash_\ell \{\phi \wedge b\}S_1\{\phi'\} \quad \bar{y}\bar{b} \vdash_\ell \{\phi \wedge \neg b\}S_2\{\phi'\}}{\bar{y} \vdash_\ell \{\phi\}\textsf{if } b \textsf{ then } S_1 \textsf{ else } S_2 \textsf{ fi}\{\phi'\}}$$

Saying that when if b then S_1 else S_2 fi is executed in a state where ϕ holds and that if b then S_1 else S_2 fi terminates then ϕ' will hold in the final state. From this the precondition is further strengthened for the two branches. If S_1 is executed in a state where $\phi \wedge b$ holds and S_1 terminates, then ϕ' will hold in the final state. Likewise for S_2 with the precondition $\phi \wedge \neg b$. Furthermore the implicit flows from the free variables in b are added to each of the branches.

The $[loop_{ts}]$ rule for iteration $\mathsf{while}\ b\ \mathsf{do}\ S\ \mathsf{od}$ is

$$\frac{\bar{y}\bar{b} \vdash_\ell \{\phi \wedge b\}S\{\phi\}}{\bar{y} \vdash_\ell \{\phi\}\mathsf{while}\ b\ \mathsf{do}\ S\ \mathsf{od}\{\phi \wedge \neg b\}}$$

which is similar to $[if_{ts}]$. Here the precondition for S is also strengthened to $\phi \wedge b$. It is furthermore seen that the precondition and postcondition ϕ is the same on both sides, making ϕ an *invariant*. This invariant is further discussed in Section 4.4. In $[loop_{ts}]$ it is furthermore straight forward to see that if the iteration terminates then $\neg b$ will be true after the termination, thus gives the postcondition $\phi \wedge \neg b$. Here the implicit flows from the free variables in b are also added to the typing rule for S.

For the $[out_{ts}]$ rule, the side conditions are

$$\underbrace{(\phi \Rightarrow \phi')}_{(1)} \wedge \underbrace{\overbrace{(\phi \Rightarrow P_\ell\langle \bar{a}_i\bar{y}/\#_i\rangle_{i\leq k})}^{(a)} \sqsubseteq \overbrace{P_{ch}[a_i/\#_i]_{i\leq k}}^{(b)} \bullet \overbrace{\{\star_\ell : \star \leftarrow \star\} \bullet \{\star_\ell : \star \rightarrow \epsilon\}}^{(d)}}_{(2)}$$

In part (1) of the side condition, the output is equivalent to $[skip_{ts}]$, as the output statement should do nothing.

For a process ℓ with a channel output $ch!(a_1, \ldots, a_k)$ the type system needs a relation between the local policy P_ℓ and the policy for the channel P_{ch}. The process may only transfer information to the channel variables if the channel policy allows more or the same influencers for integrity, and conversely fewer or the same readers for confidentiality as expressed previously for (2) in $[ass_{ts}]$. The features for the side conditions are then

(a) For the statement $ch!(a_1, \ldots, a_k)$ there is an information flow from the variables $\bar{a}_i$ to $\#_i$ (for all $i \leq k$), where $\bar{a}_i$ denotes the free variables in a_i. The local policy P_ℓ should therefore ensure that all policies concerning the variables $\bar{a}_i$ is applied to $\#_i$ for all $i \leq k$. Similar to (a) for $[ass_{ts}]$, the precondition is added for this policy.

(b) Each channel variable $\#_i$ used as a condition in P_{ch} should be related to the arithmetic expression a_i (again for all $i \leq k$), and thereby evaluated.

When such a relation is established it is only necessary to look at the policies with channel variables, since all the relevant policies have been transferred to the channel variables (which was done in (a)). One last feature is therefore needed

(d) The local policy P_ℓ also contains policies for variables not used in any of the transferred expressions $\bar{a}_i$. These should not have any effect in this relation, and top local policy $\{\star_\ell : \star \leftarrow \star\} \bullet \{\star_\ell : \star \rightarrow \epsilon\}$ is therefore appended to P_{ch}. The relation will therefore ensure (2) in relation to influencers and readers.

If the precondition ϕ holds before the execution of $ch!(a_1, \ldots, a_k)$, and if the execution terminates and the side conditions hold (i.e. no illegal information flows) then the postcondition ϕ' will hold in the final state.

The side condition for the $[out^{\dagger}_{ts}]$ rule is a combination of $[ass^{\dagger}_{ts}]$ and $[out_{ts}]$. Similar to the $[out_{ts}]$ is (1), (a), (b) and (d).

$$\overbrace{(\phi \Rightarrow \phi')}^{(1)} \wedge \overbrace{\underbrace{(\phi \Rightarrow P_\ell\langle \bar{a}_i\bar{y}/\#_i\rangle_{i\le k})}_{(a)} \sqsubseteq \underbrace{P_{ch}[a_i/\#_i]_{i\le k}}_{(b)} \bullet \underbrace{\{\star_\# : \mathcal{S}(\ell) \leftarrow \star\} \bullet \{\star_\# : \mathcal{S}(\ell) \rightarrow \epsilon\}}_{(c)} \bullet \underbrace{\{\star_\ell : \star \leftarrow \star\} \bullet \{\star_\ell : \star \rightarrow \epsilon\}}_{(d)}}^{(2)}$$

and from $[ass^{\dagger}_{ts}]$ is

(c) In the bypass output, the policies regarding $\star_\#$ should be ignored. To model this, $\{\star_\# : \mathcal{S}(\ell) \leftarrow \star\} \bullet \{\star_\# : \mathcal{S}(\ell) \rightarrow \epsilon\}$ is added on the right-hand side of the relation such that in the relation of the two sides the influencers and readers for the channels variables for the security domain $\mathcal{S}(\ell)$ does not have any effect on the result.

The side conditions the $[in_{ts}]$ rule is similar to $[ass_{ts}]$ and a reverse version of $[out_{ts}]$

$$\overbrace{(((\exists x_1, \ldots, x_k : (\phi))) \Rightarrow \phi')}^{(1)} \wedge \overbrace{\underbrace{(\phi \Rightarrow P_\ell\langle \bar{y}/x_i\rangle_{i\le k}) \bullet P_{ch}\langle \#_i/x_i\rangle_{i\le k}}_{(a)} \sqsubseteq \underbrace{P_\ell[\#_i/x_i]_{i\le k}}_{(b)} \bullet \underbrace{\{\star_\# : \star \leftarrow \star\} \bullet \{\star_\# : \star \rightarrow \epsilon\}}_{(d)}}^{(2)}$$

Part (1) is relative to the state after the execution of the statement, where there exists $x_1, \ldots, x_k$ such that ϕ holds, which implies ϕ'.

Part (2) of the side conditions follows a structure similar to $[out_{ts}]$. The channel may only transfer the information to the process if the process policy allows more or the same influencers for integrity, and conversely fewer or the same readers for confidentiality, on the variables $x_1, \ldots, x_k$ used for input.

(a) For the statement $ch?(x_1, \ldots, x_k)$ there is an information flow from the variable $\#_i$ to x_i (for all $i \le k$). The channel policy P_{ch} is therefore modified to reflect the updates made for the incoming values. Furthermore the implicit variables $\bar{y}$ are ensured for all the variables $x_1, \ldots, x_k$ in the local policy P_ℓ, and the precondition ϕ is also made a condition for the policy, as in $[ass_{ts}]$.

(b) The conditions from the left-hand side of the relation should be matched to the conditions on the right-hand side. The variables used are therefore replaced with channel variables.

(d) All the policies regarding channel variables are transferred to local variables for the processes by the last part of (a). This part is therefore like (d) from $[out_{ts}]$.

The typing rule $[cho_{ts}]$ is

$$\frac{\bar{y} \vdash_\ell \{\phi\}S_1\{\phi'\} \quad \bar{y} \vdash_\ell \{\phi\}S_2\{\phi'\}}{\bar{y} \vdash_\ell \{\phi\}((S_1) \oplus (S_2))\{\phi'\}}$$

The rule is similar to $[if_{ts}]$ where the precondition ϕ is equivalent to both typing of S_1 and S_2, but here the postconditions should be equivalent too.

The typing rule $[impl_{ts}]$ is

$$\frac{\bar{y}\bar{x} \vdash_\ell \{\phi\}S\{\phi'\}}{\bar{y} \vdash_\ell \{\phi\}\lceil\bar{x}\rceil S\{\phi'\}}$$

which is straight forward, where the flows from the variables in $\bar{x}$ are transferred.

The last rule $[cons_{ts}]$ is the *rule of consequence*

$$\frac{\bar{y} \vdash_\ell \{\psi\}S\{\psi'\}}{\bar{y} \vdash_\ell \{\phi\}S\{\phi'\}} \text{ if } (\phi \Rightarrow \psi) \wedge (\psi' \Rightarrow \phi')$$

which is used to pre-strengthen the precondition and post-weakening the postcondition in the type system. This rule can of course be implemented in all the other rules, but having this as a separate rule simplifies the the correctness proof.

3.5 Correctness result

Most papers on information flows shows a non-interference result, in which variables of high confidentiality and integrity do not interfere with variables of low confidentiality [Bou09; BH07]. In program analysis and especially for type system the correctness is shown by subject reduction. Here one reduction in the instrumented semantics together with the relevant instance in the type system yields a new instance in the type system.

The correctness of the type system is therefore shown by a classical subject reduction, where the reduction should be secure according to the security predicates we previously established in Definitions 2.4 and 2.5.

The correctness result for processes therefore amounts to the following lemma which contributes to the soundness of the type system.

Lemma 3.1. *Assume* $\bar{y} \vdash_\ell \{\phi\}S\{\phi'\}$, $\sigma \models \phi$ *and* $\vdash_\ell \langle S;\sigma\rangle \xrightarrow[\alpha]{E} \langle S';\sigma'\rangle$. *Then there exists* ψ *such that* $\bar{y} \vdash_\ell \{\psi\}S'\{\phi'\}$ *and* $\sigma' \models \psi$, *and furthermore*

- *if* $\alpha = \tau$ *then* $\mathsf{sec}(P_\ell, \sigma; \lceil\bar{y}\rceil E; \sigma', P_\ell)$,
- *if* $\alpha = ch!(v_1, \ldots, v_k)$ *then* $\mathsf{sec}(P_\ell, \sigma; \lceil\bar{y}\rceil E; \sigma'[(\#_i \mapsto v_i)_{i \leq k}], P_\ell \bullet P_{ch})$, *and*

- *if* $\alpha = ch?(v_1, \ldots, v_k)$ *then* $\mathsf{sec}(P_\ell \bullet P_{ch}, \sigma[(\#_i \mapsto v_i)_{i \leq k}]; \lceil \bar{y} \rceil E; \sigma', P_\ell)$.
  (proven by induction on the inference of $\bar{y} \vdash_\ell \{\phi\} S \{\psi\}$ *in Appendix A.8)*

Similarly, by using Definition 2.4 for secure systems, the correctness for the distributed systems then amounts to the following theorem.

Theorem 3.1. *Assume* $\vdash \{\phi_1 \& \cdots \& \phi_n\}\ l_1 : S_1 \parallel \cdots \parallel l_n : S_n$ *and* $\sigma \models \phi_1 \wedge \cdots \wedge \phi_n$ *and* $\langle l_1 : S_1 \parallel \cdots l_n : S_n, \sigma\rangle \xRightarrow[U,D]{F} \langle l_1 : S_1' \parallel \cdots \parallel l_n : S_n', \sigma'\rangle$. *Then there exist* $\psi_1, \ldots, \psi_n$ *such that* $\vdash \{\psi_1 \& \cdots \& \psi_n\}\ l_1 : S_1' \parallel \cdots \parallel l_n : S_n'$ *and* $\sigma' \models \psi_1 \wedge \cdots \wedge \psi_n$ *and* $\mathsf{sec}_{P_\bullet}(\sigma; U, F, D; \sigma')$. *(proven by the induction on the inference system and using Lemma 3.1 in Appendix A.9)*

These correctness results now finally show that with a fully defined system $(Sys, \mathcal{S}, \mathsf{NSA}, P_{Sys})$ and if $\vdash \{\phi_1 \& \cdots \& \phi_n\}\ l_1 : S_1 \parallel \cdots \parallel l_n : S_n$ holds, then $(Sys, \mathcal{S}, \mathsf{NSA}, P_{Sys})$ is secure.

3.6 Simple assignment example

Consider again $(Sys, \mathcal{S}, \mathsf{NSA}, P_{Sys})$ from Section 2.6.1 where

$$l : x := 4; y := x \tag{2.40}$$

and $\mathcal{S}(l) = s$. First consider the policy

$$P_{Sys} = P_l = \{xy : s \leftarrow s\} \tag{2.41}$$

Using the type system then yields

$$\cfrac{\cfrac{\cfrac{\epsilon \vdash_l \{\phi\} x := 4 \{\phi''\} \text{ if } c_1 \quad \epsilon \vdash_l \{\phi''\} y := x \{\phi'\} \text{ if } c_2}{\epsilon \vdash_l \{\phi\} x := 4; y := x \{\phi'\}}}{\epsilon \vdash_l \{\phi\} x := 4; y := x \{\mathsf{tt}\}}\ \phi' \Rightarrow \mathsf{tt}}{\vdash \{\phi\}\ l : x := 4; y := x}$$

where

$$\begin{aligned}
c_1 &= (\phi \Rightarrow \phi''[4/x]) \wedge \big(\phi \Rightarrow P_\ell\langle \bar{4}\bar{\epsilon}/x\rangle\big) \sqsubseteq P_\ell[4/x] \\
&= (\phi \Rightarrow \phi''[4/x]) \wedge (\phi \Rightarrow P_\ell) \sqsubseteq P_\ell[4/x] \\
c_2 &= (\phi'' \Rightarrow \phi'[x/y]) \wedge (\phi'' \Rightarrow P_\ell\langle \bar{x}\bar{\epsilon}/y\rangle) \sqsubseteq P_\ell[x/y] \\
&= (\phi'' \Rightarrow \phi'[x/y]) \wedge (\phi'' \Rightarrow P_\ell\langle x/y\rangle) \sqsubseteq P_\ell[x/y]
\end{aligned}$$

Let $\phi = \mathsf{tt}$, $\phi'' = x = 4$ and $\phi' = x = 4 \wedge y = x$ and see that all the postconditions hold if the side conditions c_1 and c_2 holds and if the statement terminates. For this policy the side conditions are

$$\begin{aligned}
c_1 &= (\mathsf{tt} \Rightarrow (x = 4)[4/x]) \wedge (\mathsf{tt} \Rightarrow \{xy : s \leftarrow s\}) \sqsubseteq \{xy : s \leftarrow s\}[4/x] \\
&= \mathsf{tt} \Rightarrow 4 = 4 \wedge \{xy : s \leftarrow s\} \sqsubseteq \{xy : s \leftarrow s\} \\
&= \mathsf{tt} \\
c_2 &= (x = 4 \Rightarrow x = 4 \wedge x = x) \wedge (x = 4 \Rightarrow \{xy : s \leftarrow s\}\langle \bar{x}/y\rangle) \sqsubseteq \{xy : s \leftarrow s\}[x/y] \\
&= (x = 4 \Rightarrow \{xy : s \leftarrow s\}\langle \bar{x}/y\rangle) \sqsubseteq \{xy : s \leftarrow s\}[x/y] \\
&= (x = 4 \Rightarrow \{xy : s \leftarrow s\}) \sqsubseteq \{xy : s \leftarrow s\}
\end{aligned}$$

This last condition is also true as the restriction $x = 4$ leaves less influencers on the left-hand side.

Next consider the other policy

$$P_{Sys} = P_l = \{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\}) \tag{2.44}$$

which yields the same inference tree as before. The side conditions can now be verified

$$\begin{aligned}
c_1 =& (\mathsf{tt} \Rightarrow (x = 4)[4/x]) \wedge (\mathsf{tt} \Rightarrow \{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\})) \\
&\sqsubseteq \{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\})[4/x] \\
=& (\mathsf{tt} \Rightarrow \{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\})) \\
&\sqsubseteq \{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\})[4/x] \\
=& (\mathsf{tt} \Rightarrow \{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\})) \\
&\sqsubseteq \{x : s \leftarrow s\} \bullet (4 > 5 \Rightarrow \{y : s \leftarrow s\}) \\
=& \{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\}) \sqsubseteq \{x : s \leftarrow s\} \bullet \{y : s \leftarrow s\} \\
=& \mathsf{tt} \\
c_2 =& (x = 4 \Rightarrow (x = 4 \wedge y = x)[x/y]) \wedge (x = 4 \Rightarrow P_\ell\langle x/y\rangle) \sqsubseteq P_\ell[x/y] \\
=& (x = 4 \Rightarrow (\{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\}))\langle x/y\rangle) \\
&\sqsubseteq (\{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\}))[x/y] \\
=& \underbrace{(x = 4 \Rightarrow (\{xy : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\})))}_{P_{\text{left}}} \\
&\sqsubseteq \underbrace{(\{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\}))}_{P_{\text{right}}}
\end{aligned}$$

For all σ, q, and u it therefore must hold that

$$\begin{aligned}
&\mathsf{Infl}\,(P_{\text{left}}, \sigma, q, u) \subseteq \mathsf{Infl}\,(P_{\text{right}}, \sigma, q, u) \\
&\wedge \mathsf{Read}\,(P_{\text{left}}, \sigma, q, u) \supseteq \mathsf{Read}\,(P_{\text{right}}, \sigma, q, u)
\end{aligned} \tag{3.4}$$

In a particular case look at the influencers for an instance where $q = s$, $u = y$ and a

σ such that $\sigma \models x = 4$

$$\begin{aligned}
&\mathsf{Infl}\,(P_{\text{left}}, \sigma, s, y) \subseteq \mathsf{Infl}\,(P_{\text{right}}, \sigma, s, y) \\
&\quad = \begin{cases} s & \text{if } \sigma \models (x = 4) \vee (x = 4 \wedge x > 5) \\ \epsilon & \text{otherwise} \end{cases} \subseteq \begin{cases} s & \text{if } \sigma \models x > 5 \\ \epsilon & \text{otherwise} \end{cases} \\
&\quad = s \subseteq \epsilon
\end{aligned}$$

Because of this instance (3.4) does not hold for all σ, the side condition c_2 is not true. All this therefore makes the system insecure according to (2.44) matching the result found by looking at the instrumented semantics and the security predicate in Section 2.6.1.

CHAPTER 4

Implementation

Implementing the type system requires ideas on how to determine the postconditions ϕ' and how to determine the preorder $P \sqsubseteq P'$ in the side conditions. The policies can be complex, contain many embedded conditions, sequences and so on, and the relation between to policies can therefore be difficult to establish, because the influencer and reader sets should hold for all $\sigma \in \textbf{State}^+$, $q \in \textbf{Pr}$ and $y \in \textbf{Var}^+$. The first section hence introduces a notion of basic policies, which is used in Section 4.2 to determine the preorder. The rest of the chapter is about how to implement the type system and how to verify the distributed system.

Most of the strategies presented in this chapter covers how to do implement the type system for influencer and reader policies, but in the concrete implementation developed for this thesis only influencer policies are considered. Section 4.5 briefly covers the prototype developed throughout this thesis.

4.1 Basic policies and implication normal form

A *basic policy* B, is denoted as a conditional policy with an influencer policy or a reader policy. The influencer policy has exactly one variable, one owner and one influencer. The reader policy has exactly one variable, one owner and a set of readers.

$$B \ ::= \ (\varphi \Rightarrow \{u : o \leftarrow s\}) \quad | \quad (\varphi \Rightarrow \{u : o \rightarrow \bar{s}\})$$

Therefore $u \neq \epsilon$, $o \neq \epsilon$, $s \neq \epsilon$. The reader variables $\bar{s}$ can be everything from ϵ to $\star$ as usual.

A policy is in *implication normal form* (INF) if it only contains a sequence of basic policies:

$$B_1 \bullet \cdots \bullet B_m \tag{4.1}$$

for all cases $m \geq 0$, where if $m = 0$ the INF is simply $\{\}$.

Take the policy $\{\bar{u_i} : \bar{o_j} \leftarrow \bar{s_k}\}$, which has a set of variables or channels $\bar{u_i}$ of size i, a set of owners $\bar{o_j}$ of size j, and a set of influencers $\bar{s_k}$ of size k. By using the distributive laws for influencer policies (2.29), (2.31) and (2.33) and an equivalence for conditional policies (2.37) we get

$$\begin{aligned}\{\bar{u_i} : \bar{o_j} \leftarrow \bar{s_k}\} &\equiv \{u_1 : o_1 \leftarrow s_1\} \bullet \ldots \bullet \{u_i : o_j \leftarrow s_k\} \\ &\equiv (\mathsf{true} \Rightarrow \{u_1 : o_1 \leftarrow s_1\}) \bullet \ldots \bullet (\mathsf{true} \Rightarrow \{u_i : o_j \leftarrow s_k\})\end{aligned}$$

and sees that the policy has an equivalent policy in INF.

To establish an algorithm to find the INF of a policy, Algorithm 4.1 to 4.3 are used. Algorithm 4.1 defines a recursive function $\mathsf{dist}(P)$ which uses the distribution laws to distribute integrity and reader policies for any P. Furthermore all sub-policies in $\mathsf{dist}(P)$ which are confidentiality or influencer policies equivalent to $\{\}$ are reduced to $\{\}$. Algorithm 4.2 defines $\mathsf{cln}(P)$ which cleans the policies such that all policies equivalent to $\{\}$ are removed. Lastly, Algorithm 4.3 defines $\mathsf{cnd}(P)$ which distributes conditions on sequences and collects conditions on succeeding conditional policies. All the algorithms use the equivalences from Section 2.3.6 and each rule is accounted for.

Fact 4.1. *The function* $\mathsf{inf}(P) = \mathsf{cnd}(\mathsf{cln}(\mathsf{dist}(P)))$ *converts any policy* P *into INF and* $P \equiv \mathsf{inf}(P)$*.*

(informally proven by the structure of P for each function in Appendix A.10)

```
$\mathsf{dist}(P) = \{\}$   if $P \equiv \{\}$ from (2.17) to (2.20)
$\mathsf{dist}(\{u : o \leftarrow s\}) = \{u : o \leftarrow s\}$
$\mathsf{dist}(\{\bar{u} : \bar{o} \leftarrow s\bar{s}'\}) = \mathsf{dist}(\{\bar{u} : \bar{o} \leftarrow s\}) \bullet \mathsf{dist}(\{\bar{u} : \bar{o} \leftarrow \bar{s}'\})$   using (2.33)
$\mathsf{dist}(\{\bar{u} : o\bar{o}' \leftarrow s\}) = \mathsf{dist}(\{\bar{u} : o \leftarrow s\}) \bullet \mathsf{dist}(\{\bar{u} : \bar{o}' \leftarrow s\})$   using (2.31)
$\mathsf{dist}(\{u\bar{u}' : o \leftarrow s\}) = \mathsf{dist}(\{u : o \leftarrow s\}) \bullet \mathsf{dist}(\{\bar{u}' : o \leftarrow s\})$   using (2.29)
$\mathsf{dist}(\{u : o \rightarrow \bar{s}\}) = \{u : o \rightarrow \bar{s}\}$
$\mathsf{dist}(\{\bar{u} : o\bar{o}' \rightarrow \bar{s}\}) = \mathsf{dist}(\{\bar{u} : o \rightarrow \bar{s}\}) \bullet \mathsf{dist}(\{\bar{u} : \bar{o}' \rightarrow \bar{s}\})$   using (2.32)
$\mathsf{dist}(\{u\bar{u}' : o \rightarrow \bar{s}\}) = \mathsf{dist}(\{u : o \rightarrow \bar{s}\}) \bullet \mathsf{dist}(\{\bar{u}' : o \rightarrow \bar{s}\})$   using (2.30)
$\mathsf{dist}((\varphi \Rightarrow P)) = (\varphi \Rightarrow \mathsf{dist}(P))$
$\mathsf{dist}(P_1 \bullet P_2) = \mathsf{dist}(P_1) \bullet \mathsf{dist}(P_2)$
```

Algorithm 4.1: Recursive function $\mathsf{dist}(P)$, distributing all confidentiality and influencer policies.

4.1.1 Sorted implication normal form

A policy is in *sorted implication normal form* (SINF) when all the basic influencer policies come first and are followed by all the basic reader policies.

$$\underbrace{B_1 \bullet \cdots \bullet B_n}_{\text{basic influencer policy}} \bullet \underbrace{B_{n+1} \bullet \cdots \bullet B_m}_{\text{basic reader policy}} \tag{4.2}$$

The policies $B_1, \ldots, B_n$ therefore have the form $(\varphi \Rightarrow \{u : o \leftarrow s\})$ and $B_{n+1}, \ldots, B_m$ have the form $(\varphi \Rightarrow \{u : o \rightarrow \bar{s}\})$. The function $\mathsf{sinf}(P)$ denotes the conversion of any policy P into SINF, and can be constructed using $\mathsf{inf}(P)$ and the fact that $\bullet$ is communicative (2.25).

$$\begin{aligned}
&\mathsf{cln}(P) = \{\} \quad \text{if } P \equiv \{\} \text{ from (2.17) to (2.20)}\\
&\mathsf{cln}(\{\bar{u} : \bar{o} \leftarrow \bar{s}\}) = \{\bar{u} : \bar{o} \leftarrow \bar{s}\}\\
&\mathsf{cln}(\{\bar{u} : \bar{o} \rightarrow \bar{s}\}) = \{\bar{u} : \bar{o} \rightarrow \bar{s}\}\\
&\mathsf{cln}((\varphi' \Rightarrow P')) = \begin{cases} \{\} & \text{if } \varphi' = \mathsf{false} \quad \text{using (2.38)}\\ \{\} & \text{if } \mathsf{cln}(P') = \{\} \quad \text{using (2.36)}\\ (\varphi' \Rightarrow \mathsf{cln}(P')) & \text{otherwise} \end{cases}\\
&\mathsf{cln}(P_1 \bullet P_2) = \begin{cases} \{\} & \text{if } \begin{array}{l}\mathsf{cln}(P_1) = \{\} \wedge\\ \quad \mathsf{cln}(P_2) = \{\}\end{array} \quad \text{using (2.27)}\\ \mathsf{cln}(P_1) & \text{if } \mathsf{cln}(P_2) = \{\} \quad \text{using (2.27)}\\ \mathsf{cln}(P_2) & \text{if } \mathsf{cln}(P_1) = \{\} \quad \text{using (2.27)}\\ \mathsf{cln}(P_1) \bullet \mathsf{cln}(P_2) & \text{otherwise} \end{cases}
\end{aligned}$$

Algorithm 4.2: Recursive function $\mathsf{cln}(P)$, cleaning up all empty policies from P, except if all policies are empty.

$$\begin{aligned}
&\mathsf{cnd}(P) = \{\} \quad \text{if } P \equiv \{\} \text{ from (2.17) to (2.20)}\\
&\mathsf{cnd}(\{\bar{u} : \bar{o} \leftarrow \bar{s}\}) = (\mathsf{true} \Rightarrow \{\bar{u} : \bar{o} \leftarrow \bar{s}\}) \quad \text{using (2.37)}\\
&\mathsf{cnd}(\{\bar{u} : \bar{o} \rightarrow \bar{s}\}) = (\mathsf{true} \Rightarrow \{\bar{u} : \bar{o} \rightarrow \bar{s}\}) \quad \text{using (2.37)}\\
&\mathsf{cnd}((\varphi' \Rightarrow P')) = \begin{cases} (\varphi' \Rightarrow P') & \text{if } P' = \{\bar{u} : \bar{o} \leftarrow \bar{s}\}\\ (\varphi' \Rightarrow P') & \text{if } P' = \{\bar{u} : \bar{o} \rightarrow \bar{s}\}\\ \mathsf{cnd}((\varphi' \wedge \varphi'' \Rightarrow P'')) & \text{if } P' = (\varphi'' \Rightarrow P'') \quad \text{using (2.35)}\\ \begin{array}{l}\mathsf{cnd}((\varphi' \Rightarrow P_1)) \bullet\\ \quad \mathsf{cnd}((\varphi' \Rightarrow P_2))\end{array} & \text{otherwise} \quad \text{using (2.34)} \end{cases}\\
&\mathsf{cnd}(P_1 \bullet P_2) = \mathsf{cnd}(P_1) \bullet \mathsf{cnd}(P_2)
\end{aligned}$$

Algorithm 4.3: Recursive function $\mathsf{cnd}(P)$, distributing conditions on sequences and collecting conditions on succeeding conditional policies.

Fact 4.2. *If B is a basic influencer policy of the form $(\varphi \Rightarrow \{u : o \leftarrow s\})$, then*

$$B \sqsubseteq B_1 \bullet \cdots \bullet B_n \bullet B_{n+1} \bullet \cdots \bullet B_m \textit{ iff } B \sqsubseteq B_1 \bullet \cdots \bullet B_n \tag{4.3}$$

else if B is a basic reader policy with the form $(\varphi \Rightarrow \{u : o \rightarrow \bar{s}\})$ then

$$B \sqsubseteq B_1 \bullet \cdots \bullet B_n \bullet B_{n+1} \bullet \cdots \bullet B_m \textit{ iff } B \sqsubseteq B_{n+1} \bullet \cdots \bullet B_m \tag{4.4}$$

where B_n is the last basic influencer policy and B_{n+1} is the first basic reader policy.
(proven by looking at the influencers and readers in Appendix A.11)

4.1.2 Properties of basic policies

The basic influencer policies have a special property which is the reverse of one of the features of the join-semilattice in (2.23).

Fact 4.3. *$(\varphi \Rightarrow \{u : o \leftarrow s\}) \sqsubseteq P_1 \bullet P_2$ if and only if $(\varphi \Rightarrow \{u : o \leftarrow s\}) \sqsubseteq P_1 \vee (\varphi \Rightarrow \{u : o \leftarrow s\}) \sqsubseteq P_2$*
(proven by looking at the influencers and readers in Appendix A.12)

Unfortunately, a similar result *does not* hold for basic reader policies, if either conjunction or disjunction is used.

Fact 4.4. *$(\varphi \Rightarrow \{u : o \rightarrow \bar{s}\}) \not\sqsubseteq P_1 \bullet P_2$ if and only if $(\varphi \Rightarrow \{u : o \rightarrow \bar{s}\}) \sqsubseteq P_1 \wedge (\varphi \Rightarrow \{u : o \rightarrow \bar{s}\}) \sqsubseteq P_2$*
(proven by looking at the influencers and readers in Appendix A.13)

Fact 4.5. *$(\varphi \Rightarrow \{u : o \rightarrow \bar{s}\}) \not\sqsubseteq P_1 \bullet P_2$ if and only if $(\varphi \Rightarrow \{u : o \rightarrow \bar{s}\}) \sqsubseteq P_1 \vee (\varphi \Rightarrow \{u : o \rightarrow \bar{s}\}) \sqsubseteq P_2$*
(proven by looking at the influencers and readers in Appendix A.13)

4.2 Preorder

The preorder $\sqsubseteq$ is used in the side conditions for the type system. Recall the definition of the preorder

$$\begin{aligned} P \sqsubseteq P' \text{ iff } & \forall \sigma, q, y : \mathsf{Infl}(P, \sigma, q, y) \subseteq \mathsf{Infl}(P', \sigma, q, y) \wedge \\ & \forall \sigma, q, y : \mathsf{Read}(P, \sigma, q, y) \supseteq \mathsf{Read}(P', \sigma, q, y) \end{aligned} \tag{2.14}$$

It can be difficult to explore all $\sigma \in \mathbf{State}^+$, $q \in \mathbf{Pr}$ and $y \in \mathbf{Var}^+$ manually, because policies can be complicated and the state space is large as all variables can have all values. To figure out if $P \sqsubseteq P'$, the policies can first be converted into SINF

$$\begin{aligned} P &\equiv \mathsf{sinf}(P) = B_1 \bullet \cdots \bullet B_n \bullet B_{n+1} \bullet \cdots \bullet B_m \\ P' &\equiv \mathsf{sinf}(P') = B'_1 \bullet \cdots \bullet B'_{n'} \bullet B'_{n'+1} \bullet \cdots \bullet B'_{m'} \end{aligned}$$

Where $B_n, B'_{n'}$ is the last basic influencer policy and $B_{n+1}, B'_{n'+1}$ is the first basic reader policy. Furthermore, it is assumed that $m > 0$. It therefore comes down to determining

$$B_1 \bullet \cdots \bullet B_n \bullet B_{n+1} \bullet \cdots \bullet B_m \sqsubseteq B'_1 \bullet \cdots \bullet B'_{n'} \bullet B'_{n'+1} \bullet \cdots \bullet B'_{m'} \tag{4.5}$$

Using that $(\mathbf{Pol}, \bullet, \sqsubseteq_{\equiv})$ is a join-semilattice (see Section 2.3.6) then

$$P_1 \bullet P_2 \sqsubseteq P_3 \quad \text{iff} \quad P_1 \sqsubseteq P_3 \wedge P_2 \sqsubseteq P_3 \tag{2.23}$$

The left-hand side of (4.5) can therefore be expanded to two predicates recursively until there is one policy on the left. A similar approach can be taken on the right-hand side with Fact 4.3, though only with the basic influencer policies. Together with (4.3) and (4.4) the ordering can be restated as

$$\begin{aligned} & B_1 \bullet \cdots \bullet B_n \bullet B_{n+1} \bullet \cdots \bullet B_m \sqsubseteq B'_1 \bullet \cdots \bullet B'_{n'} \bullet B'_{n'+1} \bullet \cdots \bullet B'_{m'} \\ & \text{iff } \left(\bigwedge_{1 \le i \le n} \bigvee_{1 \le j \le n'} B_i \sqsubseteq B'_j \right) \wedge \left(\bigwedge_{n+1 \le i \le m} B_i \sqsubseteq B'_{n'+1} \bullet \cdots \bullet B'_{m'} \right) \end{aligned} \tag{4.6}$$

The analysis then amounts to determining the relations of the basic policies.

Fact 4.6. *$(\varphi \Rightarrow \{u : o \leftarrow s\}) \sqsubseteq (\varphi' \Rightarrow \{u' : o' \leftarrow s'\})$ if $\phi \Rightarrow (\phi' \wedge u = u' \wedge o = o' \wedge s = s')$ is a tautology.*

Proof. First we look at the influencers for all σ, q, and y:

$$\begin{aligned}&\mathsf{Infl}\left((\varphi \Rightarrow \{u : o \leftarrow s\}), \sigma, q, y\right) \subseteq \mathsf{Infl}\left((\varphi' \Rightarrow \{u' : o' \leftarrow s'\}), \sigma, q, y\right)\\ &\quad= \begin{cases} s & \text{if } \varphi \models \sigma \wedge q = o \wedge y = u \\ \epsilon & \text{otherwise}\end{cases} \subseteq \begin{cases} s' & \text{if } \varphi' \models \sigma \wedge q = o' \wedge y = u' \\ \epsilon & \text{otherwise}\end{cases}\end{aligned}$$

Take a single σ, and look at the following four cases.

Case $\sigma \not\models \phi \wedge \sigma \not\models \phi'$. The influencers are ϵ on both sides, such that $\epsilon \subseteq \epsilon$. This case is always true for all q, and y.

Case $\sigma \not\models \phi \wedge \sigma \models \phi'$. The influencers on the left-hand side is always ϵ such that $\epsilon \subseteq \cdot$. This case is always true for all q, and y.

Case $\sigma \models \phi \wedge \sigma \not\models \phi'$. The influencers on the right-hand side are always ϵ. For this case to be true, there should for all q, y hold that $q \neq o$ or all $y \neq u$. Taking $q = o$ and $y = y$ this would not be true, and the case is therefore always false.

Case $\sigma \models \phi \wedge \sigma \models \phi'$. This case have two subcases. Firstly, if $q = o \wedge y = u$ and $q = o' \wedge y = u'$ then $s \subseteq s'$, which is only true if $s = s'$. Secondly, if $q \neq o$ or $y \neq u$ then the left-hand side will become ϵ such that $\epsilon \subseteq \cdot$ which is always true. The whole case is therefore true if $u = u' \wedge o = o' \wedge s = s'$ for all q and y.

