CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

A Real-Time Extension of the Formal Privacy Policy Framework

Master's Thesis in Computer Science

IVANA KELLYÉROVÁ

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2016

MASTER'S THESIS IN COMPUTER SCIENCE

A Real-Time Extension of the Formal Privacy Policy Framework

IVANA KELLYÉROVÁ

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2016

A Real-Time Extension of the Formal Privacy Policy Framework
IVANA KELLYÉROVÁ

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Göteborg
Sweden
Telephone +46 (0)31-772 1000

Supervisors: Raúl Pardo, Gerardo Schneider
Examiner: Alejandro Russo

Printed at Chalmers
Göteborg, Sweden 2016

A Real-Time Extension of the Formal Privacy Policy Framework
IVANA KELLYÉROVÁ
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Online social networks (OSNs) have become an important part of people's lives worldwide. Although users supply OSNs with large amounts of personal data, the ability to control the audience of one's own information is often limited to a number of predefined and often unclear options. In this work, we introduce two formal frameworks, $\mathcal{TFPPF}$ and $\mathcal{RTFPPF}$, with the aim to develop a time-sensitive formalization of evolving OSNs and the ways information spreads in them.

Both frameworks comprise three main components. First, a social network model is introduced to capture an OSN, together with its users, the relationships between them, and their knowledge bases, in a specific moment in time. Then, we define the syntax and semantics of a temporal knowledge-based logic used to reason about knowledge and learning in sequences of social network models representing the evolution of an OSN. Finally, we define a formal privacy policy language, powered by the knowledge-based logic, along with a conformance relation that determines whether a policy is violated in a specific OSN.

$\mathcal{TFPPF}$ and $\mathcal{RTFPPF}$ differ in the notion of time they use. $\mathcal{TFPPF}$ utilizes the standard $\Box$ and $\Diamond$ temporal operators and is thus on the logic level able to reason about time in a relative way. On the level of the privacy policy language, it enables to write policies to be enforced in fixed time windows. On the other hand, $\mathcal{RTFPPF}$ uses timestamps as a syntactic component on the logic level, allowing for more complex formulae and privacy policies.

Both frameworks allow users to define fine-grained, time-sensitive privacy policies based on formal logic, thus addressing the problem of privacy policy ambiguity in OSNs. Moreover, the logic in each framework can also be used directly to reason about knowledge in dynamic OSNs. Both frameworks constitute a step forward in the area of OSN formalizations.

Keywords: privacy policy, social network, epistemic logic, real-time logic, temporal logic, formal framework

Acknowledgements

I want to thank my supervisors Raúl Pardo and Gerardo Schneider for their invaluable guidance and support as well as my examiner Alejandro Russo for raising a number of interesting points to consider during our discussion.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Online social networks (OSNs) such as Facebook and Twitter have become an important part of people's lives worldwide. As much as 76% of all internet users in the United States use at least one OSN nowadays, which corresponds to a nearly tenfold increase from 2005 [27]. Another survey from 2015, targeting 32 different countries across the globe, ranging from Kenya, to Russia, Brazil, Poland, India, or Thailand, reports that the median of OSN users amounts to a similar number: 82% of adult internet users [18].[1]

Aside from helping make socializing number one internet activity [18], this popularity has also given rise to new concerns, the issue of preserving the privacy of social network users being one of the most fundamental. Even though the right to privacy is recognized as one of the basic human rights [29], OSN users are not at all confident about their data staying private and secure. In a recent series of interviews [22], only 11% of Americans believed their data was safe with social media sites, compared to 69% of respondents who were "not too confident" or "not at all confident". On the other hand, 93% of adults in the same study felt it was important to be able to control who was able to access information about them.

Although users provide OSNs with a great amount of personal data, the tools to govern one's own information offered by popular OSNs are limited, and sometimes even options one can set turn out to be different from the user's expectations. A study from 2011 [21], which focused on Facebook, found that privacy settings match the expectations of the users only 37% of the time, and when an instance of this disparity between expectations and reality occurs, the resulting difference is almost always undesirable: more content is exposed than expected, not the other way around.

There are, however, ways to address the issue. One suggestion is to assist users in managing their privacy, for instance by making OSNs such as Facebook more sensitive to social groups by grouping users into communities [21].

A more general solution is to target privacy policies themselves and their ex-

[1] To be more specific about what we mean when we say OSNs: In the aforementioned surveys, when the respondents were asked about their habits, the researchers most often gave Facebook and Twitter as the prime examples of OSNs. In countries where local sites were also prominent, these were mentioned as well (for example, VK in Russia and Renren in China).

More generally, we can adopt the definition in [10] by identifying three distinguishing characteristics of social network sites. These require that users on the site: (a) have uniquely identifiable profiles consisting of data provided by themselves, other users, and/or system, (b) can publicly articulate connections that can be viewed and traversed by others, and (c) are able to consume, produce and/or interact with streams of user-generated content.

pressive power and clarity. The authors of the formal privacy policy framework $\mathcal{FPPF}$ ([26, 25]) see the option of richer and more fine-grained privacy policies as a potential solution. In their framework, the user is given the opportunity to define their privacy policy using logic, and the behavior of the social network with respect to such policy can only be classified as being in compliance with it or not; there is no middle ground, no uncertainty where the user has to hope their privacy settings will work like they expect. Able to capture and work with virtually any social network structure, the framework can be used as an alternative to current privacy policies. This can be demonstrated by a working prototype implementing some of the privacy policies in an open-source social network, with full integration underway.

Writing privacy policies based on $\mathcal{FPPF}$, as opposed to the few hard-coded options currently being provided by popular OSNs, is a huge improvement. However, there are cases, interesting from the user's perspective, where the capabilities of $\mathcal{FPPF}$ fall short. One of these areas is the possibility of writing privacy policies sensitive to real time. For instance, someone might want to prevent her boss from knowing her whereabouts outside office hours. Or someone else might be interested in hiding any photos of him taken on New Year's Eve. Whatever their reason is, we firmly believe that the users should have as much control over their private data as possible. Allowing for even more fine-grained policies on top of $\mathcal{FPPF}$, by increasing their sensitivity to real time aspects, is a step closer to this goal.

1.1 Thesis Overview

Apart from the introduction (Chap. 1), the preliminaries and literature review (Chap. 2), the discussion (Chap. 9), and the conclusion (Chap. 10), the thesis is organized into two major parts.

Two standalone time-sensitive extensions of $\mathcal{FPPF}$ are introduced, each in a separate part. Since the high-level structure of both frameworks is very similar, each part is structured in the same way. There are three chapters, each describing the three major components of each framework – the underlying OSN model, the knowledge-based logic used to reason about agents and the information they possess, and the privacy policy language, built atop the knowledge-based logic.

The first framework we propose, the timed first-order privacy policy framework ($\mathcal{TFPPF}$), uses a privacy policy language enhanced with time fields, which make it possible to define a possibly recurring real-time window in which a policy should be enforced. The knowledge-based logic of $\mathcal{TFPPF}$ utilizes the standard box and diamond operators found in various temporal logics [17] to be able to reason about time. $\mathcal{TFPPF}$ is introduced in Part I.

The second framework proposed in this thesis, the real-time first-order privacy policy framework ($\mathcal{RTFPPF}$), represents an alternative way of reasoning about time in OSNs by incorporating timestamps representing a particular millisecond right into the syntax of the knowledge-based logic. Privacy policies written using the privacy policy language powered by the logic allow for even more fine-grained and flexible policies which need not be constrained by a time window, but can, for

example, react to events happening in the OSN. $\mathcal{RTFPPF}$ is described in Part II.

1.2 Scope

Our aim is to develop a time-sensitive formal framework. To this end we define a number of formalisms, summarized in the next section. Compared to the previous $\mathcal{FPPF}$, we do not formalize the notion of OSN instantiation, nor do we introduce any concrete operational semantics that transform one OSN model to another. Formalization of what it means for a framework to be privacy preserving is also out of scope of this thesis. Moreover, it is not our primary aim to explore the theoretical properties of the formalisms in greater detail in this thesis; we mainly aim to utilize them.

1.3 Contributions

The contributions presented in this thesis can be summarized as follows.

- Building on the existing framework $\mathcal{FPPF}$, we define two standalone temporal frameworks for OSNs: $\mathcal{TFPPF}$ in Part I and $\mathcal{RTFPPF}$ in Part II.
- A social graph-based OSN model is introduced for both frameworks to be able to capture an OSN at a specific point in time, together with the knowledge and relationships between its users. Additionally, we introduce the notion of OSN evolution for both frameworks using traces, that is, sequences of OSN models. The definitions and the properties of the models can be found in Chapter 3 for $\mathcal{TFPPF}$ and in Chapter 6 for $\mathcal{RTFPPF}$.
- A temporal knowledge-based logic is defined for both frameworks. Together with their associated semantics, these logics are used to reason about knowledge in OSNs in a time-sensitive context. We formally define and describe these logics in Chapters 4 for $\mathcal{TFPPF}$ and 7 for $\mathcal{RTFPPF}$.
- We define two privacy policy languages, one for each framework, that enable agents in the OSN to define their own privacy policies using a number of generic templates. A conformance relation is defined for each of the languages to determine whether a particular policy is not violated in a specific evolution of the OSN. Chapters 5 (for $\mathcal{TFPPF}$) and 8 (for $\mathcal{RTFPPF}$) are devoted to these two languages.

Chapter 2

Preliminaries and Literature Review

From the theoretical point of view, the task of designing a time-sensitive formal framework for OSNs based on $\mathcal{FPPF}$ largely overlaps with the areas of temporal and epistemic logics. We consider these in Section 2.1.1. In Section 2.1.2, we discuss frameworks for OSNs by other authors.

Section 2.2 aims to provide a high-level description of $\mathcal{FPPF}$ as the foundation for this work.

2.1 Theoretical Context

2.1.1 Epistemic, Temporal, and Real-Time Logics

First formalizations of what it means to know (or believe) something go back approximately to the 1950s. Hintikka's *Knowledge and Belief* [16] is commonly referred to as the first book-length treatment of the logic of knowledge, or epistemic logic. Since then, epistemic logic found its applications in many areas, including computer science, security, game theory, artificial intelligence and economics [24].

The system proposed by Hintikka in the aforementioned book used the so-called *possible worlds semantics* (due to Kripke [20]), an approach that would be commonly used in the future to the point where it is often referred to as the *classical model* [14]. In it, *agents* operate with possibly incomplete information about their surroundings: there are properties they are certain about as well as those they are uncertain about due to lack of knowledge. In the model, a *world* is essentially a set of facts that hold in a particular version of reality represented by the world. An agent considers some set of worlds to be possible – these are the agent's candidates for what the reality is really like. If something holds in every world the agent considers possible, the agent *knows* it for a fact. This is usually written as $K_a\varphi$, where a is the agent in question, φ is a property in the system of worlds, and K is the traditional symbol for the epistemic modality. On the other hand, if there exists at least one possible world in which the property differs from other possible worlds, the agent does not know which is the case in reality.

A common example to demonstrate this is depicted in Fig. 2.1 [11]. We will operate with two agents, Alice and Bob, three possible worlds s_1, s_2, s_3, and the primitive proposition p meaning "it is raining in Stockholm". Each agent a considers some worlds possible, which is captured by an equivalence relation $\mathcal{K}_a$. A pair of

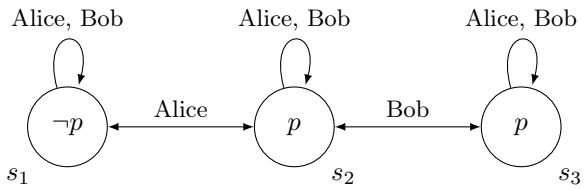

Figure 2.1: *A Kripke structure can be modeled by a labeled graph whose nodes represent worlds and edges represent the agents' relationships to pairs of worlds. More precisely, the worlds u, v are joined by an edge a whenever agent a considers world v possible given her information in world u. Such pairs of worlds are said to be* indistinguishable *to the agent.*

worlds (u, v) belongs to $\mathcal{K}_a$ if a finds v possible given the information she or he has in u. Each such pair is represented by a directed edge in Fig. 2.1. For example, if Alice thinks the world is currently in state s_1, then she considers s_1, s_2 to be possible. If, on the other hand, she thinks the world is currently in state s_3, she only considers s_3 possible. In world s_1 it is not raining in Stockholm ($\neg p$ holds), but Alice does not know that, because in world s_1, she considers both s_1, where it is not raining, and s_2, where it is raining, possible. However, in world s_3, she knows it is raining in Stockholm ($K_{Alice}\ p$) since in all the worlds she considers possible at s_3 (in this case, only s_3 itself), p holds.

Though often used in single-agent scenarios to reason about the nature of knowledge itself, epistemic logic found another major application in multi-agent systems, with notions such as *distributed* and *common knowledge* stemming from this combination. In short, φ is distributed knowledge among a group of agents ($D_G\varphi$) if φ can be obtained from their collective knowledge. Common knowledge is a more complex concept to grasp – φ is common knowledge among a group G of agents ($C_G\varphi$) if everyone in G knows φ and everyone in G knows that everyone in G knows φ and everyone in G knows that everyone in G knows that everyone in G knows φ and so on *ad infinitum*. We can take events that happen publicly, with everyone present and capable of observing the event, as a natural example of common knowledge – for instance, two people shaking hands [23], or someone making a public proclamation [11].

Aside from new group epistemic modalities, in a multi-agent setting, one can also express facts involving the knowledge of several agents at once like "Alice knows that Bob knows that Charlie does not know that David knows that it is raining in Gothenburg tonight". Coming back to the example in Fig. 2.1, in world s_2, Alice knows that Bob knows whether it is raining in Stockholm (though she does not know what the weather is herself), since in both s_1 and s_2, which are the two worlds Alice considers possible in s_2, it is the case that Bob knows what the weather is in the Swedish capital. Reasoning about knowledge of multiple agents is crucial from the point of view of applications in privacy which, in essence, is about preventing someone from learning something that someone else wants to keep hidden. This is also one of the reasons why multi-agent epistemic logic is a natural candidate for reasoning about privacy in OSNs.

Epistemic logic is also applicable in a dynamic context in which the world un-

dergoes a certain evolution. Here, the focus lies on the way knowledge evolves alongside other properties of the world. A common way to capture the semantics of this evolution is via *interpreted systems*, where each agent has a local state whose precise structure depends on the system being modeled. Moreover, the whole world is characterized by a global state which consists of the local states of all agents plus the local state of the environment, which comprises everything relevant not present in the other local states. A *run*, then, is a function of time returning the global state of the system at a specific point in time, and a *system* is a set of runs. In a system, a *point* is characterized by a run and a point in time, and we say that two points are indistinguishable to an agent if its local state is the same in both points. For a simple example, we refer to [23].

More generally, formalizing and reasoning about properties in the presence of time is a field of logic in itself, whose beginnings also date back roughly to the 1950s. Among the oldest and most well-known temporal logics is LTL (Linear Temporal Logic) [17], originally proposed in [28], which utilizes two special temporal operators, called "until" and "next". These can be used to define the perhaps more well-known box ($\Box$) and diamond ($\Diamond$), commonly read "always" and "eventually", respectively. One of the uses of linear-time logics like LTL is checking whether a system is behaving correctly. Given two propositions p and q, one can, for example, express statements like $\Box(p \rightarrow \Diamond q)$ ("it is always the case that if p occurs, then eventually q will occur") to ensure that the signal p is always followed by the appropriate response in the form of q [2].

However, pure LTL allows for reasoning about time in a relative, qualitative way only. It is, for example, not possible to specify the time interval in which q has to occur after p, only that it should occur eventually. This is where other temporal logics, either extensions of LTL, or separate logics on their own, can be used. These can be classified based on a number of attributes, such as the notion of time they use, whether they are *linear* (only accounting for one evolution of the world) or *branching* (multiple evolutions), which temporal operators they utilize [2], what their possible axiomatizations are [15, 30], or their properties in terms of expressiveness and complexity [3]. Another natural approach to modeling the behavior of real-time systems over time is to use *timed automata*, as proposed by Alur and Dill [1].

For the purposes of this thesis, in which we aim to present a way to reason about knowledge of OSN users in a linear real-time setting, we are mainly interested in a combination of the concepts described above, namely real-time epistemic logics. This particular area contains a number of formalisms created with a specific application in mind or with the aim to study a specific property of evolving knowledge [5, 7, 6, 8, 13]. In [31], Woźna and Lomuscio introduce TCTLKD, a logic to reason about knowledge, correctness and real time in the context of timed automata and interpreted systems. TCTLKD utilizes the epistemic modalities K, C and D in a standard way (for instance, an agent is said to know something if it holds in all scenarios it considers possible). In [4], Ben-Zvi and Moses revise these operators themselves by adding a *time instance* to the knowledge modalities K (written as $K_{\langle a,t\rangle}$ where a is an agent and t is a time instance) and C. These represent the knowledge at a particular moment in time. The authors also introduce a special

predicate for events, $occurred_t(e)$, whose meaning is that the event e has occurred by time t.

Though we are unaware of the existence of any real-time epistemic logics created specifically for the purpose of reasoning about knowledge in evolving OSNs, we draw syntactic inspiration from the aforementioned studies in this thesis, especially LTL and [4], which we combine with a specialized semantics built upon the one used for the original framework $\mathcal{FPPF}$ [26, 25] (Sec. 2.2).

2.1.2 OSN Formalizations

There at least three formalizations of OSNs whose main focus is on the privacy of users. Aside from $\mathcal{FPPF}$, which we are building upon and to which we devote the next section, there is also a model for Facebook-like social network systems by Fong, Anwar and Zhao [12] and a framework called Poporo by Catano, Kostakos and Oakley [9].

In the former ([12]), the authors use an access control model to capture the privacy preservation mechanism of Facebook, which can further be instantiated into other Facebook-like OSNs. They define social network systems to be made of users and objects (data that can be accessed) owned by users with the aim to model the authorization mechanism used to grant access to objects. They also show that the model can be instantiated to be able to express policies that are currently not supported by Facebook, but are interesting from the user's perspective.

The Poporo framework [9], on the other hand, consists of several parts. The main component is called Matelas and is a specification layer built on predicate calculus. It is used to model the content of a social network (SN), the privacy policies used, and the friendship relations. The predicate calculus specification is then turned into a code-level specification model which the authors call a SN core application. Additional functionalities, written for example in Java or C, can then be added (plugged in) to the core and their adherence to the policies stipulated in Matelas can be determined using a proof validator.

In this thesis, we, too, follow a formal methods-based approach, but one based on the original $\mathcal{FPPF}$.

2.2 The First-Order Privacy Policy Framework $\mathcal{FPPF}$

Since our extension builds on the foundations laid by $\mathcal{FPPF}$ [26, 25], we devote this section to a high-level overview of the framework.

$\mathcal{FPPF}$ comprises three main parts: (a) a social graph-based model for OSNs, (b) a knowledge-based logic (called $\mathcal{KBL}_{\mathcal{SN}}$) with a satisfiability relation determining when a formula holds in a network, and (c) a privacy policy language ($\mathcal{PPL}_{\mathcal{SN}}$) together with a conformance relation defining when a social network respects a specific policy. In the following we describe each of these parts in more detail.

2.2.1 The Social Network Model SNM

$\mathcal{FPPF}$ defines a generic model to capture the specifics of any social network service. This model is based on social graphs, that is, graphs whose nodes represent users (or, more generally, agents – we will use these terms interchangeably), with edges indicating different kinds of relationships between the users. In the case of $\mathcal{FPPF}$, these graphs are enriched with other components, such as information about the agents' knowledge, stored in their knowledge bases. The agents' permissions to carry out actions towards other users, their connections to one another, and their privacy policies are also included in the model.

2.2.2 The Knowledge-Based Logic $\mathcal{KBL}_{\mathcal{SN}}$

The knowledge-based logic $\mathcal{KBL}_{\mathcal{SN}}$ is used to reason about the properties of the SNM and its agents. Built on top of epistemic logic, it utilizes modalities such as $K_a\varphi$, $E_G\varphi$, and $S_G\varphi$. Intuitively, these mean, in turn: agent a knows φ; every agent in the set G knows φ; and someone in the set G knows φ. If, for instance, Alice can see Bob's location, we can write $K_{Alice}\mathit{location}(\mathit{Bob})$. Connecting this to the SNM mentioned before, this would mean that the piece of information $\mathit{location}(\mathit{Bob})$ either exists explicitly in Alice's knowledge base, or can be derived by inference from information there.

$\mathcal{KBL}_{\mathcal{SN}}$ also directly provides syntactic support for *connections* and *permissions* (or *actions*) as relationships typical for social networks. The *friendship* connection on Facebook and the *follower* connection on Twitter can serve as examples of the former. The action predicates, on the other hand, model permissions between agents. For instance, we can express that Bob is not allowed to send a friend request to Alice.

The semantics of $\mathcal{KBL}_{\mathcal{SN}}$ is given in the form of the satisfiability relation which determines whether a $\mathcal{KBL}_{\mathcal{SN}}$ formula is valid in a specific social network model by looking at its properties, such as knowledge of the agents or the connections and actions between them.

2.2.3 The Privacy Policy Language $\mathcal{PPL}_{\mathcal{SN}}$

The privacy policy language $\mathcal{PPL}_{\mathcal{SN}}$ is a formal language that can be used to write complex privacy policies based on $\mathcal{KBL}_{\mathcal{SN}}$. Each policy belongs to (is written by) an agent who is regarded as its owner. $\mathcal{PPL}_{\mathcal{SN}}$ chooses to write privacy policies in a restrictive sense, i.e., each privacy policy disallows a particular behavior to take place or a piece of information to spread. One can express simple requirements like "no one can know my location" or "Alice cannot send me a friend request", but also complex ones, for instance "if I create an event and only give certain people the permission to join it, it cannot be accessed by people outside of the group".

$\mathcal{PPL}_{\mathcal{SN}}$ provides two templates for privacy policies: direct restrictions and conditional restrictions. In direct restrictions, there is no precondition that "activates" the policy; the first two policies mentioned above are examples of direct restrictions. In conditional restrictions, like the more complicated event example above, the policy should only be enforced when the model is in a specific state.

The conformance relation provides the semantics of the privacy policy language, where the policy is checked to comply with a certain SNM. This is done with the help of $\mathcal{KBL}_{\mathcal{SN}}$'s satisfiability relation.

2.2.4 Other Features

Aside from SNM, $\mathcal{KBL}_{\mathcal{SN}}$, and $\mathcal{PPL}_{\mathcal{SN}}$, the framework $\mathcal{FPPF}$ is also characterized by a number of additional features. To be able to reason about a specific OSN, it defines the notion of framework instantiation (in [25], an example instantiation of Twitter is given).

A dynamic version of the framework is also given in [25]. Using a labeled transition system, it is possible to capture certain predecessor-successor relationships between SNMs and the actions that transformed one model to another. The dynamic behavior of an OSN is described using small-step operational semantics.

$\mathcal{FPPF}$ also defines what it means for an OSN to be privacy-preserving – no matter what events happen inside it, it can never violate a user's privacy policy.

Part I

Privacy Policies with Real-Time Windows

Chapter 3

Timed Social Network Model (TSNM)

3.1 Timed First-Order Privacy Policy Framework $\mathcal{TFPPF}$

As an extension of $\mathcal{FPPF}$, the framework proposed in this part relies on the foundations laid by the original framework. As a result, the structure of the framework proposed in this part is very similar to the original one. We, too, define three major components upon which the framework is built: (a) a timed social network model (TSNM), together with the notion of traces, that is, sequences of these models that represent the evolution of an OSN; (b) a temporal knowledge-based logic ($\mathcal{TKBL}_{\mathcal{SN}}$) with temporal modalities, inspired by temporal logics such as LTL, and one new epistemic modality; and (c) a timed privacy policy language ($\mathcal{TPPL}_{\mathcal{SN}}$), enabling the user to define a (possibly recurring) real-time window in which their policy should be enforced.

Together, these parts form the new timed first-order privacy policy framework ($\mathcal{TFPPF}$). Its formal definition follows.

Definition 1 (Timed First-Order Privacy Policy Framework)**.** The tuple

$$\mathcal{TFPPF} = \langle \mathcal{TSN}, \mathcal{TKBL}_{\mathcal{SN}}, \models, \mathcal{TPPL}_{\mathcal{SN}}, \models_C \rangle$$

is a *timed first-order privacy policy framework* where

- $\mathcal{TSN}$ is the set of all possible timed social network models;
- $\mathcal{TKBL}_{\mathcal{SN}}$ is a temporal knowledge-based logic;
- $\models$ is a satisfiability relation defined for $\mathcal{TKBL}_{\mathcal{SN}}$;
- $\mathcal{TPPL}_{\mathcal{SN}}$ is a formal timed privacy policy language;
- $\models_C$ is a conformance relation defined for $\mathcal{TPPL}_{\mathcal{SN}}$.

In what follows we devote a chapter to each of these components. We first formalize timed social network models in the remaining sections of this chapter. In Chapter 4, we give the syntax and the semantics ($\models$) of the temporal knowledge-based logic $\mathcal{TKBL}_{\mathcal{SN}}$. Finally, we describe the timed privacy policy language $\mathcal{TPPL}_{\mathcal{SN}}$ together with its conformance relation $\models_C$ in Chapter 5.

3.2 Timed Social Network Model TSNM

In the original framework $\mathcal{FPPF}$, social network models were defined as social graphs with agents, their knowledge bases and privacy policies, and a first-order relational structure. $\mathcal{TFPPF}$ retains this definition, but extends the formulae in the knowledge bases of agents with a timestamp, thus rooting each piece of information in time. What notion of time we will be working with in this thesis is discussed and formalized in Section 3.2.1.

Moreover, in addition to the definition of timed social network models, we also introduce our notion of dynamic, evolving OSNs. In $\mathcal{FPPF}$, a labeled transition system was used for a similar purpose. Here, we use sequences of timed social network models, each representing a snapshot of the modeled OSN at a specific moment in time. We describe and formalize these notions in Section 3.2.2.

3.2.1 Formalizing Time

Adding timestamps to formulae in the knowledge bases of agents enables us to tell apart pieces of information in a way that is not possible in the original framework. Let us take the simple example of Alice learning Bob's location. In $\mathcal{FPPF}$, the simplest way to capture this scenario is that the predicate $\mathit{location}(\mathit{Bob})$ either exists explicitly in Alice's knowledge base, or she is able to infer it from the knowledge she already has. At a later point, Alice might learn the location of Bob again, and again it will be available to her as $\mathit{location}(\mathit{Bob})$. As a consequence, Alice knowing Bob's location does not really tell us anything about Bob's location – the information might easily be outdated and there is no way to find out.

Another option in $\mathcal{FPPF}$ is to somehow establish the relative order of the instances when Alice learns Bob's location using so-called resource identifiers, so she would be able to access predicates like $\mathit{location}(\mathit{Bob}, 1)$ or $\mathit{location}(\mathit{Bob}, 47)$. This is closer to the design of $\mathcal{TFPPF}$, as it enables us to "refresh" knowledge without losing previous instances. Still, however, although it is possible to determine the relative order of pieces of knowledge of the same kind, there is nothing preventing the latest one from being outdated, as in the previous case.

By adding timestamps to the agents' knowledge bases, one is both able to tell apart pieces of information of the same kind, and precisely pinpoint the moment when the piece of knowledge was learned.

Let us now formalize the notion of timestamps.

Definition 2 (Timestamp)**.** A *timestamp* t is a natural number representing the number of milliseconds elapsed since January 1, 1970, 00:00:00.000.

Of course, we could have chosen a large number of equally good starting points; the beginning of 1970 was chosen simply because it is also the Unix epoch.

We will use $\mathbb{T}$ to denote the set of all timestamps.

Since referring to specific dates as, for example, $t_1 = 1458986348693$ or $t_2 = 192814041542693$ has the potential to become rather confusing, we will use the more human-readable standard ISO format [19] of `YYYY-MM-DD hh:mm:ss.sss` when

Table 3.1: *When talking about timestamps, we use the standard human-readable ISO format, optionally skipping some parts of the time component. We use the format in the first column.*

Compact	Full
2016-03-26 10:59:08.562	2016-03-26 10:59:08.562
2000-01-01	2000-01-01 00:00:00.000
1990-03-10 12:00	1990-03-10 12:00:00.000
8080-01-12 23:59:02.100	8080-01-12 23:59:02.100

talking about timestamps.[1] Optionally we might skip the time part, in which case we assume it defaults to 00:00:00.000, or its suffix, starting with either seconds or milliseconds – then we assume the number of missing (milli)seconds amounts to zero. Table 3.1 contains a number of examples. In some cases where we want to demontrate a point (most often in examples) and specific timestamps are not important, we will use dummy timestamps 1, 2, and so on.

There are two main reasons behind choosing this particular timestamp format. First, it is practical – any timestamp represents a valid date and time and one does not have to consider technicalities such as variable month and year length. The other reason is that they work well in contexts where they are meant to be used – both attached to pieces of knowledge in agents' knowledge bases, and in the privacy policy language.

Their use in the knowledge bases relies on the basic property that, given any two timestamps t_1 and t_2, it is possible to determine their relative order, i.e., considering the two points in time represented by the timestamps, determine which one happened sooner (later). As for their use in privacy policies, one of the goals of the extension in this part is to allow users to define a real-time window frame in which their privacy policy should be enforced. Therefore, when defining the time-sensitive version of the privacy policy language ($\mathcal{TPPL}_{\mathcal{SN}}$), the user has to be able to pinpoint a specific moment in time, which is done using our notion of timestamps. In addition, this simple definition makes it possible to not only determine the order of timestamps, but also to quantify the distance $|t_2 - t_1|$, which is later formalized as *duration* (Def. 11). This property is also used in $\mathcal{TPPL}_{\mathcal{SN}}$, where the most basic privacy policy window is defined using a timestamp and a duration field (as opposed to using two timestamps).

3.2.2 Capturing the Evolution of an OSN

We now provide a definition of a timed social network model, with timestamps attached to information in the knowledge bases of agents. $\mathcal{F}_{\mathcal{TKBL}}$ stands for the set of all well-formed formulae of the time-sensitive knowledge-based logic $\mathcal{TKBL}_{\mathcal{SN}}$, which is defined at a later point (Def. 7). The specific shape of the formulae used is not important at this point.

[1] Note that there is a number of technical issues that would arise if this format was to be used in practice. For instance, we do not specify whether we count leap seconds – which would have an effect on the conversion from and to the ISO format –, or what timezone we are in.

Definition 3 (Timed Social Network Model)**.** Given a set of formulae $\mathcal{F} \subseteq \mathcal{F}_{\mathcal{TKBL}}$, a set of privacy policies Π, and a finite set of agents $Ag \subseteq \mathcal{AU}$ from a universe $\mathcal{AU}$, a *timed social network model* (TSNM) is a social graph $\langle Ag, \mathcal{A}, KB, \pi \rangle$, where

- Ag is a nonempty finite set of nodes representing the agents in the OSN;
- $\mathcal{A}$ is a first-order relational structure over the TSNM, consisting of a set of domains $\{D_o\}_{o \in \mathcal{D}}$, where $\mathcal{D}$ is an index set; a set of relation symbols, function symbols and constant symbols interpreted over a domain;
- $KB : Ag \to 2^{\mathcal{F} \times \mathbb{T}}$ is a function retrieving the set of accumulated knowledge, each piece with an associated timestamp, for each agent, stored in the knowledge base of the agent; we write KB_i for $KB(i)$;
- $\pi : Ag \to 2^{\Pi}$ is a function returning the set of privacy policies of each agent; we write π_i for $\pi(i)$.

We denote $\mathcal{TSN}$ the set of all TSNMs.
Let us take a closer look at each component.

Agents We assume that, in addition to "normal" agents (that is, those who represent actual users of the OSN), there is also a special agent called the *environment* (e). The environment contains all knowledge that is true in the TSNM.

Knowledge Bases The set retrieved by the knowledge base function KB of an agent contains everything the agent has learned so far, written in the language of the temporal knowledge-based logic $\mathcal{TKBL}_{\mathcal{SN}}$. This can be anything from simple predicates meaning "I learned Alice's location on April 29, 2016 at 18:44:13.562", to more complex information such as "on July 15, 2008 at 10:00 I learned that Bob learned that Alice knew Bob's location". Note that there is only one timestamp in any formula in an agent's knowledge base, so timestamps are not nested – they always refer to the formula as a whole. For the formalization of what agents can know and what it means in the context of $\mathcal{TFPPF}$ we refer to Chapter 4, which is devoted to $\mathcal{TKBL}_{\mathcal{SN}}$ and its properties.

The First-Order Relational Structure The overall shape of $\mathcal{A}$ depends on the properties of the OSN being modeled. This is especially apparent in the case of relational symbols, which are used to represent the *connections* and *permission actions* (or just *permissions*, or *actions*) – the edges of the underlying social graph. Connections stand for relationships (not necessarily symmetric) between agents, such as *friends* (usually two-way) or *follower* (usually one-way). Permissions model what actions a user is allowed to execute toward other users. For example, Alice might not give Bob permission to send her a friend request.

More formally, given sets of indices $\mathcal{C}$ for connections and Σ for permissions, we define connections and permissions as families of binary relations on the set of agents: $\{C_i\}_{i \in \mathcal{C}} \subseteq Ag \times Ag$ and $\{A_i\}_{i \in \Sigma} \subseteq Ag \times Ag$, respectively. For better readability, we will use a predicate like $friends(Alice, Bob)$ to mean that $Alice, Bob \in Ag$ belong to the binary relation *friends*.