For all σ we therefore have

$$\begin{aligned}&(\sigma \not\models \phi \wedge \sigma \not\models \phi') \vee (\sigma \not\models \phi \wedge \sigma \models \phi') \vee (\sigma \models \phi \wedge \sigma \models \phi' \wedge u = u' \wedge o = o' \wedge s = s')\\ &= \begin{array}{c}(\sigma \models \neg\phi \wedge \sigma \models \neg\phi') \vee (\sigma \models \neg\phi \wedge \sigma \models \phi') \vee \\ (\sigma \models \phi \wedge \sigma \models \phi' \wedge u = u' \wedge o = o' \wedge s = s')\end{array}\end{aligned}$$

leaving out $\sigma \models \phi \wedge \sigma \not\models \phi'$ because this case is false. Looking at all σ in $\sigma \models \phi''$ means that ϕ'' should be a tautology. The predicate can therefore be further reduced to the desired result

$$\begin{aligned}&= (\neg\phi \wedge \neg\phi') \vee (\neg\phi \wedge \phi') \vee (\phi \wedge \phi' \wedge u = u' \wedge o = o' \wedge s = s')\\ &= \phi \Rightarrow (\phi' \wedge u = u' \wedge o = o' \wedge s = s')\end{aligned}$$

The other part is to verify the reader set for all σ, q, and y

$$\begin{aligned}&\mathsf{Read}\left((\varphi \Rightarrow \{u : o \leftarrow s\}), \sigma, q, y\right) \subseteq \mathsf{Read}\left((\varphi' \Rightarrow \{u' : o' \leftarrow s'\}), \sigma, q, y\right)\\ &\quad= \mathbf{Pr} \supseteq \mathbf{Pr} = \mathsf{true}\end{aligned}$$

which finalises the proof. □

Simplifying the part with the basic reader policy is not covered in this thesis, partly because of the result in Fact 4.4, and that this relation now covers quantities with intersections because of the sets $\bar{s}$ and $\bar{s}'$. Furthermore $\bar{s}$ can also be empty leaving some special cases.

Conjecture 4.1. *$(\varphi \Rightarrow \{u : o \rightarrow \bar{s}\}) \sqsubseteq (\varphi'_1 \Rightarrow \{u'_1 : o'_1 \rightarrow \bar{s}'_1\}) \bullet \cdots \bullet (\varphi'_n \Rightarrow \{u'_n : o'_n \rightarrow \bar{s}'_n\})$ can be simplified.*

Proof idea. Making a similar cases analysis like in Fact 4.6. □

The implication normal can be just $\{\}$ (when $m = 0$), which leads six cases, the proofs of which are not covered here.

Conjecture 4.2.

$$\begin{aligned}
\{\} &\sqsubseteq \{\} && \textit{if } \mathsf{true} \textit{ is a tautology}\\
\{\} &\sqsubseteq (\varphi' \Rightarrow \{u' : o' \leftarrow s'\}) && \textit{if } \mathsf{true} \textit{ is a tautology}\\
\{\} &\sqsubseteq (\varphi' \Rightarrow \{u' : o' \rightarrow \bar{s}'\}) && \textit{if } \mathsf{true} \textit{ is a tautology}\\
(\varphi \Rightarrow \{u : o \leftarrow s\}) &\sqsubseteq \{\} && \textit{if } \neg\phi \textit{ is a tautology}\\
(\varphi \Rightarrow \{u : o \rightarrow \bar{s}\}) &\sqsubseteq \{\} && \textit{if } \neg\phi \textit{ is a tautology}
\end{aligned}$$

Proof idea. Simply by looking at the influencer and reader set, and similar cases analysis like in Fact 4.6. □

Because of the sorted implication normal form, the two last cases with a mixed basic influencer policy and basic reader policy are not used.

Fact 4.7. *$(\varphi \Rightarrow \{u : o \leftarrow s\}) \sqsubseteq (\varphi' \Rightarrow \{u' : o' \rightarrow \bar{s}'\})$ if ϕ is unsatisfiable.*
(proven by looking at the influencers and readers in Appendix A.14)

Fact 4.8. *$(\varphi \Rightarrow \{u : o \rightarrow \bar{s}\}) \sqsubseteq (\varphi' \Rightarrow \{u' : o' \leftarrow s'\})$ if ϕ is unsatisfiable or $\bar{s} = \star$.*
(proven by looking at the influencers and readers in Appendix A.15)

4.2.1 Simple assignment example

Let us continue with the example from Section 3.6. When we used the policy

$$P_{Sys} = P_l = \{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\}) \tag{2.44}$$

the system was not secure. With that policy the type system amounted to verify the following side condition

$$\overbrace{(x = 4 \Rightarrow (\{xy : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\})))}^{P_{\text{left}}} \sqsubseteq \underbrace{(\{x : s \leftarrow s\} \bullet (x > 5 \Rightarrow \{y : s \leftarrow s\}))}_{P_{\text{right}}}$$

Converting each side to INF gives

$$\begin{aligned}\mathsf{inf}(P_{\text{left}}) =& (x = 4 \Rightarrow \{x : s \leftarrow s\})\\ &\bullet (x = 4 \Rightarrow \{y : s \leftarrow s\})\\ &\bullet (x = 4 \land x > 5 \Rightarrow \{y : s \leftarrow s\})\\ \mathsf{inf}(P_{\text{right}}) =& (\mathsf{tt} \Rightarrow \{x : s \leftarrow s\})\\ &\bullet (x > 5 \Rightarrow \{y : s \leftarrow s\})\end{aligned}$$

and because it only contains influencer policies it is also in SINF. This leaves six comparisons

$$\begin{aligned}&(x = 4 \Rightarrow \{x : s \leftarrow s\}) \sqsubseteq (\mathsf{tt} \Rightarrow \{x : s \leftarrow s\})\\ &\quad = (x = 4) \Rightarrow ((\mathsf{tt}) \land (x = x) \land (s = s) \land (s = s)) && = \mathsf{tt} \text{ using } x = 4 \Rightarrow \mathsf{tt}\\ &(x = 4 \Rightarrow \{x : s \leftarrow s\}) \sqsubseteq (x > 5 \Rightarrow \{y : s \leftarrow s\})\\ &\quad = (x = 4) \Rightarrow ((x > 5) \land (x = y) \land (s = s) \land (s = s)) && = \mathsf{ff} \text{ using } x \neq y\\ &(x = 4 \Rightarrow \{y : s \leftarrow s\}) \sqsubseteq (\mathsf{tt} \Rightarrow \{x : s \leftarrow s\})\\ &\quad = (x = 4) \Rightarrow ((\mathsf{tt}) \land (y = x) \land (s = s) \land (s = s)) && = \mathsf{ff} \text{ using } y \neq x\\ &(x = 4 \Rightarrow \{y : s \leftarrow s\}) \sqsubseteq (x > 5 \Rightarrow \{y : s \leftarrow s\})\\ &\quad = (x = 4) \Rightarrow ((x > 5) \land (y = y) \land (s = s) \land (s = s)) && = \mathsf{ff} \text{ using } x = 4 \not\Rightarrow x > 5\\ &(x = 4 \land x > 5 \Rightarrow \{y : s \leftarrow s\}) \sqsubseteq (\mathsf{tt} \Rightarrow \{x : s \leftarrow s\})\\ &\quad = (x = 4 \land x > 5) \Rightarrow ((\mathsf{tt}) \land (y = x) \land (s = s) \land (s = s)) && = \mathsf{tt} \text{ using } \mathsf{ff} \Rightarrow \mathsf{ff}\\ &(x = 4 \land x > 5 \Rightarrow \{y : s \leftarrow s\}) \sqsubseteq (x > 5 \Rightarrow \{y : s \leftarrow s\})\\ &\quad = (x = 4 \land x > 5) \Rightarrow ((x > 5) \land (y = y) \land (s = s) \land (s = s)) && = \mathsf{ff} \text{ using } \mathsf{ff} \not\Rightarrow x > 5\end{aligned}$$

leaving the result $(\mathsf{tt} \lor \mathsf{ff}) \land (\mathsf{ff} \lor \mathsf{ff}) \land (\mathsf{tt} \lor \mathsf{ff}) = \mathsf{ff}$. The side condition therefore does not hold, making the system with this policy insecure, equivalent to the result in Section 3.6.

4.3 Loop invariants

Recall $[loop_{ts}]$ from the type system

$$\frac{\bar{y}\bar{b} \vdash_\ell \{\phi \land b\}S\{\phi\}}{\bar{y} \vdash_\ell \{\phi\}\mathsf{while}\ b\ \mathsf{do}\ S\ \mathsf{od}\{\phi \land \neg b\}}$$

where ϕ is an invariant (as it must hold before and after each iteration of the loop). Invariants are not easy to generate as they require deep understanding of the statements in S. Generating invariants are therefore not considered in this thesis. To ease the implementation, the invariants are introduced in the semantics for loops such that

$$\mathsf{while}\ b\ \mathsf{do}\ S\ \mathsf{od} = \mathsf{while}\ b\ \mathsf{do}\ \{\iota\}S\ \mathsf{od}$$

where ι is an invariant, syntactical equivalent to a boolean expression b. This gives the programmer the ability to specify an invariant ι for the iteration statement. With this change $[loop_{ts}]$ then becomes

$$[loop_{ts}]\ \frac{\bar{y}\bar{b} \vdash_\ell \{\phi \wedge b\}S\{\phi\}}{\bar{y} \vdash_\ell \{\phi\}\mathsf{while}\ b\ \mathsf{do}\ \{\iota\}S\ \mathsf{od}\{\phi \wedge \neg b\}} \tag{4.7}$$

4.4 Type checker

The implementation of the type system leads to a *type checker*. The type system does not aid in determining the postconditions ϕ'. Furthermore the rule of consequence $[cons_{ts}]$ can be applied at any statement S, and it is therefore not deterministic when to use this rule. To cope with this, the rule of consequence is incorporated into the relevant rules which then gives a syntax-directed type system.

Algorithm 4.4 defines the function $\mathsf{chk}(\bar{y}, \ell, \phi, S)$ for each statement S, which leads to the following soundness property for processes.

Lemma 4.1. *If* $\mathsf{chk}(\bar{y}, \ell, \phi, S) = (\phi', c)$ *and c is a tautology then* $\bar{y} \vdash_\ell \{\phi\}S\{\phi'\}$.

Proof. The proof is done by induction on the structure of S. All the axioms in the type system yields a base case and the induction hypothesis gives the desired result for the other inferences rules in the type system.

Case $\mathsf{chk}(\bar{y}, \ell, \phi, \mathsf{skip}) = (\phi, \mathsf{true})$**.** It is clear that true is always a tautology. The side conditions for $[skip_{ts}]$ are then $\phi \Rightarrow \phi$ which is also true, and therefore $\bar{y} \vdash_\ell \{\phi\}S\{\phi\}$.

Case $\mathsf{chk}(\bar{y}, \ell, \phi, x := a) = (\phi', c)$**.** Let $\phi' = (\exists x' : ((\phi[x'/x]) \wedge (x = a[x'/x])))$ and $c = (\phi \Rightarrow P_\ell\langle \bar{a}\bar{y}/x\rangle) \sqsubseteq P_\ell[a/x]$. The side conditions for $[ass_{ts}]$ then amount to

$$\begin{aligned}
&(\phi \Rightarrow (\exists x' : ((\phi[x'/x]) \wedge (x = a[x'/x])))[a/x]) \wedge c \\
&\quad = (\phi \Rightarrow (\exists x' : (\phi[x'/x] \wedge a = a[x'/x]))) \wedge c \quad && \text{using } x \notin \mathsf{fv}(\phi[x'/x]) \cup \mathsf{fv}(a[x'/x]) \\
&\quad = (\phi \Rightarrow \phi) \wedge c && \text{using } x' = x
\end{aligned}$$

If c is a tautology then $\bar{y} \vdash_\ell \{\phi\}x := a\{\phi'\}$.

Case $\mathsf{chk}(\bar{y}, \ell, \phi, x :=^\dagger a) = (\phi', c)$**.** Let ϕ' be the same as for $\mathsf{chk}(\bar{y}, \ell, \phi, x := a)$, and $c = (\phi \Rightarrow P_\ell\langle \bar{a}\bar{y}/x\rangle) \sqsubseteq P_\ell[a/x] \bullet \{x : \mathcal{S}(\ell) \leftarrow \star\} \bullet \{x : \mathcal{S}(\ell) \rightarrow \epsilon\}$. The side conditions for $[ass^\dagger_{ts}]$ then amount to c as they did for $\mathsf{chk}(\bar{y}, \ell, \phi, x := a)$. If c is a tautology then $\bar{y} \vdash_\ell \{\phi\}x :=^\dagger a\{\phi'\}$.

Case $\mathsf{chk}(\bar{y}, \ell, \phi, S_1; S_2) = (\phi', c_1 \wedge c_2)$**.** Let $(\phi'', c_1) = \mathsf{chk}(\bar{y}, \ell, \phi, S_1)$ and $(\phi', c_2) = \mathsf{chk}(\bar{y}, \ell, \phi'', S_2)$. Assuming $c_1 \wedge c_2$ is a tautology, the induction hypothesis gives $\bar{y} \vdash_\ell \{\phi\}S_1\{\phi''\}$ and $\bar{y} \vdash_\ell \{\phi''\}S_2\{\phi'\}$ which is the premise $[comb_{ts}]$, and we have that $\bar{y} \vdash_\ell \{\phi\}S_1; S_2\{\phi'\}$.

Case $\mathsf{chk}(\bar{y}, \ell, \phi, \mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}) = (\phi_1 \vee \phi_2, c_1 \wedge c_2)$**.** Let $(\phi_1, c_1) = \mathsf{chk}(\bar{y}\bar{b}, \ell, \phi \wedge b, S_1)$ and $(\phi_2, c_2) = \mathsf{chk}(\bar{y}\bar{b}, \ell, \phi \wedge \neg b, S_2)$. Assuming $c_1 \wedge c_2$ is a tautology, the induction hypothesis gives $\bar{y}\bar{b} \vdash_\ell \{\phi \wedge b\}S_1\{\phi_1\}$ and $\bar{y}\bar{b} \vdash_\ell \{\phi \wedge \neg b\}S_2\{\phi_2\}$.

Let $\phi' = \phi_1 \vee \phi_2$, and see that $\phi_1 \Rightarrow \phi'$ and $\phi_2 \Rightarrow \phi'$ is a tautology. Using $[cons_{ts}]$ we can now get the premises for $[if_{ts}]$

$$\dfrac{\dfrac{\bar{y}\bar{b} \vdash_\ell \{\phi \wedge b\}S_1\{\phi_1\}}{\bar{y}\bar{b} \vdash_\ell \{\phi \wedge b\}S_1\{\phi'\}}\ \text{if } \phi_1 \Rightarrow \phi' \quad \dfrac{\bar{y}\bar{b} \vdash_\ell \{\phi \wedge \neg b\}S_2\{\phi_2\}}{\bar{y}\bar{b} \vdash_\ell \{\phi \wedge \neg b\}S_2\{\phi'\}}\ \text{if } \phi_2 \Rightarrow \phi'}{\bar{y} \vdash_\ell \{\phi\}\mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}\{\phi'\}}$$

which finalises this case.

Case $\mathsf{chk}(\bar{y}, \ell, \phi, \mathsf{while}\ b\ \mathsf{do}\ \{\iota\}S\ \mathsf{od}) = (\iota \wedge \neg b, c \wedge (\phi' \Rightarrow \iota) \wedge (\phi \Rightarrow \iota))$**.** Let $(\phi', c) = \mathsf{chk}(\bar{y}\bar{b}, \ell, \iota \wedge b, S)$. Assuming c is a tautology, the induction hypothesis gives $\bar{y} \vdash_\ell \{\iota \wedge b\}S\{\phi'\}$. Using $[cons_{ts}]$ we can now get the premises for $[loop_{ts}]$

$$\dfrac{\dfrac{\dfrac{\bar{y}\bar{b} \vdash_\ell \{\iota \wedge b\}S\{\phi'\}}{\bar{y}\bar{b} \vdash_\ell \{\iota \wedge b\}S\{\iota\}}\ \text{if } (\phi' \Rightarrow \iota)}{\bar{y} \vdash_\ell \{\iota\}\mathsf{while}\ b\ \mathsf{do}\ \{\iota\}S\ \mathsf{od}\{\iota \wedge \neg b\}}}{\bar{y} \vdash_\ell \{\phi\}\mathsf{while}\ b\ \mathsf{do}\ \{\iota\}S\ \mathsf{od}\{\iota \wedge \neg b\}}\ \text{if } (\phi \Rightarrow \iota)$$

Assuming that $\phi \Rightarrow \iota$ and $\phi' \Rightarrow \iota$ are also tautologies, the case is done and we have that $\bar{y} \vdash_\ell \{\phi\}\mathsf{while}\ b\ \mathsf{do}\ \{\iota\}S\ \mathsf{od}\{\iota \wedge \neg b\}$.

Case $\mathsf{chk}(\bar{y}, \ell, \phi, ch!(a_1, \ldots, a_k)) = (\phi, c)$**.** Let $c = (\phi \Rightarrow P_\ell\langle \bar{a}_i\bar{y}/\#_i\rangle_{i \le k}) \sqsubseteq P_{ch}[a_i/\#_i]_{i \le k} \bullet \{\star_\ell : \star \leftarrow \star\} \bullet \{\star_\ell : \star \to \epsilon\}$. The side conditions for $[out_{ts}]$ are then $(\phi \Rightarrow \phi) \wedge c$. If c is a tautology then the side conditions hold and $\bar{y} \vdash_\ell \{\phi\}ch!(a_1, \ldots, a_k)\{\phi\}$.

Case $\mathsf{chk}(\bar{y}, \ell, \phi, ch!^\dagger(a_1, \ldots, a_k)) = (\phi, c)$**.** Let $c = (\phi \Rightarrow P_\ell\langle \bar{a}_i\bar{y}/\#_i\rangle_{i \le k}) \sqsubseteq [a_i/\#_i]_{i \le k} \bullet \{\star_\# : \mathcal{S}(\ell) \leftarrow \star\} \bullet \{\star_\# : \mathcal{S}(\ell) \to \epsilon\} \bullet \{\star_\ell : \star \leftarrow \star\} \bullet \{\star_\ell : \star \to \epsilon\}$. The side conditions for $[out^\dagger_{ts}]$ are then $(\phi \Rightarrow \phi) \wedge c$. If c is a tautology then the side conditions hold and $\bar{y} \vdash_\ell \{\phi\}ch!^\dagger(a_1, \ldots, a_k)\{\phi\}$.

Case $\mathsf{chk}(\bar{y}, \ell, \phi, ch?(x_1, \ldots, x_k)) = (\phi', c)$**.** Let $\phi' = (\exists x'_1, \ldots, x'_k : (\phi[x'_i/x_i]_{i \le k}))$ and $c = (\phi \Rightarrow P_\ell\langle \bar{y}/x_i\rangle_{i \le k}) \bullet P_{ch}\langle \#_i/x_i\rangle_{i \le k} \sqsubseteq P_\ell[\#_i/x_i]_{i \le k} \bullet \{\star_\# : \star \leftarrow \star\} \bullet \{\star_\# : \star \to \epsilon\}$. The side conditions for $[in_{ts}]$ are then

$$\begin{aligned} &((\exists x_1, \ldots, x_k : (\phi)) \Rightarrow (\exists x'_1, \ldots, x'_k : (\phi[x'_i/x_i]_{i \le k}))) \wedge c \\ &\quad = c \qquad\qquad\qquad\qquad \text{using } x'_i = x_i \text{ for all } i \end{aligned}$$

If c is a tautology then $\bar{y} \vdash_\ell \{\phi\}ch?(x_1, \ldots, x_k)\{\phi'\}$.

Case $\mathsf{chk}(\bar{y}, \ell, \phi, ((S_1) \oplus (S_2))) = (\phi_1 \vee \phi_2, c_1 \wedge c_2)$**.** Let $(\phi_1, c_1) = \mathsf{chk}(\bar{y}, \ell, \phi, S_1)$ and $(\phi_2, c_2) = \mathsf{chk}(\bar{y}, \ell, \phi, S_2)$. Assuming $c_1 \wedge c_2$ is a tautology, the induction hypothesis gives that $\bar{y} \vdash_\ell \{\phi\}S_1\{\phi_1\}$ and $\bar{y} \vdash_\ell \{\phi\}S_2\{\phi_2\}$. Let $\phi' = \phi_1 \vee \phi_2$ and see that $\phi_1 \Rightarrow \phi'$ and $\phi_2 \Rightarrow \phi'$ is a tautology. Using $[cons_{ts}]$ we can now get the premises for $[cho_{ts}]$

$$\frac{\dfrac{\bar{y} \vdash_\ell \{\phi\}S_1\{\phi_1\}}{\bar{y} \vdash_\ell \{\phi\}S_1\{\phi'\}} \text{ if } \phi_1 \Rightarrow \phi' \quad \dfrac{\bar{y} \vdash_\ell \{\phi\}S_2\{\phi_2\}}{\bar{y} \vdash_\ell \{\phi\}S_2\{\phi'\}} \text{ if } \phi_2 \Rightarrow \phi'}{\bar{y} \vdash_\ell \{\phi\}((S_1) \oplus (S_2))\{\phi'\}}$$

which finalises this case.

Case $\mathsf{chk}(\bar{y}, \ell, \phi, \lceil \bar{x} \rceil S) = (\phi, c)$. Let $(\phi', c) = \mathsf{chk}(\bar{y}\bar{x}, \ell, \phi, S)$. Assuming c is a tautology, the induction hypothesis gives $\bar{y}\bar{x} \vdash_\ell \{\phi\}S\{\phi'\}$ which are the premises for $[impl_{ts}]$ and we have that $\bar{y} \vdash_\ell \{\phi\}\lceil \bar{x} \rceil S\{\phi'\}$.

□

A similar soundness property can be found for systems.

Theorem 4.1. *If $\mathsf{chk}(\epsilon, l_i, \phi_i, S_i) = (\phi'_i, c_i)$ for all $i \leq n$ and $(c_1 \wedge \ldots \wedge c_n)$ is a tautology then $\vdash \{\phi_1 \& \cdots \& \phi_n\}\ l_1 : S_1 \parallel \cdots \parallel l_n : S_n$.*
(proven using directly by using Lemma 4.1 Appendix A.16)

Let $\Phi(\ell) = \phi$ be a function mapping each label ℓ in a distributed system Sys to its initial precondition. Recall a fully defined system $(Sys, \mathcal{S}, \mathsf{NSA}, P_{Sys})$ and the function for verifying it is then

$$\begin{aligned} \mathsf{verify}(Sys, \mathcal{S}, \mathsf{NSA}, P_{Sys}, \Phi) &= c_1 \wedge \cdots \wedge c_i \wedge \cdots \wedge c_n \text{ where} \\ (\phi'_i, c_i) &= \mathsf{chk}(\epsilon, l_i, \Phi(l_i), S_i) \quad \text{for each label } l_i \text{ in } Sys \end{aligned}$$

If the result of $\mathsf{verify}(Sys, \mathcal{S}, \mathsf{NSA}, P_{Sys}, \Phi)$ is a tautology, the distributed system Sys is secure according to the policies P_{Sys} and the initial preconditions Φ. Otherwise it is not secure and there is an illegal information flow.

When a predicate is satisfiable it is true under at least one configuration, and therefore a tautology if the negated predicate is unsatisfiable. Using the verifier function $c = \mathsf{verify}(Sys, \mathcal{S}, \mathsf{NSA}, P_{Sys}, \Phi)$ where c should be a tautology and thus it is to figure out if $\neg c$ is unsatisfiable. As we have seen in some of the previous examples such a side condition can look like

$$c = (x = 3) \Rightarrow (x > 2 \wedge (\exists y : (y < x))) \tag{4.8}$$

and a regular SAT solver can hence not be used, because this is not a regular boolean predicate. Instead a STM solver is needed to determine if $\neg c$ is unsatisfiable, as it can determine relations on structures such as integers.

Together with the soundness of the type checker is also the completeness result (in sense of Cook). The proof is not shown here, as it can be difficult to establish.

Conjecture 4.3. *If $\bar{y} \vdash_\ell \{\phi\}S\{\phi'\}$ then there exists ϕ'' and c such that $\mathsf{chk}(\bar{y}, \ell, \phi, S) = (\phi, c)$, and $\phi'' \Rightarrow \phi'$ and $\models c$.*

Proof idea. Proof by induction on the inference system, in relation to the restrictions of ϕ and c. □

```
$\mathsf{chk}(\bar{y}, \ell, \phi, \mathsf{skip}) = (\phi, \mathsf{true})$

$\mathsf{chk}(\bar{y}, \ell, \phi, x := a) = (\phi', c)$
  where        $\phi' = (\exists x' : ((\phi[x'/x]) \wedge (x = a[x'/x])))$
               $c = (\phi \Rightarrow P_\ell\langle \bar{a}\bar{y}/x\rangle) \sqsubseteq P_\ell[a/x]$

$\mathsf{chk}(\bar{y}, \ell, \phi, x :=^\dagger a) = (\phi', c)$
  where        $\phi' = (\exists x' : ((\phi[x'/x]) \wedge (x = a[x'/x])))$
               $c = (\phi \Rightarrow P_\ell\langle \bar{a}\bar{y}/x\rangle) \sqsubseteq P_\ell[a/x] \bullet \{x : \mathcal{S}(\ell) \leftarrow \star\} \bullet \{x : \mathcal{S}(\ell) \rightarrow \epsilon\}$

$\mathsf{chk}(\bar{y}, \ell, \phi, S_1; S_2) = (\phi', c_1 \wedge c_2)$
  where $(\phi'', c_1) = \mathsf{chk}(\bar{y}, \ell, \phi, S_1)$
        $(\phi', c_2) = \mathsf{chk}(\bar{y}, \ell, \phi'', S_2)$

$\mathsf{chk}(\bar{y}, \ell, \phi, \mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}) = (\phi_1 \vee \phi_2, c_1 \wedge c_2)$
  where $(\phi_1, c_1) = \mathsf{chk}(\bar{y}\bar{b}, \ell, \phi \wedge b, S_1)$
        $(\phi_2, c_2) = \mathsf{chk}(\bar{y}\bar{b}, \ell, \phi \wedge \neg b, S_2)$

$\mathsf{chk}(\bar{y}, \ell, \phi, \mathsf{while}\ b\ \mathsf{do}\ \{\iota\}S\ \mathsf{od}) = (\iota \wedge \neg b, c \wedge (\phi' \Rightarrow \iota) \wedge (\phi \Rightarrow \iota))$
  where   $(\phi', c) = \mathsf{chk}(\bar{y}\bar{b}, \ell, \iota \wedge b, S)$

$\mathsf{chk}(\bar{y}, \ell, \phi, ch!(a_1, \ldots, a_k)) = (\phi, c)$
  where        $c = (\phi \Rightarrow P_\ell\langle \bar{a}_i\bar{y}/\#_i\rangle_{i \leq k}) \sqsubseteq P_{ch}[a_i/\#_i]_{i \leq k} \bullet \{\star_\ell : \star \leftarrow \star\} \bullet \{\star_\ell : \star \rightarrow \epsilon\}$

$\mathsf{chk}(\bar{y}, \ell, \phi, ch!^\dagger(a_1, \ldots, a_k)) = (\phi, c)$
  where        $c = (\phi \Rightarrow P_\ell\langle \bar{a}_i\bar{y}/\#_i\rangle_{i \leq k}) \sqsubseteq [a_i/\#_i]_{i \leq k} \bullet \{\star_\# : \mathcal{S}(\ell) \leftarrow \star\}$
                    $\bullet \{\star_\# : \mathcal{S}(\ell) \rightarrow \epsilon\} \bullet \{\star_\ell : \star \leftarrow \star\} \bullet \{\star_\ell : \star \rightarrow \epsilon\}$

$\mathsf{chk}(\bar{y}, \ell, \phi, ch?(x_1, \ldots, x_k)) = (\phi', c)$
  where        $\phi' = (\exists x'_1, \ldots, x'_k : (\phi[x'_i/x_i]_{i \leq k}))$
               $c = (\phi \Rightarrow P_\ell\langle \bar{y}/x_i\rangle_{i \leq k}) \bullet P_{ch}\langle \#_i/x_i\rangle_{i \leq k} \sqsubseteq P_\ell[\#_i/x_i]_{i \leq k}$
                    $\bullet \{\star_\# : \star \leftarrow \star\} \bullet \{\star_\# : \star \rightarrow \epsilon\}$

$\mathsf{chk}(\bar{y}, \ell, \phi, ((S_1) \oplus (S_2))) = (\phi_1 \vee \phi_2, c_1 \wedge c_2)$
  where $(\phi_1, c_1) = \mathsf{chk}(\bar{y}, \ell, \phi, S_1)$
        $(\phi_2, c_2) = \mathsf{chk}(\bar{y}, \ell, \phi, S_2)$

$\mathsf{chk}(\bar{y}, \ell, \phi, \lceil \bar{x} \rceil S) = (\phi', c)$
  where   $(\phi', c) = \mathsf{chk}(\bar{y}\bar{x}, \ell, \phi, S)$
```

Algorithm 4.4: Recursive function $\mathsf{chk}(\bar{y}, \ell, \phi, S)$ for type checking.

4.5 Concrete implementation

All the algorithms and auxiliary functions such as substitutions use matching on the structure, and a functional programming language is a good choice for the concrete implementation. The functional programming language Standard ML (SML) [Mil+97] which is compiled and executed with the Standard ML of New Jersey compiler [SML/NJ] has therefore been used.

An actual lexer or parser for the presented language has not been developed, and S, a, b, P, ϕ, a^+, and c are therefore datatypes used to build up an abstract syntax tree. This is only a proof-of-concept implementation and only influencer policies are considered, which fits well with the poorly defined Conjecture 4.1 for simplifying the part with the basic reader policies in the preorder. With this restriction the final algorithms for deciding the preorder $P \sqsubseteq P'$ are defined in Algorithm 4.5.

Z3 is a STM solver supporting arithmetic expressions and quantifiers [Z3]. This solver is therefore used to verify if the result $c = \mathsf{verify}(Sys, \mathcal{S}, \mathsf{NSA}, P_{Sys}, \Phi)$ is a tautology and thereby whether $\neg c$ is unsatisfiable. Verifying c from (4.8) in Z3 is written as

```
(declare-const x Int)
(assert (not
    (implies (= x 3) (and (> x 2) (exists ((y Int)) (< y x))))
))
(check-sat)
```

which gives the result `unsat` for unsatisfied.

As seen in Algorithm 4.5 the preorder $P \sqsubseteq P'$ is simplified to only contain the assertions ϕ, variable equalities $u = u'$ and equality of security principals $o = o'$ and $s = s'$. In implemented syntax for the side conditions c therefore gives

$$\begin{aligned} c \ ::= \ & \mathsf{true} \mid \mathsf{false} \mid \neg c \mid c_1 \wedge c_2 \mid c_1 \vee c_2 \\ \mid \ & a_1^+ \ rel \ a_2^+ \mid (\exists \bar{u} : (c)) \mid c_1 \Rightarrow c_2 \mid s_1 = s_2 \end{aligned}$$

only changing $P^1 \sqsubseteq P^2$ to equality of security predicates $s_1 = s_2$ as this is the only extra predicates needed for the simplification.

A more through description of the development of the type checker and how to use it can be found in Appendix B.

$$\mathsf{order}(B_1 \bullet \ldots \bullet B_n, B'_1 \bullet \ldots \bullet B'_{n'}) = \left(\bigwedge_{1 \le i \le n} \bigvee_{1 \le j \le n'} \mathsf{order}(B_i, B'_j) \right)$$

$$\mathsf{order}((\varphi \Rightarrow \{u : o \leftarrow s\}), (\varphi' \Rightarrow \{u' : o' \leftarrow s'\})) = \phi \Rightarrow (\phi' \wedge u = u' \wedge o = o' \wedge s = s')$$

$$\mathsf{order}(\{\}, \{\}) = \mathsf{true}$$

$$\mathsf{order}(\{\}, (\varphi' \Rightarrow \{u' : o' \leftarrow s'\})) = \mathsf{true}$$

$$\mathsf{order}((\varphi \Rightarrow \{u : o \leftarrow s\}), \{\}) = \neg\phi$$

$$\mathsf{order}(P, P') = \mathsf{order}(\mathsf{inf}(P), \mathsf{inf}(P'))$$

Algorithm 4.5: Function $\mathsf{order}(P, P')$ for deciding the preorder $P \sqsubseteq P'$. In all cases B and B' are basic influencer policies.

CHAPTER 5
Avionics scenarios

sing the implementation, it is now possible to type check non-trivial systems which spring from the avionics industry. The first scenario is the gateway presented previously.

The other scenario is of great relevance to the avionics industry and constructed with help from PhD student Ximeng Li from DTU Compute. The scenario is fictive and yet realistic according to some of the security problems in the airplanes. This is a much more complex example than the gateway and have therefore not been type checked, but yet it reveals a more plausible case from the industry and the description of the system is therefore included motivating further usage of the type checker.

5.1 Gateway

Recall the gateway example from Section 1.3 which are shown again in Figure 5.1. Here there are two producers p_1 and p_2 sending data to the multiplexer m using the channels in_1 and in_2. The multiplexer m then send it to the demultiplexer d using only ch. Then d demultiplex the data and send it the the consumers c_1 and c_2, while respecting that the data from p_1 only reach c_1 and respectively for c_2.

The code for all the processes are:

$$\begin{aligned}
Sys = \quad & \mathsf{p}_1 : \mathsf{while\ true\ do}\ \{\mathsf{true}\}\mathsf{in}_1!1\ \mathsf{od} \\
\| \quad & \mathsf{p}_2 : \mathsf{while\ true\ do}\ \{\mathsf{true}\}\mathsf{in}_2!2\ \mathsf{od} \\
\| \quad & \mathsf{m} : \mathsf{while\ true\ do}\ \{\mathsf{true}\}((\mathsf{in}_1?x_1; \mathsf{ch}!(1,x_1)) \oplus (\mathsf{in}_2?x_2; \mathsf{ch}!(2,x_2)))\ \mathsf{od} \\
\| \quad & \mathsf{d} : \mathsf{while\ true\ do}\ \{\mathsf{true}\}\mathsf{ch}?(y,z); \mathsf{if}\ y = 1\ \mathsf{then\ out}_1!z\ \mathsf{else\ out}_2!z\ \mathsf{fi\ od} \\
\| \quad & \mathsf{c}_1 : \mathsf{while\ true\ do}\ \{\mathsf{true}\}\mathsf{out}_1?w_1\ \mathsf{od} \\
\| \quad & \mathsf{c}_2 : \mathsf{while\ true\ do}\ \{\mathsf{true}\}\mathsf{out}_2?w_2\ \mathsf{od}
\end{aligned}$$

It is noted that each process has its own set of variables $\mathbf{Var}_\ell$.

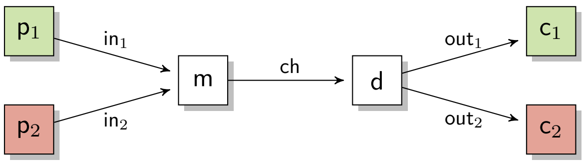

Figure 5.1: Gateway [NNL15].

As the processes p_1 and c_1 will communicate with each other, it makes sense that they belong to the same security principal, and vice versa for p_2 and c_2. The security principals for all the processes are then

$$\mathcal{S}(\mathsf{p}_1) = s_1 \quad \mathcal{S}(\mathsf{p}_2) = s_2 \quad \mathcal{S}(\mathsf{m}) = m \quad \mathcal{S}(\mathsf{d}) = d \quad \mathcal{S}(\mathsf{c}_1) = s_1 \quad \mathcal{S}(\mathsf{c}_2) = s_2$$

where our special security principal is $\mathsf{NSA} \in \mathbf{Pr}$ as usual.

We are now ready to define all the policies for the system. The two producers do not contain any variables and thus have a simple policy.

$$\begin{aligned} P_{\mathsf{p}_1} &= \{\} \\ P_{\mathsf{p}_2} &= \{\} \end{aligned}$$

The incoming values from p_1 may only be influenced by s_1 and conversely for p_2.

$$\begin{aligned} P_{\mathsf{m}} = \{&x_1 : m, s_1 \leftarrow s_1\} \\ &\bullet \{x_2 : m, s_2 \leftarrow s_2\} \end{aligned}$$

As y is the determining value for the destination in the demultiplexer, this must only be influenced by m. We now use the conditional policies to have two different policies for z.

$$\begin{aligned} P_{\mathsf{d}} = \{&y : m, d \leftarrow m\} \\ &\bullet (y = 1 \Rightarrow \{z : m, d, s_1 \leftarrow m, d, s_1\}) \\ &\bullet (y = 2 \Rightarrow \{z : m, d, s_2 \leftarrow m, d, s_2\}) \end{aligned}$$

The final process policy is for the consumers, that ensure that w_1 only is influenced by s_1, m and d, and likewise for w_2 which can be influenced by s_2, m and d.

$$\begin{aligned} P_{\mathsf{c}_1} &= \{w_1 : m, d, s_1 \leftarrow m, d, s_1\} \\ P_{\mathsf{c}_2} &= \{w_2 : m, d, s_2 \leftarrow m, d, s_2\} \end{aligned}$$

The channel policies are defined in a similar fashion where in_1 and in_2 respectively are influenced by s_1 and s_2.

$$\begin{aligned} P_{\mathsf{in}_1} &= \{\star_{\#} : m, s_1 \leftarrow s_1\} \\ P_{\mathsf{in}_2} &= \{\star_{\#} : m, s_2 \leftarrow s_2\} \end{aligned}$$

The main channel have two different policies for the second channel variable which depends on the value for the first channel variable, and the conditional policies are thus used again.

$$\begin{aligned} P_{\mathsf{ch}} = \{&\#_1 : m, d \leftarrow m\} \\ &\bullet (\#_1 = 1 \Rightarrow \{\#_2 : m, d, s_1 \leftarrow s_1\}) \\ &\bullet (\#_1 = 2 \Rightarrow \{\#_2 : m, d, s_2 \leftarrow s_2\}) \end{aligned}$$

At last is the policies for the output channel, which are equivalent to the policies for w_1 and w_2.

$$\begin{aligned} P_{\mathsf{out}_1} &= \{\star_{\#} : m, d, s_1 \leftarrow m, d, s_1\} \\ P_{\mathsf{out}_2} &= \{\star_{\#} : m, d, s_2 \leftarrow m, d, s_2\} \end{aligned}$$

It is seen that all the policies are localised, and the full system policy amounts to

$$P_{Sys} = P_{\mathsf{p}_1} \bullet P_{\mathsf{p}_2} \bullet P_{\mathsf{m}} \bullet P_{\mathsf{d}} \bullet P_{\mathsf{c}_1} \bullet P_{\mathsf{c}_2} \bullet P_{\mathsf{in}_1} \bullet P_{\mathsf{in}_2} \bullet P_{\mathsf{ch}} \bullet P_{\mathsf{out}_1} \bullet P_{\mathsf{out}_2}$$

which results in a fully defined system $(Sys, \mathcal{S}, \mathsf{NSA}, P_{Sys})$.

5.1.1 Type checking the gateway

Let $\Phi(\ell) = \mathsf{true}$ for all processes ℓ in Sys. By using the implemented type checker the systems is not considered secure

$$\mathsf{verify}(Sys, \mathcal{S}, \mathsf{NSA}, P_{Sys}, \Phi) \neq \text{Tautology} \tag{5.1}$$

A further investigation reveals that it is the process d which are not secure.

$$\begin{aligned} \mathsf{chk}(\epsilon, \mathsf{p}_1, \mathsf{true}, S_{\mathsf{p}_1}) &= (c_{\mathsf{p}_1}, \phi'_{\mathsf{p}_1}) && \text{where } c_{\mathsf{p}_1} = \text{Tautology} \\ \mathsf{chk}(\epsilon, \mathsf{p}_2, \mathsf{true}, S_{\mathsf{p}_2}) &= (c_{\mathsf{p}_2}, \phi'_{\mathsf{p}_2}) && \text{where } c_{\mathsf{p}_2} = \text{Tautology} \\ \mathsf{chk}(\epsilon, \mathsf{m}, \mathsf{true}, S_{\mathsf{m}}) &= (c_{\mathsf{m}}, \phi'_{\mathsf{m}}) && \text{where } c_{\mathsf{m}} = \text{Tautology} \\ \mathsf{chk}(\epsilon, \mathsf{d}, \mathsf{true}, S_{\mathsf{d}}) &= (c_{\mathsf{d}}, \phi'_{\mathsf{d}}) && \text{where } c_{\mathsf{d}} \neq \text{Tautology} \\ \mathsf{chk}(\epsilon, \mathsf{c}_1, \mathsf{true}, S_{\mathsf{c}_1}) &= (c_{\mathsf{c}_1}, \phi'_{\mathsf{c}_1}) && \text{where } c_{\mathsf{c}_1} = \text{Tautology} \\ \mathsf{chk}(\epsilon, \mathsf{c}_2, \mathsf{true}, S_{\mathsf{c}_2}) &= (c_{\mathsf{c}_2}, \phi'_{\mathsf{c}_2}) && \text{where } c_{\mathsf{c}_2} = \text{Tautology} \end{aligned}$$

The first execution in d is $\mathsf{ch}?(y, z)$ where the type checker makes the following caparison of policies

$$\begin{aligned} &\left(\mathsf{true} \wedge \mathsf{true} \Rightarrow \begin{array}{l} \{y : m, d \leftarrow m\} \bullet (y = 1 \Rightarrow \{z : m, d, s_1 \leftarrow m, d, s_1\}) \\ \qquad\qquad\qquad\; \bullet\, (y = 2 \Rightarrow \{z : m, d, s_2 \leftarrow m, d, s_2\}) \end{array}\right) \\ &\bullet \{\#_1, y : m, d \leftarrow m\} \\ &\bullet (\#_1 = 1 \Rightarrow \{\#_2, z : m, d, s_1 \leftarrow s_1\}) \\ &\bullet (\#_1 = 2 \Rightarrow \{\#_2, z : m, d, s_2 \leftarrow s_2\}) \\ &\quad \sqsubseteq \\ &\quad\; \{y : m, d \leftarrow m\} \\ &\quad\; \bullet (\#_1 = 1 \Rightarrow \{z : m, d, s_1 \leftarrow m, d, s_1\}) \\ &\quad\; \bullet (\#_1 = 2 \Rightarrow \{z : m, d, s_2 \leftarrow m, d, s_2\}) \\ &\quad\; \bullet \{\#_1, \#_2 : s_1, s_2, m, d, \mathsf{NSA} \leftarrow s_1, s_2, m, d, \mathsf{NSA}\} \end{aligned}$$

This comparison fails because of the policies concerning z. Some policies on the left-hand side have conditions including y and on the right-hand side the conditions includes only $\#_1$. Using Algorithm 4.5 gives among others the following basic policy relation

$$\begin{aligned}(y = 1 \Rightarrow \{z : s_1 \leftarrow s_1\}) &\sqsubseteq (\#_1 = 1 \Rightarrow \{z : s_1 \leftarrow s_1\}) \\ &= (y = 1) \Rightarrow ((\#_1 = 1) \wedge (z = z) \wedge (s_1 = s_1) \wedge (s_1 = s_1)) \\ &= y = 1 \Rightarrow \#_1 = 1\end{aligned}$$

Intuitively this is true as y is assigned in the first channel variable but the type system have no notion that these two variables are equivalent. It therefore seems to be a false positive.

Recall the policy part of the side condition for $[in_{ts}]$

$$\underbrace{(\phi \Rightarrow P_\ell\langle \bar{y}/x_i\rangle_{i\leq k})}_{\text{(a)}_1} \bullet \underbrace{P_{ch}\langle \#_i/x_i\rangle_{i\leq k}}_{\text{(a)}_2} \sqsubseteq \underbrace{P_\ell[\#_i/x_i]_{i\leq k}}_{\text{(b)}} \bullet \underbrace{\{\star_\# : \star \leftarrow \star\} \bullet \{\star_\# : \star \rightarrow \epsilon\}}_{\text{(d)}}$$

If $\ell = \mathsf{d}$ and $\phi = \mathsf{true} \wedge \mathsf{true}$ we see that this is indeed is the policy the type checker also found. Here the variables used in conditions for the conditional policies in $(a)_1$ are from $\mathbf{Var}_\ell$, and conditions in $(a)_2$ are from $\mathbf{Var}_\#$, and conditions in (b) are from $\mathbf{Var}_\#$. If there are conditional policies in P_ℓ and P_{ch} the problem we have encountered will thus be there again.

The solution of this false positive is then to modify the underlying type system, where all the conditions should be substituted to either process variables or channel variables, where the former is exemplified here

$$\underbrace{(\phi \Rightarrow P_\ell\langle \bar{y}/x_i\rangle_{i\leq k})}_{\text{(a)}_1} \bullet \underbrace{(P_{ch}\langle \#_i/x_i\rangle_{i\leq k})[x_i/\#_i]_{i\leq k}}_{\text{(a')}_2} \sqsubseteq \underbrace{P_\ell}_{\text{(b')}} \bullet \underbrace{\{\star_\# : \star \leftarrow \star\} \bullet \{\star_\# : \star \rightarrow \epsilon\}}_{\text{(d)}}$$

A quick-fix in the concrete implementation reveals that this indeed works and that d becomes secure. Left is though to proof Lemma 3.1 with this change, which is not easy and have thus not revealed any profitable results. It is therefore still an open question whether the suggested change actually follows Lemma 3.1, or if something other have to be done.

5.2 Electronic flight bag

When the airplane is on ground, map and charts for the current flight is loaded into an *electronic flight bag* which is then positioned in the AIS domain. Obviously these maps are used for navigation in the cockpit (AC domain) but they are also used by the passengers in the entertainment system (PIES domain) to show the location of the flight.

The actual position is measured using satellite-based systems (e.g. GPS) and ground-based systems (e.g. long range radio navigation). The input from these sensors is in the AC domain which is the most restrictive domain.

The information flow for the maps and position information is sketched in Figure 5.2. Here there are processes in all of the five security domains, with their corresponding channels. Obviously the positioning and map information should be viewable for both the passenger and the pilot, but the passenger should not be allowed

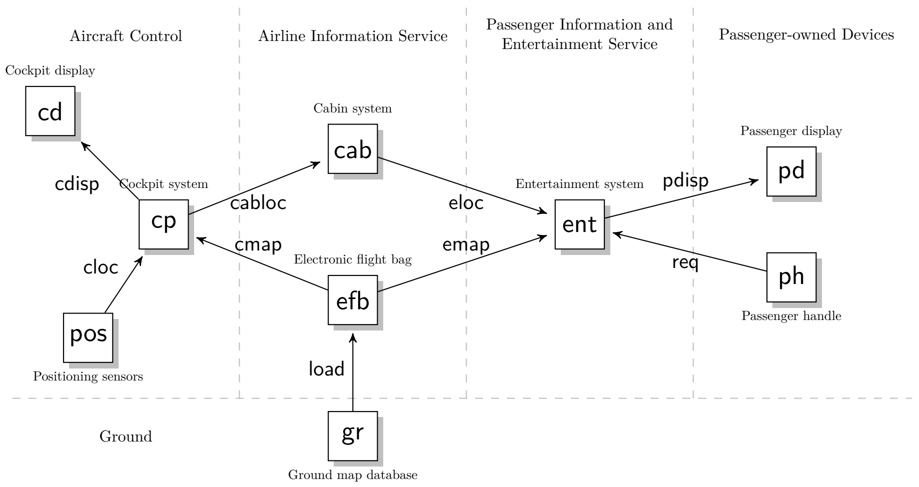

Figure 5.2: Involved processes in the distribution of location and map data.

to influence what is on the pilots screen. The challenge is therefore to annotate the different data sources with strong policies and still ensure that the passenger can not influence the cockpit display.

The processes gr, ph and pos are simple producers which sends map data, passenger requests, and the position of the flight respectively. The processes cd and ph are displays (consumers), only showing whatever they receive from their channels. The cockpit system cp continuously request the position from the sensors, and either combine it with the map and sent it to the cockpit display cd, or only sends the position to the cabin system cab. The electronic flight back efb either sends the map to the cockpit system or the entertainment system ent. The cabin system simply forwards all positioning information to the entertainment system. At last the entertainment system gets requests from the passenger handle ph, and relatively to the request either shows the combined map and position information or something else like a movie.

CHAPTER 6

Discussion

During the design and implementation different observations, errors and opportunities for improvements were found. This chapter illuminates and discusses these findings, together with alternative methods and possible extensions for different parts.

6.1 Predicates

There are multiple predicates used in the design with the following syntactic categories

$b \in \mathbf{BExp}$	boolean expressions
$\iota \in \mathbf{Invar}$	invariants
$\varphi \in \mathbf{PolCond}$	policy conditions
$\phi \in \mathbf{Asser}$	assertions (preconditions and postconditions)
$c \in \mathbf{Cond}$	side conditions

and the following syntax

$$\begin{array}{rcl}
a & ::= & n \mid x \mid a_1 \; op \; a_2 \\
a^+ & ::= & n \mid x \mid \# \mid a_1^+ \; op \; a_2^+ \\
b & ::= & \mathsf{true} \mid \mathsf{false} \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \\
 & \mid & a_1 \; rel \; a_2 \\
\iota & ::= & b \\
\varphi & ::= & \mathsf{true} \mid \mathsf{false} \mid \neg \varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \\
 & \mid & a_1^+ \; rel \; a_2^+ \mid (\exists \bar{u} : (\varphi)) \\
\phi & ::= & \mathsf{true} \mid \mathsf{false} \mid \neg \phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \\
 & \mid & a_1^+ \; rel \; a_2^+ \mid (\exists \bar{u} : (\phi)) \\
c & ::= & \mathsf{true} \mid \mathsf{false} \mid \neg c \mid c_1 \wedge c_2 \mid c_1 \vee c_2 \\
 & \mid & a_1^+ \; rel \; a_2^+ \mid (\exists \bar{u} : (c)) \mid c_1 \Rightarrow c_2 \mid P^1 \sqsubseteq P^2
\end{array}$$

It can be seen that **BExp** = **Invar** and **PolCond** = **Asser**, i.e. they respectively express the same language. Furthermore,

$$\mathbf{BExp} \subset \mathbf{PolCond} \subset \mathbf{Cond} \tag{6.1}$$

illustrating that there is an increased complexity in the predicates going from boolean expressions to side conditions. This emphasises that boolean expressions and invariants can be used in assertions, and that assertions can be used as policy conditions. Furthermore, if we want to increase the complexity of the assertion language ϕ, we must likewise increase it for policy conditions φ.

6.2 Blowup in policy predicates

Using Algorithm 4.5 on large policies results in a huge predicate due to the distribution of the predicates and the comparison of the basic policies. Let $|P|$ denote the number of policies inside P. Take a policy $P = (\phi \Rightarrow \{\bar{x}_k : \bar{o}_n \leftarrow \bar{s}_m\})$, where the size of $\bar{x}_k$ is k, $\bar{o}_n$ is n and $\bar{s}_m$ is m, and $|P| = 1$. Then $|\mathsf{inf}(P)| = k \times n \times m$, which gives that $\mathsf{inf}(P)$ contains knm basic influencer policies.

Assume P' is a slightly modified version of P, like some of the orderings $P \sqsubseteq P'$ from the side conditions in the type system, then $|\mathsf{inf}(P)| \approx |\mathsf{inf}(P')|$. Recalling (4.6), the comparison of the basic policies then gives

$$B_1 \bullet \cdots \bullet B_{knm} \sqsubseteq B'_1 \bullet \cdots \bullet B'_{knm'} = \bigwedge_{1 \le i \le knm} \bigvee_{1 \le j \le knm'} B_i \sqsubseteq B'_j \tag{6.2}$$

which amounts to $\sim (knm)^2$ comparisons for this single example. The simplification of $B \sqsubseteq B'$ from Fact 4.6 amounts to the predicate

$$\phi \Rightarrow (\phi' \land \underbrace{u = u' \land o = o' \land s = s'}_{Q})$$

where Q can be determined right away because it just contains single variables and security predicates. Therefore, the final analysis amounts to $\sim (knm)^2$ predicates of the type $\phi \Rightarrow \phi' \land Q$ when type checking with the policy $(\phi \Rightarrow \{\bar{x}_k : \bar{o}_n \leftarrow \bar{s}_m\})$. All these small predicates can be verified in parallel before taking the proper disjunction and conjunction of the individual results building up (6.2). Thus, the overall running time to verify a large system can be minimised drastically.

6.3 Tracing insecure systems

Let $(Sys, \mathcal{S}, \mathsf{NSA}, P_{Sys})$ and Φ be defined such that $c = \mathsf{verify}(Sys, \mathcal{S}, \mathsf{NSA}, P_{Sys}, \Phi)$ is *not* a tautology, meaning the system is not secure.

To get a more useful result the programmer needs some more feedback and a trace can be established to find the reason that c is not a tautology. Recall that $c = c_1 \land \cdots \land c_i \land \cdots \land c_n$, and a tautological c therefore implies for each process that c_i is a tautology. Conversely, if c is not a tautology, neither is c_i for one or more processes.

All the side conditions for the statements are present in $c_i = c_{i_1} \land \ldots$. Assume c_i is not a tautology; then at least one of the side conditions for the statement c_{i_j}

is not a tautology. This side condition is violated either by an inconsistency in the invariants from a loop statement or by a comparison of two policies.

Assume there is a problem with comparison of the policies as we first saw in the simple assignment example in Section 4.2.1. The problem could be further traced back to the origin, which leads to

$$(x = 4 \Rightarrow \{y : s \leftarrow s\}) \not\sqsubseteq (x > 5 \Rightarrow \{y : s \leftarrow s\}) \tag{6.3}$$

From here a further study reveals that x is assigned to 4 and y is influenced by x which is not allowed by the original policy $(x > 5 \Rightarrow \{y : s \leftarrow s\})$.

When the programmer encounters an insecure system it is therefore possible to get an idea of why the distributed system is not secure and uses this to change the policies, invariants or the statements such that the system becomes secure. The trace functionality is cumbersome to implement, however, because each step in the algorithms should be annotated such that a backtrace can be established. Hence, backtraces are not considered in the concrete implementation, which only reveals the two resulting policies used in the policy comparison.

6.4 Implicit flow in nondeterministic choice

Take the system

$$Sys = \ell : x := 0; ((x := 2; y := 1) \oplus (\mathsf{skip})); ch!x \tag{6.4}$$

where $\mathcal{S}(\ell) = s$ and

$$P_\ell = \{x : \star \leftarrow \star\} \bullet \{y : s \leftarrow s\} \qquad P_{ch} = \{\#_1 : \star \leftarrow \star\} \tag{6.5}$$

Looking at the first branch statement in the choice operator, a change in x is accompanied by a change in y. When x leaves the process its value can then be used to determine whether y was changed, thus entangling knowledge of y with knowledge of x. Hence, it looks like there is a flow from y to x. Take $\Phi(\ell) = \mathsf{true}$ and by using the concrete implementation $\mathsf{verify}(Sys, \mathcal{S}, \mathsf{NSA}, P_\ell \bullet P_{ch}, \Phi)$ gives a tautology. The type system does therefore not cover such a flow, which could possibly be described as a covert channel.

A similar problem is also found in conditional branching

$$\ell : x := 0; \mathsf{if}\ b\ \mathsf{then}\ x := 2; y := 1\ \mathsf{else\ skip\ fi}; ch!x$$

where there again is an entangling knowledge of y with knowledge of x. Whether these two examples can give any practical problems is left as an open question.

6.5 Cryptography

The policies enforce that all channels only can be read and influenced by those who are specified in the policies. Consider therefore an unprotected channel where everyone can influence and read the information, and we still want a secure transfer of information from one security principal to another.

First we introduced a public key infrastructure such that each security principals $s \in \mathbf{Pr}$ has a private key k^s_{priv} and a public key k^s_{pub}. A variable x encrypted with k is denoted $\mathsf{enc}(k, x)$. Conversely decrypting a variable y with k is denoted $\mathsf{dec}(k, y)$.

Consider now a distributed system where the security principals s_1 want to send x to s_2 using an unprotected channel and still want to ensure the confidentiality and integrity of the information. To guarantee confidentiality of x the message s_1 encrypts the information $x' = \mathsf{enc}(k^{s_2}_{pub}, x)$ such that only s_2 can decrypt the information. For integrity, s_1 can create a signature on x' (e.g. a hash function denoted $H(x')$) and encrypt this with its private key $y' = \mathsf{enc}(k^{s_1}_{priv}, H(x'))$, and sent it along with x'. The security principal s_2 now receives (x', y') and can calculate

$$x'' := \mathsf{dec}(k^{s_2}_{priv}, x') \quad y'' := \mathsf{dec}(k^{s_1}_{pub}, y')$$

If $H(x') = y''$ then $x'' = x$ and nobody could have influenced or read the secret information on the transferrer except s_1 and s_2 (assuming H is a collision-free hash function and the keys are large enough).

All this concludes that s_2 must be allowed to read x in the first place. Furthermore, x'' must be allowed to be influenced by x. Thus, the variable x'' needs to get the same readers and influencers as x have for s_1. Based on this, additional notation is required to ensure this before the actual assignment.

When $H(x') \neq y''$ someone has either tampered with x' or the signature y', and the integrity of x is not preserved. The message can therefore be discarded because $x'' \neq x$, and the assignment should not be allowed. Considering that s_1 and s_2 use an unprotected channel, handling information loss is therefore considered good practices when implementing such a system.

Adding cryptography could therefore simplify some parts of a distributed system due to the less restrictive channels. However, it does increase the complexity of the processes as they must handle data loss, encryption, and decryption. Furthermore, having a public key infrastructure also requires some setup as all public keys must be available for all processes. Further investigation is needed to verify if the above scheme is secure, and further examples are needed to determine to what extent adding cryptography helps making a system more secure.

6.6 Asynchronous communication

From the instrumented semantics for systems, it is clear that this thesis only covers synchronous communication. Asynchronous communication requires some sort of buffer to store the information, here termed a *buffer channel*. Using syntax inspired by KLAIM [DFP98], the statement $\mathsf{out}(a)@ch$ puts the value of a on the buffer channel ch and continues execution, while the statement $\mathsf{in}(x)@ch$ gets a value from the buffer channel ch and stores it in x.

Consider now the following two systems

$$\begin{aligned} \mathit{Sync} &= l_1 : ch_A!3; ch_B?x && \parallel l_2 : ch_B!8; ch_A?y \\ \mathit{Async} &= l_1 : \mathsf{out}(3)@ch_A; \mathsf{in}(x)@ch_B && \parallel l_2 : \mathsf{out}(8)@ch_B; \mathsf{in}(y)@ch_A \end{aligned}$$

First, it is seen that *Sync* will never terminate because l_1 waits for someone to receive the message on ch_A. The *Async* system will clearly terminate because of the channel buffers.

No structure for the buffer is proposed and there are therefore no guarantee of the ordering of the buffer. Consider the following

$$Async_2 = l_1 : \mathsf{out}(4)@ch; \mathsf{out}(8)@ch \ \|\!= l_2 : \mathsf{in}(x)@ch \tag{6.6}$$

Here there will be no guarantee whether the value for x will be 4 or 8.

Looking back at the avionics case it could be useful to reason about the security using asynchronous communication, as this could be used by some transport protocols such as UDP, where there likewise are no guarantee for the ordering of the data. Some security considerations have been made in [HNN09] for a distributed system concerning using asynchronous communication.

6.7 Greatest lower bound operator

In Section 2.3.6 it was shown that $\bullet$ was a least upper bound operator according to the defined preorder $\sqsubseteq$. Consider now an extension of the policies

$$P \ ::= \ ... \quad | \quad P^1 \circ P^2$$

with the influencers and readers are defined as the reverse of $\bullet$.

$$\mathsf{Infl}\left(P^1 \circ P^2, \sigma, q, y\right) = \mathsf{Infl}\left(P^1, \sigma, q, y\right) \cap \mathsf{Infl}\left(P^2, \sigma, q, y\right) \tag{6.7}$$
$$\mathsf{Read}\left(P^1 \circ P^2, \sigma, q, y\right) = \mathsf{Read}\left(P^1, \sigma, q, y\right) \cup \mathsf{Read}\left(P^2, \sigma, q, y\right) \tag{6.8}$$

The operator is then dual of $\bullet$ and therefore a lower bound on the policies.

Conjecture 6.1. *The operator $\circ$ is a greatest lower bound operator on* **Pol**.

Proof idea. Using same reasoning where $\bullet$ was shown to be a least upper bound in Fact 2.5. □

Consider influencers and readers for the policies similar to those presented in the end of Section 2.3.3

$$\mathsf{Infl}\left(\{s_1 : s_1, s_2 \leftarrow y\} \circ \{s_1 : s_1, s_3 \leftarrow y\}, \sigma, s_1, y\right) = \{s_1, s_2\} \cap \{s_1, s_3\} = \{s_1\}$$
$$\mathsf{Read}\left(\{s_1 : s_1, s_2 \rightarrow y\} \circ \{s_1 : s_1, s_3 \rightarrow y\}, \sigma, s_1, y\right) = \{s_1, s_2\} \cup \{s_1, s_3\} = \{s_1, s_2, s_3\}$$

Only the security principals who are in all influencer policies are allowed as influencers. Conversely, a security principal need only be allowed by one policy to be allowed as a reader.

Looking at the readers for a single variable, its confidentiality level decreases as more readers are allowed to read the information. In some sense the information gets *declassified.* Conversely, for the influencers for a variable, its integrity level decreases

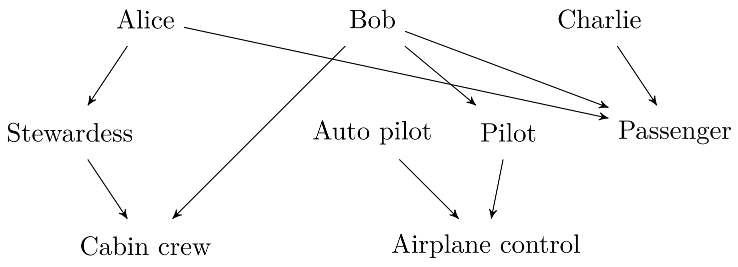

Figure 6.1: A principal hierarchy for avionics.

as we say that less principals may influence the information. The result using the operator is therefore an *endorsement* of the two policies.

Adding this operator to the policies therefore gives the programmer even more complex ways of specifying policies. Though it seems unclear if there are any advantages, as it gets more complex to verify if the policies have their intended meaning according to relevant specifications and regulations.

6.8 Principal hierarchy

DLM principals are used to denote programs, users, and groups in a *principal hierarchy* [ML97] like the one presented in Figure 6.1. Here the *user* Alice can act for the *group* stewardess and passenger, where the the stewardess also can act for the cabin crew. Bob is a pilot, and can therefore act for the passenger, cabin crew and pilot group where the latter can act for the *program* airplane control.

In this thesis the security principals have no connections. Multiple processes can use the same security principal to group processes together, but we can not assign a process to multiple security principals.

To take advantage of a principal hierarchy, some more notation is needed, such that a user can make a procedure call to another program (e.g. Bob can call the airplane control program). The program can then check if the caller have the appropriate authority to run the program (e.g. that Bob can act for the pilot group).

6.9 Related languages protecting information flow

Some existing programming languages provides information flow control similar to what is presented in this thesis.

The main example is *Jif* [Jif] which was proposed with DLM in [ML00]. Jif is an extension to the Java programming language enabling the programmer to initialise variables with DLM labels as we have seen in Section 1.2. The Jif compiler then statically checks that the integrity and confidentiality of the information holds according to the specified labels, thereby producing a secure executable Java program. The policies $\{\bar{u} : \bar{o} \leftarrow \bar{s}\}$, $\{\bar{u} : \bar{o} \rightarrow \bar{s}\}$, $P_1 \bullet P_2$ and $P_1 \circ P_2$ from this thesis can be specified directly in Jif.

A slightly different approach is used in *Jeeves*, presented in [YYS12]. The authors consider the GPS system, which until 2000 provided different precision levels depending on the context (military or civilian). They show policies which are similar to the conditional policies ($\varphi \Rightarrow P$) from this thesis. Their main implementation is in Python, using Z3 as a STM solver [Yan]. As they only do runtime checks of the policies they have a delay of the evaluation for the runtime execution.

Lastly, [Mül+14] presents *Cif* which is a verifier for the C language. The paper uses an extended version of DLM with policies very similar to those presented in this thesis, with influencer and reader policies and conditional policies. The C program is first checked with Cif and then translated into a regular C program for compiling and execution of the secure program. The authors also relate their work to the avionics scenario and security gateway described in Sections 1.3 and 5.1.

CHAPTER 7 Conclusion

This thesis shows that it is possible to build a tool to verify security policies for systems similar to those used in the avionics industry. The abstract model have the same properties as the current system running on airplanes where information flows can be checked for each execution. Furthermore, it has also been made possible to type check a system against its policies such that any execution path leads to a secure system (according to our definition of a secure system). Thus, the systems in airplanes can be made more secure than they are today.

The definition of the policies leads to a neat join-semilattice $(\mathbf{Pol}, \bullet, \sqsubseteq_{\equiv})$. Further examinations reveal that all the policies can be unified and described using the same basic policy through the implication normal form. The comparison of basic influencer policies leads to a simple result, while leaving a simplification of basic reader policies for future work.

In spite of all their careful separation, the avionics industry reveals a case where they still have multiple security domains using the same resource. This case represents a gateway connecting multiple domains using a single channel. This can not be efficiently solved with standard policies from DLM and an extension is described. Here the policies can depend on their content such that information securely can reach its destination over the shared channel.

The type checking of this gateway, though, reveals a false positive, in which a secure system is erroneously declared non-secure. Suggestions are made on how to modify the type system but is left unresolved.

Further analogies between DLM and the described policies have been illuminated in form of the the greatest lower bound operator and the principal hierarchy. The discussion also covers how non-secure systems can be traced back to the core problem of why it is non-secure. At last opportunities and problems by adding cryptography and asynchronous communication to the syntax was coverd.

APPENDIX A Proofs

This appendix contains all the proofs which are not directly relevant to have in the main thesis.

A.1 Fact 2.3

Assume $\sigma[x \mapsto [\![a]\!]\sigma] \models \varphi$, then in the evaluation of the satisfaction the variable x will be looked up in σ. If $x \notin \bar{a}$ then all occurrences of the variable x can the be substituted directly into φ by $\varphi[a/x]$ and the value of x in σ is therefore not needed, thus gives $\sigma \models \varphi[a/x]$. If $x \in \bar{a}$, the evaluation of $[\![a]\!]\sigma$ will be related to an old value of x denoted x'. Likewise after the substitution φ could contain x which should be related to x', which is available by looking up x in σ i.e. $\sigma(x)$.

For the other side, a similar deduction can be used when assuming $\sigma \models \varphi[a/x]$ to get $\sigma[x \mapsto [\![a]\!]\sigma] \models \varphi$.

A.2 Fact 2.4

The ordering $\sqsubseteq$ is *reflexive* because for all P, σ, q, and y it holds that $P \sqsubseteq P$

$$\begin{aligned}&(\mathsf{Infl}\,(P,\sigma,q,y) \subseteq \mathsf{Infl}\,(P,\sigma,q,y)) \wedge (\mathsf{Read}\,(P,\sigma,q,y) \supseteq \mathsf{Read}\,(P,\sigma,q,y))\\ &\quad = (\mathsf{Infl}\,(P,\sigma,q,y) = \mathsf{Infl}\,(P,\sigma,q,y)) \wedge (\mathsf{Read}\,(P,\sigma,q,y) = \mathsf{Read}\,(P,\sigma,q,y))\\ &\quad = \mathsf{true}\end{aligned}$$

The ordering $\sqsubseteq$ is *transitive* because if $P^1 \sqsubseteq P^2$ and $P^2 \sqsubseteq P^3$ then $P^1 \sqsubseteq P^3$. For all σ, q, and y the influencers are

$$\begin{aligned}&\mathsf{Infl}\left(P^1,\sigma,q,y\right) \subseteq \mathsf{Infl}\left(P^2,\sigma,q,y\right) \subseteq \mathsf{Infl}\left(P^3,\sigma,q,y\right)\\ &\quad \Rightarrow \mathsf{Infl}\left(P^1,\sigma,q,y\right) \subseteq \mathsf{Infl}\left(P^3,\sigma,q,y\right)\end{aligned}$$

Similar for readers this also holds

$$\begin{aligned}&\mathsf{Read}\left(P^1,\sigma,q,y\right) \supseteq \mathsf{Read}\left(P^2,\sigma,q,y\right) \supseteq \mathsf{Read}\left(P^3,\sigma,q,y\right)\\ &\quad \Rightarrow \mathsf{Read}\left(P^1,\sigma,q,y\right) \supseteq \mathsf{Read}\left(P^3,\sigma,q,y\right)\end{aligned}$$

A.3 Bottom elements

For all the equivalences (2.17) to (2.20) the influencers must be ϵ and the readers must be $\star$ for values of σ, q and y, which they are using the definition of influencers and readers directly

$$\begin{aligned}
&\mathsf{Infl}\left(\{\epsilon:\bar{o}\leftarrow\bar{s}\},\sigma,q,y\right)=\epsilon && \mathsf{Read}\left(\{\epsilon:\bar{o}\leftarrow\bar{s}\},\sigma,q,y\right)=\star && \text{(A.1)}\\
&\mathsf{Infl}\left(\{\epsilon:\bar{o}\rightarrow\bar{s}\},\sigma,q,y\right)=\epsilon && \mathsf{Read}\left(\{\epsilon:\bar{o}\rightarrow\bar{s}\},\sigma,q,y\right)=\star && \text{(A.2)}\\
&\mathsf{Infl}\left(\{\bar{u}:\epsilon\leftarrow\bar{s}\},\sigma,q,y\right)=\epsilon && \mathsf{Read}\left(\{\bar{u}:\epsilon\leftarrow\bar{s}\},\sigma,q,y\right)=\star && \text{(A.3)}\\
&\mathsf{Infl}\left(\{\bar{u}:\epsilon\rightarrow\bar{s}\},\sigma,q,y\right)=\epsilon && \mathsf{Read}\left(\{\bar{u}:\epsilon\rightarrow\bar{s}\},\sigma,q,y\right)=\star && \text{(A.4)}\\
&\mathsf{Infl}\left(\{\bar{u}:\bar{o}\leftarrow\epsilon\},\sigma,q,y\right)=\epsilon && \mathsf{Read}\left(\{\bar{u}:\bar{o}\leftarrow\epsilon\},\sigma,q,y\right)=\star && \text{(A.5)}\\
&\mathsf{Infl}\left(\{\star:\star\rightarrow\star\},\sigma,q,y\right)=\epsilon && \mathsf{Read}\left(\{\star:\star\rightarrow\star\},\sigma,q,y\right)=\star && \text{(A.6)}
\end{aligned}$$

It is here assumed that $\bar{u}$, $\bar{s}$, and $\bar{o}$ can be any set from ϵ to $\star$.

For the inequality $\{\}\not\equiv\{\bar{u}:\bar{o}\rightarrow\epsilon\}$, there exist one $q\in\bar{o}$, and $y\in\bar{u}$, such that

$$\mathsf{Infl}\left(\{\bar{u}:\bar{o}\rightarrow\epsilon\},\sigma,q,y\right)=\epsilon \qquad \mathsf{Read}\left(\{\bar{u}:\bar{o}\rightarrow\epsilon\},\sigma,q,y\right)=\epsilon \qquad \text{(A.7)}$$

The influencer set is ϵ as required, but the reader sets not $\star$.