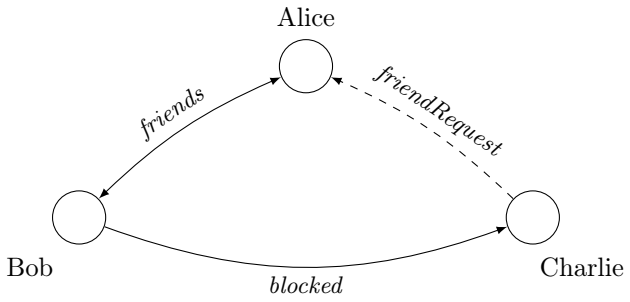

Figure 3.1: *A simple social graph with three agents, two connections (the bidirectional* friends *and the one-way* blocked*) and one permission (*friendRequest*). The connections are represented by normal edges, the permission uses a dashed one. The information from this graph can be summarized as follows: Alice is friends with Bob and vice-versa, Bob is blocking Charlie, and Charlie is allowed to send a friend request to Alice.*

Privacy Policies In addition to possessing a set of knowledge, each agent is able to define their own set of privacy policies using the timed privacy policy language $\mathcal{TPPL}_{\mathcal{SN}}$. Generally speaking, the goal of these policies is to restrict the audience of something in the OSN, for example a post or a picture. The language itself and its attributes are described in detail in Chapter 5.

Example 1. We give a simple example of a TSNM in Fig. 3.1 with $Ag = \{Alice, Bob, Charlie\}$, $\mathcal{C} = \{friends, blocked\}$ and $\Sigma = \{friendRequest\}$. The agents' knowledge bases and privacy policies are not depicted at this point – we will revisit this example later once we have established the general shape of formulae in the users' knowledge bases.

Since $\mathcal{TFPPF}$ is by definition a dynamic framework, we need a way to capture the evolution of an OSN. This is done by using sequences of TSNMs, so that every TSNM in the sequence represents a snapshot of the OSN at some point. This structure is called a *trace*.

More specifically, a trace is a sequence of pairs consisting of a specific $TSN \in \mathcal{TSN}$ together with a timestamp t. The intuitive meaning is that each such TSN is a snapshot of the OSN at point t, as if we froze the network, along with the knowledge and relationships between its agents, at that moment.

We demand traces be finite. This makes working with them more practical, especially in terms of the semantics we give at a later point (Def. 10), which relies on access to all social network models in the trace.

Definition 4 (Trace)**.** Given $k \in \mathbb{N}$, a trace σ is a finite sequence

$$\sigma = \langle (TSN_0, t_0), (TSN_1, t_1), \ldots, (TSN_k, t_k) \rangle$$

such that, for all $0 \geq i \geq k$, $TSN_i \in \mathcal{TSN}$ and $t_i \in \mathbb{T}$.

This basic definition does not impose any restrictions on the timestamps or TSNMs used, so even sequences of TSNMs that have little in common, with arbitrary, and potentially repeating, timestamps, are traces by definition. To single out meaningful traces, that is, those that actually capture the evolution of an OSN, we

introduce the notion of *well-formed traces*. To be well-formed, a trace has to satisfy three conditions.

Order We place a restriction on the ordering of the pairs in the trace with regards to the timestamps, which we require to be strictly ordered from smallest to largest. This allows us to immediately identify the successors and predecessors and the gradual changes happening between TSNMs at different positions of the trace.

Plausible Knowledge Moreover, for each (TSN, t) in the trace, the timestamps used inside the agents' knowledge bases must be at most t. Intuitively, if a snapshot of an OSN was taken at time t, then t should be the latest point at which the agents could have obtained new knowledge.

Continuity Finally, for the successor-predecessor relationships between two adjacent TSNMs to make sense, each TSNM, starting from the second one, has to be the result of some events happening in the TSNM that comes immediately before. For this purpose we use the transition relation $\rightarrow$ defined for $\mathcal{FPPF}$ [25], extended with a timestamp capturing when a particular set of events happens. More formally, assuming that EVT is the set of all events, the $\rightarrow$ relation here is characterized as $\rightarrow \subseteq \mathcal{TSN} \times 2^{EVT} \times \mathbb{T} \times \mathcal{TSN}$ and the tuple $\langle TSN_1, E, t, TSN_2 \rangle$ is in $\rightarrow$ if TSN_2 is the result of the nonempty set of events E happening in TSN_1 at time t. We will write this as $TSN_1 \xrightarrow{E,t} TSN_2$.

Definition 5 (Well-Formed Trace)**.** Let

$$\sigma = \langle (TSN_0, t_0), (TSN_1, t_1), \ldots, (TSN_k, t_k) \rangle$$

be a trace. σ is *well-formed* if the following conditions hold:

1. For any i, j such that $0 \geq i, j \geq k$ and $i < j$, it is the case that $t_i < t_j$.
2. Let KB^{TSN} denote the knowledge base function of model TSN, and similarly for Ag^{TSN}. For all $(TSN, t) \in \sigma$, for all $a \in Ag^{TSN}$, for all $(\varphi, t_\varphi) \in KB_a^{TSN}$, it is the case that $t_\varphi \leq t$.
3. For all i such that $0 \leq i \leq k-1$, it is the case that $TSN_i \xrightarrow{E,\, t_{i+1}} TSN_{i+1}$, where $E \subseteq EVT$ and E is nonempty.

We will use TCS to refer to the set of all well-formed traces.

3.2.3 Notation

In the previous text we established a number of interconnected notions. In order to simplify notation used in the future, we introduce the following shortcuts. We assume σ is a well-formed trace.

Trace Properties

We name the set of all timestamps associated with the TSNMs in the trace $\mathbb{T}_\sigma$. In other words, $\mathbb{T}_\sigma = \{t \mid (TSN, t) \in \sigma\}$. In a similar manner, we use $\mathcal{TSN}_\sigma$ to denote the set of all TSNMs in the trace: $\mathcal{TSN}_\sigma = \{TSN \mid (TSN, t) \in \sigma\}$.

Example 2. Let us say that

$$\begin{aligned}\sigma = \langle &(TSN_0,\ \text{2016-04-30 19:57}),\\ &(TSN_1,\ \text{2016-04-30 19:59}),\\ &(TSN_2,\ \text{2016-04-30 20:00}),\\ &(TSN_3,\ \text{2016-04-30 20:37}),\\ &(TSN_4,\ \text{2016-04-30 20:47})\rangle.\end{aligned}$$

Retrieving the set of all timestamps or TSNMs present in σ is straightforward:

$$\begin{aligned}\mathbb{T}_\sigma = \{&\text{2016-04-30 19:57}, \text{2016-04-30 19:59}, \text{2016-04-30 20:00},\\ &\text{2016-04-30 20:37}, \text{2016-04-30 20:47}\}\\ \mathcal{TSN}_\sigma = \{&TSN_0, TSN_1, TSN_2, TSN_3, TSN_4\}\end{aligned}$$

Accessing Parts of a Trace

Specific TSNMs Often we will need to refer to a specific TSNM in a trace. For this purpose, $\sigma[t]$ for a timestamp $t \in \mathbb{T}_\sigma$ is the model $TSN \in \mathcal{TSN}$ belonging to the pair $(TSN, t_{TSN}) \in \sigma$ for which $t_{TSN} = t$. Note that in a well-formed trace, there is exactly one such model.

Once we have retrieved a specific TSNM $TSN \in \mathcal{TSN}_\sigma$, we will often need to refer to its components directly. We will write ${Ag}^{TSN}$, ${\mathcal{A}}^{TSN}$, ${KB}^{TSN}$, ${\Pi}^{TSN}$ to access TSN's agent set, relational structure, knowledge base, and privacy policy function, respectively.

Example 3. Let σ be the trace from Example 2. The indexing function can be used to access any of the three models in a straightforward way:

$$\begin{aligned}\sigma[\text{2016-04-30 19:57}] &= TSN_0\\ \sigma[\text{2016-04-30 19:59}] &= TSN_1\\ \sigma[\text{2016-04-30 20:00}] &= TSN_2\end{aligned}$$

Note that by definition, only timestamps that actually exist in the trace (that is, those that belong to $\mathbb{T}_\sigma$) can be used, so for example σ[2016-04-30 19:58] is invalid.

If we want to get the knowledge base function of TSN_1, we can write

$${KB}^{\sigma[\text{2016-04-30 19:59}]}$$

and similarly for other TSNMs and their components.

Subtraces We define $\sigma[t_1 \mathinner{..} t_2]$, where $t_1 \leq t_2$, to be a function returning a specific subtrace of σ. The first element of the subtrace is the first $(TSN, t) \in \sigma$ for which $t \geq t_1$; the last element of the subtrace is the last $(TSN, t) \in \sigma$ such that $t \leq t_2$.

Essentially, the function simply extracts all the TSNMs with timestamps that fall into the interval from t_1 to t_2, inclusive.

Note that unlike the indexing function $\sigma[t]$, here the index timestamps need not be actual timestamps of the models in the trace. It might even be the case that t_1 is greater than the last timestamp of σ (using the notation introduced previously, $t_1 \geq \max(\mathbb{T}_\sigma)$) or that t_2 is less than the very first timestamp of σ ($t_2 \leq \min(\mathbb{T}_\sigma)$). In these cases, the subtrace returned is empty.

We will also use $\sigma[t \mathinner{..}]$ and $\sigma[\mathinner{..} t]$ to retrieve the suffix (prefix) of σ that satisfies the corresponding part of the previous description.

Example 4. Once again, we will use σ from the previous Examples 2 and 3. Imagine we want to take the subtrace starting with TSN_2 and ending with TSN_4. There is a number ways to achieve this. For instance:

$$\sigma[\text{2016-04-30 20:00} \mathinner{..} \text{2016-04-30 20:47}]$$
$$\sigma[\text{2016-04-30 19:59:30.587} \mathinner{..} \text{2016-05-01}]$$
$$\sigma[\text{2016-04-30 19:59:11} \mathinner{..}]$$

Chapter 4

Temporal Knowledge-Based Logic ($\mathcal{TKBL}_{\mathcal{SN}}$)

In this chapter we introduce a logic that will be used to reason about knowledge in $\mathcal{TFPPF}$. It will also be the base upon which the timed privacy policy language $\mathcal{TPPL}_{\mathcal{SN}}$ (Chap. 5) will be built.

Again, we follow in the footsteps of the authors of $\mathcal{FPPF}$ [25], extending their knowledge-based $\mathcal{KBL}_{\mathcal{SN}}$ with two temporal operators and a new epistemic operator for learning. The resulting logic is called temporal knowledge-based logic, or $\mathcal{TKBL}_{\mathcal{SN}}$.

4.1 Syntax of $\mathcal{TKBL}_{\mathcal{SN}}$

We assume that the most basic building blocks of the logic – the function, relation, and constant symbols – are parts of a vocabulary. We also assume that the former two have an implicit arity, corresponding to the number of arguments they take. Furthermore, we assume we have an infinite supply of variables we can use.

Definition 6 (Term)**.** Let c be a constant, f be a function symbol, and x be a variable. A *term* s is inductively defined as

$$s ::= c \mid x \mid f(\vec{s}),$$

where $\vec{s}$ is a tuple of terms respecting the arity of f.

One of the parts that sets $\mathcal{TKBL}_{\mathcal{SN}}$ (and the original $\mathcal{KBL}_{\mathcal{SN}}$) apart from other logics is the existence of two special types of predicates: connections and actions. These mirror the different kinds of edges in the social graph of TSNMs (Def. 3). To recapitulate: Connections $\mathcal{C}$ represent relationships between agents, as defined by the authors of the specific OSN they are modeling. They can be symmetric (*friends*) as well as asymmetric (*follower*). Actions (or permissions) Σ connect two users a, b if a is allowed to execute a specific action towards b, for example send a friend request.

Definition 7 (Syntax of $\mathcal{TKBL}_{\mathcal{SN}}$)**.** Given agents $a, b \in Ag$, a nonempty set of agents $G \subseteq Ag$, predicate symbols $a_n(a, b)$, $c_m(a, b)$, $p(\vec{s})$ where $m \in \mathcal{C}$ and $n \in \Sigma$, and a variable x, the *syntax of the temporal knowledge-based logic* $\mathcal{TKBL}_{\mathcal{SN}}$ is

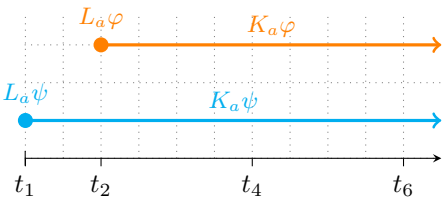

Figure 4.1: *Our interpretation of learning and knowing can be illustrated by this picture. Suppose we have two formulae φ and ψ and an agent a. t_1 and t_2 represent the point in time when a came to possess (learned) ψ and φ, respectively. If this picture captures all knowledge transfer, then $L_a\psi$ should only be true at time t_1 and $L_a\varphi$ only at time t_2. However, at any time $t \geq t_1$ (for example t_4) we have $K_a\psi$ and at any time $t' \geq t_2$ (such as t_6), $K_a\varphi$.*

inductively defined as:

$$\varphi ::= \rho \mid \varphi \wedge \varphi \mid \neg\varphi \mid \forall x.\varphi \mid K_a\psi \mid L_a\psi \mid C_G\psi \mid D_G\psi \mid \Box\varphi \mid \Diamond\varphi$$
$$\psi ::= \rho \mid \psi \wedge \psi \mid \neg\psi \mid \forall x.\psi \mid K_a\psi \mid L_a\psi \mid C_G\psi \mid D_G\psi$$
$$\rho ::= c_m(a,b) \mid a_n(a,b) \mid p(\vec{s})$$

We use $\mathcal{F}_{\mathcal{TKBL}}$ to denote the set of all well-formed formulae of $\mathcal{TKBL}_{\mathcal{SN}}$.

The epistemic modalities used here are read, in turn: $K_a\psi$ as "agent a knows ψ", $L_a\psi$ as "agent a learns ψ", $C_G\psi$ as "it is common knowledge in group G that ψ", and $D_G\psi$ as "it is distributed knowledge in group G that ψ". The temporal operator $\Box$ is read "always", $\Diamond$ is "eventually".

There are two main differences between $\mathcal{TKBL}_{\mathcal{SN}}$ and $\mathcal{KBL}_{\mathcal{SN}}$.

Temporal Operators $\mathcal{TKBL}_{\mathcal{SN}}$ utilizes the temporal operators $\Box$ and $\Diamond$. These can be found in many temporal logics, where they are usually defined using a more basic *until* operator. We do not have such operator here since we were unable to find interesting use cases in the context of OSNs.

Note also that no temporal operator is allowed inside an epistemic modality. The reason is that, as we are about to show, the temporal operators are used with regards to traces, to be able to iterate through them, whereas once a formula is inside an epistemic modality, it is checked in a single knowledge base which itself is static.

Learning Modality In a static context, the K modality is quite enough to express that an agent possesses a specific piece of knowledge. In a dynamic context, however, we found it more natural to separate *knowing* something and *learning* it. The distinction is captured by what we later on establish as the learning axiom: if an agent learns something, she knows it.

Learning can be intuitively described as the instant in time when an agent comes into contact with a piece of information. On the other hand, knowing something is not a single point in time, it is more of an interval (Fig. 4.1).

4.1.1 Notation

We will use the following syntactic sugar borrowed from $\mathcal{KBL}_{\mathcal{SN}}$. Given agents $a, b \in Ag$, a nonempty group $G \subseteq Ag$, and an action a_n, we have:

$$\begin{array}{rcl} P^c_b a_n & := & a_n(b,c) \\ SP^c_G a_n & := & \bigvee_{b \in G} a_n(b,c) \\ GP^c_G a_n & := & \bigwedge_{b \in G} a_n(b,c) \end{array}$$

These stand for, in turn: "b is permitted to execute a_n to c", "someone in G is permitted to execute a_n to c", and "everyone in G is permitted to execute a_n to c".

Often it is useful to be able to express that, in a group G, everyone (E) knows (or learns) something. The same goes for someone (S) in a group G learning or knowing something. These notions are straightforward to define using the basic K and L modalities:

$$\begin{array}{ll} S_G\varphi \triangleq \bigvee_{a \in G} K_a\varphi & SL_G\varphi \triangleq \bigvee_{a \in G} L_a\varphi \\ E_G\varphi \triangleq \bigwedge_{a \in G} K_a\varphi & EL_G\varphi \triangleq \bigwedge_{a \in G} L_a\varphi \end{array}$$

These are read "someone in G knows (learns) φ" and "everyone in G knows (learns) φ".

4.2 Agents and Their Knowledge

Let us now retrace a bit and take a closer look at the knowledge bases of agents, first defined in Def. 3. The set retrieved by the KB function contains everything the agent knows, in form of timestamped $\mathcal{TKBL}_{\mathcal{SN}}$ formulae.

In addition to possessing specific knowledge, we also want to make the agents smarter by enabling them to gain new knowledge on their own, if it follows from what they already know. For this purpose we define the notion of *timed closure of a knowledge base* (Cl_T). In short, the closure of a knowledge base KB of an agent contains all the knowledge already in KB, plus all knowledge that can be inferred from it according to a set of rules. Of course, for this to be useful, the process should be sound – agents should not be able to infer something that is not true in the TSNM.

Following is the formal definition of Cl_T and the related notion of timed derivation.

Definition 8 (Timed Closure of a Knowledge Base)**.** Given the knowledge base of an agent a, KB_a, the *timed closure of* KB_a, $Cl_T(KB_a)$, satisfies the following properties:

1. For all $\varphi \in \mathcal{F}_{\mathcal{TKBL}}$ and $t \in \mathbb{T}$, if $(\varphi, t) \in Cl_T(KB_a)$ then $(\neg\varphi, t) \notin Cl_T(KB_a)$.
2. Introduction and elimination rules for conjunction:

   $\wedge_I$ - If $(\varphi, t) \in Cl_T(KB_a)$ and $(\psi, t') \in Cl_T(KB_a)$, then $(\varphi \wedge \psi, \max(t, t')) \in Cl_T(KB_a)$.