A.4 Semilattice

Using the definition of the influencer set and reader set, it can easily be seen that for all σ, q and y that $\bullet$ on **Pol** is *idempotent* (from (2.24))

$$\begin{gathered}
\mathsf{Infl}\left(P\bullet P,\sigma,q,y\right)=\mathsf{Infl}\left(P,\sigma,q,y\right)\cup\mathsf{Infl}\left(P,\sigma,q,y\right)=\mathsf{Infl}\left(P,\sigma,q,y\right)\\
\mathsf{Read}\left(P\bullet P,\sigma,q,y\right)=\mathsf{Read}\left(P,\sigma,q,y\right)\cap\mathsf{Read}\left(P,\sigma,q,y\right)=\mathsf{Read}\left(P,\sigma,q,y\right)
\end{gathered}$$

commutative (from (2.25))

$$\begin{aligned}
\mathsf{Infl}\left(P^1\bullet P^2,\sigma,q,y\right)&=\mathsf{Infl}\left(P^1,\sigma,q,y\right)\cup\mathsf{Infl}\left(P^2,\sigma,q,y\right)\\
&=\mathsf{Infl}\left(P^2,\sigma,q,y\right)\cup\mathsf{Infl}\left(P^1,\sigma,q,y\right)\\
&=\mathsf{Infl}\left(P^2\bullet P^1,\sigma,q,y\right)\\
\mathsf{Read}\left(P^1\bullet P^2,\sigma,q,y\right)&=\mathsf{Read}\left(P^1,\sigma,q,y\right)\cap\mathsf{Read}\left(P^2,\sigma,q,y\right)\\
&=\mathsf{Read}\left(P^2,\sigma,q,y\right)\cap\mathsf{Read}\left(P^1,\sigma,q,y\right)\\
&=\mathsf{Read}\left(P^2\bullet P^1,\sigma,q,y\right)
\end{aligned}$$

and *associative* (from (2.26))

$$\begin{aligned}
&\mathsf{Infl}\left((P^1 \bullet P^2) \bullet P^3, \sigma, q, y\right) \\
&\qquad = \left(\mathsf{Infl}\left(P^1, \sigma, q, y\right) \cup \mathsf{Infl}\left(P^2, \sigma, q, y\right)\right) \cup \mathsf{Infl}\left(P^3, \sigma, q, y\right) \\
&\qquad = \mathsf{Infl}\left(P^1, \sigma, q, y\right) \cup \mathsf{Infl}\left(P^2, \sigma, q, y\right) \cup \mathsf{Infl}\left(P^3, \sigma, q, y\right) \\
&\qquad = \mathsf{Infl}\left(P^1, \sigma, q, y\right) \cup \left(\mathsf{Infl}\left(P^2, \sigma, q, y\right) \cup \mathsf{Infl}\left(P^3, \sigma, q, y\right)\right) \\
&\qquad = \mathsf{Infl}\left(P^1 \bullet (P^2 \bullet P^3), \sigma, q, y\right) \\
&\mathsf{Read}\left((P^1 \bullet P^2) \bullet P^3, \sigma, q, y\right) \\
&\qquad = \left(\mathsf{Read}\left(P^1, \sigma, q, y\right) \cap \mathsf{Read}\left(P^2, \sigma, q, y\right)\right) \cap \mathsf{Read}\left(P^3, \sigma, q, y\right) \\
&\qquad = \mathsf{Read}\left(P^1, \sigma, q, y\right) \cap \mathsf{Read}\left(P^2, \sigma, q, y\right) \cap \mathsf{Read}\left(P^3, \sigma, q, y\right) \\
&\qquad = \mathsf{Read}\left(P^1, \sigma, q, y\right) \cap \left(\mathsf{Read}\left(P^2, \sigma, q, y\right) \cap \mathsf{Read}\left(P^3, \sigma, q, y\right)\right) \\
&\qquad = \mathsf{Read}\left(P^1 \bullet (P^2 \bullet P^3), \sigma, q, y\right)
\end{aligned}$$

A.5 Equivalences

A.5.1 Distribution laws

For all $\bar{u}_1$, $\bar{u}_2$, $\bar{o}$, $\bar{s}$, σ, q and y the influencers are the same on both sides of the $\{\bar{u}_1\bar{u}_2 : \bar{o} \leftarrow \bar{s}\} \equiv \{\bar{u}_1 : \bar{o} \leftarrow \bar{s}\} \bullet \{\bar{u}_2 : \bar{o} \leftarrow \bar{s}\}$

$$\begin{aligned}
&\mathsf{Infl}\left(\{\bar{u}_1\bar{u}_2 : \bar{o} \leftarrow \bar{s}\}, \sigma, q, y\right) \\
&\qquad = \begin{cases} \bar{s} & \text{if } y \in \bar{u}_1 \cup \bar{u}_2 \wedge q \in \bar{o} \\ \epsilon & \text{otherwise} \end{cases} \\
&\qquad = \begin{cases} \bar{s} & \text{if } y \in \bar{u}_1 \wedge q \in \bar{o} \\ \epsilon & \text{otherwise} \end{cases} \cup \begin{cases} \bar{s} & \text{if } y \in \bar{u}_2 \wedge q \in \bar{o} \\ \epsilon & \text{otherwise} \end{cases} \\
&\qquad = \mathsf{Infl}\left(\{\bar{u}_1 : \bar{o} \leftarrow \bar{s}\}, \sigma, q, y\right) \cup \mathsf{Infl}\left(\{\bar{u}_2 : \bar{o} \leftarrow \bar{s}\}, \sigma, q, y\right) \\
&\qquad = \mathsf{Infl}\left(\{\bar{u}_1 : \bar{o} \leftarrow \bar{s}\} \bullet \{\bar{u}_1 : \bar{o} \leftarrow \bar{s}\}, \sigma, q, y\right)
\end{aligned}$$

and for readers

$$\begin{aligned}
&\mathsf{Read}\left(\{\bar{u}_1\bar{u}_2 : \bar{o} \leftarrow \bar{s}\}, \sigma, q, y\right) \\
&\qquad = \mathbf{Pr} \\
&\qquad = \mathbf{Pr} \cap \mathbf{Pr} \\
&\qquad = \mathsf{Read}\left(\{\bar{u}_1 : \bar{o} \leftarrow \bar{s}\}, \sigma, q, y\right) \cap \mathsf{Read}\left(\{\bar{u}_2 : \bar{o} \leftarrow \bar{s}\}, \sigma, q, y\right) \\
&\qquad = \mathsf{Read}\left(\{\bar{u}_1 : \bar{o} \leftarrow \bar{s}\} \bullet \{\bar{u}_1 : \bar{o} \leftarrow \bar{s}\}, \sigma, q, y\right)
\end{aligned}$$

For all $\bar{u}_1$, $\bar{u}_2$, $\bar{o}$, $\bar{s}$, σ, q and y the influencers are the same on both sides of

$\{\bar{u}_1\bar{u}_2 : \bar{o} \rightarrow \bar{s}\} \equiv \{\bar{u}_1 : \bar{o} \rightarrow \bar{s}\} \bullet \{\bar{u}_2 : \bar{o} \rightarrow \bar{s}\}$

$$\begin{aligned}
&\mathsf{Infl}\,(\{\bar{u}_1\bar{u}_2 : \bar{o} \rightarrow \bar{s}\}, \sigma, q, y)\\
&\quad = \epsilon\\
&\quad = \epsilon \cup \epsilon\\
&\quad = \mathsf{Infl}\,(\{\bar{u}_1 : \bar{o} \rightarrow \bar{s}\}, \sigma, q, y) \cup \mathsf{Infl}\,(\{\bar{u}_2 : \bar{o} \rightarrow \bar{s}\}, \sigma, q, y)\\
&\quad = \mathsf{Infl}\,(\{\bar{u}_1 : \bar{o} \rightarrow \bar{s}\} \bullet \{\bar{u}_2 : \bar{o} \rightarrow \bar{s}\}, \sigma, q, y)
\end{aligned}$$

and for readers

$$\begin{aligned}
&\mathsf{Read}\,(\{\bar{u}_1\bar{u}_2 : \bar{o} \rightarrow \bar{s}\}, \sigma, q, y)\\
&\quad = \begin{cases} \bar{s} & \text{if } y \in \bar{u}_1 \cup \bar{u}_2 \wedge q \in \bar{o} \\ \mathbf{Pr} & \text{otherwise} \end{cases}\\
&\quad = \begin{cases} \bar{s} & \text{if } y \in \bar{u}_1 \wedge y \in \bar{u}_2 \wedge q \in \bar{o} \\ \bar{s} & \text{if } y \in \bar{u}_1 \wedge y \notin \bar{u}_2 \wedge q \in \bar{o} \\ \bar{s} & \text{if } y \notin \bar{u}_1 \wedge y \in \bar{u}_2 \wedge q \in \bar{o} \\ \mathbf{Pr} & \text{otherwise} \end{cases}\\
&\quad = \begin{cases} \bar{s} & \text{if } y \in \bar{u}_1 \wedge q \in \bar{o} \\ \mathbf{Pr} & \text{otherwise} \end{cases} \cap \begin{cases} \bar{s} & \text{if } y \in \bar{u}_2 \wedge q \in \bar{o} \\ \mathbf{Pr} & \text{otherwise} \end{cases}\\
&\quad = \mathsf{Read}\,(\{\bar{u}_1 : \bar{o} \rightarrow \bar{s}\}, \sigma, q, y) \cap \mathsf{Read}\,(\{\bar{u}_2 : \bar{o} \rightarrow \bar{s}\}, \sigma, q, y)\\
&\quad = \mathsf{Read}\,(\{\bar{u}_1 : \bar{o} \rightarrow \bar{s}\} \bullet \{\bar{u}_2 : \bar{o} \rightarrow \bar{s}\}, \sigma, q, y)
\end{aligned}$$

For all $\bar{u}$, $\bar{o}_1$, $\bar{o}_2$, $\bar{s}$, σ, q and y the influencers are the same on both sides of $\{\bar{u} : \bar{o}_1\bar{o}_2 \leftarrow \bar{s}\} \equiv \{\bar{u} : \bar{o}_1 \leftarrow \bar{s}\} \bullet \{\bar{u} : \bar{o}_2 \leftarrow \bar{s}\}$

$$\begin{aligned}
&\mathsf{Infl}\,(\{\bar{u} : \bar{o}_1\bar{o}_2 \leftarrow \bar{s}\}, \sigma, q, y)\\
&\quad = \begin{cases} \bar{s} & \text{if } y \in \bar{u} \wedge q \in \bar{o}_1 \cup \bar{o}_2 \\ \epsilon & \text{otherwise} \end{cases}\\
&\quad = \begin{cases} \bar{s} & \text{if } y \in \bar{u} \wedge q \in \bar{o}_1 \\ \epsilon & \text{otherwise} \end{cases} \cup \begin{cases} \bar{s} & \text{if } y \in \bar{u} \wedge q \in \bar{o}_2 \\ \epsilon & \text{otherwise} \end{cases}\\
&\quad = \mathsf{Infl}\,(\{\bar{u} : \bar{o}_1 \leftarrow \bar{s}\}, \sigma, q, y) \cup \mathsf{Infl}\,(\{\bar{u} : \bar{o}_2 \leftarrow \bar{s}\}, \sigma, q, y)\\
&\quad = \mathsf{Infl}\,(\{\bar{u} : \bar{o}_1 \leftarrow \bar{s}\} \bullet \{\bar{u} : \bar{o}_2 \leftarrow \bar{s}\}, \sigma, q, y)
\end{aligned}$$

and readders

$$\begin{aligned}
&\mathsf{Read}\,(\{\bar{u} : \bar{o}_1\bar{o}_2 \leftarrow \bar{s}\}, \sigma, q, y)\\
&\quad = \mathbf{Pr}\\
&\quad = \mathbf{Pr} \cap \mathbf{Pr}\\
&\quad = \mathsf{Read}\,(\{\bar{u} : \bar{o}_1 \leftarrow \bar{s}\}, \sigma, q, y) \cap \mathsf{Read}\,(\{\bar{u} : \bar{o}_2 \leftarrow \bar{s}\}, \sigma, q, y)\\
&\quad = \mathsf{Read}\,(\{\bar{u} : \bar{o}_1 \leftarrow \bar{s}\} \bullet \{\bar{u} : \bar{o}_2 \leftarrow \bar{s}\}, \sigma, q, y)
\end{aligned}$$

For all $\bar{u}$, $\bar{o}_1$, $\bar{o}_2$, $\bar{s}$, σ, q and y the influencers are the same on both sides of $\{\bar{u} : \bar{o}_1\bar{o}_2 \leftarrow \bar{s}\} \equiv \{\bar{u} : \bar{o}_1 \leftarrow \bar{s}\} \bullet \{\bar{u} : \bar{o}_2 \leftarrow \bar{s}\}$

$$
\begin{aligned}
&\mathsf{Infl}\,(\{\bar{u} : \bar{o}_1\bar{o}_2 \rightarrow \bar{s}\}, \sigma, q, y) \\
&\quad = \epsilon \\
&\quad = \epsilon \cup \epsilon \\
&\quad = \mathsf{Infl}\,(\{\bar{u} : \bar{o}_1 \rightarrow \bar{s}\}, \sigma, q, y) \cup \mathsf{Infl}\,(\{\bar{u} : \bar{o}_2 \rightarrow \bar{s}\}, \sigma, q, y) \\
&\quad = \mathsf{Infl}\,(\{\bar{u} : \bar{o}_1 \rightarrow \bar{s}\} \bullet \{\bar{u} : \bar{o}_2 \rightarrow \bar{s}\}, \sigma, q, y)
\end{aligned}
$$

and for readers

$$
\begin{aligned}
&\mathsf{Read}\,(\{\bar{u} : \bar{o}_1\bar{o}_2 \rightarrow \bar{s}\}, \sigma, q, y) \\
&\quad = \begin{cases} \bar{s} & \text{if } y \in \bar{u} \wedge q \in \bar{o}_1 \cup \bar{o}_2 \\ \mathbf{Pr} & \text{otherwise} \end{cases} \\
&\quad = \begin{cases} \bar{s} & \text{if } y \in \bar{u} \wedge q \in \bar{o}_1 \wedge q \in \bar{o}_2 \\ \bar{s} & \text{if } y \in \bar{u} \wedge q \in \bar{o}_1 \wedge q \notin \bar{o}_2 \\ \bar{s} & \text{if } y \in \bar{u} \wedge q \notin \bar{o}_1 \wedge q \in \bar{o}_2 \\ \mathbf{Pr} & \text{otherwise} \end{cases} \\
&\quad = \begin{cases} \bar{s} & \text{if } y \in \bar{u} \wedge q \in \bar{o}_1 \\ \mathbf{Pr} & \text{otherwise} \end{cases} \cap \begin{cases} \bar{s} & \text{if } y \in \bar{u} \wedge q \in \bar{o}_2 \\ \mathbf{Pr} & \text{otherwise} \end{cases} \\
&\quad = \mathsf{Read}\,(\{\bar{u} : \bar{o}_1 \rightarrow \bar{s}\}, \sigma, q, y) \cap \mathsf{Read}\,(\{\bar{u} : \bar{o}_2 \rightarrow \bar{s}\}, \sigma, q, y) \\
&\quad = \mathsf{Read}\,(\{\bar{u} : \bar{o}_1 \rightarrow \bar{s}\} \bullet \{\bar{u} : \bar{o}_2 \rightarrow \bar{s}\}, \sigma, q, y)
\end{aligned}
$$

For all $\bar{u}$, $\bar{o}$, $\bar{s}_1$, $\bar{s}_2$, σ, q and y the influencers are the same on both sides of $\{\bar{u} : \bar{o} \leftarrow \bar{s}_1\bar{s}_2\} \equiv \{\bar{u} : \bar{o} \leftarrow \bar{s}_1\} \bullet \{\bar{u} : \bar{o} \leftarrow \bar{s}_2\}$

$$
\begin{aligned}
&\mathsf{Infl}\,(\{\bar{u} : \bar{o} \leftarrow \bar{s}_1\bar{s}_2\}, \sigma, q, y) \\
&\quad = \begin{cases} \bar{s}_1 \cup \bar{s}_2 & \text{if } y \in \bar{u} \wedge q \in \bar{o} \\ \epsilon & \text{otherwise} \end{cases} \\
&\quad = \begin{cases} \bar{s}_1 & \text{if } y \in \bar{u} \wedge q \in \bar{o} \\ \epsilon & \text{otherwise} \end{cases} \cup \begin{cases} \bar{s}_2 & \text{if } y \in \bar{u} \wedge q \in \bar{o} \\ \epsilon & \text{otherwise} \end{cases} \\
&\quad = \mathsf{Infl}\,(\{\bar{u} : \bar{o} \leftarrow \bar{s}_1\}, \sigma, q, y) \cup \mathsf{Infl}\,(\{\bar{u} : \bar{o} \leftarrow \bar{s}_2\}, \sigma, q, y) \\
&\quad = \mathsf{Infl}\,(\{\bar{u} : \bar{o} \leftarrow \bar{s}_1\} \bullet \{\bar{u} : \bar{o} \leftarrow \bar{s}_2\}, \sigma, q, y)
\end{aligned}
$$

and for readers

$$
\begin{aligned}
&\mathsf{Read}\,(\{\bar{u} : \bar{o} \leftarrow \bar{s}_1\bar{s}_2\}, \sigma, q, y) \\
&\quad = \mathbf{Pr} \\
&\quad = \mathbf{Pr} \cap \mathbf{Pr} \\
&\quad = \mathsf{Read}\,(\{\bar{u} : \bar{o} \leftarrow \bar{s}_1\}, \sigma, q, y) \cap \mathsf{Read}\,(\{\bar{u} : \bar{o} \leftarrow \bar{s}_2\}, \sigma, q, y) \\
&\quad = \mathsf{Read}\,(\{\bar{u} : \bar{o} \leftarrow \bar{s}_1\} \bullet \{\bar{u} : \bar{o} \leftarrow \bar{s}_2\}, \sigma, q, y)
\end{aligned}
$$

A.5.2 Conditional policies

For all φ, P^1, P^2, σ, q and y the influencers are the same on both sides of $\left(\varphi \Rightarrow P^1 \bullet P^2\right) \equiv \left(\varphi \Rightarrow P^1\right) \bullet \left(\varphi \Rightarrow P^2\right)$

$$\begin{aligned}
&\mathsf{Infl}\left(\left(\varphi \Rightarrow P^1 \bullet P^2\right), \sigma, q, y\right) \\
&\quad = \begin{cases} \mathsf{Infl}\left(P^1 \bullet P^2, \sigma, q, y\right) & \text{if } \sigma \models \varphi \\ \epsilon & \text{otherwise} \end{cases} \\
&\quad = \begin{cases} \mathsf{Infl}\left(P^1, \sigma, q, y\right) \cup \mathsf{Infl}\left(P^2, \sigma, q, y\right) & \text{if } \sigma \models \varphi \\ \epsilon & \text{otherwise} \end{cases} \\
&\quad = \begin{cases} \mathsf{Infl}\left(P^1, \sigma, q, y\right) & \text{if } \sigma \models \varphi \\ \epsilon & \text{otherwise} \end{cases} \cup \begin{cases} \mathsf{Infl}\left(P^2, \sigma, q, y\right) & \text{if } \sigma \models \varphi \\ \epsilon & \text{otherwise} \end{cases} \\
&\quad = \mathsf{Infl}\left(\left(\varphi \Rightarrow P^1\right), \sigma, q, y\right) \cup \mathsf{Infl}\left(\left(\varphi \Rightarrow P^2\right), \sigma, q, y\right) \\
&\quad = \mathsf{Infl}\left(\left(\varphi \Rightarrow P^1\right) \bullet \left(\varphi \Rightarrow P^2\right), \sigma, q, y\right)
\end{aligned}$$

and for readers

$$\begin{aligned}
&\mathsf{Read}\left(\left(\varphi \Rightarrow P^1 \bullet P^2\right), \sigma, q, y\right) \\
&\quad = \begin{cases} \mathsf{Read}\left(P^1 \bullet P^2, \sigma, q, y\right) & \text{if } \sigma \models \varphi \\ \mathbf{Pr} & \text{otherwise} \end{cases} \\
&\quad = \begin{cases} \mathsf{Read}\left(P^1, \sigma, q, y\right) \cap \mathsf{Read}\left(P^2, \sigma, q, y\right) & \text{if } \sigma \models \varphi \\ \mathbf{Pr} & \text{otherwise} \end{cases} \\
&\quad = \begin{cases} \mathsf{Read}\left(P^1, \sigma, q, y\right) & \text{if } \sigma \models \varphi \\ \mathbf{Pr} & \text{otherwise} \end{cases} \cap \begin{cases} \mathsf{Read}\left(P^2, \sigma, q, y\right) & \text{if } \sigma \models \varphi \\ \mathbf{Pr} & \text{otherwise} \end{cases} \\
&\quad = \mathsf{Read}\left(\left(\varphi \Rightarrow P^1\right), \sigma, q, y\right) \cap \mathsf{Read}\left(\left(\varphi \Rightarrow P^2\right), \sigma, q, y\right) \\
&\quad = \mathsf{Read}\left(\left(\varphi \Rightarrow P^1\right) \bullet \left(\varphi \Rightarrow P^2\right), \sigma, q, y\right)
\end{aligned}$$

For all φ_1, φ_2, P, σ, q and y the influencers are the same on both sides $(\varphi_1 \wedge \varphi_2 \Rightarrow P) \equiv$

$(\varphi_1 \Rightarrow (\varphi_2 \Rightarrow P))$

$$\begin{aligned}
&\mathsf{Infl}\left((\varphi_1 \wedge \varphi_2 \Rightarrow P), \sigma, q, y\right) \\
&\quad = \begin{cases} \mathsf{Infl}\left(P, \sigma, q, y\right) & \text{if } \sigma \models (\varphi_1 \wedge \varphi_2) \\ \epsilon & \text{otherwise} \end{cases} \\
&\quad = \begin{cases} \mathsf{Infl}\left(P, \sigma, q, y\right) & \text{if } \sigma \models \varphi_1 \wedge \sigma \models \varphi_2 \\ \epsilon & \text{otherwise} \end{cases} \\
&\quad = \begin{cases} \begin{cases} \mathsf{Infl}\left(P, \sigma, q, y\right) & \text{if } \sigma \models \varphi_2 \\ \epsilon & \text{otherwise} \end{cases} & \text{if } \sigma \models \varphi_1 \\ \epsilon & \text{otherwise} \end{cases} \\
&\quad = \begin{cases} \mathsf{Infl}\left((\varphi_2 \Rightarrow P), \sigma, q, y\right) & \text{if } \sigma \models \varphi_1 \\ \epsilon & \text{otherwise} \end{cases} \\
&\quad = \mathsf{Infl}\left((\varphi_1 \Rightarrow (\varphi_2 \Rightarrow P)), \sigma, q, y\right)
\end{aligned}$$

and for readers

$$\begin{aligned}
&\mathsf{Read}\left((\varphi_1 \wedge \varphi_2 \Rightarrow P), \sigma, q, y\right) \\
&\quad = \begin{cases} \mathsf{Read}\left(P, \sigma, q, y\right) & \text{if } \sigma \models (\varphi_1 \wedge \varphi_2) \\ \mathbf{Pr} & \text{otherwise} \end{cases} \\
&\quad = \begin{cases} \mathsf{Read}\left(P, \sigma, q, y\right) & \text{if } \sigma \models \varphi_1 \wedge \sigma \models \varphi_2 \\ \mathbf{Pr} & \text{otherwise} \end{cases} \\
&\quad = \begin{cases} \begin{cases} \mathsf{Read}\left(P, \sigma, q, y\right) & \text{if } \sigma \models \varphi_2 \\ \mathbf{Pr} & \text{otherwise} \end{cases} & \text{if } \sigma \models \varphi_1 \\ \mathbf{Pr} & \text{otherwise} \end{cases} \\
&\quad = \begin{cases} \mathsf{Read}\left((\varphi_2 \Rightarrow P), \sigma, q, y\right) & \text{if } \sigma \models \varphi_1 \\ \mathbf{Pr} & \text{otherwise} \end{cases} \\
&\quad = \mathsf{Read}\left((\varphi_1 \Rightarrow (\varphi_2 \Rightarrow P)), \sigma, q, y\right)
\end{aligned}$$

For all φ, σ, q and y the influencers are the same on both sides of $(\varphi \Rightarrow \{\}) \equiv \{\}$

$$\begin{aligned}
&\mathsf{Infl}\left((\varphi \Rightarrow \{\}), \sigma, q, y\right) \\
&\quad = \begin{cases} \mathsf{Infl}\left(\{\}, \sigma, q, y\right) & \text{if } \sigma \models \varphi \\ \epsilon & \text{otherwise} \end{cases} \\
&\quad = \begin{cases} \epsilon & \text{if } \sigma \models \varphi \\ \epsilon & \text{otherwise} \end{cases} \\
&\quad = \epsilon \\
&\quad = \mathsf{Infl}\left(\{\}, \sigma, q, y\right)
\end{aligned}$$

and for readers

$$\begin{aligned}
&\mathsf{Read}\left(\left(\varphi \Rightarrow \{\}\right), \sigma, q, y\right)\\
&\quad= \begin{cases} \mathsf{Read}\left(\{\}, \sigma, q, y\right) & \text{if } \sigma \models \varphi \\ \mathbf{Pr} & \text{otherwise} \end{cases}\\
&\quad= \begin{cases} \mathbf{Pr} & \text{if } \sigma \models \varphi \\ \mathbf{Pr} & \text{otherwise} \end{cases}\\
&\quad= \mathbf{Pr}\\
&\quad= \mathsf{Read}\left(\{\}, \sigma, q, y\right)
\end{aligned}$$

For all P, σ, q and y the influencers are the same on both sides of $(\mathsf{true} \Rightarrow P) \equiv P$

$$\begin{aligned}
&\mathsf{Infl}\left(\left(\mathsf{true} \Rightarrow P\right), \sigma, q, y\right)\\
&\quad= \begin{cases} \mathsf{Infl}\left(P, \sigma, q, y\right) & \text{if } \sigma \models \mathsf{true} \\ \epsilon & \text{otherwise} \end{cases}\\
&\quad= \mathsf{Infl}\left(P, \sigma, q, y\right)
\end{aligned}$$

and for readers

$$\begin{aligned}
&\mathsf{Read}\left(\left(\mathsf{true} \Rightarrow P\right), \sigma, q, y\right)\\
&\quad= \begin{cases} \mathsf{Read}\left(P, \sigma, q, y\right) & \text{if } \sigma \models \mathsf{true} \\ \mathbf{Pr} & \text{otherwise} \end{cases}\\
&\quad= \mathsf{Read}\left(P, \sigma, q, y\right)
\end{aligned}$$

For all P, σ, q and y the influencers are the same on both sides of $(\mathsf{false} \Rightarrow P) \equiv \{\}$

$$\begin{aligned}
&\mathsf{Infl}\left(\left(\mathsf{false} \Rightarrow P\right), \sigma, q, y\right)\\
&\quad= \begin{cases} \mathsf{Infl}\left(P, \sigma, q, y\right) & \text{if } \sigma \models \mathsf{false} \\ \epsilon & \text{otherwise} \end{cases}\\
&\quad= \mathsf{Infl}\left(\{\}, \sigma, q, y\right)
\end{aligned}$$

and for readers

$$\begin{aligned}
&\mathsf{Read}\left(\left(\mathsf{false} \Rightarrow P\right), \sigma, q, y\right)\\
&\quad= \begin{cases} \mathsf{Read}\left(P, \sigma, q, y\right) & \text{if } \sigma \models \mathsf{false} \\ \mathbf{Pr} & \text{otherwise} \end{cases}\\
&\quad= \mathsf{Read}\left(\{\}, \sigma, q, y\right)
\end{aligned}$$

A.6 Fact 3.2 and Fact 3.1

The proof is done by structural induction on P. There are recursion in conditional policy $(\varphi \Rightarrow P)$ and sequential policies $P^1 \bullet P^2$. First all the base cases are established. Then the induction step is taken, and the recursion is proven.

Case $P = \{\bar{x} : \bar{o} \leftarrow \bar{s}\}[a/x]$. The influencers hold

$$\begin{aligned}
&\mathsf{Infl}\left(\{\bar{x} : \bar{o} \leftarrow \bar{s}\}[a/x], \sigma, q, z\right) \\
&\quad = \mathsf{Infl}\left(\{\bar{x} : \bar{o} \leftarrow \bar{s}\}, \sigma, q, z\right) \\
&\quad = \begin{cases} \bar{s} & \text{if } y \in \bar{x} \wedge q \in \bar{o} \\ \epsilon & \text{otherwise} \end{cases} \\
&\quad = \mathsf{Infl}\left(\{\bar{x} : \bar{o} \leftarrow \bar{s}\}, \sigma[x \mapsto [\![a]\!]\sigma], q, z\right)
\end{aligned}$$

and the readers hold

$$\begin{aligned}
&\mathsf{Read}\left(\{\bar{x} : \bar{o} \leftarrow \bar{s}\}[a/x], \sigma, q, z\right) \\
&\quad = \mathsf{Read}\left(\{\bar{x} : \bar{o} \leftarrow \bar{s}\}, \sigma, q, z\right) \\
&\quad = \mathbf{Pr} \\
&\quad = \mathsf{Read}\left(\{\bar{x} : \bar{o} \leftarrow \bar{s}\}, \sigma[x \mapsto [\![a]\!]\sigma], q, z\right)
\end{aligned}$$

Case $P = \{\bar{x} : \bar{o} \rightarrow \bar{s}\}[a/x]$. The influencers hold

$$\begin{aligned}
&\mathsf{Infl}\left(\{\bar{x} : \bar{o} \rightarrow \bar{s}\}[a/x], \sigma, q, z\right) \\
&\quad = \mathsf{Infl}\left(\{\bar{x} : \bar{o} \rightarrow \bar{s}\}, \sigma, q, z\right) \\
&\quad = \epsilon \\
&\quad = \mathsf{Infl}\left(\{\bar{x} : \bar{o} \rightarrow \bar{s}\}, \sigma[x \mapsto [\![a]\!]\sigma], q, z\right)
\end{aligned}$$

and the readers hold

$$\begin{aligned}
&\mathsf{Read}\left(\{\bar{x} : \bar{o} \rightarrow \bar{s}\}[a/x], \sigma, q, z\right) \\
&\quad = \mathsf{Read}\left(\{\bar{x} : \bar{o} \rightarrow \bar{s}\}, \sigma, q, z\right) \\
&\quad = \begin{cases} \bar{s} & \text{if } y \in \bar{x} \wedge q \in \bar{o} \\ \mathbf{Pr} & \text{otherwise} \end{cases} \\
&\quad = \mathsf{Read}\left(\{\bar{x} : \bar{o} \rightarrow \bar{s}\}, \sigma[x \mapsto [\![a]\!]\sigma], q, z\right)
\end{aligned}$$

Case $P = ((\varphi \Rightarrow \{\bar{x} : \bar{o} \leftarrow \bar{s}\}))[a/x]$ **or** $P = ((\varphi \Rightarrow \{\bar{x} : \bar{o} \rightarrow \bar{s}\}))[a/x]$. The influencers hold

$$\begin{aligned}
&\mathsf{Infl}\left((\varphi \Rightarrow P')[a/x], \sigma, q, z\right) \\
&\quad = \mathsf{Infl}\left((\varphi[a/x] \Rightarrow P'[a/x]), \sigma, q, z\right) \\
&\quad = \begin{cases} \mathsf{Infl}\left(P'[a/x], \sigma, q, z\right) & \text{if } \sigma \models \varphi[a/x] \\ \epsilon & \text{otherwise} \end{cases} \\
&\quad = \begin{cases} \mathsf{Infl}\left(P', \sigma[x \mapsto [\![a]\!]\sigma], q, z\right) & \text{if } \sigma \models \varphi[a/x] \\ \epsilon & \text{otherwise} \end{cases} \\
&\quad = \begin{cases} \mathsf{Infl}\left(P', \sigma[x \mapsto [\![a]\!]\sigma], q, z\right) & \text{if } \sigma[x \mapsto [\![a]\!]\sigma] \models \varphi \\ \epsilon & \text{otherwise} \end{cases} \qquad \text{using Fact 2.3} \\
&\quad = \mathsf{Infl}\left((\varphi \Rightarrow P'), \sigma[x \mapsto [\![a]\!]\sigma], q, z\right)
\end{aligned}$$

and the readers hold

$$\begin{aligned}
&\mathsf{Read}\left((\varphi \Rightarrow P')[a/x], \sigma, q, z\right) \\
&\quad = \mathsf{Read}\left((\varphi[a/x] \Rightarrow P'[a/x]), \sigma, q, z\right) \\
&\quad = \begin{cases} \mathsf{Read}\left(P'[a/x], \sigma, q, z\right) & \text{if } \sigma \models \varphi[a/x] \\ \mathbf{Pr} & \text{otherwise} \end{cases} \\
&\quad = \begin{cases} \mathsf{Read}\left(P', \sigma[x \mapsto [\![a]\!]\sigma], q, z\right) & \text{if } \sigma \models \varphi[a/x] \\ \mathbf{Pr} & \text{otherwise} \end{cases} \\
&\quad = \begin{cases} \mathsf{Read}\left(P', \sigma[x \mapsto [\![a]\!]\sigma], q, z\right) & \text{if } \sigma[x \mapsto [\![a]\!]\sigma] \models \varphi \\ \mathbf{Pr} & \text{otherwise} \end{cases} \qquad \text{using Fact 2.3} \\
&\quad = \mathsf{Read}\left((\varphi \Rightarrow P'), \sigma[x \mapsto [\![a]\!]\sigma], q, z\right)
\end{aligned}$$

The induction steps can begin, and it is now assumed that

$$\begin{aligned}
\mathsf{Infl}\left((\varphi \Rightarrow P)[a/x], \sigma, q, z\right) &= \mathsf{Infl}\left((\varphi \Rightarrow P), \sigma[x \mapsto [\![a]\!]\sigma], q, z\right) \\
\mathsf{Read}\left((\varphi \Rightarrow P)[a/x], \sigma, q, z\right) &= \mathsf{Read}\left((\varphi \Rightarrow P), \sigma[x \mapsto [\![a]\!]\sigma], q, z\right)
\end{aligned}$$

Induction step: $P = ((\varphi \Rightarrow (\varphi' \Rightarrow P')))[a/x]$. The influencers hold

$$\begin{aligned}
&\mathsf{Infl}\left((\varphi \Rightarrow (\varphi' \Rightarrow P'))[a/x], \sigma, q, z\right) \\
&\quad = \mathsf{Infl}\left((\varphi \wedge \varphi' \Rightarrow P')[a/x], \sigma, q, z\right) \\
&\quad = \mathsf{Infl}\left((\varphi \wedge \varphi' \Rightarrow P'), \sigma[x \mapsto [\![a]\!]\sigma], q, u\right) \\
&\quad = \mathsf{Infl}\left((\varphi \Rightarrow (\varphi' \Rightarrow P')), \sigma[x \mapsto [\![a]\!]\sigma], q, u\right)
\end{aligned}$$

and the readers hold

$$\begin{aligned}
&\mathsf{Read}\left((\varphi \Rightarrow (\varphi' \Rightarrow P'))[a/x], \sigma, q, z\right) \\
&\quad = \mathsf{Read}\left((\varphi \wedge \varphi' \Rightarrow P')[a/x], \sigma, q, z\right) \\
&\quad = \mathsf{Read}\left((\varphi \wedge \varphi' \Rightarrow P'), \sigma[x \mapsto [\![a]\!]\sigma], q, u\right) \\
&\quad = \mathsf{Read}\left((\varphi \Rightarrow (\varphi' \Rightarrow P')), \sigma[x \mapsto [\![a]\!]\sigma], q, u\right)
\end{aligned}$$