$\wedge_{E_1}$ - If $(\varphi \wedge \psi, t) \in Cl_T(KB_a)$ and $\wedge_I$ was not used to derive $\varphi \wedge \psi$, then $(\varphi, t) \in Cl_T(KB_a)$.

$\wedge_{E_2}$ - Analogous to $\wedge_{E_1}$ but for ψ.

3. If $(\varphi, t) \in Cl_T(KB_a)$ and $(\varphi \implies \psi, t') \in Cl_T(KB_a)$, then $(\psi, \max(t, t')) \in Cl_T(KB_a)$.

4. If $(\varphi, t) \in Cl_T(KB_a)$ then $(K_a\varphi, t) \in Cl_T(KB_a)$.

5. If φ is provable in the axiomatization **S5** ([11]) from $Cl_T(KB_a)$, then $\varphi \in Cl_T(KB_a)$. Formally:

A1 - If φ is an instance of a first-order tautology, then $(\varphi, t^\top) \in Cl_T(KB_a)$.

A2 - If $(K_a\varphi, t) \in Cl_T(KB_a)$ and $(K_a(\varphi \implies \psi), t') \in Cl_T(KB_a)$, then $(K_a\psi, \max(t, t')) \in Cl_T(KB_a)$.

A3 - If $(K_a\varphi, t) \in Cl_T(KB_a)$, then $(\varphi, t) \in Cl_T(KB_a)$.

A4 - If $(K_a\varphi, t) \in Cl_T(KB_a)$, then $(K_aK_a\varphi, t) \in Cl_T(KB_a)$.

A5 - If $(\varphi, t) \notin Cl_T(KB_a)$, then $(\neg K_a\varphi, t) \in Cl_T(KB_a)$.

R1 - *Modus ponens*, defined as 3).

R2 - If (φ, t) is provable from no assumptions (i.e., φ is a tautology), then $(K_a\varphi, t) \in Cl_T(KB_a)$.

C1 - $(E_G\varphi, t) \in Cl_T(KB_a)$ iff $(\bigwedge_{i \in G} K_i\varphi, t) \in Cl_T(KB_a)$.

C2 - $(C_G\varphi, t) \in Cl_T(KB_a)$ iff $(E_G(\varphi \wedge C_G\varphi), t) \in Cl_T(KB_a)$.

RC1 - If $(\varphi \implies E_G(\psi \wedge \varphi), t)$ is provable from no assumptions, then $(\varphi \implies C_G\psi, t) \in Cl_T(KB_a)$.

D1 - $(D_{\{a\}}\varphi, t) \in Cl_T(KB_a)$ iff $(K_a\varphi, t) \in Cl_T(KB_a)$.

D2 - If $(D_G\varphi, t) \in Cl_T(KB_a)$, then $(D_{G'}\varphi, t) \in Cl_T(KB_a)$ if $G \subseteq G'$.

DA2-DA5 Properties A2, A3, A4 and A5, replacing the modality K_a with the modality D_G for each axiom.

Definition 9 (Timed Derivation)**.** A *timed derivation* of a formula $\varphi \in \mathcal{F}_{\mathcal{TKBL}}$ with timestamp $t \in \mathbb{T}$ is a finite sequence of formulae and timestamps

$$(\varphi_1, t_1), (\varphi_2, t_2), \ldots, (\varphi_k, t_k) = (\varphi, t),$$

where each φ_i, for $1 \leq i \leq k$, is either an instance of the axioms or the conclusion of one of the derivation rules of which premises has already been derived, i.e., it appears as φ_j with $j < i$ of $Cl_T(KB_a)$.

The previous definitions build on those defined for the original $\mathcal{FPPF}$. The main difference is the handling of timestamps in knowledge bases of agents. More precisely, we want to ensure that a timestamp attached to a formula in a KB has the intended meaning, i.e., it should be the time when that particular information was either learned or inferred.

Combining Knowledge For instance, we allow agents to combine individual pieces of information using the conjuction introduction rule $\wedge_I$. The timestamp of the resulting conjunction is the maximum of the timestamps of the individual formulae. The reasoning behind this is that one can claim they know both φ and ψ only once they obtain both φ and ψ – more precisely, at the point in time when the last missing piece was obtained. This is also the case in the *modus ponens* rule in 3: the resulting knowledge could only be inferred at the point when the agent is aware of both the precondition and the rule itself.

An agent is also capable of breaking a conjunction $\varphi \wedge \psi$ down into its two conjuncts φ and ψ. This is, however, only possible if it was not obtained using the introduction rule. The reason for this is to avoid obtaining knowledge with incorrect timestamps. If the restriction were not in place and we attempted to break down a conjunction obtained by $\wedge_I$, we could end up with either φ or ψ with an illegal timestamp.

Tautologies Tautologies are a special case as they should be true at any point in time. To reflect this, we use a special timestamp $t^\top$ which stands for any timestamp (for all $t \in \mathbb{T}$, $t^\top = t$). This guarantees that, when using a tautology in a derivation including other premises, we will delegate the timestamp t of the premise that is not a tautology since $\max(t, t^\top) = t$.

Knowledge Evolution It should be noted that in our model, the knowledge of agents grows monotonically – there is no notion of forgetting and the agents remember everything they have learned so far. We do not provide a formal proof here, but it follows from examining Def. 8 case by case as none of the properties results in inferred knowledge with a timestamp less than those of the premises used. Neither is it the case that inferred knowledge would get a timestamp strictly greater than the maximum of its premises.

4.3 Semantics of $\mathcal{TKBL}_{\mathcal{SN}}$

Now that we have defined both the syntax of $\mathcal{TKBL}_{\mathcal{SN}}$ and the notion of a trace, we are ready to introduce a precise way to determine whether a formula holds in a trace.

Definition 10 (Satisfiability Relation for $\mathcal{TKBL}_{\mathcal{SN}}$)**.** Given a well-formed trace $\sigma \in TCS$, agents $a, b \in Ag_\sigma$, a finite set of agents $G \subseteq Ag$, formulae $\varphi, \psi \in \mathcal{F}_{\mathcal{TKBL}}$, $m \in \mathcal{C}$, $n \in \Sigma$, $o \in \mathcal{D}$, and $t \in \mathbb{T}_\sigma$, the *satisfiability relation* $\models$ is defined as shown in Fig. 4.2.

In the semantics, we use a timestamp t to help determine which TSNM in the trace we are interested in. For the temporal operators $\Box$ and $\Diamond$, which are used to manipulate t, this simply means that we have to check either all TSNMs starting with t, or we have to find at least one at time t or greater for which the inner formula holds.

The cases of negation, conjunction, and quantification are dealt with in a standard way. For connections and actions, we simply look into the relational structure

$$
\begin{array}{lll}
\sigma, t \models \Box\varphi & \text{iff} & \text{for all } t' \in \mathbb{T}_\sigma,\ t' \geq t,\ \sigma, t' \models \varphi \\
\sigma, t \models \Diamond\varphi & \text{iff} & \text{there exists } t' \in \mathbb{T}_\sigma,\ t' \geq t, \\
 & & \text{such that } \sigma, t' \models \varphi \\
 & & \\
\sigma, t \models \neg\varphi & \text{iff} & \sigma, t \not\models \varphi \\
\sigma, t \models \varphi \wedge \psi & \text{iff} & \sigma, t \models \varphi \text{ and } \sigma, t \models \psi \\
\sigma, t \models \forall x.\varphi & \text{iff} & \text{for all } v \in D_o^{\sigma[t]},\ \sigma, t \models \varphi[v/x] \\
 & & \\
\sigma, t \models c_m(a, b) & \text{iff} & (a, b) \in C_m^{\sigma[t]} \\
\sigma, t \models a_n(a, b) & \text{iff} & (a, b) \in A_n^{\sigma[t]} \\
 & & \\
\sigma, t \models p(\vec{s}) & \text{iff} & \text{there exists } t' \in \mathbb{T}_\sigma \text{ such that } (p(\vec{s}), t') \in KB_e^{\sigma[t]} \\
 & & \\
\sigma, t \models K_a\varphi & \text{iff} & \text{there exists } t' \in \mathbb{T}_\sigma \text{ such that } (\varphi, t') \in Cl_T(KB_a^{\sigma[t]}) \\
\sigma, t \models L_a\varphi & \text{iff} & (\varphi, t) \in Cl_T(KB_a^{\sigma[t]}) \\
 & & \\
\sigma, t \models C_G\varphi & \text{iff} & \sigma, t \models E_G^k\varphi \text{ for } k = 1, 2, \ldots \\
\sigma, t \models D_G\varphi & \text{iff} & \text{there exists } t' \in \mathbb{T}_\sigma \text{ such that } (\varphi, t') \in Cl_T(\bigcup_{i \in G} KB_i^{\sigma[t]})
\end{array}
$$

Figure 4.2: *The semantics for $\mathcal{TKBL}_{\mathcal{SN}}$ is given in terms of the satisfiability relation.*

of the TSNM at time t to determine whether the agents in question were in the relation at that point.

Predicates are checked with respect to the environment, which contains everything that is true in the TSNM. A predicate is considered to hold at time t if the knowledge base of the environment at time t contains said predicate with any timestamp (note that since σ is well-formed, only timestamps less than t and t itself may exist there). The knowledge modality K is treated in essentially the same way, with the only exception being that we look into the timed closure of the knowledge of the agent in question. Learning φ at time t in terms of the semantics given here means that φ with timestamp t has to be in the timed closure of the knowledge base at time t – in other words, φ has to have appeared in the closure at precisely t.

The semantics given for common knowledge are defined using the E shortcut modality (Sec. 4.1.1). For φ to be distributed knowledge among agents of G at time t it has to be the case that φ appears with some timestamp (again, by definition this can never be more than t) in the closure of the collective knowledge bases of all agents in G at time t.

4.4 Properties of $\mathcal{TKBL}_{\mathcal{SN}}$

As mentioned before, while L is used to represent the time instant in which someone learns a piece of information, K is the lasting effect of learning something. The relationship between the two operators is demonstrated by the following two properties.

Given an agent a and a $\mathcal{TKBL}_{\mathcal{SN}}$ formula φ, we can formulate the following as

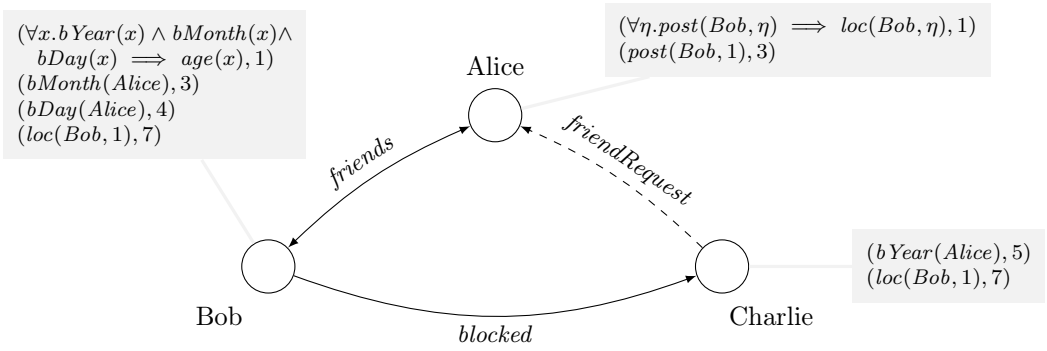

Figure 4.3: *We revisit the previous Example 1 by making the knowledge bases of agents explicit. In the picture, these are depicted as grey rectangles connected to the node representing the owner agent. Note that we use simple dummy timestamps to simplify the picture (originally, timestamp $t = 1$ would mean January 1, 1970, 00:00:00.001).*

the *learning axiom*:

$$L_a\varphi \implies K_a\varphi \tag{L}$$

The intuition behind L is that in order for the premise to be true in any trace $\sigma \in TCS$ at time $t \in \mathbb{T}_\sigma$, it has to be the case that $(\varphi, t) \in Cl_T(KB_a^{\sigma[t]})$. But then the conclusion is trivially satisfied by the same (φ, t) being in a's knowledge base closure.

However, the converse does not hold. It is often the case that $(\psi, t') \in Cl_T(KB_b^{\sigma[t]})$ and $t' < t$, in which case ψ is simply knowledge obtained in the past (t'). Then by definition $\sigma, t \models K_b\psi$ since ψ is in b's knowledge base closure with any timestamp, but at the same time $\sigma, t \not\models L_b\psi$ since $t' < t$.

Moreover, the following property, henceforth called the *perfect recall axiom*, expresses the relationship between knowledge and time in $\mathcal{TKBL}_{\mathcal{SN}}$:

$$K_a\varphi \implies \Box K_a\varphi \tag{PR}$$

In other words, if an agent knows something, then they will always know it. As mentioned before, knowledge here is monotonic – agents cannot forget anything they have learned. Once (φ, t) is in an agent's knowledge base closure, there is no way to remove it in the future since Cl_T only adds new knowledge and we have not formalized any notion of taking away knowledge, or forgetting, so the same (φ, t) will still be there at any point in the future, thus satisfying the consequence of PR.

By combining L and PR, we also have the property that once an agent learns something, they will always know it:

$$L_a\varphi \implies \Box K_a\varphi$$

4.5 Examples

Example 5. At this point we can revisit the previous example (Ex. 1). Figure 4.3 contains the same set of agents, connections and permissions as before, but

now the knowledge bases have been made explicit. Each rectangle represents the accumulated knowledge of an agent.

The knowledge bases of both Bob and Charlie contain the timestamped formula $(loc(Bob, 1), 7)$, which means that they both learned Bob's location 1 at time 7. We use the second argument of *loc*, 1, as a resource identifier to be able to tie together related information. This is used for example in the first formula in Alice's knowledge base, which says that she can derive the location of Bob if she can access his post, where the $loc(Bob, \eta)$ and $post(Bob, \eta)$ have the same identifier, meaning that Bob's location is attached to his post in some way. Since Alice learned $post(Bob, 1)$ at time 3, she can use this rule to derive the timestamped formula $(loc(Bob, 1), 3)$. Note that according to the closure definition (Def. 8), the timestamp of the new piece of information is the maximum out of the two formulae used to derive it (1 and 3).

Agents can also combine their knowledge. For instance, if $G = \{Bob, Charlie\}$, then $D_G age(Alice)$ holds at the time of this particular TSNM. Bob knows Alice's day and month of birth, Charlie knows the year Alice was born. Moreover, Bob knows that once he knows someone's day, month, and year of birth, he can infer the age of the person in question. And so, if Bob and Charlie combined their knowledge at time 5 (since that is when the last "piece of the puzzle" was obtained) or later, they would be able to find out Alice's age.

Example 6. To demonstrate how the satisfiability relation works, consider an OSN with at least two events: *checkIn*, which discloses a user's location to all their friends, and *openFeed*, which retrieves all information a user has access to, including locations. Let us assume *TSN* is the TSNM from Fig. 4.3 and it represents the OSN at time 7. Afterwards, it undergoes the following evolution:

$$TSN \xrightarrow{\{checkIn(Alice)\},\, 8} TSN' \xrightarrow{\{openFeed(Bob)\},\, 9} TSN''$$

In other words, Alice makes her location public to her friends at time 8 and then Bob opens his news feed at time 9. Using the more commonly used notation, we can write

$$\sigma = \langle (TSN, 7), (TSN', 8), (TSN'', 9) \rangle.$$

We can use the satisfiability relation (Def. 10) to determine whether Bob learns Alice's location after she discloses it to her friends. More precisely, we are interested in whether it is the case that

$$\sigma, 7 \models \Box(friends(Alice, Bob) \wedge checkIn(Alice, 7) \implies \exists x. \Diamond L_{Bob} loc(Alice, x)).$$

According to the definition, in order for $\Box\varphi$ to hold, φ has to hold in all TSNMs that are not older than the guiding timestamp on the left-hand side, which in our case is 7. Therefore, we must in fact check every TSNM in σ.

As for the premise, the predicate $friends(Alice, Bob)$ was true in *TSN* and since no *unfriend* event took place, it is the case that $(Alice, Bob) \in A^{\sigma[t]}_{friends}$ for all $t \geq 7$, that is, Alice and Bob are friends in all three TSNMs in σ – at time 7, 8 and 9. The predicate $checkIn(Alice, 7)$ in this case represents that Alice executed the *checkIn* event after time 7. Since according to the transition relation the *checkIn*

event was executed at time 8, the associated predicate is true at time 8 and 9: $checkIn(Alice, 7) \in KB_e^{\sigma[t]}$ for $t \geq 8$.

Moving over to the right-hand side of the implication, $\Diamond\psi$ requires there to be at least one TSNM not older than the guiding timestamp such that ψ holds. In our case, there has to be some TSNM in which a timestamped $loc(Alice, x)$ for some x is in $Cl_T(KB_{Bob}^{\sigma[t]})$ for some t satisfying the above condition. In this case Bob does not need to infer any knowledge: once he opens his news feed at time 9, he will see Alice's location, so $(loc(Alice, 1), 9)$ (where 1 is the resource identifier of the location) will appear in $KB_{Bob}^{\sigma[9]}$.

To conclude, since the premise holds at time 8 and 9 and in both cases, Bob eventually learns Alice's location at 9, the property holds.

Chapter 5

Timed Privacy Policy Language ($\mathcal{TPPL}_{\mathcal{SN}}$)

One of the main goals of $\mathcal{TFPPF}$ is to equip users of OSNs with additional power when defining their privacy policies. Building on $\mathcal{FPPF}$, this is done by extending the original privacy policy language $\mathcal{PPL}_{\mathcal{SN}}$ with time fields, which enable the user to specify a (possibly recurring) real-time window in which their policy should be enforced. The resulting language is called timed privacy policy language ($\mathcal{TPPL}_{\mathcal{SN}}$).

In the following sections, we first describe how to capture a specific time window (or windows) using $\mathcal{TPPL}_{\mathcal{SN}}$ (Sec. 5.1), followed by the formal definition, semantics and properties of $\mathcal{TPPL}_{\mathcal{SN}}$ (Sec. 5.2, 5.3, 5.4). The remaining Sec. 5.5 is devoted to examples of policies written in $\mathcal{TPPL}_{\mathcal{SN}}$ and their properties and consequences.

5.1 Privacy Policies in Real Time

Our aim is to provide a way to enforce a privacy policy in a specific real-time window. Additionally, we also want to be able to repeat this window after a chosen time period has passed. To this end $\mathcal{TPPL}_{\mathcal{SN}}$ offers a total of three time fields (Fig. 5.1), called *start*, *duration*, and *recurrence*.

Starting Point The first field, *start*, is a timestamp (Def. 2). It is the only compulsory field out of the three. If a policy only uses the *start* field, it is meant to be enforced from the point represented by its value forward. Such policies (which only have a *start*) are the $\mathcal{TPPL}_{\mathcal{SN}}$ version of the original $\mathcal{PPL}_{\mathcal{SN}}$ policies.

Window Duration The second field is *duration*. Unlike the *start* field, it does not contain a timestamp, but a slightly different notion with the same name as that

$$\begin{array}{ll} [\![\neg\alpha]\!]_a^{[\text{ start }]} & [\![\varphi \implies \neg\alpha]\!]_a^{[\text{ start }]} \\ [\![\neg\alpha]\!]_a^{[\text{ start } \mid \text{ duration }]} & [\![\varphi \implies \neg\alpha]\!]_a^{[\text{ start } \mid \text{ duration }]} \\ [\![\neg\alpha]\!]_a^{[\text{ start } \mid \text{ duration } \mid \text{ recurrence }]} & [\![\varphi \implies \neg\alpha]\!]_a^{[\text{ start } \mid \text{ duration } \mid \text{ recurrence }]} \end{array}$$

Figure 5.1: *Basic forms of privacy policies one can define in $\mathcal{TPPL}_{\mathcal{SN}}$. There are two basic shapes for the inner content and three possible time fields.*

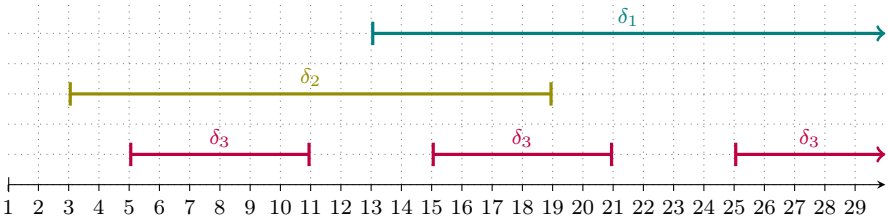

Figure 5.2: *Consider three policies $\delta_1, \delta_2, \delta_3$, each with as many time fields defined as its index. If δ_1 only has the start field defined and set to 13, it will hold from that point forward, assuming the owner user does not deactivate it altogether. If δ_2 has a start and a duration, it will be enforced in one fixed time window. In this case, the respective fields are 3 and 16, making the starting time of δ_2 3 and the ending time $3 + 16 = 19$. In case of δ_3, we also have the recurrence field defined. In this case, the values of the fields are, in order, 5, 6, and 10, meaning that the starting points will be $5, 5+10, 5+2*10, \ldots$, while the ending points will be $5+6, 5+10+6, 5+2*10+6, \ldots$.*

Table 5.1: *Given the time fields start, duration, recurrence, a privacy policy should be enforced in intervals $0, 1, \ldots$, specified by the table. If the recurrence field is not defined, then a policy should be only enforced in interval 0. This table is not applicable when only the start field has been defined – in that case, there is no end of the time frame.*

Interval	Start	End
0	start	start + duration
1	start + recurrence	start + recurrence + duration
2	start + 2 recurrence	start + 2 recurrence + duration
3	start + 3 recurrence	start + 3 recurrence + duration
⋮	⋮	⋮
i	start + i recurrence	start + i recurrence + duration

of the field itself: duration (Def. 11). A duration simply stands for the amount of time between two points in time. If a policy is defined with both a *start* field and a *duration* field, but the last field is missing, it is meant to be enforced in the time window starting at *start* and ending at *start* + *duration*.

Recurrence Offset The last field is *recurrence* and it also uses the duration format. It can only be defined if both the *start* and the *duration* fields have been defined. The *recurrence* field is essentially the offset between the starting point of each pair of adjacent time windows. A policy with all three fields is meant to be enforced first in the window from *start* to *start* + *duration*. The second window starts at *start* + *recurrence* and ends at *start* + *recurrence* + *duration*. The third window is offset by twice the value of the *recurrence* field, and so on (Table 5.1).

For a more intuitive explanation of how the three fields work together, see Table 5.1 and Fig. 5.2. We also provide a simple example (Ex. 7) at the end of this section.

Let us now formalize the aforementioned notion of *durations*. The definition

provided below is very similar to the definition of timestamps (Def. 2) – it is also essentially just a natural number representing a number of milliseconds. In this case, however, it stands for the absolute difference of two timestamps $|t_2 - t_1|$, i.e., the time elapsed between t_1 and t_2.

Definition 11 (Duration)**.** A *duration* d is a positive natural number representing the number of milliseconds elapsed between two points in time.

As was the case of timestamps, here, too, we will use a more human-readable format for durations. For instance, $d = 60000$ will be *1 minute*, $d = 2167236000$ will be *25 days, 2 hours and 36 seconds*, and so on. To avoid unnecessary ambiguity, we will avoid referring to months and years, since their length varies.[1]

Example 7. A privacy policy defined with the time field

$$[\ \text{2016-02-02 14:00}\ ]$$

is meant to be enforced from February 2, 2016, 14:00 onwards. If the fields defined are

$$[\ \text{2016-02-02 14:00} \mid \text{6 hours}\ ],$$

it gives the policy an ending point – now it should be enforced on February 2, 2016, from 14:00 to 20:00. And finally,

$$[\ \text{2016-02-02 14:00} \mid \text{6 hours} \mid \text{1 day}\ ]$$

stands for every afternoon from 14:00 to 20:00, starting on February 2, 2016.

5.2 Writing Privacy Policies

With the intuition behind the time fields explained and the notion of durations introduced, we are now ready to define the general shape of formulae used in the timed privacy policy language $\mathcal{TPPL}_{\mathcal{SN}}$. The language is very similar to that of the original $\mathcal{PPL}_{\mathcal{SN}}$, as given in [25].

Definition 12 (Syntax of $\mathcal{TPPL}_{\mathcal{SN}}$)**.** Given agents $a, b \in Ag$, a nonempty set $G \subseteq Ag$, a timestamp s, durations d, r, a variable x, a formula $\varphi \in \mathcal{F}_{\mathcal{TKBL}}$, relation symbols $c_m(a,b), a_n(a,b), p(\vec{s})$ where $m \in \mathcal{C}$ and $n \in \Sigma$, the *syntax of the timed privacy policy language* $\mathcal{TPPL}_{\mathcal{SN}}$ is inductively defined as:

$$\begin{array}{lcl}
\delta & ::= & \delta \wedge \delta \mid \forall x.\delta \mid \tau \\
\tau & ::= & [\![\neg\alpha]\!]_a^{[\ s\ ]} \mid [\![\neg\alpha]\!]_a^{[\ s \mid d\ ]} \mid [\![\neg\alpha]\!]_a^{[\ s \mid d \mid r\ ]} \mid \\
 & & [\![\varphi \implies \neg\alpha]\!]_a^{[\ s\ ]} \mid [\![\varphi \implies \neg\alpha]\!]_a^{[\ s \mid d\ ]} \mid [\![\varphi \implies \neg\alpha]\!]_a^{[\ s \mid d \mid r\ ]} \\
\alpha & ::= & \alpha \wedge \alpha \mid \forall x.\alpha \mid \psi \mid \gamma\prime \\
\gamma\prime & ::= & K_a\gamma \mid L_a\gamma \mid D_G\gamma \mid C_G\gamma \\
\gamma & ::= & \gamma \wedge \gamma \mid \neg\gamma \mid p(\vec{s}) \mid \gamma' \mid \psi \mid \forall x.\gamma \\
\psi & ::= & c_m(a,b) \mid a_n(a,b)
\end{array}$$

Furthermore, let $\mathcal{F}_{\mathcal{TPPL}}$ denote the set of all formulae of $\mathcal{TPPL}_{\mathcal{SN}}$ and $\mathcal{F}^{\mathcal{R}}_{\mathcal{TPPL}}$ the set of all formulae created using the α category (the restrictions).

[1] In general, variable month and year length has a number of consequences from the point of view of practical usability of $\mathcal{TPPL}_{\mathcal{SN}}$; we discuss these in Sec. 5.4.

$$
\begin{array}{lll}
\sigma \models_C \delta_1 \wedge \delta_2 & \text{iff} & \sigma \models_C \delta_1 \wedge \sigma \models_C \delta_2 \\
\sigma \models_C \forall x.\delta & \text{iff} & \text{for all } v \in D_o^{\sigma[t]},\ \sigma \models_C \delta[v/x] \\
 & & \\
\sigma \models_C [\![\neg\alpha]\!]_a^{[\ s\ |\ d\ |\ r\ ]} & \text{iff} & \text{for all } c \in \mathbb{N}^0 \\
 & & \text{such that } 0 \leq s + cr \leq \max(\mathbb{T}_\sigma), \\
 & & \sigma \models_C [\![\neg\alpha]\!]_a^{[\ s\ +\ cr\ |\ d\ ]} \\
\sigma \models_C [\![\neg\alpha]\!]_a^{[\ s\ |\ d\ ]} & \text{iff} & \sigma[s\ ..\ s+d], s \models \Box(\neg\alpha) \\
\sigma \models_C [\![\neg\alpha]\!]_a^{[\ s\ ]} & \text{iff} & \sigma[s\ ..\ ], s \models \Box(\neg\alpha) \\
 & & \\
\sigma \models_C [\![\varphi \implies \neg\alpha]\!]_a^{[\ s\ |\ d\ |\ r\ ]} & \text{iff} & \text{for all } c \in \mathbb{N}^0 \\
 & & \text{such that } 0 \leq s + cr \leq \max(\mathbb{T}_\sigma), \\
 & & \sigma \models_C [\![\varphi \implies \neg\alpha]\!]_a^{[\ s\ +\ cr\ |\ d\ ]} \\
\sigma \models_C [\![\varphi \implies \neg\alpha]\!]_a^{[\ s\ |\ d\ ]} & \text{iff} & \sigma[s\ ..\ s+d], s \models \Box(\varphi \implies \neg\alpha) \\
\sigma \models_C [\![\varphi \implies \neg\alpha]\!]_a^{[\ s\ ]} & \text{iff} & \sigma[s\ ..\ ], s \models \Box(\varphi \implies \neg\alpha)
\end{array}
$$

Figure 5.3: *The semantics of the timed privacy policy language $\mathcal{TPPL}_{\mathcal{SN}}$ is given in terms of the conformance relation $\models_C$. We use $\mathbb{N}^0$ to denote the set of all natural numbers and zero. Both the direct and the conditional restriction policy form have two base cases, based on the time fields defined. These base cases take the relevant subtrace of the original trace and the content of the policy, with $\Box$ attached, is then checked using the satisfiability relation.*

5.3 Checking Privacy Policies

Now that we are able to define policies to be enforced in real-time, we need to be able to determine whether they actually work in an OSN. For this purpose we formalize the semantics of $\mathcal{TPPL}_{\mathcal{SN}}$, which define whether a privacy policy is or is not violated during the evolution of an OSN, encoded in a well-formed trace.

The actual checking process uses notions defined earlier: it combines basic trace operations (Sec. 3.2.3) with the satisfiability relation for $\mathcal{TKBL}_{\mathcal{SN}}$ (Def. 10).

Definition 13 (Conformance Relation)**.** Given a well-formed trace $\sigma \in TCS$, an agent $a \in Ag$, formulae $\varphi \in \mathcal{F}_{\mathcal{TKBL}}, \alpha \in \mathcal{F}^{\mathcal{R}}_{\mathcal{TPPL}}$, and $\delta, \delta_1, \delta_2 \in \mathcal{F}_{\mathcal{TPPL}}$, the conformance relation $\models_C$ is defined as shown in Fig. 5.3.

The conformance relation treats both direct and conditional restrictions in the same way: both are interpreted using the satisfiability relation based on the time fields they have; the shape of the content of the policy is not relevant. The three cases mirror the three possible configurations of policy time fields: by Def. 12, a policy either only has a *start*, or it has a *start* and a *duration*, or a *start*, a *duration* and a *recurrence*.

One-Field Policies This is a base case in the conformance relation. With only the *start* field defined, we take the suffix of the original trace σ starting at *start*: since we are only interested in enforcing the policy from that point forward, we only need to look at those TSNMs in the trace whose timestamps are *start* or greater. We also set the trace position variable to *start*. Finally, we attach the $\Box$ (*always*) temporal

operator to the policy, to make sure that it will be checked in every TSNM of the subtrace, and we delegate the checking process to the satisfiability relation.

Two-Field Policies This is also a base case in the conformance relation. It is very similar to the previous one, the only difference being that we do not delegate a whole suffix of the original trace, but a subtrace starting at *start* and ending at *start* + *duration*. The philosophy behind this is exactly the same as before: we are only interested in the time window between these two time points.

Three-Field Policies This is the only complex form. The intuition behind the reduction to a simpler case is illustrated by Fig. 5.2, which was mentioned before. Essentially, we cut the original trace into the windows specified by the time fields and check each of the windows separately by reducing to a simpler case. This is quite straightforward, since we can easily calculate the starting and ending point of any time window from the three fields, as shown previously by Table 5.1.

Note that even though the counter c (which basically stands for the window number) is a nonnegative integer, we only check a finite number of windows (it would not make much sense otherwise, since the trace itself is finite). There is also one notable corner case: if the last window is not complete (because the trace ends somewhere in it), we just check the part that can be accessed. This is because of the way the subtrace function (Sec. 3.2.3) works when handling timestamps out of range.

5.4 Clarifications

5.4.1 Checking Outside Policy Windows

The behavior and meaning of $\mathcal{TPPL}_{\mathcal{SN}}$ policies might differ from initial expectations. For example, imagine Alice wants to define a policy that says that Bob cannot learn about any of her pictures during the weekend.

Now let us say that on Wednesday, Bob learns about Alice's picture, taken at a party on Saturday. Is this a violation of Alice's policy? Not in terms of the conformance relation. This is because the relation trims away any part of the trace that does not have a weekend timestamp, so the policy is really checked only in the time frame it is defined for. Essentially, it says that Bob is not allowed to learn about Alice's picture during any weekend.

To further clarify what the policy actually means, consider now the opposite scenario: on Saturday, Bob learns about Alice's picture, which was taken on Wednesday. This *is* a violation of Alice's policy, because the conformance (and satisfiability, by extension) relation does not care about the original date of something happening or appearing in the OSN, it only checks whether something was learned at a specific point (e.g. on Saturday).

For similar potentially ambiguous cases and their handling in $\mathcal{TPPL}_{\mathcal{SN}}$, we refer to the examples in Sec. 5.5.

5.4.2 Variable Window Offset

Shortly after defining the notion of durations, we briefly mentioned that we will avoid using months and years in the human-readable format to avoid confusion caused by the variability in their length. Take month lengths, for example: a month can mean any number of days from 28 to 31, depending on a number of factors. In year length, leap years are the cause of ambiguity.

It might seem that this is not actually a problem; we can just use the precise number of days in a specific month or year to translate a duration unambiguously. The issue, however, extends further.

Imagine a user wants to define a policy that should be enforced on the last day of each month. There is no way to achieve this with just one policy with all three time fields: as the *recurrence* field value is used as the offset between the starting point of windows, the duration between each window will be the same and there is no way to offset the second window by 31 days, the next one by 28 days, and so on.

Nevertheless, there is a number of solutions that can be used in scenarios like these. For example, one can define multiple policies (or one longer policy that is a conjunction of several basic policies) with the same content, but different time fields. Since the number of options one needs to account for is finite, the number of policies needed to capture them is finite as well. Alternatively, we could have a notion of months and years that would be automatically translated to the right number of days based on the starting point of the policy window. For example, if we are currently checking a policy in a time window starting on November 5, 2016, and the next window based on the *recurrence* field is supposed to be a month from that point, we could calculate the number of days between November 5 and December 5 and use that as the next starting point.