Induction step: $P = (\left(\varphi \Rightarrow P^1 \bullet P^2\right))[a/x]$. The influencers hold

$$\begin{aligned}
&\mathsf{Infl}\left(\left(\varphi \Rightarrow P^1 \bullet P^2\right)[a/x], \sigma, q, z\right) \\
&\quad = \mathsf{Infl}\left(\left(\left(\varphi \Rightarrow P^1\right) \bullet \left(\varphi \Rightarrow P^2\right)\right)[a/x], \sigma, q, z\right) \\
&\quad = \mathsf{Infl}\left(\left(\varphi \Rightarrow P^1\right)[a/x] \bullet \left(\varphi \Rightarrow P^2\right)[a/x], \sigma, q, z\right) \\
&\quad = \mathsf{Infl}\left(\left(\varphi \Rightarrow P^1\right)[a/x], \sigma, q, z\right) \cap \mathsf{Infl}\left(\left(\varphi \Rightarrow P^2\right)[a/x], \sigma, q, z\right) \\
&\quad = \mathsf{Infl}\left(\left(\varphi \Rightarrow P^1\right), \sigma[x \mapsto [\![a]\!]\sigma], q, z\right) \cap \mathsf{Infl}\left(\left(\varphi \Rightarrow P^2\right), \sigma[x \mapsto [\![a]\!]\sigma], q, z\right) \\
&\quad = \mathsf{Infl}\left(\left(\varphi \Rightarrow P^1\right) \bullet \left(\varphi \Rightarrow P^2\right), \sigma[x \mapsto [\![a]\!]\sigma], q, z\right) \\
&\quad = \mathsf{Infl}\left(\left(\varphi \Rightarrow P^1 \bullet P^2\right), \sigma[x \mapsto [\![a]\!]\sigma], q, z\right)
\end{aligned}$$

and the readers hold

$$
\begin{aligned}
&\mathsf{Read}\left(\left(\varphi \Rightarrow P^1 \bullet P^2\right)[a/x], \sigma, q, z\right)\\
&\quad= \mathsf{Read}\left(\left(\left(\varphi \Rightarrow P^1\right) \bullet \left(\varphi \Rightarrow P^2\right)\right)[a/x], \sigma, q, z\right)\\
&\quad= \mathsf{Read}\left(\left(\varphi \Rightarrow P^1\right)[a/x] \bullet \left(\varphi \Rightarrow P^2\right)[a/x], \sigma, q, z\right)\\
&\quad= \mathsf{Read}\left(\left(\varphi \Rightarrow P^1\right)[a/x], \sigma, q, z\right) \cap \mathsf{Read}\left(\left(\varphi \Rightarrow P^2\right)[a/x], \sigma, q, z\right)\\
&\quad= \mathsf{Read}\left(\left(\varphi \Rightarrow P^1\right), \sigma[x \mapsto [\![a]\!]\sigma], q, z\right) \cap \mathsf{Read}\left(\left(\varphi \Rightarrow P^2\right), \sigma[x \mapsto [\![a]\!]\sigma], q, z\right)\\
&\quad= \mathsf{Read}\left(\left(\varphi \Rightarrow P^1\right) \bullet \left(\varphi \Rightarrow P^2\right), \sigma[x \mapsto [\![a]\!]\sigma], q, z\right)\\
&\quad= \mathsf{Read}\left(\left(\varphi \Rightarrow P^1 \bullet P^2\right), \sigma[x \mapsto [\![a]\!]\sigma], q, z\right)
\end{aligned}
$$

Base case: $P = (P^1 \bullet P^2)[a/x]$ Using all combinations for P^1 and P^2 from the cases above the base case can be established for sequential policies using this template. First the influencers hold

$$
\begin{aligned}
&\mathsf{Infl}\left((P^1 \bullet P^2)[a/x], \sigma, q, z\right)\\
&\quad= \mathsf{Infl}\left(P^1[a/x] \bullet P^2[a/x], \sigma, q, z\right)\\
&\quad= \mathsf{Infl}\left(P^1[a/x], \sigma, q, z\right) \cup \mathsf{Infl}\left(P^2[a/x], \sigma, q, z\right)\\
&\quad= \mathsf{Infl}\left(P^1, \sigma[x \mapsto [\![a]\!]\sigma], q, z\right) \cup \mathsf{Infl}\left(P^2, \sigma[x \mapsto [\![a]\!]\sigma], q, z\right)\\
&\quad= \mathsf{Infl}\left(P^1 \bullet P^2, \sigma[x \mapsto [\![a]\!]\sigma], q, z\right)
\end{aligned}
$$

and the readers hold

$$
\begin{aligned}
&\mathsf{Read}\left((P^1 \bullet P^2)[a/x], \sigma, q, z\right)\\
&\quad= \mathsf{Read}\left(P^1[a/x] \bullet P^2[a/x], \sigma, q, z\right)\\
&\quad= \mathsf{Read}\left(P^1[a/x], \sigma, q, z\right) \cap \mathsf{Read}\left(P^2[a/x], \sigma, q, z\right)\\
&\quad= \mathsf{Read}\left(P^1, \sigma[x \mapsto [\![a]\!]\sigma], q, z\right) \cap \mathsf{Read}\left(P^2, \sigma[x \mapsto [\![a]\!]\sigma], q, z\right)\\
&\quad= \mathsf{Read}\left(P^1 \bullet P^2, \sigma[x \mapsto [\![a]\!]\sigma], q, z\right)
\end{aligned}
$$

The last induction step can begin, and it is now assumed that

$$
\mathsf{Read}\left((P^1 \bullet P^2)[a/x], \sigma, q, z\right) = \mathsf{Read}\left(P^1 \bullet P^2, \sigma[x \mapsto [\![a]\!]\sigma], q, z\right)
$$

Induction step: $P = (P^1 \bullet P^2 \bullet P^3)[a/x]$. First for influencers it holds

$$
\begin{aligned}
&\mathsf{Infl}\left((P^1 \bullet P^2 \bullet P^3)[a/x], \sigma, q, z\right)\\
&\quad= \mathsf{Infl}\left(((P^1 \bullet P^2) \bullet P^3)[a/x], \sigma, q, z\right)\\
&\quad= \mathsf{Infl}\left((P^1 \bullet P^2)[a/x] \bullet P^3[a/x], \sigma, q, z\right)\\
&\quad= \mathsf{Infl}\left((P^1 \bullet P^2)[a/x], \sigma, q, z\right) \cup \mathsf{Infl}\left(P^3[a/x], \sigma, q, z\right)\\
&\quad= \mathsf{Infl}\left(P^1 \bullet P^2, \sigma[x \mapsto [\![a]\!]\sigma], q, z\right) \cup \mathsf{Infl}\left(P^3, \sigma[x \mapsto [\![a]\!]\sigma], q, z\right)\\
&\quad= \mathsf{Infl}\left((P^1 \bullet P^2) \bullet P^3, \sigma[x \mapsto [\![a]\!]\sigma], q, z\right)\\
&\quad= \mathsf{Infl}\left((P^1 \bullet P^2 \bullet P^3), \sigma[x \mapsto [\![a]\!]\sigma], q, z\right)
\end{aligned}
$$

and for readers

$$\begin{aligned}
&\mathsf{Read}\left((P^1 \bullet P^2 \bullet P^3)[a/x]\,, \sigma, q, z\right)\\
&\quad= \mathsf{Read}\left(((P^1 \bullet P^2) \bullet P^3)[a/x]\,, \sigma, q, z\right)\\
&\quad= \mathsf{Read}\left((P^1 \bullet P^2)[a/x] \bullet P^3[a/x]\,, \sigma, q, z\right)\\
&\quad= \mathsf{Read}\left((P^1 \bullet P^2)[a/x]\,, \sigma, q, z\right) \cap \mathsf{Read}\left(P^3[a/x]\,, \sigma, q, z\right)\\
&\quad= \mathsf{Read}\left(P^1 \bullet P^2, \sigma[x \mapsto [\![a]\!]\sigma], q, z\right) \cap \mathsf{Read}\left(P^3, \sigma[x \mapsto [\![a]\!]\sigma], q, z\right)\\
&\quad= \mathsf{Read}\left((P^1 \bullet P^2) \bullet P^3, \sigma[x \mapsto [\![a]\!]\sigma], q, z\right)\\
&\quad= \mathsf{Read}\left((P^1 \bullet P^2 \bullet P^3), \sigma[x \mapsto [\![a]\!]\sigma], q, z\right)
\end{aligned}$$

Which finalises the proof.

A.7 Fact 3.4 and Fact 3.3

Similar to the proof for Fact 3.1 and Fact 3.2. It is done by induction on P.

First note the following

$$\bigcup_{z\in\bar{y}} \mathsf{Infl}\left(\{\bar{x} : \bar{o} \leftarrow \bar{s}\}, \sigma, q, z\right) = \begin{cases} \bar{s} & \text{if } q \in \bar{o} \wedge (\bar{y} \cap \bar{x} \neq \epsilon) \\ \epsilon & \text{otherwise} \end{cases} \tag{A.8}$$

$$\bigcap_{z\in\bar{y}} \mathsf{Read}\left(\{\bar{x} : \bar{o} \rightarrow \bar{s}\}, \sigma, q, z\right) = \begin{cases} \bar{s} & \text{if } q \in \bar{o} \wedge (\bar{y} \cap \bar{x} \neq \epsilon) \\ \mathbf{Pr} & \text{otherwise} \end{cases} \tag{A.9}$$

If at least one element $z \in \bar{y}$ is also in $\bar{x}$, then due to the union/intersections the influencers/readers are $\bar{s}$ (if $q \in \bar{o}$). If $q \in \bar{o}$, then the only way to get $\epsilon/\mathbf{Pr}$ is if $\bar{z} \cap \bar{x} = \epsilon$.

Shown here is only the proof for the readers, but the proof for influencers follows the same structure.

Base case: $\{\bar{x} : \bar{o} \rightarrow \bar{s}\}\langle \bar{y}/x\rangle$ **and** $x \neq z$.

$$\begin{aligned}
&\mathsf{Read}\left(\{\bar{x} : \bar{o} \rightarrow \bar{s}\}\langle \bar{y}/x\rangle, \sigma, q, z\right)\\
&\quad= \begin{cases} \mathsf{Read}\left(\{\bar{x}x : \bar{o} \rightarrow \bar{s}\}, \sigma, q, z\right) & \text{if } \bar{x} \cap \bar{y} \neq \epsilon \\ \mathsf{Read}\left(\{\bar{x} : \bar{o} \rightarrow \bar{s}\}, \sigma, q, z\right) & \text{otherwise} \end{cases}\\
&\quad= \begin{cases} \bar{s} & \text{if } \bar{x} \cap \bar{y} \neq \epsilon \wedge z \in \bar{x}x \wedge q \in \bar{o} \\ \mathbf{Pr} & \text{if } \bar{x} \cap \bar{y} \neq \epsilon \\ \bar{s} & \text{if } \bar{x} \cap \bar{y} = \epsilon \wedge z \in \bar{x} \wedge q \in \bar{o} \\ \mathbf{Pr} & \text{if } \bar{x} \cap \bar{y} = \epsilon \end{cases}\\
&\quad= \begin{cases} \bar{s} & \text{if } z \in \bar{x} \wedge q \in \bar{o} \\ \mathbf{Pr} & \text{otherwise} \end{cases} && \text{using } x \neq z\\
&\quad= \mathsf{Read}\left(\{\bar{x} : \bar{o} \rightarrow \bar{s}\}, \sigma, q, z\right)
\end{aligned}$$

Base case: $\{\bar{x} : \bar{o} \to \bar{s}\}\langle\bar{y}/x\rangle$ **and** $x = z$

$$\begin{aligned}
&\mathsf{Read}\,(\{\bar{x} : \bar{o} \to \bar{s}\}\langle\bar{y}/x\rangle, \sigma, q, z)\\
&\quad= \mathsf{Read}\,(\{\bar{x} : \bar{o} \to \bar{s}\}\langle\bar{y}/x\rangle, \sigma, q, x)\\
&\quad= \begin{cases} \mathsf{Read}\,(\{\bar{x}x : \bar{o} \to \bar{s}\}, \sigma, q, x) & \text{if } \bar{x} \cap \bar{y} \neq \epsilon \\ \mathsf{Read}\,(\{\bar{x} : \bar{o} \to \bar{s}\}, \sigma, q, x) & \text{otherwise} \end{cases}
\end{aligned}$$

This yields two cases. *First case:* $\bar{x} \cap \bar{y} \neq \epsilon$. The readers of the left-hand is then

$$\begin{aligned}
&\mathsf{Read}\,(\{\bar{x}x : \bar{o} \to \bar{s}\}, \sigma, q, x)\\
&\quad= \mathsf{Read}\,(\{\bar{x} : \bar{o} \to \bar{s}\}, \sigma, q, x) \cap \mathsf{Read}\,(\{x : \bar{o} \to \bar{s}\}, \sigma, q, x)\\
&\quad= \begin{cases} \bar{s} & \text{if } x \in \bar{x} \wedge q \in \bar{o} \\ \mathbf{Pr} & \text{otherwise} \end{cases} \cap \begin{cases} \bar{s} & \text{if } x \in x \wedge q \in \bar{o} \\ \mathbf{Pr} & \text{otherwise} \end{cases}\\
&\quad= \begin{cases} \bar{s} & \text{if } x \in \bar{x} \wedge q \in \bar{o} \\ \mathbf{Pr} & \text{otherwise} \end{cases} \cap \begin{cases} \bar{s} & \text{if } q \in \bar{o} \\ \mathbf{Pr} & \text{otherwise} \end{cases}
\end{aligned}$$

Because $\bar{x} \cap \bar{y} \neq \epsilon$ then the readers of the right-hand side is

$$\begin{aligned}
&\bigcap_{u \in x\bar{y}} \mathsf{Read}\,(\{\bar{x} : \bar{o} \to \bar{s}\}, \sigma, q, u)\\
&\quad= \mathsf{Read}\,(\{\bar{x} : \bar{o} \to \bar{s}\}, \sigma, q, x) \cap \bigcap_{v \in \bar{y}} \mathsf{Read}\,(\{\bar{x} : \bar{o} \to \bar{s}\}, \sigma, q, v)\\
&\quad= \begin{cases} \bar{s} & \text{if } x \in \bar{x} \wedge q \in \bar{o} \\ \mathbf{Pr} & \text{otherwise} \end{cases} \cap \begin{cases} \bar{s} & \text{if } q \in \bar{o} \wedge (\bar{x} \cap \bar{y} \neq \epsilon) \\ \mathbf{Pr} & \text{otherwise} \end{cases}\\
&\quad= \begin{cases} \bar{s} & \text{if } x \in \bar{x} \wedge q \in \bar{o} \\ \mathbf{Pr} & \text{otherwise} \end{cases} \cap \begin{cases} \bar{s} & \text{if } q \in \bar{o} \\ \mathbf{Pr} & \text{otherwise} \end{cases}
\end{aligned}$$

Which is equal to the left-hand side.

Second case: $\bar{x} \cap \bar{y} = \epsilon$. Again the value of the left-hand side gives:

$$\mathsf{Read}\,(\{\bar{x} : \bar{o} \to \bar{s}\}, \sigma, q, x) = \begin{cases} \bar{s} & \text{if } x \in \bar{x} \wedge q \in \bar{o} \\ \mathbf{Pr} & \text{otherwise} \end{cases}$$

Because $\bar{x} \cap \bar{y} = \epsilon$ and $\bar{s} \subseteq \mathbf{Pr} = \bar{s}$ the readers of the right-hand side is

$$\begin{aligned}
&\bigcap_{u \in x\bar{y}} \mathsf{Read}\left(\{\bar{x} : \bar{o} \to \bar{s}\}, \sigma, q, u\right) \\
&\quad = \mathsf{Read}\left(\{\bar{x} : \bar{o} \to \bar{s}\}, \sigma, q, x\right) \cap \bigcap_{v \in \bar{y}} \mathsf{Read}\left(\{\bar{x} : \bar{o} \to \bar{s}\}, \sigma, q, v\right) \\
&\quad = \begin{cases} \bar{s} & \text{if } x \in \bar{x} \wedge q \in \bar{o} \\ \mathbf{Pr} & \text{otherwise} \end{cases} \cap \begin{cases} \bar{s} & \text{if } q \in \bar{o} \wedge (\bar{x} \cap \bar{y} \neq \epsilon) \\ \mathbf{Pr} & \text{otherwise} \end{cases} \\
&\quad = \begin{cases} \bar{s} & \text{if } x \in \bar{x} \wedge q \in \bar{o} \\ \mathbf{Pr} & \text{otherwise} \end{cases} \cap \mathbf{Pr} \\
&\quad = \begin{cases} \bar{s} & \text{if } x \in \bar{x} \wedge q \in \bar{o} \\ \mathbf{Pr} & \text{otherwise} \end{cases}
\end{aligned}$$

Which is equal to the left-hand side.

Base case: $\{\bar{x} : \bar{o} \leftarrow \bar{s}\}\langle \bar{y}/x \rangle$ **and** $x \neq z$. The readers on both sides are equal.

$$\begin{aligned}
&\mathsf{Read}\left(\{\bar{x} : \bar{o} \leftarrow \bar{s}\}\langle \bar{y}/x \rangle, \sigma, q, z\right) \\
&\quad = \begin{cases} \mathsf{Read}\left(\{\bar{x}x : \bar{o} \leftarrow \bar{s}\}, \sigma, q, z\right) & \text{if } \bar{x} \cap \bar{y} \neq \epsilon \\ \mathsf{Read}\left(\{\bar{x} : \bar{o} \leftarrow \bar{s}\}, \sigma, q, z\right) & \text{otherwise} \end{cases} \\
&\quad = \mathbf{Pr} \\
&\quad = \mathsf{Read}\left(\{\bar{x} : \bar{o} \leftarrow \bar{s}\}, \sigma, q, z\right)
\end{aligned}$$

Base case: $\{\bar{x} : \bar{o} \leftarrow \bar{s}\}\langle \bar{y}/x \rangle$ **and** $x = z$. The readers on both sides are equal.

$$\begin{aligned}
&\mathsf{Read}\left(\{\bar{x} : \bar{o} \leftarrow \bar{s}\}\langle \bar{y}/x \rangle, \sigma, q, z\right) \\
&\quad = \begin{cases} \mathsf{Read}\left(\{\bar{x}x : \bar{o} \leftarrow \bar{s}\}, \sigma, q, z\right) & \text{if } \bar{x} \cap \bar{y} \neq \epsilon \\ \mathsf{Read}\left(\{\bar{x} : \bar{o} \leftarrow \bar{s}\}, \sigma, q, z\right) & \text{otherwise} \end{cases} \\
&\quad = \mathbf{Pr} \\
&\quad = \bigcap_{u \in x\bar{y}} \mathbf{Pr} \\
&\quad = \bigcap_{u \in x\bar{y}} \mathsf{Read}\left(\{\bar{x} : \bar{o} \leftarrow \bar{s}\}, \sigma, q, u\right)
\end{aligned}$$

Base case: $(\varphi \Rightarrow \{\bar{x} : \bar{o} \leftarrow \bar{s}\})\,\langle \bar{y}/x \rangle$ **or** $(\varphi \Rightarrow \{\bar{x} : \bar{o} \to \bar{s}\})\,\langle \bar{y}/x \rangle$ **and** $x \neq z$. The readers on both sides are equal.

$$\begin{aligned}
&\mathsf{Read}\left((\varphi \Rightarrow P')\,\langle \bar{y}/x\rangle, \sigma, q, z\right)\\
&\quad= \mathsf{Read}\left((\varphi \Rightarrow P'\langle \bar{y}/x\rangle), \sigma, q, z\right)\\
&\quad= \begin{cases} \mathsf{Read}\left(P'\langle \bar{y}/x\rangle, \sigma, q, z\right) & \text{if } \sigma \models \varphi\\ \mathbf{Pr} & \text{otherwise}\end{cases}\\
&\quad= \begin{cases} \mathsf{Read}\left(P', \sigma, q, z\right) & \text{if } \sigma \models \varphi\\ \mathbf{Pr} & \text{otherwise}\end{cases}\\
&\quad= \mathsf{Read}\left((\varphi \Rightarrow P'), \sigma, q, z\right)
\end{aligned}$$

Base case: $(\varphi \Rightarrow \{\bar{x} : \bar{o} \leftarrow \bar{s}\})$ **or** $(\varphi \Rightarrow \{\bar{x} : \bar{o} \rightarrow \bar{s}\})$ **and** $x = z$. The readers on both sides are equal.

$$\begin{aligned}
&\mathsf{Read}\left((\varphi \Rightarrow P')\,\langle \bar{y}/x\rangle, \sigma, q, x\right)\\
&\quad= \mathsf{Read}\left((\varphi \Rightarrow P'\langle \bar{y}/x\rangle), \sigma, q, x\right)\\
&\quad= \begin{cases} \mathsf{Read}\left(P'\langle \bar{y}/x\rangle, \sigma, q, x\right) & \text{if } \sigma \models \varphi\\ \mathbf{Pr} & \text{otherwise}\end{cases}\\
&\quad= \begin{cases} \bigcap_{u \in x\bar{y}} \mathsf{Read}\left(P', \sigma, q, u\right) & \text{if } \sigma \models \varphi\\ \mathbf{Pr} & \text{otherwise}\end{cases}\\
&\quad= \begin{cases} \bigcap_{u \in x\bar{y}} \mathsf{Read}\left(P', \sigma, q, u\right) & \text{if } \sigma \models \varphi\\ \bigcap_{u \in x\bar{y}} \mathbf{Pr} & \text{otherwise}\end{cases}\\
&\quad= \bigcap_{u \in x\bar{y}} \begin{cases} \mathsf{Read}\left(P', \sigma, q, u\right) & \text{if } \sigma \models \varphi\\ \mathbf{Pr} & \text{otherwise}\end{cases}\\
&\quad= \bigcap_{u \in x\bar{y}} \mathsf{Read}\left((\varphi \Rightarrow P'), \sigma, q, u\right)
\end{aligned}$$

Induction hypothesis. All the base cases holds. It is now assumed that

$$\mathsf{Read}\left((\varphi \Rightarrow P)\,\langle \bar{y}/x\rangle, \sigma, q, z\right) = \begin{cases} \mathsf{Read}\left((\varphi \Rightarrow P), \sigma, q, z\right) & \text{if } x \neq z\\ \bigcap_{u \in x\bar{y}} \mathsf{Read}\left((\varphi \Rightarrow P), \sigma, q, u\right) & \text{otherwise}\end{cases}$$

Which yields two induction steps.

Induction step: $(\varphi \Rightarrow (\varphi' \Rightarrow P'))\,\langle \bar{y}/x\rangle$. The readers on both sides are equal.

$$\begin{aligned}
&\mathsf{Read}\left((\varphi \Rightarrow (\varphi' \Rightarrow P'))\,\langle a/x\rangle, \sigma, q, z\right)\\
&\quad= \mathsf{Read}\left((\varphi \wedge \varphi' \Rightarrow P')\,\langle a/x\rangle, \sigma, q, z\right)\\
&\quad= \begin{cases} \mathsf{Read}\left((\varphi \wedge \varphi' \Rightarrow P'), \sigma, q, z\right) & \text{if } x \neq z\\ \bigcap_{u \in x\bar{y}} \mathsf{Read}\left((\varphi \wedge \varphi' \Rightarrow P'), \sigma, q, u\right) & \text{otherwise}\end{cases}\\
&\quad= \begin{cases} \mathsf{Read}\left((\varphi \Rightarrow (\varphi' \Rightarrow P')), \sigma, q, z\right) & \text{if } x \neq z\\ \bigcap_{u \in x\bar{y}} \mathsf{Read}\left((\varphi \Rightarrow (\varphi' \Rightarrow P')), \sigma, q, u\right) & \text{otherwise}\end{cases}
\end{aligned}$$

Induction step: $\left(\varphi \Rightarrow P^1 \bullet P^2\right)$. The readers on both sides are equal.

$$
\begin{aligned}
&\mathsf{Read}\left(\left(\varphi \Rightarrow P^1 \bullet P^2\right)\langle a/x\rangle, \sigma, q, z\right)\\
&\quad= \mathsf{Read}\left(\left(\left(\varphi \Rightarrow P^1\right) \bullet \left(\varphi \Rightarrow P^2\right)\right)\langle a/x\rangle, \sigma, q, z\right)\\
&\quad= \mathsf{Read}\left(\left(\varphi \Rightarrow P^1\right)\langle a/x\rangle \bullet \left(\varphi \Rightarrow P^2\right)\langle a/x\rangle, \sigma, q, z\right)\\
&\quad= \mathsf{Read}\left(\left(\varphi \Rightarrow P^1\right)\langle a/x\rangle, \sigma, q, z\right) \cap \mathsf{Read}\left(\left(\varphi \Rightarrow P^2\right)\langle a/x\rangle, \sigma, q, z\right)\\
&\quad= \begin{cases}\mathsf{Read}\left(\left(\varphi \Rightarrow P^1\right), \sigma, q, z\right)\\ \bigcap_{u\in x\bar{y}} \mathsf{Read}\left(\left(\varphi \Rightarrow P^1\right), \sigma, q, u\right)\end{cases} \cap \begin{cases}\mathsf{Read}\left(\left(\varphi \Rightarrow P^2\right), \sigma, q, z\right) & \text{if } x \neq z\\ \bigcap_{u\in x\bar{y}} \mathsf{Read}\left(\left(\varphi \Rightarrow P^2\right), \sigma, q, u\right) & \text{otherwise}\end{cases}\\
&\quad= \begin{cases}\mathsf{Read}\left(\left(\varphi \Rightarrow P^1\right), \sigma, q, z\right) \cap \mathsf{Read}\left(\left(\varphi \Rightarrow P^2\right), \sigma, q, z\right) & \text{if } x \neq z\\ \bigcap_{u\in x\bar{y}} \mathsf{Read}\left(\left(\varphi \Rightarrow P^1\right), \sigma, q, u\right) \cap \bigcap_{u\in x\bar{y}} \mathsf{Read}\left(\left(\varphi \Rightarrow P^2\right), \sigma, q, u\right) & \text{otherwise}\end{cases}\\
&\quad= \begin{cases}\mathsf{Read}\left(\left(\varphi \Rightarrow P^1\right), \sigma, q, z\right) \cap \mathsf{Read}\left(\left(\varphi \Rightarrow P^2\right), \sigma, q, z\right) & \text{if } x \neq z\\ \bigcap_{u\in x\bar{y}} \mathsf{Read}\left(\left(\varphi \Rightarrow P^1\right), \sigma, q, u\right) \cap \mathsf{Read}\left(\left(\varphi \Rightarrow P^2\right), \sigma, q, u\right) & \text{otherwise}\end{cases}\\
&\quad= \begin{cases}\mathsf{Read}\left(\left(\varphi \Rightarrow P^1\right) \bullet \left(\varphi \Rightarrow P^2\right), \sigma, q, z\right) & \text{if } x \neq z\\ \bigcap_{u\in x\bar{y}} \mathsf{Read}\left(\left(\varphi \Rightarrow P^1\right) \bullet \left(\varphi \Rightarrow P^2\right), \sigma, q, u\right) & \text{otherwise}\end{cases}\\
&\quad= \begin{cases}\mathsf{Read}\left(\left(\varphi \Rightarrow P^1 \bullet P^2\right), \sigma, q, z\right) & \text{if } x \neq z\\ \bigcap_{u\in x\bar{y}} \mathsf{Read}\left(\left(\varphi \Rightarrow P^1 \bullet P^2\right), \sigma, q, u\right) & \text{otherwise}\end{cases}
\end{aligned}
$$

Base case: $(P^1 \bullet P^2)\langle \bar{y}/x\rangle$. Using all combinations of base cases for P^1 and P^2, the following template can be used to prove this last base case

$$
\begin{aligned}
&\mathsf{Read}\left((P^1 \bullet P^2)\langle \bar{y}/x\rangle, \sigma, q, z\right)\\
&\quad= \mathsf{Read}\left(P^1\langle \bar{y}/x\rangle \bullet P^2\langle \bar{y}/x\rangle, \sigma, q, z\right)\\
&\quad= \mathsf{Read}\left(P^1\langle \bar{y}/x\rangle, \sigma, q, z\right) \cap \mathsf{Read}\left(P^2\langle \bar{y}/x\rangle, \sigma, q, z\right)\\
&\quad= \begin{cases}\mathsf{Read}\left(P^1, \sigma, q, z\right) & \text{if } x \neq z\\ \bigcap_{u\in x\bar{y}} \mathsf{Read}\left(P^1, \sigma, q, u\right) & \text{otherwise}\end{cases} \cap \begin{cases}\mathsf{Read}\left(P^2, \sigma, q, z\right) & \text{if } x \neq z\\ \bigcap_{u\in x\bar{y}} \mathsf{Read}\left(P^2, \sigma, q, u\right) & \text{otherwise}\end{cases}\\
&\quad= \begin{cases}\mathsf{Read}\left(P^1, \sigma, q, z\right) \cap \mathsf{Read}\left(P^2, \sigma, q, z\right) & \text{if } x \neq z\\ \bigcap_{u\in x\bar{y}} \mathsf{Read}\left(P^1, \sigma, q, u\right) \cap \mathsf{Read}\left(P^2, \sigma, q, u\right) & \text{otherwise}\end{cases}\\
&\quad= \begin{cases}\mathsf{Read}\left(P^1 \bullet P^2, \sigma, q, z\right) & \text{if } x \neq z\\ \bigcap_{u\in x\bar{y}} \mathsf{Read}\left(P^1 \bullet P^2, \sigma, q, u\right) & \text{otherwise}\end{cases}
\end{aligned}
$$

Induction hypothesis. It is now assumed that the following holds

$$
\mathsf{Read}\left((P^1 \bullet P^2)\langle \bar{y}/x\rangle, \sigma, q, z\right) \begin{cases}\mathsf{Read}\left(P^1 \bullet P^2, \sigma, q, z\right) & \text{if } x \neq z\\ \bigcap_{u\in x\bar{y}} \mathsf{Read}\left(P^1 \bullet P^2, \sigma, q, u\right) & \text{otherwise}\end{cases} \tag{A.10}
$$

Induction step: $((P^1 \bullet P^2) \bullet P^3)\langle \bar{y}/x\rangle$. The readers on both sides are equal.

$$\begin{aligned}
&\mathsf{Read}\left((P^1 \bullet P^2 \bullet P^3)\langle a/x\rangle, \sigma, q, z\right)\\
&\quad= \mathsf{Read}\left(((P^1 \bullet P^2) \bullet P^3)\langle a/x\rangle, \sigma, q, z\right)\\
&\quad= \mathsf{Read}\left((P^1 \bullet P^2)\langle a/x\rangle \bullet P^3\langle a/x\rangle, \sigma, q, z\right)\\
&\quad= \mathsf{Read}\left((P^1 \bullet P^2)\langle a/x\rangle, \sigma, q, z\right) \cap \mathsf{Read}\left(P^3\langle a/x\rangle, \sigma, q, z\right)\\
&\quad= \begin{cases} \mathsf{Read}\left(P^1 \bullet P^2, \sigma, q, z\right) & \text{if } x \neq z\\ \bigcap_{u \in x\bar{y}} \mathsf{Read}\left(P^1 \bullet P^2, \sigma, q, u\right) & \text{otherwise}\end{cases} \cap \begin{cases} \mathsf{Read}\left(P^3, \sigma, q, z\right) & \text{if } x \neq z\\ \bigcap_{u \in x\bar{y}} \mathsf{Read}\left(P^3, \sigma, q, u\right) & \text{otherwise}\end{cases}\\
&\quad= \begin{cases} \mathsf{Read}\left(P^1 \bullet P^2, \sigma, q, z\right) \cap \mathsf{Read}\left(P^3, \sigma, q, z\right) & \text{if } x \neq z\\ \bigcap_{u \in x\bar{y}} \mathsf{Read}\left(P^1 \bullet P^2, \sigma, q, u\right) \cap \mathsf{Read}\left(P^3, \sigma, q, u\right) & \text{otherwise}\end{cases}\\
&\quad= \begin{cases} \mathsf{Read}\left(P^1 \bullet P^2 \bullet P^3, \sigma, q, z\right) & \text{if } x \neq z\\ \bigcap_{u \in x\bar{y}} \mathsf{Read}\left(P^1 \bullet P^2 \bullet P^3, \sigma, q, u\right) & \text{otherwise}\end{cases}
\end{aligned}$$

which finalises the proof.

A.8 Lemma 3.1

The proof is by induction on the inference of $\bar{y} \vdash_\ell \{\phi\}S\{\psi\}$. Each case for S is therefore individually proven.

Let $E = \bar{x} \times \mathbf{Pr} \times \bar{z}$ then the extension with the implicit flows from the set of variables $\bar{y}$ is

$$\lceil \bar{y} \rceil E = \bar{x} \times \mathbf{Pr} \times \bar{z} \cup \bar{y} \times \mathbf{Pr} \times \bar{z} = \bar{x}\bar{y} \times \mathbf{Pr} \times \bar{z} \tag{A.11}$$

A.8.1 Case $[skip_{ts}]$

Assume $\bar{y} \vdash_\ell \{\phi\}\mathsf{skip}\{\phi'\}$. The type system gives that $\phi \Rightarrow \phi'$. From the lemma it is assumed that

$$\sigma \models \phi \tag{A.12}$$

$$\vdash_\ell \langle \mathsf{skip}; \sigma\rangle \xrightarrow[\alpha]{E} \langle S'; \sigma'\rangle \tag{A.13}$$

The instrumented semantics for the skip statement gives

$$\begin{aligned} E &= \diamond \times \mathbf{Pr} \times \diamond\\ \alpha &= \tau\\ S' &= \mathsf{skip}\\ \sigma' &= \sigma \end{aligned}$$

Then ψ needs to exists such that $\bar{y} \vdash_\ell \{\psi\}S'\{\phi'\}$. Using $\psi = \phi'$ then $\bar{y} \vdash_\ell \{\psi\}\mathsf{skip}\{\phi'\}$ because $\psi \Rightarrow \phi'$.

Because $\sigma \models \phi$ and $\sigma = \sigma'$ then $\sigma' \models \phi$. Because $\phi \Rightarrow \phi'$ then $\sigma' \models \phi'$. Because $\psi = \phi'$ then $\sigma' \models \psi$, also required by the lemma.

Because $\alpha = \tau$ then $\mathsf{sec}(P_\ell, \sigma; \lceil \bar{y} \rceil E; \sigma', P_\ell)$ needs to hold. Looking at the relevant extended flows used in the first part of the security predicate gives

$$E^- = \lceil \bar{y} \rceil E \cap (\mathbf{Var}^+ \times \mathbf{Pr} \times \mathbf{Var}^+) = \lceil \bar{y} \rceil (\diamond \times \mathbf{Pr} \times \diamond) \cap (\mathbf{Var}^+ \times \mathbf{Pr} \times \mathbf{Var}^+) = \emptyset$$

due to the lack of variables in the third part of the extended flow. This part therefore holds. Next the variables used in the other part of the security predicate are determined

$$\bar{u} = \mathbf{Var} \setminus \mathsf{trd}(\lceil \bar{y} \rceil E) = \mathbf{Var} \setminus \mathsf{trd}(\lceil \bar{y} \rceil (\diamond \times \mathbf{Pr} \times \diamond)) = \mathbf{Var} \setminus \{\diamond\} = \mathbf{Var}$$

By considering all $s \in \mathbf{Pr}$ and $u \in \bar{u}$, it is then clear that

$$\begin{aligned} \mathsf{Infl}\,(P_\ell, \sigma, s, u) &\subseteq \mathsf{Infl}\,(P_\ell, \sigma', s, u) \\ \mathsf{Read}\,(P_\ell, \sigma, s, u) &\supseteq \mathsf{Read}\,(P_\ell, \sigma', s, u) \end{aligned}$$

because $\sigma = \sigma'$. The security predicate $\mathsf{sec}(P_\ell, \sigma; \lceil \bar{y} \rceil E; \sigma', P_\ell)$ therefore holds, finalising the case.