5.5 Example Privacy Policies

Example 8. On Friday, April 15, 2016, Alice decides that she wants to keep her private life separate from her life as a graduate student. In an OSN with privacy policies using $\mathcal{TPPL}_{\mathcal{SN}}$, she can keep her supervisor Bob from learning her location on weekends by defining the following privacy policy:

$$\delta = [\![\neg L_{Bob}\, location(Alice)]\!]^{[\text{ 2016-04-16 | 2 days | 1 week }]}_{Alice}$$

Given a trace of the OSN Alice and Bob use, δ would be checked first in the subtrace from Saturday 16th, 00:00, to Monday 18th, 00:00, then again from Saturday 23rd, 00:00, to the end of Sunday 24th, and so on.

In order for the trace to be in conformance with δ, in each of these subtraces, the $\mathcal{TKBL}_{\mathcal{SN}}$ formula $\Box\,(\neg L_{Bob}\, location(Alice))$ needs to be satisfied. Based on the satisfiability relation (Def. 10), this is only the case if the formula $\neg L_{Bob}\, location(Alice)$ is satisfied at every point of the subtrace. This, in turn, means that it must not be the case that $L_{Bob}\, location(Alice)$ is satisfied in any of the TSNMs of the subtrace.

To determine whether $L_{Bob}\, location(Alice)$, we consult the closure of Bob's knowledge base to see whether it contains the pair $(location(Alice), t)$, where t is the

timestamp of the TSNM currently being checked. In other words, we have to determine whether Bob learned (either directly, or by inference) $location(Alice)$ at point t. As t is a time in one of the time windows δ is defined for (i.e., a weekend sometime after April 16th), Bob having access to this particular piece of knowledge would be a violation of Alice's policy, since it would mean Bob managed to learn Alice's location on a weekend. Note, however, that Bob learning Alice's weekend location at any point *not* during a weekend is not considered a violation of the policy. This is due to the fact that the policy is not checked outside the time windows it is defined for.

There is also an alternative way of writing the policy using the knowledge modality K. In fact, although it is not always so, in this simple case L and K are interchangeable. Consider the policy

$$\delta' = [\![\neg K_{Bob} location(Alice)]\!]^{[\text{ 2016-04-16 | 2 days | 1 week }]}_{Alice}$$

and the way it is checked with respect to a trace, according to the conformance and satisfiability relations. The process is the same as before – we slice the trace into relevant windows and delegate the actual checking of these subtraces to the sastifiability relation – until the point when we are looking for a certain piece of information in the closure of Bob's knowledge base in every relevant model of the subtrace. While in the previous case we were looking for a "fresh" $location(Alice)$ (that is, one with the same timestamp as that of its model), this time we are interested in any $location(Alice)$ whose timestamp falls anywhere within the start of the time window being checked and the timestamp of the model. In order for such $location(Alice)$ to be in the closure of Bob's knowledge base, he must have learned it sometime in the time frame being checked, which is exactly what the original policy δ meant.

Note that when checking a specific time frame starting at time s, there might be instances of $location(Alice)$ with timestamps older than s in Bob's knowledge base. This, however, does not violate δ' since by Def. 13, we are only checking the subtrace starting at s, and by Def. 10, in order for Bob to know something, that particular piece of knowledge has to have a timestamp greater than or equal to the oldest timestamp of the trace, which in this case is s, so older knowledge is not taken into account.

Example 9. Charlie will start a new one-month job on July 1st, 2016, and she would like to ensure that, during this period, only her friends can learn about her posts when she is at home, and only her colleagues can learn about her posts when she is at work. In $\mathcal{TPPL}_{\mathcal{SN}}$, she can accomplish this by defining two privacy policies δ_1, δ_2:

$$\delta_1 = \forall x.[\![home(Charlie) \wedge \neg friends(Charlie, x) \implies \neg L_x post(Charlie, text)]\!]^{[\text{ 2016-07-01, 31 days }]}_{Charlie}$$

$$\delta_2 = \forall x.[\![working(Charlie) \wedge \neg colleagues(Charlie, x) \implies \neg L_x post(Charlie, text)]\!]^{[\text{ 2016-07-01, 31 days }]}_{Charlie}$$

The predicates $home(Charlie)$ and $working(Charlie)$ are checked by consulting the environment, which contains everything that holds in a specific model. Checking whether δ_1 or δ_2 are violated ultimately boils down to checking whether all TSNMs in the trace, starting at 2016-07-01 00:00 and ending at 2016-08-01 00:00, satisfy the whole formula inside the policy. It should be noted, however, that the time when the post was actually posted is irrelevant; what matters is when the users learn about it. So if Charlie is working and her colleague Daniel somehow gains access to her post that was originally posted when she was at home, it is not a violation of either of the policies.

In Example 8, the learning and knowledge modalities could be swapped without changing the meaning of the policy. This is usually not possible in conditional policies like δ_1 and δ_2. Consider the following variation of a simplified δ_1:

$$\delta_1' = \forall x.[\![\neg friends(Charlie, x) \implies \neg K_x post(Charlie, text)]\!]^{[\text{ 2016-07-01, 31 days }]}_{Charlie}$$

Imagine we want to check δ_1' on a trace with a user Greg whose relationship with Charlie is a little complicated. They first meet and become friends on July 15th, but for some reason Greg unfriends Charlie on July 25th. Both dates fall into the time window of the policy.

Now let us say Greg learns Charlie's location when they are friends, on July 20th. Since agents do not forget anything, it will still be in his knowledge base after he unfriends Charlie. This, however, means that δ_1' will be violated, since at any point after July 25th, someone who is not a friend of Charlie's knows her location with a timestamp greater than July 1st.

If we use the learning modality instead, this will not be a problem. The difference is that in order for δ_1' to be violated, the state of Charlie and Greg not being friends and Greg *knowing* Charlie's location have to happen at the same time (in the same model). If L is used instead of K, the violating condition changes to not being friends and Greg *learning* Charlie's location at the same time.

Example 10. Another example of a privacy policy could be "if we break up, then you can no longer learn about pictures I am tagged in". Let us say it is Frank who wants to enforce this and Eve is his current girlfriend. He is free to write the following in $\mathcal{TPPL}_{\mathcal{SN}}$:

$$\delta = [\![brokenUp(Frank, Eve) \wedge taggedIn(Frank, \eta) \implies \neg L_{Eve} picture(\eta)]\!]^{[\ t\ ]}_{Frank}$$

To rephrase the policy to better capture the meaning of the individual predicates, we could say: "If Frank and Eve are in the state of being broken up according to the environment and Frank is tagged in picture η, then Eve does not learn about the picture η", where η is a unique identifier tying the *taggedIn* and *picture* predicates together.

Here, too, checking whether δ is violated with respect to a trace means checking the contents of the policy in all TSNMs in the trace, starting at time t. So if Eve gains access to a picture Frank is tagged in that is new to her (no matter when it was originally posted) when they are no longer together, Frank's policy will be violated.

Part II

Towards More Variability in Privacy Policies

Chapter 6

Real-Time Social Network Model (RTSNM)

6.1 Real-Time First-Order Privacy Policy Framework $\mathcal{RTFPPF}$

The second part of the thesis is devoted to the real-time first-order privacy policy framework $\mathcal{RTFPPF}$. This framework aims to solve the delimitations of $\mathcal{TFPPF}$, using it as an intermediate step towards a more fine-grained time-sensitive epistemic logic and privacy policy language.

Some of the delimitations of $\mathcal{TFPPF}$ were described in the previous chapters. Essentially, they can be summed up as follows:

- Inflexibility. The policies of $\mathcal{TPPL}_{\mathcal{SN}}$ are only enforced in a fixed time window, though that may not be the desired behavior, as illustrated in Sec. 5.4.1 and 5.5.

- Inability to determine the time when a piece of information was up-to-date, or more precisely, when it appeared in the OSN. The timestamps in the knowledge bases of agents only represent the point of learning. If an agent learns some piece of information from a year ago at the same time as a piece of information that has just been posted, both will be in their knowledge base with a recent timestamp.

- Limited ability to reason about time relative to what happens in the OSN. One cannot write policies that react to events in the OSN, nor is it possible to tie predicates together based on when they were or were not true.

The logic proposed in this part, $\mathcal{RTKBL}_{\mathcal{SN}}$ (Chap. 7), marks the transition from the temporal $\mathcal{TKBL}_{\mathcal{SN}}$ to a logic which contains timestamps as a syntactic component. In $\mathcal{RTKBL}_{\mathcal{SN}}$, timestamps become part of every predicate and knowledge modality, making it possible to distinguish between the point when a piece of information appeared in the OSN and the point when an agent learns it. To progress from the implicit notion of events in $\mathcal{TFPPF}$, they become an explicit part of the OSN trace in $\mathcal{RTFPPF}$. These and other differences will be described in the following chapters.

Formally, we define:

Definition 14 (Real-Time First-Order Privacy Policy Framework)**.** The tuple $\mathcal{RTFPPF} = \langle \mathcal{RTSN}, \mathcal{RTKBL}_{\mathcal{SN}}, \models, \mathcal{RTPPL}_{\mathcal{SN}}, \models_C \rangle$ is a *real-time first-order privacy policy framework* where

- $\mathcal{RTSN}$ is the set of all possible real-time social network models;
- $\mathcal{RTKBL}_{\mathcal{SN}}$ is a real-time knowledge-based logic;
- $\models$ is a satisfiability relation defined for $\mathcal{RTKBL}_{\mathcal{SN}}$;
- $\mathcal{RTPPL}_{\mathcal{SN}}$ is a formal real-time privacy policy language;
- $\models_C$ is a conformance relation defined for $\mathcal{RTPPL}_{\mathcal{SN}}$.

The structure of $\mathcal{RTFPPF}$ is very similar to that of $\mathcal{TFPPF}$. In fact, some of the notions defined in $\mathcal{TFPPF}$, such as timestamps, can and will be reused. To reflect this continuity, the rest of this part of the thesis is organized in the same way as the previous one. There are three chapters, each describing one of the major components (the underlying model in the rest of Chapter 6, the logic to reason about knowledge in Chapter 7, and the privacy policy language in Chapter 8).

Some of the notions in this part will reuse the names from Part I, even in cases where the definition might be different. This is to avoid heavy notation, which could be the case if we decided to use subscripts and superscripts to distinguish between names used in both frameworks. In cases when we want to use a previously established name to refer to the notion from the previous framework, it will be explicitly stated.

6.2 Real-Time Social Network Model RTSNM

The shape of the model used to capture a specific OSN in a given moment in $\mathcal{RTFPPF}$ is the same as in $\mathcal{TFPPF}$. It is a social graph with agents, the relationships between them, and their privacy policies. Each agent, including the environment, has their own knowledge base with timestamped pieces of knowledge. There are two differences compared to TSNM: the shape of formulae in the users' knowledge bases is different, and their privacy policies use a different language.

As mentioned before, we will use the original definition for timestamps (Def. 2). Also, $\mathbb{T}$ stands for the set of all timestamps as before, while $\mathcal{F}_{\mathcal{RTKBL}}$ denotes the set of all formulae of the logic $\mathcal{RTKBL}_{\mathcal{SN}}$ defined in the next chapter.

Definition 15 (Real-Time Social Network Model)**.** Given a set of formulae $\mathcal{F} \subseteq \mathcal{F}_{\mathcal{RTKBL}}$, a set of privacy policies Π, and a finite set of agents $Ag \subseteq \mathcal{AU}$ from a universe $\mathcal{AU}$, a *real-time social network model* (RTSNM) is a social graph $\langle Ag, \mathcal{A}, KB, \pi \rangle$, where

- Ag is a nonempty finite set of nodes representing the agents in the social network;
- $\mathcal{A}$ is a first-order relational structure over the RTSNM, consisting of a set of domains $\{D_o\}_{o \in \mathcal{D}}$, where $\mathcal{D}$ is an index set, and a set of relation symbols, function symbols and constant symbols interpreted over a domain;

- $KB : Ag \to 2^{\mathcal{F} \times \mathbb{T}}$ is a function retrieving the set of accumulated knowledge, each piece with an associated timestamp, for each agent, stored in the knowledge base of the agent; we write KB_i for $KB(i)$;
- $\pi : Ag \to 2^{\Pi}$ is a function returning the set of privacy policies of each agent; we write π_i for $\pi(i)$.

We will use $\mathcal{RTSN}$ to denote the set of all RTSNMs.

6.2.1 Evolving OSNs

In $\mathcal{RTFPPF}$, we extend the traces of $\mathcal{TFPPF}$ with *events* from a universe EVT. What these events look like and how they change the structure of the RTSNM – the transition relation – depends on the OSN being modeled. A *tweet* event could model a tweet being published on Twitter, for example [25]. The meaning of the events is the same as previously (Sec. 3.2.2) – they transform one RTSNM to another.

Definition 16 (Trace)**.** Given $k \in \mathbb{N}$, a trace σ is a finite sequence

$$\sigma = \langle (RTSN_0, E_0, t_0), (RTSN_1, E_1, t_1), \ldots, (RTSN_k, E_k, t_k) \rangle$$

such that, for all $0 \geq i \geq k$, $RTSN_i \in \mathcal{RTSN}$, $E_i \subseteq EVT$, and $t_i \in \mathbb{T}$.

As before, given a trace σ, we define $\mathbb{T}_\sigma = \{t \mid (RTSN, E, t) \in \sigma\}$.

The conditions a trace must fullfil in order to be meaningful (*well-formed*) closely follow the ones established before for $\mathcal{TFPPF}$ (Def. 5), with two notable differences.

Accounting for Events The definition has to account for events being explicit in the trace. We use an updated version of the transition relation described before (Sec. 3.2.2), this time as $\to \subseteq \mathcal{RTSN} \times 2^{EVT} \times \mathbb{T} \times \mathcal{RTSN}$. We have $\langle RTSN_1, E, t, RTSN_2 \rangle \in \to$ if $RTSN_2$ is the result of the set of events $E \in EVT$ happening in $RTSN_1$ at time t. Note that we allow E to be empty, in which case $RTSN_2 = RTSN_1$. Again, we will use the more compact notation of

$$RTSN_1 \xrightarrow{E,t} RTSN_2$$

where appropriate.

Traces without Gaps We require that there be a RTSNM in the trace for every time instant between the beginning of the trace and its end. Formally:

Definition 17 (Complete Trace)**.** Let σ be a trace. σ is *complete* if for all $t \in \mathbb{T}$ such that $\min(\mathbb{T}_\sigma) \leq t \leq \max(\mathbb{T}_\sigma)$, $t \in \mathbb{T}_\sigma$.

Example 11. Suppose we have the following trace

$$\begin{aligned} \sigma = \langle & (RTSN_0,\ E_0,\ \text{2016-05-27 00:00:00.000}), \\ & (RTSN_1,\ E_1,\ \text{2016-05-27 00:00:00.001}), \\ & (RTSN_2,\ E_2,\ \text{2016-05-27 00:00:00.003}) \rangle. \end{aligned}$$

σ is missing at least one entry to be complete, since there is one valid timestamp in the range from 2016-05-27 00:00:00.000 to 2016-05-27 00:00:00.003 that is part of $\mathbb{T}$ but not $\mathbb{T}_\sigma$, and that is 2016-05-27 00:00:00.002.

Using trace completeness as a prerequisite, we can now provide a formal definition of well-formed RTSNM traces.

Definition 18 (Well-Formed Trace)**.** Let

$$\sigma = \langle (RTSN_0, E_0, t_0), (RTSN_1, E_1, t_1), \ldots, (RTSN_k, E_k, t_k) \rangle$$

be a complete trace. σ is *well-formed* if the following conditions hold:

1. For any i, j such that $0 \geq i, j \geq k$ and $i < j$, it is the case that $t_i < t_j$.
2. Let KB^{RTSN} denote the knowledge base function of model $RTSN$, and similarly for Ag^{RTSN}. For all $(RTSN, t) \in \sigma$, for all $a \in Ag^{RTSN}$, for all $(\varphi, t_\varphi) \in KB_a^{RTSN}$, it is the case that $t_\varphi \leq t$.
3. For all i such that $0 \leq i \leq k-1$, it is the case that $RTSN_i \xrightarrow{E_{i+1},\, t_{i+1}} RTSN_{i+1}$.

Again, we will use TCS to refer to the set of all well-formed $\mathcal{RTFPPF}$ traces.

6.2.2 Notation

Accessing a specific RTSNM $RTSN$ in a well-formed trace is done using the same notation as before: $\sigma[t]$ for a timestamp $t \in \mathbb{T}$. If $t \in \mathbb{T}_\sigma$, then the result is the RTSNM $RTSN \in \mathcal{RTSN}$ belonging to the tuple $(RTSN, E, t') \in \sigma$ for which $t' = t$, as previously.

There are two corner cases arising from the fact that this time we do not require that $t \in \mathbb{T}_\sigma$: t can be out of bounds if it represents a time instant before or after every RTSNM in the trace. If $t > t'$ for all $t' \in \mathbb{T}_\sigma$, we define $\sigma[t]$ to be the very last RTSNM in σ. Conversely, if $t < t'$ for all $t' \in \mathbb{T}_\sigma$, then $\sigma[t]$ will be the very first RTSNM in σ. A demonstration of how the indexing function works can be found in Example 12.