A.8.2 Case $[ass_{ts}]$

Assume $\bar{y} \vdash_\ell \{\phi\}x := a\{\phi'\}$. In the type system the following two side conditions must hold

$$\phi \Rightarrow \phi'[a/x] \tag{A.14}$$

$$(\phi \Rightarrow P_\ell\langle \bar{a}\bar{y}/x \rangle) \sqsubseteq P_\ell[a/x] \tag{A.15}$$

and from the lemma, it is assumed that

$$\sigma \models \phi \tag{A.16}$$

$$\vdash_\ell \langle x := y; \sigma \rangle \xrightarrow[\alpha]{E} \langle S'; \sigma' \rangle \tag{A.17}$$

The instrumented semantics for assignments gives

$$E = \bar{a} \diamond \times \mathbf{Pr} \times x \tag{A.18}$$

$$\alpha = \tau \tag{A.19}$$

$$S' = \mathsf{skip} \tag{A.20}$$

$$\sigma' = \sigma[x \mapsto [\![a]\!]\sigma] \tag{A.21}$$

Then ψ needs to exists such that $\bar{y} \vdash_\ell \{\psi\}S'\{\phi'\}$. Taking $\psi = \phi'$, then $\bar{y} \vdash_\ell \{\psi\}\mathsf{skip}\{\phi'\}$ holds because $\psi \Rightarrow \phi'$. Using that (A.14) holds, then $\sigma \models \phi \Rightarrow \sigma \models \phi'[a/x]$, and by using Fact 2.3

$$\begin{aligned} \sigma \models \phi'[a/x] &= \sigma \models \psi[a/x] \\ &= \sigma[x \mapsto [\![a]\!]\sigma] \models \psi \\ &= \sigma' \models \psi \end{aligned}$$

which is also required. Because α is τ, then $\mathsf{sec}(P_\ell, \sigma; \lceil\bar{y}\rceil E; \sigma', P_\ell)$ needs to hold. First we determine the extended flows used

$$\begin{aligned} E^- = \lceil\bar{y}\rceil E \cap (\mathbf{Var}^+ \times \mathbf{Pr} \times \mathbf{Var}^+) &= (\lceil\bar{y}\rceil\bar{a} \diamond \times\mathbf{Pr} \times x) \cap (\mathbf{Var}^+ \times \mathbf{Pr} \times \mathbf{Var}^+) \\ &= \bar{a}\bar{y} \times \mathbf{Pr} \times x \end{aligned}$$

Consider $(u, s, x) \in E^-$, the influencer set is then

$$\begin{aligned} \mathsf{Infl}\,(P_\ell, \sigma, s, u) &\subseteq \bigcup_{u \in x\bar{a}\bar{y}} \mathsf{Infl}\,(P_\ell, \sigma, s, u) && \text{using } u \in \bar{a}\bar{y} \\ &= \mathsf{Infl}\,(P_\ell\langle\bar{a}\bar{y}/x\rangle, \sigma, s, x) && \text{using Fact 3.3} \\ &= \mathsf{Infl}\,((\phi \Rightarrow P_\ell\langle\bar{a}\bar{y}/x\rangle)\,, \sigma, s, x) && \text{using } \sigma \models \phi \\ &\subseteq \mathsf{Infl}\,(P_\ell[a/x]\,, \sigma, s, x) && \text{using (A.15)} \\ &= \mathsf{Infl}\,(P_\ell, \sigma[x \models [\![a]\!]\sigma], s, x) && \text{using Fact 3.1} \\ &= \mathsf{Infl}\,(P_\ell, \sigma', s, x) && \text{using (A.21)} \end{aligned}$$

and the reader set is

$$\begin{aligned} \mathsf{Read}\,(P_\ell, \sigma, s, u) &\supseteq \bigcap_{u \in x\bar{a}\bar{y}} \mathsf{Read}\,(P_\ell, \sigma, s, u) && \text{using } u \in \bar{a}\bar{y} \\ &= \mathsf{Read}\,(P_\ell\langle\bar{a}\bar{y}/x\rangle, \sigma, s, x) && \text{using Fact 3.4} \\ &= \mathsf{Read}\,((\phi \Rightarrow P_\ell\langle\bar{a}\bar{y}/x\rangle)\,, \sigma, s, x) && \text{using } \sigma \models \phi \\ &\supseteq \mathsf{Read}\,(P_\ell[a/x]\,, \sigma, s, x) && \text{using (A.15)} \\ &= \mathsf{Read}\,(P_\ell, \sigma[x \mapsto [\![a]\!]\sigma], s, x) && \text{using Fact 3.2} \\ &= \mathsf{Read}\,(P_\ell, \sigma', s, x) && \text{using (A.21)} \end{aligned}$$

Next the variables used in the other part of the security predicate are determined

$$\bar{u} = \mathbf{Var} \setminus \mathsf{trd}(\lceil\bar{y}\rceil E) = \mathbf{Var} \setminus \mathsf{trd}(\lceil\bar{y}\rceil(\bar{a} \diamond \times\mathbf{Pr} \times x)) = \mathbf{Var} \setminus \{x\}$$

By considering all $u \in \bar{u}$, it is noted that $u \neq x$. The influencer set is then found for all $s \in \mathbf{Pr}$

$$\begin{aligned} \mathsf{Infl}\,(P_\ell, \sigma, s, u) &= \mathsf{Infl}\,(P\langle\bar{y}/x\rangle, \sigma, s, u) && \text{using Fact 3.3 and } u \neq x \\ &= \mathsf{Infl}\,((\phi \Rightarrow P\langle\bar{y}/x\rangle)\,, \sigma, s, u) && \text{using } \sigma \models \phi \\ &\subseteq \mathsf{Infl}\,(P_\ell[a/x]\,, \sigma, s, u) && \text{using (A.15)} \\ &= \mathsf{Infl}\,(P_\ell, \sigma[x \models [\![a]\!]\sigma], s, u) && \text{using Fact 3.1} \\ &= \mathsf{Infl}\,(P_\ell, \sigma', s, u) && \text{using (A.21)} \end{aligned}$$

and for the reader set

$$\begin{aligned} \mathsf{Read}\,(P_\ell, \sigma, s, u) &= \mathsf{Read}\,(P\langle\bar{y}/x\rangle, \sigma, s, u) && \text{using Fact 3.4 and } u \neq x \\ &= \mathsf{Read}\,((\phi \Rightarrow P\langle\bar{y}/x\rangle)\,, \sigma, s, u) && \text{using } \sigma \models \phi \\ &\supseteq \mathsf{Read}\,(P_\ell[a/x]\,, \sigma, s, u) && \text{using (A.15)} \\ &= \mathsf{Read}\,(P_\ell, \sigma[x \mapsto [\![a]\!]\sigma], s, u) && \text{using Fact 3.2} \\ &= \mathsf{Read}\,(P_\ell, \sigma', s, u) && \text{using (A.21)} \end{aligned}$$

The two parts in $\mathsf{sec}(P_\ell, \sigma; \lceil\bar{y}\rceil E; \sigma', P_\ell)$ therefore holds, which finalises the case.

A.8.3 Case $[ass^{\dagger}_{ts}]$

Assume $\bar{y} \vdash_\ell \{\phi\}x :=^{\dagger} a\{\phi'\}$. In the type system the following two side conditions must hold

$$\phi \Rightarrow \phi'[a/x] \tag{A.22}$$

$$(\phi \Rightarrow P_\ell\langle \bar{a}\bar{y}/x\rangle) \sqsubseteq (P_\ell[a/x] \bullet \{x : \mathcal{S}(\ell) \leftarrow \star\} \bullet \{x : \mathcal{S}(\ell) \rightarrow \epsilon\}) \tag{A.23}$$

and from the lemma, it is assumed that

$$\sigma \models \phi \tag{A.24}$$

$$\vdash_\ell \langle x := y; \sigma\rangle \xrightarrow[\alpha]{E} \langle S'; \sigma'\rangle \tag{A.25}$$

The instrumented semantics for bypass assignments gives

$$E = \bar{a} \diamond \times \mathbf{Pr}_{\setminus \ell} \times x \tag{A.26}$$

$$\alpha = \tau \tag{A.27}$$

$$S' = \mathsf{skip} \tag{A.28}$$

$$\sigma' = \sigma[x \mapsto [\![a]\!]\sigma] \tag{A.29}$$

Then ψ needs to exists such that $\bar{y} \vdash_\ell \{\psi\}S'\{\phi'\}$. Taking $\psi = \phi'$, then $\bar{y} \vdash_\ell \{\psi\}\mathsf{skip}\{\phi'\}$ holds because $\psi \Rightarrow \phi'$. Using that (A.14) holds, then $\sigma \models \phi \Rightarrow \sigma \models \phi'[a/x]$, and by using Fact 2.3

$$\begin{aligned}\sigma \models \phi'[a/x] &= \sigma \models \psi[a/x]\\ &= \sigma[x \mapsto [\![a]\!]\sigma] \models \psi\\ &= \sigma' \models \psi\end{aligned}$$

which is also required. Because α is τ, then $\mathsf{sec}(P_\ell, \sigma; \lceil\rceil E; \sigma', P_\ell)$ needs to hold. First we determine the extended flows used

$$E^- \triangleq \lceil\bar{y}\rceil E \cap (\mathbf{Var}^+ \times \mathbf{Pr} \times \mathbf{Var}^+) = \bar{a}\bar{y} \times \mathbf{Pr} \setminus \mathcal{S}(\ell) \times x$$

Consider $(u, s, x) \in E^-$, and realise that $s \neq \mathcal{S}(\ell)$. The influencer set is then

$$\begin{aligned}
&\mathsf{Infl}\,(P_\ell, \sigma, s, u)\\
&\quad\subseteq \bigcup_{u \in x\bar{a}\bar{y}} \mathsf{Infl}\,(P_\ell, \sigma, s, u) && \text{using } u \in \bar{a}\bar{y}\\
&\quad= \mathsf{Infl}\,(P_\ell\langle\bar{a}\bar{y}/x\rangle, \sigma, s, x) && \text{using Fact 3.4}\\
&\quad= \mathsf{Infl}\,((\phi \Rightarrow P_\ell\langle\bar{a}\bar{y}/x\rangle)\,, \sigma, s, x) && \text{using } \sigma \models \phi\\
&\quad\subseteq \mathsf{Infl}\,((P_\ell[a/x] \bullet \{x : \mathcal{S}(\ell) \leftarrow \star\} \bullet \{x : \mathcal{S}(\ell) \rightarrow \epsilon\}), \sigma, s, x) && \text{using (A.23)}\\
&\quad= \mathsf{Infl}\,(P_\ell[a/x]\,, \sigma, s, x) \cup \epsilon \cup \epsilon && \text{using } s \neq \mathcal{S}(\ell)\\
&\quad= \mathsf{Infl}\,(P_\ell[a/x]\,, \sigma, s, x)\\
&\quad= \mathsf{Infl}\,(P_\ell, \sigma[x \models [\![a]\!]\sigma], s, x) && \text{using Fact 3.2}\\
&\quad= \mathsf{Infl}\,(P_\ell, \sigma', s, x) && \text{using (A.29)}
\end{aligned}$$

and the reader set is

$$\begin{aligned}
&\mathsf{Read}\,(P_\ell, \sigma, s, u) \\
&\qquad \supseteq \bigcap_{u \in x\bar{a}\bar{y}} \mathsf{Read}\,(P_\ell, \sigma, s, u) && \text{using } u \in \bar{a}\bar{y} \\
&\qquad = \mathsf{Read}\,(P_\ell\langle \bar{a}\bar{y}/x\rangle, \sigma, s, x) && \text{using Fact 3.4} \\
&\qquad = \mathsf{Read}\,((\phi \Rightarrow P_\ell\langle \bar{a}\bar{y}/x\rangle), \sigma, s, x) && \text{using } \sigma \models \phi \\
&\qquad \supseteq \mathsf{Read}\,((P_\ell[a/x] \bullet \{x : \mathcal{S}(\ell) \leftarrow \star\} \bullet \{x : \mathcal{S}(\ell) \rightarrow \epsilon\}), \sigma, s, x) && \text{using (A.23)} \\
&\qquad = \mathsf{Read}\,(P_\ell[a/x], \sigma, s, x) \cap \mathbf{Pr} \cap \mathbf{Pr} && \text{using } s \neq \mathcal{S}(\ell) \\
&\qquad = \mathsf{Read}\,(P_\ell[a/x], \sigma, s, x) \\
&\qquad = \mathsf{Read}\,(P_\ell, \sigma[x \mapsto [\![a]\!]\sigma], s, x) && \text{using Fact 3.2} \\
&\qquad = \mathsf{Read}\,(P_\ell, \sigma', s, x) && \text{using (A.29)}
\end{aligned}$$

Next the variables used in the other part of the security predicate are determined

$$\bar{u} = \mathbf{Var} \setminus \mathsf{trd}(\lceil \bar{y} \rceil E) = \mathbf{Var} \setminus \mathsf{trd}(\lceil \bar{y} \rceil (\bar{a} \diamond \times \mathbf{Pr}_{\setminus \ell} \times x)) = \mathbf{Var} \setminus \{x\}$$

By considering $u \in \bar{u}$, it is noted that $u \neq x$. The influencer set is then found

$$\begin{aligned}
&\mathsf{Infl}\,(P_\ell, \sigma, s, u) \\
&\qquad = \mathsf{Infl}\,(P\langle \bar{y}/x\rangle, \sigma, s, u) && \text{using Fact 3.3} \\
&\qquad = \mathsf{Infl}\,((\phi \Rightarrow P\langle \bar{y}/x\rangle), \sigma, s, u) && \text{using } \sigma \models \phi \\
&\qquad \subseteq \mathsf{Infl}\,((P_\ell[a/x] \bullet \{x : \mathcal{S}(\ell) \leftarrow \star\} \bullet \{x : \mathcal{S}(\ell) \rightarrow \epsilon\}), \sigma, s, u) && \text{using (A.23)} \\
&\qquad = \mathsf{Infl}\,(P_\ell[a/x], \sigma, s, u) \cup \epsilon \cup \epsilon && \text{using } u \neq x \\
&\qquad = \mathsf{Infl}\,(P_\ell[a/x], \sigma, s, u) \\
&\qquad = \mathsf{Infl}\,(P_\ell, \sigma[x \mapsto [\![a]\!]\sigma], s, u) && \text{using Fact 3.1} \\
&\qquad = \mathsf{Infl}\,(P_\ell, \sigma', s, u) && \text{using (A.29)}
\end{aligned}$$

and for the reader set

$$\begin{aligned}
&\mathsf{Read}\,(P_\ell, \sigma, s, u) \\
&\qquad = \mathsf{Read}\,(P\langle \bar{y}/x\rangle, \sigma, s, u) && \text{using Fact 3.4} \\
&\qquad = \mathsf{Read}\,((\phi \Rightarrow P\langle \bar{y}/x\rangle), \sigma, s, u) && \text{using } \sigma \models \phi \\
&\qquad \supseteq \mathsf{Read}\,((P_\ell[a/x] \bullet \{x : \mathcal{S}(\ell) \leftarrow \star\} \bullet \{x : \mathcal{S}(\ell) \rightarrow \epsilon\}), \sigma, s, u) && \text{using (A.23)} \\
&\qquad = \mathsf{Read}\,(P_\ell[a/x], \sigma, s, u) \cap \mathbf{Pr} \cap \mathbf{Pr} && \text{using } u \neq x \\
&\qquad = \mathsf{Read}\,(P_\ell[a/x], \sigma, s, u) \\
&\qquad = \mathsf{Read}\,(P_\ell, \sigma[x \models [\![a]\!]\sigma], s, u) && \text{using Fact 3.2} \\
&\qquad = \mathsf{Read}\,(P_\ell, \sigma', s, u) && \text{using (A.29)}
\end{aligned}$$

The two parts in $\mathsf{sec}(P_\ell, \sigma; \lceil \bar{y} \rceil E; \sigma', P_\ell)$ therefore holds, which finalises the case.

A.8.4 Case $[comb_{ts}]$

Assume $\bar{y} \vdash_\ell \{\phi\}S_1; S_2\{\phi'\}$. From the type system the following two holds

$$\bar{y} \vdash_\ell \{\phi\}S_1\{\phi''\} \qquad \bar{y} \vdash_\ell \{\phi''\}S_2\{\phi'\}$$

and from the lemma it is assumed that

$$\sigma \models \phi$$
$$\vdash_\ell \langle S_1; S_2; \sigma\rangle \xrightarrow[\alpha]{E} \langle S'; \sigma'\rangle$$

From the instrumented semantic there are two cases for sequential composition. If $S_1 \neq \mathsf{skip}$ then $S' = S_1'; S_2$ and the premises for the rule then yields the assumption

$$\vdash_\ell \langle S_1; \sigma\rangle \xrightarrow[\alpha]{E} \langle S_1'; \sigma'\rangle$$

thereby defining E, α and σ'. The induction hypothesis gives that there exists ψ such that $\bar{y} \vdash_\ell \{\psi\}S_1'\{\phi''\}$, $\sigma' \models \psi$ and $\mathsf{sec}(P, \sigma; \lceil\bar{y}\rceil E; \sigma', P')$, where P and P' are dependent on α. All this then gives $\bar{y} \vdash_\ell \{\psi\}S_1'; S_2\{\phi'\}$, which completes the first case.

If $S_1 = \mathsf{skip}$ then $S' = S_2$ and the premises for the rule gives

$$\vdash_\ell \langle S_1; \sigma\rangle \xrightarrow[\alpha]{E} \langle \mathsf{skip}; \sigma'\rangle$$

The induction hypothesis gives that there exists ψ such that $\bar{y} \vdash_\ell \{\psi\}\mathsf{skip}\{\phi''\}$ and that $\sigma' \models \psi$ and therefore $\mathsf{sec}(P, \sigma; \lceil\bar{y}\rceil E; \sigma', P')$, where P and P' are dependent on α. The previous $\bar{y} \vdash_\ell \{\psi\}\mathsf{skip}\{\phi''\}$ gives that $\psi \Rightarrow \phi''$ and therefore $\bar{y} \vdash_\ell \{\phi''\}S_2\{\phi'\}$ can be written as $\bar{y} \vdash_\ell \{\psi\}S_2\{\phi'\}$, completing the case.

A.8.5 Case $[if_{ts}]$

Assume $\bar{y} \vdash_\ell \{\phi\}\mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}\{\phi'\}$. From the type system the following two holds

$$\bar{y}\bar{b} \vdash_\ell \{\phi \wedge b\}S_1\{\phi'\} \qquad \bar{y}\bar{b} \vdash_\ell \{\phi \wedge \neg b\}S_2\{\phi'\}$$

and from the lemma it is assumed that

$$\sigma \models \phi$$
$$\vdash_\ell \langle \mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}; \sigma\rangle \xrightarrow[\tau]{E} \langle S'; \sigma'\rangle$$

From the instrumented semantic there are two cases for conditional branching. If $[\![b]\!]\sigma = \mathsf{true}$ then the semantics gives

$$E = \bar{b} \diamond \times \mathbf{Pr} \times \diamond \tag{A.30}$$
$$\alpha = \tau \tag{A.31}$$
$$S' = \lceil\bar{b}\rceil S_1 \tag{A.32}$$
$$\sigma' = \sigma \tag{A.33}$$

Set $\psi = \phi \wedge b$ and it follows that

$$\frac{\bar{y}\bar{b} \vdash_{\ell} \{\psi\}S_1\{\phi'\}}{\bar{y} \vdash_{\ell} \{\psi\}\lceil\bar{b}\rceil S_1\{\phi'\}}$$

as required by the lemma. Because $\sigma \models \phi$, $\psi = \phi \wedge b$, $[\![b]\!]\sigma = \mathsf{true}$, $\sigma \models \psi$, and $\sigma' = \sigma$ then $\sigma' \models \psi$. Because $\alpha = \tau$, then $\mathsf{sec}(P_\ell, \sigma; \lceil\bar{y}\rceil E; \sigma', P_\ell)$ needs to hold. First the extended flows are determined

$$E^- = \lceil\bar{y}\rceil E \cap (\mathbf{Var}^+ \times \mathbf{Pr} \times \mathbf{Var}^+) = \lceil\bar{y}\rceil(\bar{b} \diamond \times \mathbf{Pr} \times \diamond) \cap (\mathbf{Var}^+ \times \mathbf{Pr} \times \mathbf{Var}^+) = \emptyset \tag{A.34}$$

and clearly the first part of the security predicate holds, because there are no flows. The variables used in the other part of the security predicate are determined

$$\bar{u} = \mathbf{Var} \setminus \mathsf{trd}(\lceil\bar{y}\rceil E) = \mathbf{Var} \setminus \mathsf{trd}(\lceil\bar{y}\rceil(\bar{b} \diamond \times \mathbf{Pr} \times \diamond)) = \mathbf{Var} \tag{A.35}$$

Consider all $s \in \mathbf{Pr}$ and $u \in \bar{u}$, and it is then clear that

$$\mathsf{Infl}\,(P_\ell, \sigma, s, u) \subseteq \mathsf{Infl}\,(P_\ell, \sigma', s, u)$$
$$\mathsf{Read}\,(P_\ell, \sigma, s, u) \supseteq \mathsf{Read}\,(P_\ell, \sigma', s, u)$$

because $\sigma = \sigma'$, and $\mathsf{sec}(P_\ell, \sigma; \lceil\bar{y}\rceil E; \sigma', P_\ell)$ therefore holds.

Similar can be done for the case where $[\![b]\!]\sigma = \mathsf{false}$. The semantics gives

$$E = \bar{b} \diamond \times \mathbf{Pr} \times \diamond \tag{A.36}$$
$$\alpha = \tau \tag{A.37}$$
$$S' = \lceil\bar{b}\rceil S_2 \tag{A.38}$$
$$\sigma' = \sigma \tag{A.39}$$

Set $\psi = \phi \wedge \neg b$ and it follows that

$$\frac{\bar{y}\bar{b} \vdash_{\ell} \{\psi\}S_2\{\phi'\}}{\bar{y} \vdash_{\ell} \{\psi\}\lceil\bar{b}\rceil S_2\{\phi'\}}$$

as required by the lemma. Because $\sigma \models \phi$, $\psi = \phi \wedge \neg b$, $[\![b]\!]\sigma = \mathsf{false}$, $\sigma \models \psi$, and $\sigma' = \sigma$ then $\sigma' \models \psi$. Because $\alpha = \tau$, then $\mathsf{sec}(P_\ell, \sigma; \lceil\bar{y}\rceil E; \sigma', P_\ell)$ needs to hold, which it does because E^- and $\bar{u}$ are the same as in the first subcase, thereby completing the whole case.

A.8.6 Case $[loop_{ts}]$

Assume $\bar{y} \vdash_\ell \{\phi\}\mathsf{while}\ b\ \mathsf{do}\ S\ \mathsf{od}\{\phi \wedge \neg b\}$. In the type system the following premises holds

$$\bar{y}\bar{b} \vdash_{\ell} \{\phi \wedge b\}S\{\phi\}$$

and from the lemma, it is assumed that

$$\sigma \models \phi$$
$$\vdash_\ell \langle \textsf{while } b \textsf{ do } S \textsf{ od}; \sigma\rangle \xrightarrow[\tau]{E} \langle S'; \sigma'\rangle$$

The instrumented semantics for iteration has two axioms. In the first subcase, where $[\![b]\!]\sigma = \textsf{true}$, the semantics gives

$$\begin{aligned} E &= \bar{b} \diamond \times \mathbf{Pr} \times \diamond \\ \alpha &= \tau \\ S' &= \lceil \bar{b} \rceil S; \textsf{while } b \textsf{ do } S \textsf{ od} \\ \sigma' &= \sigma \end{aligned}$$

Then there needs to exist a ψ such that $\bar{y} \vdash_\ell \{\psi\}S'\{\phi \wedge \neg b\}$. Let $\psi = \phi \wedge b$, then it would be accomplished with

$$\dfrac{\dfrac{\bar{y}\bar{b} \vdash_\ell \{\psi\}S\{\phi\}}{\bar{y} \vdash_\ell \{\psi\}\lceil \bar{b} \rceil S\{\phi\}} \quad \bar{y} \vdash_\ell \{\phi\}\textsf{while } b \textsf{ do } S \textsf{ od}\{\phi \wedge \neg b\}}{\bar{y}\bar{b} \vdash_\ell \{\psi\}\lceil \bar{b} \rceil S; \textsf{while } b \textsf{ do } S \textsf{ od}\{\phi \wedge \neg b\}}$$

as required. Clearly $\sigma' \models \psi$, because $\sigma \models \phi$, $\sigma = \sigma'$, $\psi = \phi \wedge b$ and $[\![b]\!]\sigma = \textsf{true}$. The security predicate holds $\textsf{sec}(P_\ell, \sigma; \lceil \bar{y} \rceil E; \sigma', P_\ell)$, following same structure as from (A.34) and (A.35).

In the second subcase, where $[\![b]\!]\sigma = \textsf{false}$, the semantics give the same E, α, σ and with

$$S' = \textsf{skip}$$

Then there needs to exists ψ such that $\bar{y} \vdash_\ell \{\psi\}S'\{\phi \wedge \neg b\}$. Take $\psi = \phi \wedge \neg b$; then it is clear that

$$\bar{y} \vdash_\ell \{\psi\}\textsf{skip}\{\phi \wedge \neg b\}$$

because $\psi \Rightarrow \phi \wedge \neg b$. Furthermore $\sigma' \models \psi$, because $\sigma \models \phi$, $\sigma = \sigma'$, $\psi = \phi \wedge \neg b$ and $[\![b]\!]\sigma = \textsf{false}$, and the security predicate $\textsf{sec}(P_\ell, \sigma; \lceil \bar{y} \rceil E; \sigma', P_\ell)$ follows from (A.34) and (A.35).

A.8.7 Case $[out_{ts}]$

Assume $\bar{y} \vdash_\ell \{\phi\}ch!(a_1, \ldots, a_k)\{\phi'\}$. From the type system the following two holds

$$\phi \Rightarrow \phi' \tag{A.40}$$
$$(\phi \Rightarrow P_\ell\langle \bar{a}_i\bar{y}/\#_i\rangle_{i\leq k}) \sqsubseteq P_{ch}[a_i/\#_i]_{i\leq k} \bullet \{\star_\ell : \star \leftarrow \star\} \bullet \{\star_\ell : \star \rightarrow \epsilon\} \tag{A.41}$$

and from the lemma, it is assumed that

$$\sigma \models \phi$$
$$\vdash_\ell \langle ch!(a_1, \ldots, a_k); \sigma\rangle \xrightarrow[\alpha]{E} \langle S'; \sigma'\rangle$$

The semantics for the channel send statement then gives

$$\begin{aligned} E &= \bigcup_{i \le k} \bar{a}_i \diamond \times \mathbf{Pr} \times \#_i \\ \alpha &= ch!(v_1, \ldots, v_k) \\ S' &= \mathsf{skip} \\ \sigma' &= \sigma \end{aligned}$$

Then ψ needs to exist such that $\bar{y} \vdash_\ell \{\psi\}S'\{\phi'\}$. Using $\psi = \phi$ then

$$\bar{y} \vdash_\ell \{\psi\}\mathsf{skip}\{\phi'\}$$

holds, because $\psi \Rightarrow \phi'$. Clearly $\sigma' \models \psi$ because $\sigma \models \phi$, $\sigma' = \sigma$ and $\psi = \phi$.

Because α is $ch!(v_1, \ldots, v_k)$, then

$$\mathsf{sec}(P_\ell, \sigma; \lceil \bar{y} \rceil E; \sigma'[(\#_i \mapsto v_i)_{i \le k}], P_\ell \bullet P_{ch})$$

needs to hold.

Firstly the extended flows are determined

$$\begin{aligned} E^- &= \lceil \bar{y} \rceil E \cap (\mathbf{Var}^+ \times \mathbf{Pr} \times \mathbf{Var}^+) \\ &= \left(\lceil \bar{y} \rceil \bigcup_{i \le k} \bar{a}_i \diamond \times \mathbf{Pr} \times \#_i \right) \cap (\mathbf{Var}^+ \times \mathbf{Pr} \times \mathbf{Var}^+) \\ &= \bar{y}\bar{a}_1 \ldots \bar{a}_k \times \mathbf{Pr} \times \#_1 \ldots \#_k \end{aligned}$$

Now consider $(u, s, \#_j) \in E^-$, then the influencer set is

$$\begin{aligned}
&\mathsf{Infl}\,(P_\ell, \sigma, s, u) \\
&\quad \subseteq \bigcup_{u' \in \#_j \bar{a}_j \bar{y}} \mathsf{Infl}\,(P_\ell, \sigma, s, u') && \text{using } u \in \bar{a}_j \bar{y} \\
&\quad = \mathsf{Infl}\,(P_\ell \langle \bar{a}_j \bar{y} / \#_j \rangle, \sigma, s, \#_j) && \text{using Fact 3.3} \\
&\quad = \mathsf{Infl}\,((P_\ell \langle \bar{a}_j \bar{y} / \#_j \rangle) \langle \bar{a}_{j'} \bar{y} / \#_{j'} \rangle, \sigma, s, \#_j) && \text{using Fact 3.3, } j' \ne j \\
&\quad \vdots && \text{for all } j' \ne j \\
&\quad = \mathsf{Infl}\,(P_\ell \langle \bar{a}_i \bar{y} / \#_i \rangle_{i \le k}, \sigma, s, \#_j) \\
&\quad = \mathsf{Infl}\,((\phi \Rightarrow P_\ell \langle \bar{a}_i \bar{y} / \#_i \rangle_{i \le k}), \sigma, s, \#_j) && \text{using } \sigma \models \phi \\
&\quad \subseteq \mathsf{Infl}\left(P_{ch}[a_i/\#_i]_{i \le k} \bullet \{\star_\ell : \star \leftarrow \star\} \bullet \{\star_\ell : \star \to \epsilon\}, \sigma, s, \#_j \right) && \text{using (A.41)} \\
&\quad = \mathsf{Infl}\left(P_{ch}[a_i/\#_i]_{i \le k}, \sigma, s, \#_j \right) \cup \epsilon \cup \epsilon && \text{using } \#_j \notin \star_\ell \\
&\quad = \mathsf{Infl}\left(P_{ch}[a_i/\#_i]_{i \le k}, \sigma, s, \#_j \right) \\
&\quad = \mathsf{Infl}\left(P_{ch}[v_i/\#_i]_{i \le k}, \sigma, s, \#_j \right) && \text{using } v_i = [\![a_i]\!]\sigma \\
&\quad = \mathsf{Infl}\,(P_{ch}, \sigma[(\#_i \mapsto v_i)_{i \le k}], s, \#_j) && \text{using Fact 3.1} \\
&\quad = \mathsf{Infl}\,(P_{ch}, \sigma'[(\#_i \mapsto v_i)_{i \le k}], s, \#_j) && \text{using } \sigma' = \sigma \\
&\quad = \mathsf{Infl}\,(P_\ell \bullet P_{ch}, \sigma'[(\#_i \mapsto v_i)_{i \le k}], s, \#_j) && \text{since } \#_j \notin \mathbf{Var}_\ell
\end{aligned}$$

and the reader set is

$$
\begin{aligned}
&\mathsf{Read}\,(P_\ell, \sigma, s, u) \\
&\quad \supseteq \bigcap_{u' \in \#_j \bar{a}_j \bar{y}} \mathsf{Read}\,(P_\ell, \sigma, s, u') && \text{using } u \in \bar{a}_j \bar{y} \\
&\quad = \mathsf{Read}\,(P_\ell\langle \bar{a}_j \bar{y}/\#_j\rangle, \sigma, s, \#_j) && \text{using Fact 3.4} \\
&\quad = \mathsf{Read}\,((P_\ell\langle \bar{a}_j \bar{y}/\#_j\rangle)\langle \bar{a}_{j'} \bar{y}/\#_{j'}\rangle, \sigma, s, \#_j) && \text{using Fact 3.4, } j' \neq j \\
&\quad \vdots && \text{for all } j' \neq j \\
&\quad = \mathsf{Read}\,(P_\ell\langle \bar{a}_i \bar{y}/\#_i\rangle_{i \leq k}, \sigma, s, \#_j) \\
&\quad = \mathsf{Read}\,((\phi \Rightarrow P_\ell\langle \bar{a}_i \bar{y}/\#_i\rangle_{i \leq k}), \sigma, s, \#_j) && \text{using } \sigma \models \phi \\
&\quad \supseteq \mathsf{Read}\left(P_{ch}[a_i/\#_i]_{i \leq k} \bullet \{\star_\ell : \star \leftarrow \star\} \bullet \{\star_\ell : \star \rightarrow \epsilon\}, \sigma, s, \#_j\right) && \text{using (A.41)} \\
&\quad = \mathsf{Read}\left(P_{ch}[a_i/\#_i]_{i \leq k}, \sigma, s, \#_j\right) \cap \mathbf{Pr} \cap \mathbf{Pr} && \text{using } \#_j \notin \star_\ell \\
&\quad = \mathsf{Read}\left(P_{ch}[a_i/\#_i]_{i \leq k}, \sigma, s, \#_j\right) \\
&\quad = \mathsf{Read}\left(P_{ch}[v_i/\#_i]_{i \leq k}, \sigma, s, \#_j\right) && \text{using } v_i = [\![a_i]\!]\sigma \\
&\quad = \mathsf{Read}\,(P_{ch}, \sigma[(\#_i \mapsto v_i)_{i \leq k}], s, \#_j) && \text{using Fact 3.2} \\
&\quad = \mathsf{Read}\,(P_{ch}, \sigma'[(\#_i \mapsto v_i)_{i \leq k}], s, \#_j) && \text{using } \sigma' = \sigma \\
&\quad = \mathsf{Read}\,(P_\ell \bullet P_{ch}, \sigma'[(\#_i \mapsto v_i)_{i \leq k}], s, \#_j) && \text{since } \#_j \notin \mathbf{Var}_\ell
\end{aligned}
$$

as required.

Next the variables used in the other part of the security predicate are determined

$$
\begin{aligned}
\bar{u} &= \mathbf{Var} \setminus \mathsf{trd}(\lceil \bar{y} \rceil E) \\
&= \mathbf{Var} \setminus \mathsf{trd}(\lceil \bar{y} \rceil \bigcup_{i \leq k} \bar{a}_i \diamond \times \mathbf{Pr} \times \#_i) \\
&= \mathbf{Var} \setminus \bigcup_{i \leq k} \#_i \\
&= \mathbf{Var}
\end{aligned}
$$

Next consider all $s \in \mathbf{Pr}$ and $u \in \bar{u}$, and notice that $u \neq \#_j$ for all $j \leq k$. The influencer and reader set can be determined

$$
\begin{aligned}
&\mathsf{Infl}\,(P_\ell, \sigma, s, u) \\
&\quad = \mathsf{Infl}\,(P_\ell, \sigma[(\#_i \mapsto v_i)_{i \leq k}], s, u) && \text{using } \#_j \notin \mathsf{fv}(P_\ell) \text{ for all } j \leq k \\
&\quad = \mathsf{Infl}\,(P_\ell, \sigma'[(\#_i \mapsto v_i)_{i \leq k}], s, u) && \text{using } \sigma' = \sigma \\
&\quad = \mathsf{Infl}\,(P_\ell \bullet P_{ch}, \sigma'[(\#_i \mapsto v_i)_{i \leq k}], s, u) && \text{using } u \notin \mathsf{fv}(P_{ch}) \\
&\mathsf{Read}\,(P_\ell, \sigma, s, u) \\
&\quad = \mathsf{Read}\,(P_\ell, \sigma[(\#_i \mapsto v_i)_{i \leq k}], s, u) && \text{using } \#_j \notin \mathsf{fv}(P_\ell) \text{ for all } j \leq k \\
&\quad = \mathsf{Read}\,(P_\ell, \sigma'[(\#_i \mapsto v_i)_{i \leq k}], s, u) && \text{using } \sigma' = \sigma \\
&\quad = \mathsf{Read}\,(P_\ell \bullet P_{ch}, \sigma'[(\#_i \mapsto v_i)_{i \leq k}], s, u) && \text{using } u \notin \mathsf{fv}(P_{ch})
\end{aligned}
$$

which also holds as required. The security predicate $\mathsf{sec}(P_\ell, \sigma; \lceil\bar{y}\rceil E; \sigma'[(\#_i \mapsto v_i)_{i\leq k}], P_\ell \bullet P_{ch})$ therefore holds, concluding the case.

A.8.8 Case $[out^\dagger_{ts}]$

The proof for this case follows as the same pattern as for $[out_{ts}]$ and $[ass^\dagger_{ts}]$, and is thus omitted.

A.8.9 Case $[in_{ts}]$

Assume $\bar{y} \vdash_\ell \{\phi\}ch?(x_1, \ldots, x_k)\{\phi'\}$. From the type system the following two holds

$$((\exists x_1, \ldots, x_k.\phi) \Rightarrow \phi') \tag{A.42}$$

$$(\phi \Rightarrow P_\ell\langle\bar{y}/x_i\rangle_{i\leq k}) \bullet P_{ch}\langle\#_i/x_i\rangle_{i\leq k} \sqsubseteq P_\ell[\#_i/x_i]_{i\leq k} \bullet \{\star_\# : \star \leftarrow \star\} \bullet \{\star_\# : \star \rightarrow \epsilon\} \tag{A.43}$$

and from the lemma, it is assumed that

$$\sigma \models \phi$$
$$\vdash_\ell \langle ch?(x_1, \ldots, x_k); \sigma\rangle \xrightarrow[\alpha]{E} \langle S'; \sigma'\rangle$$

The semantics for the channel receive statement then gives

$$\begin{aligned} E &= \bigcup_{i\leq k} \#_i \times \mathbf{Pr} \times x_i \\ \alpha &= ch?(v_1, \ldots, v_k) \\ S' &= \mathsf{skip} \\ \sigma' &= \sigma[(x_i \mapsto v_i)_{i\leq k}] \end{aligned}$$

Then ψ needs to exist such that $\bar{y} \vdash_\ell \{\psi\}S'\{\phi'\}$. Using $\psi = (\exists x_1, \ldots, x_k.\phi)$ then

$$\bar{y} \vdash_\ell \{\psi\}\mathsf{skip}\{\phi'\}$$

holds, because $\psi \Rightarrow \phi'$. Furthermore $\sigma \models \phi$ implies $\sigma[(x_i \mapsto v_i)_{i\leq k}] \models \exists x_1, \ldots, x_k.\phi$ (x_i is then fixed in σ, and therefore there exists values for x_i in the satisfaction relation) and therefore $\sigma' \models \psi$.

Because α is $ch?(v_1, \ldots, v_k)$, then

$$\mathsf{sec}(P_\ell \bullet P_{ch}, \sigma[(\#_i \mapsto v_i)_{i\leq k}]; \lceil\bar{y}\rceil E; \sigma', P_\ell)$$

needs to hold. Firstly the extended flows are determined

$$\begin{aligned} E^- &= \lceil\bar{y}\rceil E \cap (\mathbf{Var}^+ \times \mathbf{Pr} \times \mathbf{Var}^+) \\ &= \left(\lceil\bar{y}\rceil \bigcup_{i\leq k} \#_i \times \mathbf{Pr} \times x_i\right) \cap (\mathbf{Var}^+ \times \mathbf{Pr} \times \mathbf{Var}^+) \\ &= \bar{y}\#_1 \ldots \#_k \times \mathbf{Pr} \times x_1 \ldots x_k \end{aligned}$$

Now consider all $(u, s, x_j) \in E^-$, and $\sigma^+ = \sigma[(\#_i \mapsto v_i)_{i \leq k}]$ for simplified notation. It is seen that $\sigma^+ \models \phi$ because $\sigma \models \phi$, so having more fixed variables in σ^+ the satisfaction relation would still hold. The influencer set is then

$$\begin{aligned}
&\mathsf{Infl}\left(P_\ell \bullet P_{ch}, \sigma^+, s, u\right) \\
&\subseteq \bigcup_{u' \in x_j \bar{y}\#_j} \mathsf{Infl}\left(P_\ell \bullet P_{ch}, \sigma^+, s, u'\right) && \text{using } u \in \bar{y}\#_j \\
&\quad = \mathsf{Infl}\left((P_\ell \bullet P_{ch})\langle \bar{y}\#_j / x_j\rangle, \sigma^+, s, x_j\right) && \text{using Fact 3.3} \\
&\quad = \mathsf{Infl}\left(((P_\ell \bullet P_{ch})\langle \bar{y}\#_j / x_j\rangle)\langle \bar{y}\#_{j'} / x_{j'}\rangle, \sigma^+, s, x_j\right) && \text{using Fact 3.3, } j' \neq j \\
&\quad \vdots && \text{for all } j' \neq j \\
&\quad = \mathsf{Infl}\left((P_\ell \bullet P_{ch})\langle \bar{y}\#_i / x_i\rangle_{i \leq k}, \sigma^+, s, x_j\right) \\
&\quad = \mathsf{Infl}\left(P_\ell\langle \bar{y}/x_i\rangle_{i \leq k} \bullet P_{ch}\langle \#_i / x_i\rangle_{i \leq k}, \sigma^+, s, x_j\right) && \text{using } \begin{array}{r}\forall y \in \bar{y} : y \notin \mathsf{fv}(P_{ch}) \wedge \\ \forall i \leq k : \#_i \notin \mathsf{fv}(P_\ell)\end{array} \\
&\quad = \mathsf{Infl}\left((\phi \Rightarrow P_\ell\langle \bar{y}/x_i\rangle_{i \leq k}) \bullet P_{ch}\langle \#_i / x_i\rangle_{i \leq k}, \sigma^+, s, x_j\right) && \text{using } \sigma^+ \models \phi \\
&\subseteq \mathsf{Infl}\left(P_\ell[\#_i / x_i]_{i \leq k} \bullet \{\star_\# : \star \leftarrow \star\} \bullet \{\star_\# : \star \rightarrow \epsilon\}, \sigma^+, s, x_j\right) && \text{using (A.43)} \\
&\quad = \mathsf{Infl}\left(P_\ell[\#_i / x_i]_{i \leq k}, \sigma^+, s, x_j\right) \cup \epsilon \cup \epsilon && \text{using } x_j \notin \star_\# \\
&\quad = \mathsf{Infl}\left(P_\ell[\#_i / x_i]_{i \leq k}, \sigma^+, s, x_j\right) \\
&\quad = \mathsf{Infl}\left(P_\ell, \sigma^+[(x_i \mapsto v_i)_{i \leq k}], s, x_j\right) && \text{using Fact 3.1} \\
&\quad = \mathsf{Infl}\left(P_\ell, \sigma[(x_i \mapsto v_i)_{i \leq k}], s, x_j\right) && \text{using } \#_j \notin \mathsf{fv}(P_\ell)
\end{aligned}$$

and likewise the reader set is then

$$\begin{aligned}
&\mathsf{Read}\left(P_\ell \bullet P_{ch}, \sigma^+, s, u\right) \\
&\supseteq \bigcap_{u' \in x_j \bar{y}\#_j} \mathsf{Read}\left(P_\ell \bullet P_{ch}, \sigma^+, s, u'\right) && \text{using } u \in \bar{y}\#_j \\
&\quad = \mathsf{Read}\left((P_\ell \bullet P_{ch})\langle \bar{y}\#_j / x_j\rangle, \sigma^+, s, x_j\right) && \text{using Fact 3.4} \\
&\quad = \mathsf{Read}\left(((P_\ell \bullet P_{ch})\langle \bar{y}\#_j / x_j\rangle)\langle \bar{y}\#_{j'} / x_{j'}\rangle, \sigma^+, s, x_j\right) && \text{using Fact 3.4, } j' \neq j \\
&\quad \vdots && \text{for all } j' \neq j \\
&\quad = \mathsf{Read}\left((P_\ell \bullet P_{ch})\langle \bar{y}\#_i / x_i\rangle_{i \leq k}, \sigma^+, s, x_j\right) \\
&\quad = \mathsf{Read}\left(P_\ell\langle \bar{y}/x_i\rangle_{i \leq k} \bullet P_{ch}\langle \#_i / x_i\rangle_{i \leq k}, \sigma^+, s, x_j\right) && \text{using } \begin{array}{r}\forall y \in \bar{y} : y \notin \mathsf{fv}(P_{ch}) \wedge \\ \forall i \leq k : \#_i \notin \mathsf{fv}(P_\ell)\end{array} \\
&\quad = \mathsf{Read}\left((\phi \Rightarrow P_\ell\langle \bar{y}/x_i\rangle_{i \leq k}) \bullet P_{ch}\langle \#_i / x_i\rangle_{i \leq k}, \sigma^+, s, x_j\right) && \text{using } \sigma^+ \models \phi \\
&\supseteq \mathsf{Read}\left(P_\ell[\#_i / x_i]_{i \leq k} \bullet \{\star_\# : \star \leftarrow \star\} \bullet \{\star_\# : \star \rightarrow \epsilon\}, \sigma^+, s, x_j\right) && \text{using (A.43)} \\
&\quad = \mathsf{Read}\left(P_\ell[\#_i / x_i]_{i \leq k}, \sigma^+, s, x_j\right) \cap \epsilon \cap \epsilon && \text{using } x_j \notin \star_\# \\
&\quad = \mathsf{Read}\left(P_\ell[\#_i / x_i]_{i \leq k}, \sigma^+, s, x_j\right) \\
&\quad = \mathsf{Read}\left(P_\ell, \sigma^+[(x_i \mapsto v_i)_{i \leq k}], s, x_j\right) && \text{using Fact 3.2} \\
&\quad = \mathsf{Read}\left(P_\ell, \sigma[(x_i \mapsto v_i)_{i \leq k}], s, x_j\right) && \text{using } \#_j \notin \mathsf{fv}(P_\ell)
\end{aligned}$$

as required.

Next the variables used in the other part of the security predicate are determined

$$\begin{aligned}\bar{u} &= \mathbf{Var} \setminus \mathsf{trd}(\lceil \bar{y} \rceil E) \\ &= \mathbf{Var} \setminus \mathsf{trd}(\lceil \bar{y} \rceil \bigcup_{i \leq k} \#_i \times \mathbf{Pr} \times x_i) \\ &= \mathbf{Var} \setminus \bigcup_{i \leq k} x_i\end{aligned}$$

Next consider all $s \in \mathbf{Pr}$ and $u \in \bar{u}$, and notice that $u \neq x_j$ for all $j \leq k$. The influencer and reader sets can be determined

$$\begin{aligned}
&\mathsf{Infl}\left(P_\ell \bullet P_{ch}, \sigma^+, s, u\right) && \\
&\quad = \mathsf{Infl}\left(P_\ell, \sigma^+, s, u\right) && \text{using } u \notin \mathsf{fv}(P_{ch}) \text{ for all } j \leq k \\
&\quad = \mathsf{Infl}\left(P_\ell\langle \bar{y}/x_i \rangle_{i \leq k}, \sigma^+, s, u\right) && \text{using Fact 3.3 and } u \neq x_j \\
&\quad = \mathsf{Infl}\left((\phi \Rightarrow P_\ell\langle \bar{y}/x_i \rangle_{i \leq k}), \sigma^+, s, u\right) && \text{using } \sigma^+ \models \phi \\
&\subseteq \mathsf{Infl}\left((\phi \Rightarrow P_\ell\langle \bar{y}/x_i \rangle_{i \leq k}) \bullet P_{ch}\langle \#_i/x_i \rangle_{i \leq k}, \sigma^+, s, u\right) && \\
&\subseteq \mathsf{Infl}\left(P_\ell[\#_i/x_i]_{i \leq k} \bullet \{\star_\# : \star \leftarrow \star\} \bullet \{\star_\# : \star \rightarrow \epsilon\}, \sigma^+, s, u\right) && \text{using (A.43)} \\
&\quad = \mathsf{Infl}\left(P_\ell[\#_i/x_i]_{i \leq k}, \sigma^+, s, u\right) \cup \epsilon \cup \epsilon && \text{using } u \notin \star_\# \\
&\quad = \mathsf{Infl}\left(P_\ell, \sigma^+[(x_i \mapsto v_i)_{i \leq k}], s, u\right) && \text{using } \sigma^+(\#_i) = v_i \text{ for all } i \leq k \\
&\quad = \mathsf{Infl}\left(P_\ell, \sigma[(x_i \mapsto v_i)_{i \leq k}], s, u\right) && \text{using } u \notin \star_\#
\end{aligned}$$

which also holds as required. The security predicate $\mathsf{sec}(P_\ell, \sigma; \lceil \bar{y} \rceil E; \sigma'[(\#_i \mapsto v_i)_{i \leq k}], P_\ell \bullet P_{ch})$ therefore holds, concluding the case.

A.8.10 Case $[cho_{ts}]$

Assume $\bar{y} \vdash_\ell \{\phi\}((S_1) \oplus (S))\{\phi'\}$. From the type system the following two holds

$$\bar{y} \vdash_\ell \{\phi\} S_1 \{\phi'\} \qquad \bar{y} \vdash_\ell \{\phi\} S_2 \{\phi'\}$$

and from the lemma it is assumed that

$$\sigma \models \phi$$
$$\vdash_\ell \langle ((S_1) \oplus (S_2)); \sigma \rangle \xrightarrow[\alpha]{E} \langle S'; \sigma' \rangle$$

From the instrumented semantic there are two cases for nondeterministic choice. Nondeterministic then it gives $S' = S_1'; S_2$ and the premises for the rule then gives the assumption

$$\vdash_\ell \langle S_1; \sigma \rangle \xrightarrow[\alpha]{E} \langle S_1'; \sigma' \rangle$$

thereby defining E, α and σ'. The induction hypothesis gives that there exists ψ such that $\bar{y} \vdash_\ell \{\psi\}S_1'\{\phi''\}$, $\sigma' \models \psi$ and $\mathsf{sec}(P, \sigma; \lceil\bar{y}\rceil E; \sigma', P')$, where P and P' are depended on α. All this then gives $\bar{y} \vdash_\ell \{\psi\}S_1'; S_2\{\phi'\}$, which completes the first case.

If $S_1 = \mathsf{skip}$ then $S' = S_2$ and the premises for the rule gives

$$\vdash_\ell \langle S_1; \sigma\rangle \xrightarrow[\alpha]{E} \langle \mathsf{skip}; \sigma'\rangle$$

The induction hypothesis gives that there exists ψ such that $\bar{y} \vdash_\ell \{\psi\}\mathsf{skip}\{\phi''\}$ and that $\sigma' \models \psi$ and therefore $\mathsf{sec}(P, \sigma; \lceil\bar{y}\rceil E; \sigma', P')$, where P and P' are depended on α. From previous $\bar{y} \vdash_\ell \{\psi\}\mathsf{skip}\{\phi''\}$ gives that $\psi \Rightarrow \phi''$ and therefore $\bar{y} \vdash_\ell \{\phi''\}S_2\{\phi'\}$ can be written as $\bar{y} \vdash_\ell \{\psi\}S_2\{\phi'\}$ completing the case.

A.8.11 Case $[impl_{ts}]$

Assume $\bar{y} \vdash_\ell \{\phi\}\lceil\bar{x}\rceil S\{\phi'\}$. In the type system the following premises holds for $\bar{y} \vdash_\ell \{\phi\}\lceil\bar{x}\rceil S\{\phi'\}$

$$\bar{y}\bar{x} \vdash_\ell \{\phi\}S\{\phi'\}$$

and from the lemma, it is assumed that

$$\sigma \models \phi$$
$$\vdash_\ell \langle \lceil\bar{x}\rceil S; \sigma\rangle \xrightarrow[\alpha]{E} \langle S'; \sigma'\rangle$$

The instrumented semantics for this statement have two matching rules. Assuming that $S' = \lceil\bar{x}\rceil S''$ and $S'' \neq \mathsf{skip}$ then the premise for the semantic rule is

$$\vdash_\ell \langle S; \sigma\rangle \xrightarrow[\alpha]{E'} \langle S''; \sigma'\rangle$$

which determines α, σ' and $E = \lceil\bar{x}\rceil E'$. The induction hypothesis then gives that there exists ψ such that $\bar{y}\bar{x} \vdash_\ell \{\psi\}S''\{\phi'\}$ and that $\sigma' \models \psi$ and therefore $\mathsf{sec}(P, \sigma;$ $\lceil\bar{y}\bar{x}\rceil E; \sigma', P')$, where P and P' are dependent on α. It can now be seen that

$$\frac{\bar{y}\bar{x} \vdash_\ell \{\psi\}S''\{\phi'\}}{\bar{y} \vdash_\ell \{\psi\}\lceil\bar{x}\rceil S''\{\phi'\}}$$

and thereby that $\bar{y} \vdash_\ell \{\psi\}S'\{\phi'\}$ because $S' = \lceil\bar{x}\rceil S''$. Still $\sigma' \models \psi$, and the security predicate becomes $\mathsf{sec}(P, \sigma; \lceil\bar{y}\rceil(\lceil\bar{x}\rceil E); \sigma', P')$ which holds looking at the influencers and readers as with previous cases.

A.8.12 Case $[cons_{ts}]$

The proof is straight forward as in $[skip_{ts}]$ and are thus omitted.

A.9 Theorem 3.1

The instrumented semantics for systems have two rules and the proof therefore have two cases

A.9.1 Case $[sys_{is}^{\tau}]$

From the theorem it is assumed that

$$\begin{gathered}\langle l_1 : S_1 \parallel \cdots \parallel l_i : S_i \parallel \cdots \parallel l_n : S_n, \sigma\rangle \xRightarrow[U,D]{F} \\ \langle l_1 : S_1 \parallel \cdots \parallel l_i : S_i' \parallel \cdots \parallel l_n : S_n, \sigma'\rangle\end{gathered} \tag{A.44}$$

$$\vdash \{\phi_1 \& \cdots \& \phi_i \& \cdots \& \phi_n\}\ l_1 : S_1 \parallel \cdots \parallel l_i : S_i \parallel \cdots \parallel l_n : S_n \tag{A.45}$$

$$\sigma \models \phi_1 \wedge \cdots \wedge \phi_i \wedge \cdots \wedge \phi_n \tag{A.46}$$

The rule $[sys_{is}^{\tau}]$ gives

$$\vdash_{l_i} \langle S_i; \sigma\rangle \xrightarrow[\tau]{E_i} \langle S_i'; \sigma'\rangle \tag{A.47}$$

and hence $F = E_i \cap (\mathbf{Var} \times \mathbf{Pr} \times \mathbf{Var})$, $U = \mathsf{fst}(E_i) \cap \mathbf{Var}$, and $D = \mathsf{trd}(E_i) \cap \mathbf{Var}$. The assumption in (A.45) gives

$$\epsilon \vdash_{l_i} \{\phi_i\} S_i \{\mathsf{true}\} \tag{A.48}$$

and from (A.46) we have $\sigma \models \phi_i$, which together with (A.47) and (A.48) gives the assumptions for Lemma 3.1. The lemma gives that there exists ψ_i such that $\epsilon \vdash_{l_i} \{\psi_i\} S_i' \{\mathsf{true}\}$ and $\sigma' \models \psi_i$ and

$$\mathsf{sec}(P_{l_i}, \sigma; E_i; \sigma', P_{l_i}) \tag{A.49}$$

Now let $\psi_j = \phi_j$ and $S_j' = S_j$ for all $j \neq i$, and there now exists $\psi_1, \ldots, \psi_i, \ldots, \psi_n$ such that

$$\vdash \{\psi_1 \& \cdots \& \psi_i \& \cdots \& \psi_n\}\ l_1 : S_1' \parallel \cdots \parallel l_i : S_i' \parallel \cdots \parallel l_n : S_n'$$

as required by the theorem. Furthermore because $\sigma(x) = \sigma'(x)$ when $x \notin \mathbf{Var}_{l_i}$ then $\sigma' \models \psi_1 \wedge \cdots \wedge \psi_i \wedge \cdots \wedge \psi_n$, which is also required.

For the last part we need to prove the security predicate $\mathsf{sec}_{P_\bullet}(\sigma; U, F, D; \sigma')$. By using (2.6) and (2.11) it is noted that $\mathsf{Infl}\,(P_\ell, \sigma, s, u) = \epsilon$ and $\mathsf{Read}\,(P_\ell, \sigma, s, u) = \mathbf{Pr}$ if $u \notin \mathbf{Var}_\ell$, and therefore

$$\mathsf{Infl}\,(P_\bullet, \sigma, s, u) = \mathsf{Infl}\,(P_\ell, \sigma, s, u) \text{ if } u \in \mathbf{Var}_\ell \tag{A.50}$$

$$\mathsf{Read}\,(P_\bullet, \sigma, s, u) = \mathsf{Read}\,(P_\ell, \sigma, s, u) \text{ if } u \in \mathbf{Var}_\ell \tag{A.51}$$

Take now $(u, s, u') \in F$, and see that we have that $u, u' \in \mathbf{Var}_{l_i}$. The first property of the security predicate for systems holds for influencers and readers

$$\begin{aligned}
\mathsf{Infl}\,(P_\bullet, \sigma, s, u) &= \mathsf{Infl}\,(P_{l_i}, \sigma, s, u) && \text{using (A.50)}\\
&\supseteq \mathsf{Infl}\,(P_{l_i}, \sigma', s, u') && \text{using (A.49)}\\
&= \mathsf{Infl}\,(P_\bullet, \sigma', s, u') && \text{using (A.50)}\\
\mathsf{Read}\,(P_\bullet, \sigma, s, u) &= \mathsf{Read}\,(P_{l_i}, \sigma, s, u) && \text{using (A.51)}\\
&\supseteq \mathsf{Read}\,(P_{l_i}, \sigma', s, u') && \text{using (A.49)}\\
&= \mathsf{Read}\,(P_\bullet, \sigma', s, u') && \text{using (A.51)}
\end{aligned}$$

For the other part of the security predicate take all s and $u \in \mathbf{Var} \setminus D$. There are now two cases, take first where $u \in \mathbf{Var}_{l_i}$ then

$$\begin{aligned}
\mathsf{Infl}\,(P_\bullet, \sigma, s, u) &= \mathsf{Infl}\,(P_{l_i}, \sigma, s, u) && \text{using (A.50)}\\
&\supseteq \mathsf{Infl}\,(P_{l_i}, \sigma', s, u) && \text{using (A.49)}\\
&= \mathsf{Infl}\,(P_\bullet, \sigma', s, u) && \text{using (A.50)}\\
\mathsf{Read}\,(P_\bullet, \sigma, s, u) &= \mathsf{Read}\,(P_{l_i}, \sigma, s, u) && \text{using (A.51)}\\
&\supseteq \mathsf{Read}\,(P_{l_i}, \sigma', s, u) && \text{using (A.49)}\\
&= \mathsf{Read}\,(P_\bullet, \sigma', s, u) && \text{using (A.51)}
\end{aligned}$$

Secondly, if $u \notin \mathbf{Var}_{l_i}$ the same result holds, following (A.50) and (A.51) because that $\sigma(x) = \sigma'(x)$ when $x \notin \mathbf{Var}_{l_i}$.

A.9.2 Case $[sys^{ch}_{is}]$

From the theorem it is assumed that

$$\begin{aligned}
&\langle l_1 : S_1 \parallel \cdots \parallel l_i : S_i \parallel \cdots \parallel l_i : S_j \parallel \cdots \parallel l_n : S_n, \sigma\rangle \xRightarrow[U,D]{F} \\
&\qquad \langle l_1 : S_1 \parallel \cdots \parallel l_i : S_i' \parallel \cdots \parallel l_i : S_j' \parallel \cdots \parallel l_n : S_n, \sigma''\rangle
\end{aligned} \tag{A.52}$$

$$\vdash \{\phi_1 \& \cdots \& \phi_i \& \cdots \& \phi_j \& \cdots \& \phi_n\}\ l_1 : S_1 \parallel \cdots \parallel l_i : S_i \parallel \cdots \parallel l_j : S_j \parallel \cdots \parallel l_n : S_n \tag{A.53}$$

$$\sigma \models \phi_1 \wedge \cdots \wedge \phi_i \wedge \cdots \wedge \phi_j \wedge \cdots \wedge \phi_n \tag{A.54}$$

The rule $[sys^{ch}_{is}]$ gives

$$\vdash_{l_i} \langle S_i; \sigma\rangle \xrightarrow[ch!(v_1,\ldots,v_k)]{E_i} \langle S_i'; \sigma'\rangle \tag{A.55}$$

$$\vdash_{l_j} \langle S_j; \sigma'\rangle \xrightarrow[ch?(v_1,\ldots,v_k)]{E_j} \langle S_j'; \sigma''\rangle \tag{A.56}$$

and hence $F = E_i \parallel E_j \cap (\mathbf{Var} \times \mathbf{Pr} \times \mathbf{Var})$, $U = \mathsf{fst}(E_i \parallel E_j) \cap \mathbf{Var}$, and $D = \mathsf{trd}(E_i \parallel E_j) \cap \mathbf{Var}$, where $i \neq j$.

The assumption in (A.53) gives

$$\epsilon \vdash_{l_i} \{\phi_i\}S_i\{\mathsf{true}\} \tag{A.57}$$

$$\epsilon \vdash_{l_j} \{\phi_j\}S_j\{\mathsf{true}\} \tag{A.58}$$

and from (A.54) we have $\sigma \models \phi_i$ which together with (A.55) and (A.57) gives the assumptions for Lemma 3.1. The lemma gives that there exists ψ_i such that $\epsilon \vdash_{l_i} \{\psi_i\}S'_i\{\mathsf{true}\}$ and $\sigma' \models \psi_i$ and

$$\mathsf{sec}(P_{l_i}, \sigma; E_i; \sigma'[(\#_{k'} \mapsto v_{k'})_{k' \le k}], P_{l_i} \bullet P_{ch}) \tag{A.59}$$

Furthermore $\sigma \models \phi_j$ and because $i \neq j$ then $\sigma' \models \phi_j$ because l_i only changes its own values. The lemma therefore also gives that there exists ψ_i such that $\epsilon \vdash_{l_j} \{\psi_j\}S'_j\{\mathsf{true}\}$ and $\sigma'' \models \psi_j$ and

$$\mathsf{sec}(P_{l_j} \bullet P_{ch}, \sigma'[(\#_{k'} \mapsto v_{k'})_{k' \le k}]; E_j; \sigma'', P_{l_j}) \tag{A.60}$$

Now let $\psi_\ell = \phi_\ell$ and $S'_\ell = S_\ell$ for all $\ell \neq i, j$, and there now exists $\psi_1, \ldots, \psi_i, \ldots,$ $\psi_j, \ldots, \psi_n$ such that

$$\vdash \{\psi_1 \& \cdots \& \psi_i \& \cdots \& \psi_j \& \cdots \& \psi_n\}\ l_1 : S'_1 \parallel \cdots \parallel l_i : S'_i \parallel \cdots \parallel l_j : S'_j \parallel \cdots \parallel l_n : S'_n$$

as required by the theorem. Furthermore because $\sigma(x) = \sigma'(x)$ when $x \notin \mathbf{Var}_{l_i}$ and $\sigma'(x) = \sigma''(x)$ when $x \notin \mathbf{Var}_{l_j}$ then $\sigma'' \models \psi_1 \wedge \cdots \wedge \psi_i \wedge \cdots \wedge \psi_j \wedge \cdots \wedge \psi_n$, which also is required.

For the second part we need to prove the security predicate $\mathsf{sec}_{P_\bullet}(\sigma; U, F, D; \sigma'')$. The flows F contains the combination of $(u, s, \#_m) \in E_i$ and $(\#_m, s, u') \in E_j$, considerer therefore such $(u, s, u') \in F$, and then we have

$$\begin{aligned}
&\mathsf{Infl}\,(P_\bullet, \sigma, s, u) \\
&\quad = \mathsf{Infl}\,(P_{l_i}, \sigma, s, u) && \text{using (A.50)} \\
&\quad \subseteq \mathsf{Infl}\,(P_{l_i} \bullet P_{ch}, \sigma'[(\#_{k'} \mapsto v_{k'})_{k' \le k}], s, \#_m) && \text{using (A.59)} \\
&\quad = \mathsf{Infl}\,\big(P_{l_j} \bullet P_{ch}, \sigma'[(\#_{k'} \mapsto v_{k'})_{k' \le k}], s, \#_m\big) && \text{using } \#_m \notin \mathbf{Var} \\
&\quad \subseteq \mathsf{Infl}\,\big(P_{l_j}, \sigma'', s, u'\big) && \text{using (A.60)} \\
&\quad = \mathsf{Infl}\,(P_\bullet, \sigma'', s, u') && \text{using (A.50)}
\end{aligned}$$

and the same goes for the readers

$$\begin{aligned}
&\mathsf{Read}\,(P_\bullet, \sigma, s, u) \\
&\quad = \mathsf{Read}\,(P_{l_i}, \sigma, s, u) && \text{using (A.51)} \\
&\quad \supseteq \mathsf{Read}\,(P_{l_i} \bullet P_{ch}, \sigma'[(\#_{k'} \mapsto v_{k'})_{k' \le k}], s, \#_m) && \text{using (A.59)} \\
&\quad = \mathsf{Read}\,\big(P_{l_j} \bullet P_{ch}, \sigma'[(\#_{k'} \mapsto v_{k'})_{k' \le k}], s, \#_m\big) && \text{using } \#_m \notin \mathbf{Var} \\
&\quad \supseteq \mathsf{Read}\,\big(P_{l_j}, \sigma'', s, u'\big) && \text{using (A.60)} \\
&\quad = \mathsf{Read}\,(P_\bullet, \sigma'', s, u') && \text{using (A.51)}
\end{aligned}$$

Furthermore consider the rest of the flows $(u, s, u') \in F$, which original is from the flows $(u, s, u') \in E_j$ (from the other part of the combiner function (2.3)), we have

$$\begin{aligned}
&\mathsf{Infl}\,(P_\bullet, \sigma, s, u) \\
&\quad = \mathsf{Infl}\,\big(P_{l_j}, \sigma, s, u\big) && \text{using (A.50)} \\
&\quad = \mathsf{Infl}\,\big(P_{l_j}, \sigma', s, u\big) && \text{using } u \notin \mathbf{Var}_{l_i} \\
&\quad = \mathsf{Infl}\,\big(P_{l_j} \bullet P_{ch}, \sigma'[(\#_{k'} \mapsto v_{k'})_{k' \leq k}], s, u\big) && \text{using } \#_m \notin \mathbf{Var},\ u \in \mathbf{Var}_{l_j} \\
&\quad \subseteq \mathsf{Infl}\,\big(P_{l_j}, \sigma'', s, u'\big) && \text{using (A.60)} \\
&\quad = \mathsf{Infl}\,(P_\bullet, \sigma'', s, u') && \text{using (A.50)}
\end{aligned}$$

and likewise for readers

$$\begin{aligned}
&\mathsf{Read}\,(P_\bullet, \sigma, s, u) \\
&\quad = \mathsf{Read}\,\big(P_{l_j}, \sigma, s, u\big) && \text{using (A.51)} \\
&\quad = \mathsf{Read}\,\big(P_{l_j}, \sigma', s, u\big) && \text{using } u \notin \mathbf{Var}_{l_i} \\
&\quad = \mathsf{Read}\,\big(P_{l_j} \bullet P_{ch}, \sigma'[(\#_{k'} \mapsto v_{k'})_{k' \leq k}], s, u\big) && \text{using } \#_m \notin \mathbf{Var},\ u \in \mathbf{Var}_{l_j} \\
&\quad \supseteq \mathsf{Read}\,\big(P_{l_j}, \sigma'', s, u'\big) && \text{using (A.60)} \\
&\quad = \mathsf{Read}\,(P_\bullet, \sigma'', s, u') && \text{using (A.51)}
\end{aligned}$$

For the other part of the security predicate take now all s and $u \in \mathbf{Var} \setminus D$. There are now two cases, take first where $u \in \mathbf{Var}_{l_j}$, which is similar as before

$$\begin{aligned}
&\mathsf{Infl}\,(P_\bullet, \sigma, s, u) \\
&\quad = \mathsf{Infl}\,\big(P_{l_j}, \sigma, s, u\big) && \text{using (A.50)} \\
&\quad = \mathsf{Infl}\,\big(P_{l_j}, \sigma', s, u\big) && \text{using } u \notin \mathbf{Var}_{l_i} \\
&\quad = \mathsf{Infl}\,\big(P_{l_j} \bullet P_{ch}, \sigma'[(\#_{k'} \mapsto v_{k'})_{k' \leq k}], s, u\big) && \text{using } \#_m \notin \mathbf{Var},\ u \in \mathbf{Var}_{l_j} \\
&\quad \subseteq \mathsf{Infl}\,\big(P_{l_j}, \sigma'', s, u\big) && \text{using (A.60)} \\
&\quad = \mathsf{Infl}\,(P_\bullet, \sigma'', s, u) && \text{using (A.50)}
\end{aligned}$$

and for the readers

$$\begin{aligned}
&\mathsf{Read}\,(P_\bullet, \sigma, s, u) \\
&\quad = \mathsf{Read}\,\big(P_{l_j}, \sigma, s, u\big) && \text{using (A.51)} \\
&\quad = \mathsf{Read}\,\big(P_{l_j}, \sigma', s, u\big) && \text{using } u \notin \mathbf{Var}_{l_i} \\
&\quad = \mathsf{Read}\,\big(P_{l_j} \bullet P_{ch}, \sigma'[(\#_{k'} \mapsto v_{k'})_{k' \leq k}], s, u\big) && \text{using } \#_m \notin \mathbf{Var},\ u \in \mathbf{Var}_{l_j} \\
&\quad \supseteq \mathsf{Read}\,\big(P_{l_j}, \sigma'', s, u\big) && \text{using (A.60)} \\
&\quad = \mathsf{Read}\,(P_\bullet, \sigma'', s, u) && \text{using (A.51)}
\end{aligned}$$

The last case is where $u \in \mathbf{Var}_\ell$ and $\ell \neq l_j$ and gives

$$\begin{aligned}
&\mathsf{Infl}\,(P_\bullet, \sigma, s, u) && \\
&\qquad = \mathsf{Infl}\,(P_\ell, \sigma, s, u) && \text{using (A.50)} \\
&\qquad = \mathsf{Infl}\,(P_\ell, \sigma'', s, u) && \text{using } u \in \mathbf{Var}_\ell \\
&\qquad = \mathsf{Infl}\,(P_\bullet, \sigma'', s, u) && \text{using (A.50)} \\
&\mathsf{Read}\,(P_\bullet, \sigma, s, u) && \\
&\qquad = \mathsf{Read}\,(P_\ell, \sigma, s, u) && \text{using (A.51)} \\
&\qquad = \mathsf{Read}\,(P_\ell, \sigma'', s, u) && \text{using } u \in \mathbf{Var}_\ell \\
&\qquad = \mathsf{Read}\,(P_\bullet, \sigma'', s, u) && \text{using (A.51)}
\end{aligned}$$

which all finalises the proof.