Given a specific RTSNM $RTSN$, we will write Ag^{RTSN}, $\mathcal{A}^{RTSN}$, KB^{RTSN}, Π^{RTSN} to access $RTSN$'s agent set, relational structure, knowledge base, and privacy policy function, respectively.

Example 12. We revisit and expand Example 11 to illustrate how the new indexing function works. Suppose we have the following well-formed $\sigma \in TCS$:

$$\begin{aligned}\sigma = \langle &(RTSN_0,\ E_0,\ \text{2016-05-27 00:00:00.000}),\\ &(RTSN_1,\ E_1,\ \text{2016-05-27 00:00:00.001}),\\ &(RTSN_2,\ \emptyset,\ \text{2016-05-27 00:00:00.002}),\\ &(RTSN_3,\ E_3,\ \text{2016-05-27 00:00:00.003}),\\ &(RTSN_4,\ E_4,\ \text{2016-05-27 00:00:00.004})\rangle.\end{aligned}$$

Given $t \in \mathbb{T}_\sigma$, the indexing function returns the corresponding RTSNM, as before.

$$\sigma[\text{2016-05-27 00:00:00.002}] = RTSN_2$$
$$\sigma[\text{2016-05-27 00:00:00.004}] = RTSN_4$$

Given a timestamp $t \notin \mathbb{T}_\sigma$, it returns the closest RTSNM in the trace.

$$\sigma[\text{2006-04-30 20:42}] = RTSN_0$$
$$\sigma[\text{2016-05-26 19:58}] = RTSN_0$$
$$\sigma[\text{2016-05-27 00:00:00.005}] = RTSN_4$$
$$\sigma[\text{2016-05-28 00:47}] = RTSN_4$$

Real-Time Knowledge-Based Logic ($\mathcal{RTKBL}_{\mathcal{SN}}$)

While $\mathcal{TKBL}_{\mathcal{SN}}$ utilized the temporal operators $\Box$ and $\Diamond$ to capture the dynamic character of the framework, $\mathcal{RTKBL}_{\mathcal{SN}}$ promotes timestamps themselves to a syntactic component of the language. Since both $\Box$ and $\Diamond$ are derivable in $\mathcal{RTKBL}_{\mathcal{SN}}$, these operators are not present in the basic syntax of $\mathcal{RTKBL}_{\mathcal{SN}}$.

This transition from temporal operators to timestamps being part of the syntax is the main difference between the temporal $\mathcal{TKBL}_{\mathcal{SN}}$ and the real-time $\mathcal{RTKBL}_{\mathcal{SN}}$.

7.1 Syntax of $\mathcal{RTKBL}_{\mathcal{SN}}$

We reuse the standard definition of terms from before (Def. 6) to build the syntax of the new logic, where each of the basic building blocks contains a timestamp.

Definition 19 (Syntax of $\mathcal{RTKBL}_{\mathcal{SN}}$)**.** Given agents $a, b \in Ag$, a nonempty set of agents $G \subseteq Ag$, a timestamp t, an event $e \in EVT$, a variable x, predicate symbols $a_n(a, b, t), c_m(a, b, t), p(\vec{s}, t)$ where $m \in \mathcal{C}$ and $n \in \Sigma$, the *syntax of the real-time knowledge-based logic* $\mathcal{RTKBL}_{\mathcal{SN}}$ is inductively defined as:

$$\varphi ::= \rho \mid \varphi \wedge \varphi \mid \neg\varphi \mid \forall x.\varphi \mid K_a^t\varphi \mid C_G^t\varphi \mid D_G^t\varphi$$
$$\rho ::= c_m(a, b, t) \mid a_n(a, b, t) \mid p(\vec{s}, t) \mid \mathit{happened}(e, t)$$

We will use $\mathcal{F}_{\mathcal{RTKBL}}$ to denote the set of all well-formed $\mathcal{RTKBL}_{\mathcal{SN}}$ formulae.

As the definition suggests, apart from the now absent temporal operators mentioned earlier, the syntax introduces a number of new notions.

Timestamped Predicates Timestamps are explicitly part of each predicate, including connections and actions. The meaning of the timestamp attached to a predicate is that it should capture the moment where that particular predicate was true. In this sense, timestamped predicates can be seen as functions of time. For instance, if Alice and Bob were friends in a certain time period, then the predicate $\mathit{friends}(\mathit{Alice}, \mathit{Bob}, t)$ should be true for all t falling into the period, and false for all t outside.

Timestamped Knowledge Modalities Timestamps are now also part of the knowledge modalities K, C, D. These timestamps represent the moment in time when a

particular piece of knowledge was obtained. For example, the meaning of the formula $K^t_{Alice} friends(Bob, Charlie, t')$ is that Alice learns at time t that Charlie was friends with Bob at time t'.

It should also be noted that $\mathcal{RTKBL}_{\mathcal{SN}}$ does not include separate modalities for learning and knowledge, as was the case in $\mathcal{TKBL}_{\mathcal{SN}}$. Semantically, here the K modality stands for learning: it is true if the agent in question acquires a certain piece of information at the moment given by the timestamp (Def. 21). Knowledge in the sense of the K of $\mathcal{TFPPF}$, i.e., simply having a piece of information in the knowledge base closure, can be derived from the K modality of $\mathcal{RTFPPF}$ thanks to the ability to quantify over timestamps – we can use this to translate the meta language quantification in Def. 10 to quantification on the syntactic level of $\mathcal{RTKBL}_{\mathcal{SN}}$. Semantically, evaluating $K_a\varphi$ (for the $\mathcal{TKBL}_{\mathcal{SN}}$ K) at time t in a trace has the same interpretation as the $\mathcal{RTKBL}_{\mathcal{SN}}$ formula $\exists t'.t' \leq t \wedge K^{t'}_a \varphi$, which can be read as "there is a time in the past when a learned φ".

Using timestamps in both the knowledge modalities and predicates enables us to make a crucial distinction impossible in $\mathcal{TFPPF}$: we can now separate and capture the time of a predicate being valid in the OSN and the time when an agent learns it. This allows for more fine-grained privacy policies (Chapter 8).

Special Event Predicate Now that events are explicitly part of any $\mathcal{RTFPPF}$ trace, the $happened(e, t)$ predicate has been introduced to be able to syntactically capture the moment when a specific event occurred. This makes it possible to reason about time relative to the time of the event, enabling the user to define policies such as "if someone unfriends me, they are not allowed to send me a friend request". Note that now that events are explicit in the trace, they need not be included in the KB of the environment.

7.1.1 Notation

Given agents $a, b \in Ag$, a nonempty group $G \subseteq Ag$, and an action a_n, we define, as before in $\mathcal{TFPPF}$ (Sec. 7.1.1):

$$\begin{array}{rcl} P^b_a a_n & := & a_n(a, b) \\ SP^b_G a_n & := & \bigvee_{a \in G} a_n(a, b) \\ GP^b_G a_n & := & \bigwedge_{a \in G} a_n(a, b) \end{array}$$

Again, these are read: "a is permitted to execute a_n to b", "someone in G is permitted to execute a_n to b", and "everyone in G is permitted to execute a_n to b".

Furthermore, we also capture the statements "someone in group G knows φ" and "everyone in group G knows φ" by the following shortcut modalities:

$$S_G\varphi \triangleq \bigvee_{a \in G} K_a\varphi \quad E_G\varphi \triangleq \bigwedge_{a \in G} K_a\varphi$$

Note that semantically, these will be the $\mathcal{RTKBL}_{\mathcal{SN}}$ equivalents of $\mathcal{TKBL}_{\mathcal{SN}}$'s SL ("someone learned") and EL ("everyone learned") modalities.

7.2 Semantics of $\mathcal{RTKBL}_{\mathcal{SN}}$

In order to define a $\mathcal{RTKBL}_{\mathcal{SN}}$-based inference engine for agents, the definition of timed derivation (Def. 9) can be reused entirely, just replacing Cl_T with the real-time version we are about to define, Cl_{RT}. We will also reuse most of the Cl_T definition (Def. 8), since the way it captures learning and inferring new knowledge applies here as well. We have to, however, update it with timestamps where missing. To make the following definition more readable, we use the placeholder variable name _ when the variable in question is not used again in the same context. If there are more instances of _ in the same formula, they refer to different variables.

Definition 20 (Real-Time Closure of a Knowledge Base)**.** Given the knowledge base of an agent a, KB_a, the *real-time closure of* KB_a, $Cl_{RT}(KB_a)$, satisfies the following properties:

1. Properties 1, 2, 3 in Def. 8, with Cl_{RT} instead of Cl_T.
2. If $(\varphi, t) \in Cl_{RT}(KB_a)$ then $(K_a^t\varphi, t) \in Cl_{RT}(KB_a)$.
3. If φ is provable in the axiomatization **S5** from $Cl_{RT}(KB_a)$ then $\varphi \in Cl_{RT}(KB_a)$. Formally:

A1, R2 - As A1, R2 in Def. 8, with Cl_{RT} instead of Cl_T.

A2 - If $(K_{\bar{a}}\varphi, t) \in Cl_{RT}(KB_a)$ and $(K_{\bar{a}}(\varphi \implies \psi), t') \in Cl_{RT}(KB_a)$, then $(K_a^{\max(t,t')}\psi, \max(t,t')) \in Cl_{RT}(KB_a)$.

A3 - If $(K_{\bar{a}}\varphi, t) \in Cl_{RT}(KB_a)$, then $(\varphi, t) \in Cl_{RT}(KB_a)$.

A4 - If $(K_a^{t'}\varphi, t) \in Cl_{RT}(KB_a)$, then $(K_a^t K_a^{t'}\varphi, t) \in Cl_{RT}(KB_a)$.

A5 - If $(\varphi, t) \notin Cl_{RT}(KB_a)$, then $(\neg K_a^t\varphi, t) \in Cl_{RT}(KB_a)$.

R2 - If (φ, t) is provable from no assumptions (i.e., φ is a tautology), then $(K_a^t\varphi, t) \in Cl_{RT}(KB_a)$.

C1 - $(E_G^{t'}\varphi, t) \in Cl_{RT}(KB_a)$ iff $(\bigwedge_{i\in G} K_i^{t'}\varphi, t) \in Cl_{RT}(KB_a)$.

C2 - $(C_G^{t'}\varphi, t) \in Cl_{RT}(KB_a)$ iff $(E_G^{t'}(\varphi \wedge C_G^{t'}\varphi), t) \in Cl_{RT}(KB_a)$.

RC1 - If $(\varphi \implies E_G^{t'}(\psi \wedge \varphi), t)$ is provable from no assumptions, then $(\varphi \implies C_G^{t'}\psi, t) \in Cl_{RT}(KB_a)$.

D1 - $(D_{\{a\}}^{t'}\varphi, t) \in Cl_{RT}(KB_a)$ iff $(K_a^{t'}\varphi, t) \in Cl_{RT}(KB_a)$.

D2 - If $(D_G^{t'}\varphi, t) \in Cl_{RT}(KB_a)$, then $(D_{G'}^{t'}\varphi, t) \in Cl_{RT}(KB_a)$ if $G \subseteq G'$.

DA2-DA5 Properties A2, A3, A4 and A5, replacing the modality K_a^t with the modality D_G^t for each axiom.

The basic principle behind the rules in the previous definition is the same as before. As was the case of $\mathcal{TKBL}_{\mathcal{SN}}$, the semantics of $\mathcal{RTKBL}_{\mathcal{SN}}$ is also given in terms of a satisfiability relation.

Definition 21 (Satisfiability Relation)**.** Given a well-formed trace $\sigma \in TCS$, agents $a, b \in Ag$, a finite set of agents $G \subseteq Ag$, formulae $\varphi, \psi \in \mathcal{F}_{\mathcal{RTKBL}}$, $m \in \mathcal{C}$, $n \in \Sigma$, $o \in \mathcal{D}$, a variable x, an event $e \in EVT$, and a timestamp t, the *satisfiability relation* $\models \subseteq TCS \times \mathcal{F}_{\mathcal{RTKBL}}$ is defined as shown in Fig. 7.1.

$$
\begin{array}{lll}
\sigma, t \models \mathit{happened}(e, t') & \text{iff} & e \in E \text{ where } (\mathit{RTSN}, E, t') \in \sigma \\
\\
\sigma, t \models \neg\varphi & \text{iff} & \sigma, t \not\models \varphi \\
\sigma, t \models \varphi \wedge \psi & \text{iff} & \sigma, t \models \varphi \text{ and } \sigma, t \models \psi \\
\sigma, t \models \forall x.\varphi & \text{iff} & \text{for all } v \in D_o^{\sigma[t]},\ \sigma, t \models \varphi[v/x] \\
\\
\sigma, t \models c_m(a, b, t') & \text{iff} & (a, b) \in C_m^{\sigma[t']} \\
\sigma, t \models a_n(a, b, t') & \text{iff} & (a, b) \in A_n^{\sigma[t']} \\
\\
\sigma, t \models p(\vec{s}, t') & \text{iff} & (p(\vec{s}, t'), t') \in KB_e^{\sigma[t']} \\
\\
\sigma, t \models K_a^{t'}\varphi & \text{iff} & (\varphi, t') \in Cl_{RT}(KB_a^{\sigma[t']}) \\
\\
\sigma, t \models C_G^{t'}\varphi & \text{iff} & \sigma, t' \models E_G^{t',k}\varphi \text{ for } k = 1, 2, \ldots \\
\sigma, t \models D_G^{t'}\varphi & \text{iff} & (\varphi, t') \in Cl_{RT}(\bigcup_{i \in G} KB_i^{\sigma[t']})
\end{array}
$$

Figure 7.1: *The semantics for $\mathcal{RTKBL}_{\mathcal{SN}}$ is given in terms of the satisfiability relation.*

One of the differences from the previous satisfiability relation (Def. 10) is the meaning of the timestamp on the left-hand side. In $\mathcal{TKBL}_{\mathcal{SN}}$, this timestamp marked the TSNM in the trace in which the right-hand side formula was meant to be checked. Here, we use the timestamps in the syntax to guide the checking process to the corresponding RTSNMs. When checking connections and actions at time t, we check whether the corresponding relation of the RTSNM at time t contains the agents in question. Checking a predicate with timestamp t is equivalent to looking into the knowledge base of the environment at time t and looking for the predicate. Determining whether an agent a knows φ at time t translates to looking into the closure of a's knowledge base at time t to see whether a learns φ at time t. The left-hand-side timestamp is only used in the quantification case to determine the RTSNM in the trace whose relational structure should be used to obtain the values to substitute for the quantified variable.

The semantics of the new special event predicate $\mathit{happened}(e, t)$ boils down to looking into the set of events that happened in the trace at time t and determining whether e is one of them.

7.3 Examples

Example 13. Consider Fig. 7.2 which shows a RTSNM with Alice, Bob and Charlie, the connections between them, and their knowledge bases. In this case, the predicates used have the following meaning; we use $\mathit{picture}(x, y, z)$ for "there is a picture posted by x with resource identifier y", and similarly for *location*; $\mathit{event}(x, y)$ stands for "there is an event with resource identifier x", and $\mathit{attending}(x, y, z)$ is "agent x is attending event with identifier y". In all cases, the remaining argument is the timestamp representing when that particular piece of information was true.

Let RTSN_0 be the RTSNM modeled in the picture and let σ be the well-formed

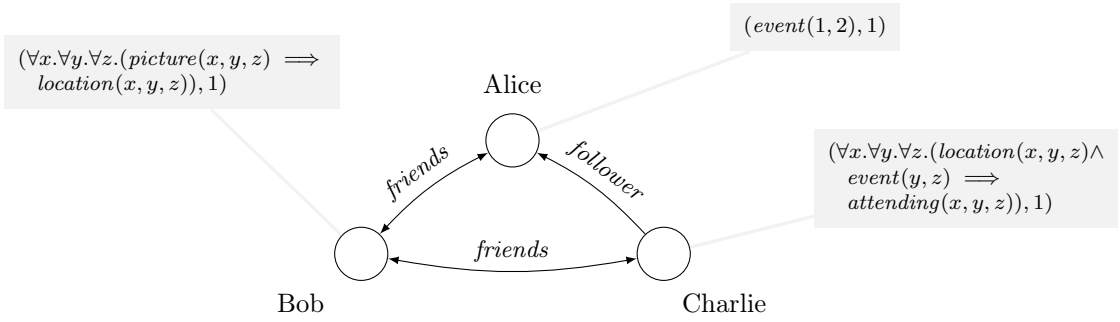

Figure 7.2: *A RTSNM with three agents, the connections between them, and their knowledge bases. Here, Bob is able to derive the location of a picture if they share the same resource identifier, and Charlie is able to conclude that if the time and place of someone's location correspond to that of a known event, the person in question is attending the event. Meanwhile, Alice found out about an event with identifier 1 which happens at time 2.*

trace

$$\begin{aligned}\sigma = \langle &(RTSN_0, E_0, 1),\\ &(RTSN_1, \{postPicture(Alice)\}, 2),\\ &(RTSN_2, \{openFeed(Charlie)\}, 3),\\ &(RTSN_3, \{openFeed(Bob)\}, 4)\rangle,\end{aligned}$$

where the $postPicture(Alice)$ event – which transforms $RTSN_0$ to $RTSN_1$ at time 2 – represents Alice making a picture public to her friends and followers and the meaning of the *openFeed* event is the same as before: it refers to a user accessing all information available to them in the OSN.