A.10 Fact 4.1

By looking at the resulting form of the different functions the result can be obtained. Using $\mathsf{dist}(P)$ gives the form P^β

$$P^\beta ::= \{\} \quad | \quad \{u : o \leftarrow s\} \quad | \quad \{u : o \rightarrow \bar{s}\} \quad | \quad \left(\varphi \Rightarrow P^\beta\right) \quad | \quad P_1^\beta \bullet P_2^\beta$$

Using $\mathsf{cln}(P)$ gives the form P^γ

$$\begin{aligned}
P^\gamma &::= \{\} \quad | \quad P^\delta \\
P^\delta &::= \{\bar{u} : \bar{o} \leftarrow \bar{s}\} \quad | \quad \{\bar{u} : \bar{o} \rightarrow \bar{s}\} \quad | \quad \left(\varphi \Rightarrow P^\delta\right) \quad | \quad P^\delta \bullet P^\delta
\end{aligned}$$

Using $\mathsf{cnd}(P)$ gives the form P^ζ

$$P^\zeta ::= \{\} \quad | \quad (\varphi \Rightarrow \{\bar{u} : \bar{o} \leftarrow \bar{s}\}) \quad | \quad (\varphi \Rightarrow \{\bar{u} : \bar{o} \rightarrow \bar{s}\}) \quad | \quad P_1^\zeta \bullet P_2^\zeta$$

Now consider $P^\beta = \mathsf{dist}(P)$. Using $\mathsf{cln}(P^\beta)$ gives the form $P^{\gamma'}$

$$\begin{aligned}
P^{\gamma'} &::= \{\} \quad | \quad P^{\delta'} \\
P^{\delta'} &::= \{u : o \leftarrow s\} \quad | \quad \{u : o \rightarrow \bar{s}\} \quad | \quad \left(\varphi \Rightarrow P^{\delta'}\right) \quad | \quad P^{\delta'} \bullet P^{\delta'}
\end{aligned}$$

Using $\mathsf{cnd}(P^{\gamma'})$ gives the form $P^{\gamma''}$

$$\begin{aligned}
P^{\gamma''} &::= \{\} \quad | \quad P^{\zeta'} \\
P^{\zeta''} &::= (\varphi \Rightarrow \{u : o \leftarrow s\}) \quad | \quad (\varphi \Rightarrow \{u : o \rightarrow \bar{s}\}) \quad | \quad P_1^{\zeta''} \bullet P_2^{\zeta''}
\end{aligned}$$

It is now seen that $P^{\zeta''}$ is syntactic equivalent to INF.

All the modifications done by $\mathsf{dist}(P), \mathsf{cln}(P), \mathsf{cnd}(P)$ is accounted for in the their respective definitions in Algorithm 4.1 to 4.3. We can therefore state that $P \equiv \mathsf{inf}(P)$, finalising the proof.

A.11 Fact 4.2

For $B = (\varphi \Rightarrow \{u : o \leftarrow s\})$, the influencers and readers are

$$\begin{aligned}
&\mathsf{Infl}\,(B,\sigma,q,y) \subseteq \mathsf{Infl}\,(B_1 \bullet \cdots \bullet B_n \bullet B_{n+1} \bullet \cdots \bullet B_m,\sigma,q,y)\\
&\quad\Leftrightarrow \mathsf{Infl}\,(B,\sigma,q,y) \subseteq \mathsf{Infl}\,(B_1,\sigma,q,y) \cup \cdots \cup \mathsf{Infl}\,(B_m,\sigma,q,y)\\
&\quad\Leftrightarrow \mathsf{Infl}\,(B,\sigma,q,y) \subseteq \mathsf{Infl}\,(B_1,\sigma,q,y) \cup \cdots \cup \mathsf{Infl}\,(B_n,\sigma,q,y) \cup \epsilon \cup \cdots \cup \epsilon\\
&\quad\Leftrightarrow \mathsf{Infl}\,(B,\sigma,q,y) \subseteq \mathsf{Infl}\,(B_1,\sigma,q,y) \cup \cdots \cup \mathsf{Infl}\,(B_n,\sigma,q,y)\\
&\mathsf{Read}\,(B,\sigma,q,y) \supseteq \mathsf{Read}\,(B_1 \bullet \cdots \bullet B_n \bullet B_{n+1} \bullet \cdots \bullet B_m,\sigma,q,y)\\
&\quad\Leftrightarrow \mathbf{Pr} \supseteq \mathsf{Read}\,(B_1 \bullet \cdots \bullet B_n \bullet B_{n+1} \bullet \cdots \bullet B_m,\sigma,q,y)\\
&\quad\Leftrightarrow \mathsf{true}\\
&\quad\Leftrightarrow \mathbf{Pr} \supseteq \mathsf{Read}\,(B_1 \bullet \cdots \bullet B_n,\sigma,q,y)\\
&\quad\Leftrightarrow \mathsf{Read}\,(B,\sigma,q,y) \supseteq \mathsf{Read}\,(B_1 \bullet \cdots \bullet B_n,\sigma,q,y)
\end{aligned}$$

Similar for $B = (\varphi \Rightarrow \{u : o \rightarrow \bar{s}\})$ the influencers and readers are

$$\begin{aligned}
&\mathsf{Infl}\,(B,\sigma,q,y) \subseteq \mathsf{Infl}\,(B_1 \bullet \cdots \bullet B_n \bullet B_{n+1} \bullet \cdots \bullet B_m,\sigma,q,y)\\
&\quad\Leftrightarrow \epsilon \subseteq \mathsf{Infl}\,(B_1 \bullet \cdots \bullet B_n \bullet B_{n+1} \bullet \cdots \bullet B_m,\sigma,q,y)\\
&\quad\Leftrightarrow \mathsf{true}\\
&\quad\Leftrightarrow \epsilon \subseteq \mathsf{Infl}\,(B_{n+1} \bullet \cdots \bullet B_m,\sigma,q,y)\\
&\quad\Leftrightarrow \mathsf{Infl}\,(B,\sigma,q,y) \subseteq \mathsf{Infl}\,(B_{n+1} \bullet \cdots \bullet B_m,\sigma,q,y)\\
&\mathsf{Read}\,(B,\sigma,q,y) \supseteq \mathsf{Read}\,(B_1 \bullet \cdots \bullet B_n \bullet B_{n+1} \bullet \cdots \bullet B_m,\sigma,q,y)\\
&\quad\Leftrightarrow \mathsf{Read}\,(B,\sigma,q,y) \supseteq \mathsf{Read}\,(B_1,\sigma,q,y) \cap \cdots \cap \mathsf{Read}\,(B_m,\sigma,q,y)\\
&\quad\Leftrightarrow \mathsf{Read}\,(B,\sigma,q,y) \supseteq \mathbf{Pr} \cap \cdots \cap \mathbf{Pr} \cap \mathsf{Read}\,(B_{n+1},\sigma,q,y) \cap \cdots \cap \mathsf{Read}\,(B_m,\sigma,q,y)\\
&\quad\Leftrightarrow \mathsf{Read}\,(B,\sigma,q,y) \supseteq \mathsf{Read}\,(B_{n+1},\sigma,q,y) \cap \cdots \cap \mathsf{Read}\,(B_m,\sigma,q,y)
\end{aligned}$$

A.12 Fact 4.3

Let $B^I = (\varphi \Rightarrow \{u : o \leftarrow s\})$. The influencers for $B^I \sqsubseteq P_1 \bullet P_2$ for all σ, q and y is then

$$\begin{aligned}
&\mathsf{Infl}\left(B^I,\sigma,q,y\right) \subseteq \mathsf{Infl}\,(P_1 \bullet P_2,\sigma,q,y)\\
&\quad\Leftrightarrow \mathsf{Infl}\left(B^I,\sigma,q,y\right) \subseteq \mathsf{Infl}\,(P_1,\sigma,q,y) \cup \mathsf{Infl}\,(P_2,\sigma,q,y)\\
&\quad\Leftrightarrow \mathsf{Infl}\left(B^I,\sigma,q,y\right) \subseteq \mathsf{Infl}\,(P_1,\sigma,q,y) \vee \mathsf{Infl}\left(B^I,\sigma,q,y\right) \subseteq \mathsf{Infl}\,(P_2,\sigma,q,y)
\end{aligned}$$

and likewise for the readers

$$\begin{aligned}
&\mathsf{Read}\left(B^I,\sigma,q,y\right) \supseteq \mathsf{Read}\,(P_1 \bullet P_2,\sigma,q,y)\\
&\quad\Leftrightarrow \mathbf{Pr} \supseteq \mathsf{Read}\,(P_1 \bullet P_2,\sigma,q,y)\\
&\quad\Leftrightarrow \mathsf{true}\\
&\quad\Leftrightarrow \mathbf{Pr} \supseteq \mathsf{Read}\,(P_1,\sigma,q,y) \vee \mathbf{Pr} \supseteq \mathsf{Read}\,(P_2,\sigma,q,y)\\
&\quad\Leftrightarrow \mathsf{Read}\left(B^I,\sigma,q,y\right) \supseteq \mathsf{Read}\,(P_1,\sigma,q,y) \vee \mathsf{Read}\left(B^I,\sigma,q,y\right) \supseteq \mathsf{Read}\,(P_2,\sigma,q,y)
\end{aligned}$$

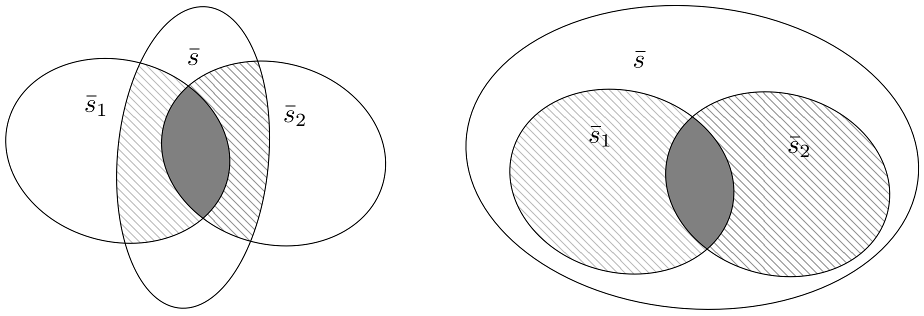

Figure A.1: In the left Venn diagram $\bar{s} \supseteq \bar{s}_1 \cap \bar{s}_2$ but $\bar{s} \not\supseteq \bar{s}_i$ for $i \in 1, 2$. In the right Venn diagram $\bar{s} \supseteq \bar{s}_i$ for $i \in 1, 2$ which makes $\bar{s} \supseteq \bar{s}_1 \cap \bar{s}_2$.

Which is the same as finding the influencers and readers for $B^I \sqsubseteq P_1 \vee B^I \sqsubseteq P_2$.

A.13 Fact 4.4 and 4.5

Proof by counterexample. Let $B^R = (\varphi \Rightarrow \{u : o \to \bar{s}\})$. For readers it does not hold, here shown by counterexample. Let $P_i = (\varphi \Rightarrow \{u : o \to \bar{s}_i\})$ for $i \in 1, 2$, where $\bar{s} \neq \bar{s}_i$ but not necessarily disjoint (e.g. $\bar{s} = \{s, s_3\}$ and $\bar{s}_i = \{s_i, s_3\}$).

$$\begin{aligned}
&\mathsf{Read}\left(B^R, \sigma, q, y\right) \supseteq \mathsf{Read}\left(P_1 \bullet P_2, \sigma, q, y\right) \\
&\quad\Leftrightarrow \mathsf{Read}\left(B^R, \sigma, q, y\right) \supseteq \mathsf{Read}\left(P_1, \sigma, q, y\right) \cap \mathsf{Read}\left(P_2, \sigma, q, y\right) \\
&\quad\Leftrightarrow \begin{cases} \bar{s} \supseteq \bar{s}_1 \cap \bar{s}_2 & \text{if } \sigma \models \varphi \wedge o = q \wedge u = y \\ \mathbf{Pr} \supseteq \mathbf{Pr} \cap \mathbf{Pr} & \text{otherwise} \end{cases} \\
&\quad\Leftarrow \begin{cases} \bar{s} \supseteq \bar{s}_1 \wedge \bar{s} \supseteq \bar{s}_2 & \text{if } \sigma \models \varphi \wedge o = q \wedge u = y \\ \mathbf{Pr} \supseteq \mathbf{Pr} \wedge \mathbf{Pr} \supseteq \mathbf{Pr} & \text{otherwise} \end{cases}
\end{aligned}$$

In the last equation "$\Rightarrow$" does not hold, also illustrated in Figure A.1. If a disjunction was used in the false statement, a similar proof would show the same e.g. $\bar{s} \supseteq \bar{s}_1 \cap \bar{s}_2 \not\Rightarrow \bar{s} \supseteq \bar{s}_1 \vee \bar{s} \supseteq \bar{s}_2$, but "$\Leftarrow$" would hold.

A.14 Fact 4.7

The influencer and reader set is

$$\begin{aligned}
&\mathsf{Infl}\left(\left(\varphi \Rightarrow \{u : o \leftarrow s\}\right), \sigma, q, y\right) \subseteq \mathsf{Infl}\left(\left(\varphi' \Rightarrow \{u' : o' \rightarrow \bar{s}'\}\right), \sigma, q, y\right)\\
&\qquad = \mathsf{Infl}\left(\left(\varphi \Rightarrow \{u : o \leftarrow s\}\right), \sigma, q, y\right) \subseteq \epsilon\\
&\mathsf{Read}\left(\left(\varphi \Rightarrow \{u : o \leftarrow s\}\right), \sigma, q, y\right) \supseteq \mathsf{Read}\left(\left(\varphi' \Rightarrow \{u' : o' \rightarrow \bar{s}'\}\right), \sigma, q, y\right)\\
&\qquad = \mathbf{Pr} \supseteq \mathsf{Read}\left(\left(\varphi' \Rightarrow \{u' : o' \rightarrow \bar{s}'\}\right), \sigma, q, y\right)\\
&\qquad = \mathsf{true}
\end{aligned}$$

If φ is satisfiable then there exists a σ, q, y such that $\sigma \models \varphi$, $q = o$ and $y = u$ such that the influencers becomes s, thereby making $\{s\} \subseteq \epsilon$ false. If φ is unsatisfiable then the case becomes true.

A.15 Fact 4.8

The influencer and reader set is

$$\begin{aligned}
&\mathsf{Infl}\left(\left(\varphi \Rightarrow \{u : o \rightarrow \bar{s}\}\right), \sigma, q, y\right) \subseteq \mathsf{Infl}\left(\left(\varphi' \Rightarrow \{u' : o' \leftarrow s'\}\right), \sigma, q, y\right)\\
&\qquad = \epsilon \subseteq \mathsf{Infl}\left(\left(\varphi' \Rightarrow \{u' : o' \leftarrow s'\}\right), \sigma, q, y\right)\\
&\qquad = \mathsf{true}\\
&\mathsf{Read}\left(\left(\varphi \Rightarrow \{u : o \rightarrow \bar{s}\}\right), \sigma, q, y\right) \supseteq \mathsf{Read}\left(\left(\varphi' \Rightarrow \{u' : o' \leftarrow s'\}\right), \sigma, q, y\right)\\
&\qquad = \mathsf{Read}\left(\left(\varphi \Rightarrow \{u : o \rightarrow \bar{s}\}\right), \sigma, q, y\right) \supseteq \mathbf{Pr}
\end{aligned}$$

If φ is satisfiable and $\bar{s} \subsetneq \mathbf{Pr}$ (e.g. $\bar{s} \neq \star$) then there exists a σ, q, y such that $\sigma \models \varphi$, $q = o$ and $y = u$ such that the readers becomes $\bar{s}$, thereby making $\bar{s} \supseteq \mathbf{Pr} = \mathsf{false}$. If φ is unsatisfiable or $\bar{s} = \star = \mathbf{Pr}$ then the whole case becomes true.

A.16 Theorem 4.1

Assume $\mathsf{chk}(\epsilon, l_i, \phi_i, S_i) = (\phi_i', c_i)$ for all $i \leq n$ and that c_i is a tautology for all $i \leq n$. Then $(c_1 \wedge \ldots \wedge c_n)$ will be a tautology too. We then have have $\epsilon \vdash_{l_i} \{\phi_i\}S\{\phi_i'\}$ for all $i \leq n$ from Lemma 4.1. Using the rule of consequence, for all $i \leq n$ we have

$$\frac{\epsilon \vdash_{l_i} \{\phi_i\}S\{\phi_i'\}}{\epsilon \vdash_{l_i} \{\phi_i\}S\{\mathsf{true}\}}\ \phi_i' \Rightarrow \mathsf{true}$$

Because $\phi_i' \Rightarrow \mathsf{true}$ is always true, the premises for $[sys_{ts}]$ are satisfied and we have that $\vdash \{\phi_1 \& \cdots \& \phi_n\}\ l_1 : S_1 \parallel \cdots \parallel l_n : S_n$ as required.

APPENDIX B
Type checker manual

This appendix describes a brief overview of the code and how to use the developed type checker to verify a system. The actual code should be available with this thesis.

B.1 Requirements

The following software is required to run and get a result from the type checker:

- Standard ML of New Jersey (SML/NJ) compiler
- Z3 solver (or for small examples use `http://rise4fun.com/z3`)
- SML Testing library `https://github.com/kvalle/sml-testing.git` (optional)

B.2 Overview

`example_`X`.sml` Some small examples (unit test for the type checker) and some larger examples. Most generates a file `example_`X`.z3` with the resulting Z3 predicate.

`inf.sml` Implication normal form $\mathsf{inf}(P)$ together with its auxiliary functions $\mathsf{cnd}(P)$, $\mathsf{cln}(P)$, $\mathsf{dist}(P)$.

`preorder.sml` Preorder function $\mathsf{order}(P, P')$.

`Set.sml` Helper functions simulating sets using lists.

`substitutions.sml` Substitutions used on policies P and conditions ϕ such as $P[a/z]$ and $P\langle\bar{y}/z\rangle$.

`syntax.sml` Datatypes for S, a, b, P, ϕ, a^+, and c, including converter functions e.g. from ϕ to c and c to Z3 notation. Furthermore, it contains functions for pretty printing all the data types and getting free variables for different data types.

`tests.sml` Unit tests, for printing, substitutions, preorder and type checker. The SML Testing library should be loaded into `sml-testing/`.

`typechecker.sml` The functions for the type checker $\mathsf{chk}(\bar{y}, \ell, \phi, S)$ and for a fully defined system $\mathsf{verify}(Sys, \mathcal{S}, \mathsf{NSA}, P_{Sys}, \Phi)$.

The type checker does not contain any sanity check in terms of variable separation and if policies are truly localised. It is therefore assumed that the programmer consider these when writing examples.

B.3 Usage

Let $(Sys, \mathcal{S}, \mathsf{NSA}, P_{Sys})$ be defined as

$$Sys = l_1 : x := 4; y := x \parallel l_2 : w := 4; z := w$$
$$P_{l_1} = \{x, y : s \leftarrow s\}$$
$$P_{l_2} = \{w : s \leftarrow s\} \bullet (w > 5 \Rightarrow \{z : s \leftarrow s\})$$

where $\mathcal{S}(l_i) = s$ for $i \in 1, 2$ and $P_{Sys} = P_{l_1} \bullet P_{l_2}$. Let $\Phi(l_i) = \mathsf{true}$ for $i \in 1, 2$.

The result of $\mathsf{verify}(Sys, \mathcal{S}, \mathsf{NSA}, P_{Sys}, \Phi)$ can be found by the following code

```
use "typechecker.sml";
val Sys = [
    ("l1", Seq(Assign("x", Int(4)), Assign("y", Var("x")))),
    ("l2", Seq(Assign("w", Int(4)), Assign("z", Var("w"))))
];
val SecurityDomains = [
    (["l1", "l2"], "s")
];
val NSA = "NSA";
val P_Sys = [
    ("l1",  Pol(["x", "y"], ["s"], ["s"])),
    ("l2",  SeqPol( Pol(["w"], ["s"], ["s"]),
                    CondPol(phiGt(a_Var "w", a_Int 5),
                        Pol(["z"], ["s"], ["s"]
    ))))
];
val initPhi = [
    ("l1", phiTrue),
    ("l2", phiTrue)
];

val result = verify(Sys,
    SecurityDomains,
    NSA,
    P_Sys,
    initPhi
);

map print (map toZ3tautology result);
(*  Result
    l1: unsat
    l2: sat
    ---------
    Sys: sat
*)
```

The output is Z3 code for verifying if each of the processes is satisfiable or not. l_1 is unsatisfiable thereby meaning this process is secure. l_2 is satisfiable and is not secure. The overall result for $\mathsf{verify}(Sys, \mathcal{S}, \mathsf{NSA}, P_{Sys}, \Phi)$ is therefore not secure.

APPENDIX C

Thesis formalities

This appendix contains the original project plan, the final project plan, and a brief auto-evaluation of the project process as required in the program specifications from DTU for writing a master thesis (`http://sdb.dtu.dk`).

C.1 Original project plan

The *aviation electronic* (avionics) industry has developed safety critical and reliable hardware and software for many years. In classical avionics each function was put in separate avionics controller following the "federated architecture", to keep a high independent reliability. In a combination of more data communication and growth in signal interfaces the weight, volume and power consumption of all the different controllers hit the boundaries of the airplane in the mid 1990s. Meanwhile to keep maintaining all the controllers, the catalog for spare parts for the controllers kept increasing, and thereby cost more in the ongoing maintenance [But07].

The avionics industry therefore developed *Integrated Modular Avionics* (IMA), where software with different security domains is integrated on commercial off-the-shelf components. This indeed lowered the weight, volume, power consumption, and hardware maintenance cost. The idea of IMA is to have an operating system which can handle multiple applications separately (a separation kernel), and if one application is altered or a new applications is added, this should have minimal or no effect on the other applications. With IMA the cost for the avionics modules was therefore moved from the separate hardware components to the development and certification of the separation kernel and insurance that the separate applications is kept separate [But07].

The *Multiple Independent Levels of Security* (MILS) approach gives a guideline on how to design, construct, integrate and evaluate secure systems. This approach decomposes components and locates the vulnerable parts, and suggest strict separation and information flow control [Rus08]. The MILS approach is therefore guidelines on how to achieve the safe and secure separation mechanisms from the IMA requirements.

Aeronautical Radio, Incorporated (ARINC) is a major company producing communication solutions to the aviation industry, among others. They have a huge number of standards, and within one is the ARINC report 811 [A81105], which provides security concepts for airborne networks. This report tells the story again with the decomposition of a system into several domains. Here they provide an actual example as shown in Figure C.1 with four different domains, *Aircraft Control Domain*,

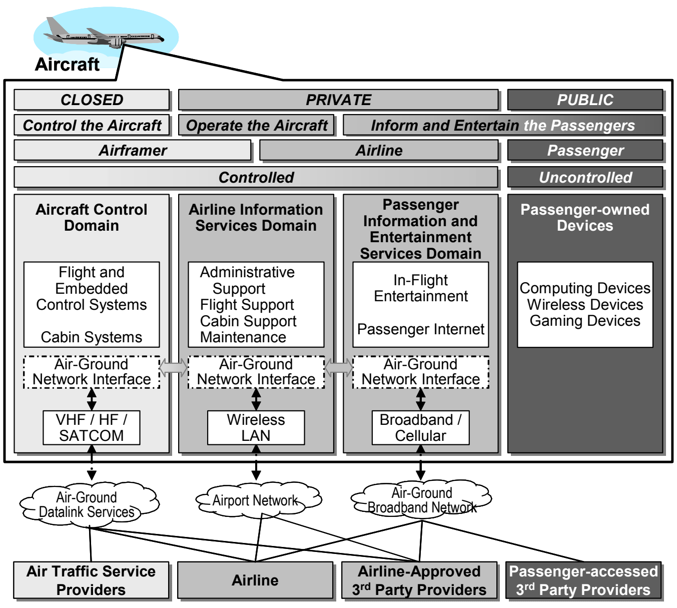
Aircraft
CLOSED
PRIVATE
PUBLIC
Control the Aircraft
Operate the Aircraft
Inform and Entertain the Passengers
Airframer
Airline
Passenger
Controlled
Uncontrolled
Aircraft Control Domain
Airline Information Services Domain
Passenger Information and Entertainment Services Domain
Passenger-owned Devices
Flight and Embedded Control Systems
Cabin Systems
Administrative Support
Flight Support
Cabin Support
Maintenance
In-Flight Entertainment
Passenger Internet
Computing Devices
Wireless Devices
Gaming Devices
Air-Ground Network Interface
Air-Ground Network Interface
Air-Ground Network Interface
VHF / HF / SATCOM
Wireless LAN
Broadband / Cellular
Air-Ground Datalink Services
Airport Network
Air-Ground Broadband Network
Air Traffic Service Providers
Airline
Airline-Approved 3rd Party Providers
Passenger-accessed 3rd Party Providers

Figure C.1: Security domains according to ARINC report 811 [A81105].

Airline Information Service Domain, *Passenger Information and Entertainment Service*, and *Passenger-owned Devices*. These domains is only a minimal subset, where e.g. the Aircraft Control Domain could be split up into several domains, some for navigation and others for ground communication. The ARINC report 811 also provides a “aircraft security process framework”, which more or less is the *Common Criteria* (CC) [CC12a; CC12b; CC12c] in avionics terminology.

There has been put a lot of effort into develop and certify a separation kernel, one main example is the PikeOS by SYSGO, which is used in the avionics industry, and fulfils and implements IMA and the MILS standard [SYSGO]. The applications from the previous separation is then stacked as virtual machine on top of a separation kernel. All these application can be part of different security domains, and the separation kernel then ensures that the information flow is correct across these domains at run time. The separation kernel therefore specifies some common ground for defining policies, and either block or allow data flows. Depended on the complexity of the system all these checks and lookup in policies could take up much time, and can therefore become a bottleneck in a real time setting.

[Mül+12] presents and discuss the software architecture of an abstraction for a security gateway, which looks at the flow from two different domains. This security gateway is further developed in [NNL15], where the *Decentralised Label Model* (DLM) [ML97; ML00] is used. When using this model, static analysis can be used on the software, to ensure that confidentiality and integrity is preserved in the information flow. The run-time checks can therefore be minimised.

Internet of Things is a vision that all embedded electronics, sensors, and software will be connected and exchange information, to achieve better products with interconnected services across multiple domains, machines, etc. With the strict separation and time consuming security checks the aviation industry will not be ready for this future vision.

Previously the codebase has been developed as separate instances, and the same code for some common procedures is therefore copied across all applications (e.g. this could be a sorting procedure). This is not a modern way of programming and the overall codebase will strictly increase if the same procedures is used across many applications with different security restrictions. Having a label model like DLM ensuring secure information flow, functions can have polymorphic labels which is relative to the context of the caller, and thereby remove duplicated code.

C.1.1 Contributions

Due to the duality between integrity and confidentiality in DLM the policies in [NNL15] can be simplified to only contain integrity. The policies in [NNL15] also contains a value range for variables which gives the content-based information flow. This project will take the underlying basics of [NNL15] and remove the confidentiality part. Furthermore, the article is not considering the programmer, and this project will therefore specify the syntax for policies and how they will interact.

Constructs such as “acts for” and “endorse” from DLM not shown in [NNL15] will also be introduced to see if this gives a more expressiveness language. Furthermore the language in [NNL15] is only a limited set compared to C which is commonly used in avionics. This project will therefore look at the complexity issues of adding arrays and still have to deal with distinct policies and preserve confidentiality and integrity.

Lastly the analysis should be formally proven, to ensure that it actually provides a secure information flow. This could be further emphasised with a prototype, showing some not-so-trivial examples.

C.1.2 Bibliography

[A81105] Airlines Electronic Engineering Committee. *ARINC report 811: Commercial Aircraft Information Security Concepts of Operation and Process Framework*. Aeronautical Radio, INC., 2005 (cited on pages 113, 114).

[But07] Henning Butz. “The Airbus Approach to Open Integrated Modular Avionics (IMA): Technology, Methods, Processes and Future Road Map”. In: *Workshop on Aircraft System Technologies* (2007) (cited on page 113).

[CC12a] Common Criteria. *Common Criteria for Information Technology Security Evaluation Part 1: Introduction and general model*. 2012 (cited on page 114).

[CC12b] Common Criteria. *Common Criteria for Information Technology Security Evaluation Part 2: Security functional components*. 2012 (cited on page 114).

[CC12c] Common Criteria. *Common Criteria for Information Technology Security Evaluation Part 3: Security assurance components.* 2012 (cited on page 114).

[ML00] Andrew C. Myers and Barbara Liskov. "Protecting privacy using the decentralized label model". In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 9.4 (2000), pages 410–442 (cited on page 114).

[ML97] Andrew C. Myers and Barbara Liskov. "A Decentralized Model for Information Flow Control". In: *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP)* (1997) (cited on page 114).

[Mül+12] Kevin Müller et al. "MILS-related information flow control in the avionic domain: A view on security-enhancing software architectures". In: *Proceedings of the International Conference on Dependable Systems and Networks* (2012) (cited on page 114).

[NNL15] Hanne R. Nielson, Flemming Nielson, and Ximeng Li. "Disjunctive Information Flow". In: *DTU Compute* (2015) (cited on pages 114, 115).

[Rus08] John Rushby. "Separation and Integration in MILS (The MILS Constitution)". In: February (2008) (cited on page 113).

[SYSGO] SYSGO. *SYSGO – Embedding innocations.* URL: `http://www.sysgo.com/` (visited on January 28, 2015) (cited on page 114).

C.2 Final project plan

The final project plan is a revision of the original project plan, which is rewritten and extended into Chapter 1.

C.3 Auto-evaluation

This project started with a literature study about the aviation industry and DLM. From this most of terminology was settled in the rest of the project, and most of the case study was settled.

From previous work done in [NNL15] it was clear that DLM had some shortcomings and needed to be extended with the conditional policies. Furthermore, I unsuccessful tried to map DLM work [ML00] to the new work in [NNL15]. The work done for this part was therefore discarded.

After the first third of the project period the draft paper [NNd] was presented to me, and the project took a turnover. The idea was now to verify the many of the facts, lemmas and proofs in [NNd] and to implement the type system also presented there.

The goal for the project was clear and a more regular day began. Thus, I used a much time understanding the type system and its correctness result. Furthermore, I wrote the case about the electronic flight bag with help from PhD student Ximeng Li from DTU Compute. This was good for the project to have another semi realistic case.

Along the way I took many notes, which have been really helpful in the end phase where all the smaller parts are merged together to form this thesis.

The overall project process has therefore been a mix of experimental and analytic approach.

Bibliography

[A811105] Airlines Electronic Engineering Committee. *ARINC report 811: Commercial Aircraft Information Security Concepts of Operation and Process Framework*. Aeronautical Radio, INC., 2005 (cited on pages 2, 3).

[Asta] Astrée. *Astrée Run-Time Error Analyzer*. URL: `http://www.absint.com/astree/index.htm` (visited on January 27, 2015) (cited on page 6).

[Astb] Astrée Static Analyzer. *The Astrée Static Analyzer*. URL: `http://www.astree.ens.fr/` (visited on January 27, 2015) (cited on page 6).

[BH07] Lennart Beringer and Martin Hofmann. "Secure information flow and program logics". In: *Proceedings – IEEE Computer Security Foundations Symposium* (2007), pages 233–245 (cited on page 39).

[Bou09] Ǵerard Boudol. *Secure information flow as a safety property, In: Formal Aspects in Security and Trust*. Edited by Pierpaolo Degano, Joshua Guttman, and Fabio Martinelli. Volume 5491. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pages 20–34 (cited on page 39).

[But07] Henning Butz. "The Airbus Approach to Open Integrated Modular Avionics (IMA): Technology, Methods, Processes and Future Road Map". In: *Workshop on Aircraft System Technologies* (2007) (cited on pages 1, 2).

[CC12a] Common Criteria. *Common Criteria for Information Technology Security Evaluation Part 1: Introduction and general model*. 2012 (cited on page 2).

[CC12b] Common Criteria. *Common Criteria for Information Technology Security Evaluation Part 2: Security functional components*. 2012 (cited on page 2).

[CC12c] Common Criteria. *Common Criteria for Information Technology Security Evaluation Part 3: Security assurance components*. 2012 (cited on page 2).

[DD12] Anthony Dessiatnikoff and Yves Deswarte. "Potential Attacks on Onboard Aerospace Systems". In: August (2012), pages 71–74 (cited on pages 1, 6).

[DD77] Dorothy E. Denning and Peter J. Denning. "Certification of programs for secure information flow". In: *Communications of the ACM* 20 (1977), pages 504–513 (cited on page 9).

[DFP98] Rocco De Nicola, Gian Luigi Ferrari, and Rosario Pugliese. “Klaim: A kernel language for agents interaction and mobility”. In: *IEEE Transactions on Software Engineering* 24.5 (1998), pages 315–330 (cited on page 66).

[GAO15] United States Government Accountability Office. “Air Traffic Control: FAA Needs a More Comprehensive Approach to Address Cybersecurity As Agency Transitions to NextGen”. In: April (2015) (cited on page 1).

[HNN09] Chris Hankin, Flemming Nielson, and Hanne R. Nielson. “Advice from belnap policies”. In: *Proceedings - IEEE Computer Security Foundations Symposium* (2009), pages 234–247 (cited on page 67).

[Hur15] Mark S. Hurley. “Search warrant for Chris Robberts’ electronics”. In: (2015), pages 1–22. URL: `http://aptn.ca/news/wp-content/uploads/sites/4/2015/05/warrant-for-Roberts-electronics.pdf` (cited on page 1).

[Jif] Jif software. *Jif: Java + information flow.* URL: `http://www.cs.cornell.edu/jif/` (visited on July 7, 2015) (cited on page 68).

[Mil+97] Robin Milner et al. “The Definition of Standard ML, revised edition”. In: *MIT Press* 1.2 (1997), pages 2–3 (cited on page 54).

[ML00] Andrew C. Myers and Barbara Liskov. “Protecting privacy using the decentralized label model”. In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 9.4 (2000), pages 410–442 (cited on pages 3, 4, 68, 116).

[ML97] Andrew C. Myers and Barbara Liskov. “A Decentralized Model for Information Flow Control”. In: *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP)* (1997) (cited on pages 3, 68).

[Mül+12] Kevin Müller et al. “MILS-related information flow control in the avionic domain: A view on security-enhancing software architectures”. In: *Proceedings of the International Conference on Dependable Systems and Networks* (2012) (cited on page 5).

[Mül+14] Kevin Müller et al. “Secure Information Flow Control in Safety-Critical Systems”. In: (2014) (cited on page 69).

[NN07] Hanne R. Nielson and Flemming Nielson. *Semantics with Applications, An Appetizer (Undergraduate Topics in Computer Science).* 2007 (cited on pages iii, 31, 35).

[NNd] Hanne R. Nielson and Flemming Nielson. “Content-Dependent Information Flow Control”. In: *DTU Compute* (2015). DRAFT (cited on pages 6, 10, 116).

[NNL15] Hanne R. Nielson, Flemming Nielson, and Ximeng Li. “Disjunctive Information Flow”. In: *DTU Compute* (2015) (cited on pages 5, 6, 10, 57, 116).

[SL15] The Security Ledger. *GAO Warns of Cyber Risks In-Flight.* 2015. URL: `https://securityledger.com/2015/04/gao-warns-of-cyber-risks-in-flight/` (visited on June 29, 2015) (cited on page 1).

[SML/NJ] SML/NJ Fellowship. *Standard ML of New Jersey (SML/NJ).* URL: `http://www.smlnj.org/` (visited on June 27, 2015) (cited on page 54).

[SYSGO] SYSGO. *SYSGO – Embedding innocations.* URL: `http://www.sysgo.com/` (visited on January 28, 2015) (cited on page 2).

[VSI09] Dennis Volpano, Geoffrey Smith, and Cynthia Irvine. "A Sound Type System for Secure Flow Analysis". In: *Journal of Computer Security* (2009), pages 1–20 (cited on page 4).

[WIR15] WIRED. *Feds Say That Banned Researcher Commandeered a Plane.* 2015. URL: `http://www.wired.com/2015/05/feds-say-banned-researcher-commandeered-plane/` (visited on June 29, 2015) (cited on page 1).

[Yan] Jean Yang. *Jeeves Programming Language.* URL: `http://projects.csail.mit.edu/jeeves/` (visited on July 7, 2015) (cited on page 69).

[YYS12] Jean Yang, Kuat Yessenov, and Armando Solar-Lezama. "A language for automatically enforcing privacy policies". In: *ACM SIGPLAN Notices* 47 (2012), page 85 (cited on page 69).

[Z3] Microsoft Research. *Z3 – Efficient Theorem Prover.* URL: `https://github.com/Z3Prover/z3` (visited on June 27, 2015) (cited on page 54).