The knowledge evolution from $\mathcal{RTSN}_0$ to $\mathcal{RTSN}_3$ is as follows. At time 2, a picture with resource identifier 1 (that is, tied to event 1) taken by Alice appears in the network. Charlie then learns about the picture at time 3, while Bob learns about it at time 4. In addition, let us assume both Bob and Charlie learn about event 1 when they open their news feed.

Let us now check the following properties on σ:

$$\sigma, 3 \models \neg K^3_{Bob} location(Alice, 1, 2) \tag{1}$$

$$\sigma, 4 \models K^4_{Bob} location(Alice, 1, 2) \tag{2}$$

To check whether Bob does not learn Alice's location 1 from time 2 at time 3, we consult the satisfiability definition (Def. 21). We look into the closure of Bob's knowledge base at time $\sigma[3] = RTSN_2$. Since Bob could not learn or infer Alice's location from time 2 at time 3, $(location(Alice, 1, 2), 3) \notin Cl_{RT}(KB^{\sigma[3]}_{Bob})$, and so property (1) holds.

However, as soon as Bob opens his news feed at time 4, he learns about Alice's picture, so $(picture(Alice, 1, 2), 4) \in Cl_{RT}(KB^{\sigma[4]}_{Bob})$. He is then able to use the rule in his knowledge base to learn $location(Alice, 1, 2), 4)$, which will be in the closure of his knowledge base at time 4, thus fullfiling (2).

Example 14. Using everything established in Example 13, let us also check a property regarding the knowledge of multiple agents at once. Let $G = \{Bob, Charlie\}$.

$$\sigma, 4 \models D_G^4 attending(Alice, 1, 2) \tag{1}$$

To answer whether Bob and Charlie together are able to learn at time 4 about Alice attending event 1 at time 2, let us consider what they know independently at 4. According to the previous example, $(location(Alice, 1, 2), 4) \in Cl_{RT}(KB_{Bob}^{\sigma[4]})$. Charlie learns about Alice's picture 1 from time 2 as well as about event 1 from time 2 once he opens his news feed at time 3. He is, however, unable to use the rule he has had in his KB from time 1 to infer anything new from the two pieces of information.

The collective knowledge of Bob and Charlie at time 4 contains all of this. And so, since both $(location(Alice, 1, 2), 4)$ and $(event(1, 2), 3)$ are in $Cl_{RT}(\bigcup_{i \in G} KB_i^{\sigma[4]})$, the old rule of Charlie's can be used so that we get

$$(attending(Alice, 1, 2), 4) \in Cl_{RT}\left(\bigcup_{i \in G} KB_i^{\sigma[4]}\right),$$

which is the interpretation of (1).

Chapter 8

Real-Time Privacy Policy Language ($\mathcal{RTPPL}_{\mathcal{SN}}$)

Since $\mathcal{RTKBL}_{\mathcal{SN}}$ treats timestamps like any other variable, we do not need to define any special time fields to take care of the time aspect artificially, as we did previously. Instead, we rely on the power of $\mathcal{RTKBL}_{\mathcal{SN}}$ itself.

8.1 Writing Privacy Policies

The definition of the real-time privacy policy language ($\mathcal{RTPPL}_{\mathcal{SN}}$) contains less rules than that for $\mathcal{TPPL}_{\mathcal{SN}}$. It closely follows the description of the original $\mathcal{FPPF}$, adding timestamps where necessary.

Definition 22 (Syntax of $\mathcal{RTPPL}_{\mathcal{SN}}$)**.** Given agents $a, b \in Ag$, a nonempty set of agents $G \subseteq Ag$, timestamps s, t, a variable x, relation symbols $c_m(a, b, t), a_n(a, b, t)$, $p(\vec{s}, t)$, and a formula $\varphi \in \mathcal{F}_{\mathcal{RTKBL}}$, the *syntax of the real-time privacy policy language* $\mathcal{RTPPL}_{\mathcal{SN}}$ is inductively defined as:

$$\begin{array}{rcl} \delta & ::= & \delta \wedge \delta \mid \forall x.\delta \mid [\![\neg\alpha]\!]^s_a \mid [\![\varphi \implies \neg\alpha]\!]^s_a \\ \alpha & ::= & \alpha \wedge \alpha \mid \forall x.\alpha \mid \exists x.\alpha \mid \psi \mid \gamma' \\ \psi & ::= & c_m(a, b, t) \mid a_n(a, b, t) \\ \gamma' & ::= & K^t_a\gamma \mid D^t_G\gamma \mid C^t_G\gamma \\ \gamma & ::= & \gamma \wedge \gamma \mid \neg\gamma \mid p(\vec{s}, t) \mid \gamma' \mid \psi \mid \forall x.\gamma \end{array}$$

We will use $\mathcal{F}_{\mathcal{RTPPL}}$ to denote the set of all privacy policies created according to the previous definition.

8.2 Checking Privacy Policies

To determine whether a policy gets violated in an evolving OSN, we formalize the notion of conformance for $\mathcal{RTPPL}_{\mathcal{SN}}$.

Definition 23 (Conformance Relation)**.** Given a well-formed trace $\sigma \in TCS$, a variable x, a timestamp s, and an agent $a \in Ag$, the *conformance relation* $\models_C$ is defined as shown in Fig. 8.1.

$$\sigma \models_C \forall x.\delta \quad \text{iff} \quad \text{for all } v \in D_o,\ \sigma \models_C \delta[v/x]$$

$$\sigma \models_C [\![\neg\alpha]\!]^s_a \quad \text{iff} \quad \sigma \models \neg\alpha$$
$$\sigma \models_C [\![\varphi \implies \neg\alpha]\!]^s_a \quad \text{iff} \quad \sigma \models \varphi \implies \neg\alpha$$

Figure 8.1: *The semantics of the real-time privacy policy language $\mathcal{RTPPL}_{\mathcal{SN}}$ is given in terms of the conformance relation $\models_C$.*

The definition is quite simple, especially compared to that of conformance of $\mathcal{TPPL}_{\mathcal{SN}}$ (Def. 13). If the policy is quantified, we substitute in the usual way. The main body of the policy in double brackets is dealt with by simply delegating to the satisfiability relation.

8.3 Clarifications

8.3.1 Indefinitely Recurring Windows

While $\mathcal{RTPPL}_{\mathcal{SN}}$ allows users to define more fine-grained policies than $\mathcal{TPPL}_{\mathcal{SN}}$ (Sec. 8.4), it has its limits. For example, defining a recurring privacy policy cannot be done without help of specialized predicates, and some more complex recurrent windows cannot be defined at all.

Suppose a user wants to define a policy which utilizes a timestamp that should represent a weekend. We can introduce a specialized predicate for this, called *weekend*(t), which is true if t belongs to a weekend. Analogously, we could have predicates like *weekday*, *evening*, or *Monday*, that would help anchor a timestamp in correct context.

But how does one deal with contexts that are more arbitrary, for example, from a fixed date to a fixed date each year? This is not easily done. Though we can compare timestamps – so it is possible to write statements like $t <$ 2016-28-05 $\wedge\, t >$ 2016-02-03 –, there is no other way to capture multiple windows (like "from February 3 to May 28 every year") than to restrict the timestamps by adding more intervals into the conjunction, which is not ideal, especially when the interval should repeat more often and many times – perhaps even indefinitely.

8.3.2 Restricting in the Past

The ability to use timestamps in almost arbitrary ways in $\mathcal{RTPPL}_{\mathcal{SN}}$ creates a problem that has to do with restricting past behavior. Consider the following policy:

$$\delta = [\![\neg K^1_b \varphi]\!]^4_a$$

Here, a is attempting to prevent b from learning φ at time 1, but the activation time of the policy is 4. In other words, δ wants to restrict something that may have already happened, in which case it would be trivially violated.

8.4 Examples

Example 15. Revisiting Example 8, assume Alice decides to hide all her weekend locations from her supervisor Bob. She has a number of options how to achieve this, depending on what the precise meaning of the policy should be.

If the idea she has is to restrict Bob learning her weekend location directly when she posts it, she can define

$$\delta_1 = \forall x.[\![\mathit{weekend}(x) \implies \neg K^x_{Bob}\mathit{location}(\mathit{Alice}, x)]\!]^{\text{2016-04-16}}_{\mathit{Alice}}$$

where the *weekend* predicate is true if the timestamp supplied represents a time during a weekend. This policy can be read as "if x is a time instant during a weekend, then Bob is not allowed to learn at x Alice's location from time x".

This, however, is a very specialized scenario that captures only a small number of situations. Bob is, for example, free to learn Alice's location at any point not during the weekend, or at any point during the weekend when Alice's location is no longer up-to-date. Though there might be scenarios where this might be the desired behavior, we can define a policy that seems much closer to the intuitive meaning of learning someone's location on a weekend. Consider

$$\delta_2 = \forall x.[\![\mathit{weekend}(x) \implies \neg\exists y.(K^y_{Bob}\mathit{location}(\mathit{Alice}, x))]\!]^{\text{2016-04-16}}_{\mathit{Alice}}.$$

Here, Bob is not allowed to learn Alice's location from a weekend, no matter when. If the policy does not get violated, then Alice's weekend locations will be completely safe from Bob – on the OSN, at least.

Example 16. One of the advantages of $\mathcal{RTPPL}_{\mathcal{SN}}$ is the ability to distinguish between the original time of a piece of information and the time when it should be hidden. Suppose Diane activates the following policy:

$$\delta_1 = \forall x.\forall y.[\![\neg\mathit{friends}(\mathit{Diane}, x, y) \implies \neg\exists z.(K^y_x\mathit{post}(\mathit{Diane}, z))]\!]^{\text{2016-05-28}}_{\mathit{Diane}}$$

This aims to prevent anyone who is not a friend of Diane's from learning any of her posts (here we assume that the *friends* connection is reflexive for simplicity, otherwise the restriction would target Diane herself, too).

Though δ_1 may seem reasonable enough, it might be unnecessarily restrictive. Let us say there is another user, Ethan. Diane becomes friends with Ethan on May 31, so when her policy is already in effect. Should Ethan be able to learn about Diane's posts from when they were not friends? Not according to δ_1, which says that no one, regardless of their relationship with Diane at the moment, is able to learn about her posts from when they were not friends.

Note that while this may indeed be the desired behavior, it is, for example, not what happens on Facebook, where when two users become friends, they are free to access each other's timeline including past events and posts. $\mathcal{RTPPL}_{\mathcal{SN}}$ is expressive enough to model this behavior as well. We can define:

$$\delta_2 = \forall x.\forall y.\forall y'.[\![\neg\mathit{friends}(\mathit{Diane}, x, y) \wedge \neg\mathit{friends}(\mathit{Diane}, x, y') \implies \neg K^{y'}_x\mathit{post}(\mathit{Diane}, y)]\!]^{\text{2016-05-28}}_{\mathit{Diane}}$$

This policy precisely defines the point in time *from* when to hide information, y, as well as the point in time *when* to hide it, y'. It says, "if Diane is not friends with someone, then that someone cannot learn her posts, but only if they come from a time when they were not friends". Note that δ_2 says nothing about users who are currently friends of Diane's, which is different from δ_1 – here her friends can learn anything, including past posts from when they were not friends with her.

Chapter 9

Discussion

Both $\mathcal{TFPPF}$ and $\mathcal{RTFPPF}$ have their strengths and weaknesses. Both allow for fine-grained policies and even though $\mathcal{TFPPF}$ is in many aspects limited compared to $\mathcal{RTFPPF}$, it carries out its specialized task of enabling policies in fixed, recurring windows *ad infinitum* well. $\mathcal{TKBL}_{\mathcal{SN}}$ acts as an important milestone from the original $\mathcal{KBL}_{\mathcal{SN}}$ by allowing to reason about time, though it is arguably $\mathcal{RTKBL}_{\mathcal{SN}}$ that actually achieves what one would consider reasoning about knowledge in real time. However, it is still our belief that there is a large number of use cases in which $\mathcal{TFPPF}$ would be a good candidate, especially if one does not need the additional machinery introduced in $\mathcal{RTFPPF}$, or if one wants to define truly indefinitely recurrent policies. $\mathcal{RTFPPF}$, on the other hand, is preferable where even more fine-grained policies are needed.

$\mathcal{TFPPF}$ has a number of areas to improve, many of which $\mathcal{RTFPPF}$ manages to address – after all, they were the reason why $\mathcal{RTFPPF}$ was developed. Most of these issues stem from the limited expressive power of $\mathcal{TKBL}_{\mathcal{SN}}$, which introduces time, but in a relative way only. This is because we utilize the $\Box$ and $\Diamond$ operators without any notion of real time, so while it is possible to ensure, for example, that something will eventually hold, one cannot specify the actual time when it should. This is partially remedied on the $\mathcal{TPPL}_{\mathcal{SN}}$ level, but again only in a limited way – while policies can be defined for fixed time windows, they are very rigid and incapable of carrying out any more complex requests a user might have. They only specify a part of the trace where a property should hold; anything outside is not taken into account and even the ability to navigate the subtrace is limited.

While $\mathcal{RTFPPF}$ manages to address most of these issues, it comes with its own set of shortcomings. The definition of the trace indexing function (Sec. 6.2.2) is problematic in its handling of out-of-bounds timestamps. To recapitulate, if one tries to access a RTSNM with a timestamp that is greater than the last one in the trace, the function just returns the last one, and analogously for a timestamp lower than the very first. This can be an issue if one looks at the definition of the satisfiability relation (Def. 21) – when quantifying over timestamps, it treats them like any other variable, but when evaluating a basic formula, the timestamp in it has a very special interpretation and is used to navigate the trace. So when interpreting a predicate like $\mathit{location}(\mathit{Alice}, \text{2000-01-01})$ over a trace that starts at time 2016-01-01, we actually end up checking something entirely different, $\mathit{location}(\mathit{Alice}, \text{2016-01-01})$. Probably the most reasonable solution is that accessing an out-of-bounds RTSNM

should be undefined, since we cannot say anything about what the OSN looked like at that point. This, however, has other implications, as it essentially results in a three-valued logic.

As hinted at in the previous paragraph, timestamps are really a very specific part of the $\mathcal{RTKBL}_{\mathcal{SN}}$ syntax. The timestamp domain is naturally ordered and they receive special treatment in the satisfiability relation. These are good reasons for promoting the timestamp domain to a special one, but in this thesis we just assume they have the same status and importance as other domains.

Also, looking back at the $\mathcal{RTKBL}_{\mathcal{SN}}$ satisfiability relation, we mentioned that the timestamp on the left-hand side is only used in one case – to deal with quantification. This is, actually, the reason why we introduced this special timestamp in the first place: to be able to determine which relational structure to use when looking for candidates for a certain variable. This choice has its issues, as does the $\mathcal{TKBL}_{\mathcal{SN}}$ satisfiability relation (Def. 10), since it uses the same principle. The problem is that the structure of the RTSNMs in the trace changes over time – for example, users may leave the OSN. However, when quantifying over the domain of agents, we take the set of agents at time t, when the quantified formula is being checked, which may result in problems when evaluating any inner formulae at a different point in time.

As we mentioned in the chapter about $\mathcal{RTPPL}_{\mathcal{SN}}$ (Chap. 8), the language has some disadvantages, some even compared to $\mathcal{TPPL}_{\mathcal{SN}}$. Unlimited recurrence is hard to achieve unless the user does not mind activating a new policy everytime it exhausts the finite number of time windows it can capture. This could be remedied by adding arithmetic to $\mathcal{RTPPL}_{\mathcal{SN}}$.

9.1 Future Work

We can imagine a large number of potential paths future research can take to improve the frameworks we propose in this work. Some of them have been hinted at in the previous section. Other ideas are:

- We mentioned early on that the formalization of OSN *instantiation* and *privacy preservation* are out of the scope of this work. These two notions are natural candidates for future investigation.

- Some information and knowledge can be overriden by new data of the same kind. Consider, for example, knowing someone's birthday and knowing their location. While the location of an agent may and probably will change a lot, their birthday is set in stone. If Alice learns Bob's location on Monday afternoon, it does not tell her anything about his location a year from then, but learning Bob's birthday is permanent. We could therefore find a categorization for information, for instance into *transient* and *permanent*. This would bring the framework closer to the actual notion of learning and knowing in reality.

- A related notion is that of *knowledge validity in time*. Each piece of information has an implicit timeout – the time after which it is no longer up-to-date. For

permanent knowledge, like someone's birthday, this would be infinity. However, the real challenge would be to reasonably determine this timeout for transient knowledge. For example, how long should someone's location be valid for? Probably not until they post a new one, because the two may be years apart. This is an interesting question both philosophically and from the privacy point of view, which could be explored in future work.

- When we talk about knowledge in this work, we actually mean *belief* as there is no universal way to check whether an arbitrary piece of information one comes across on an OSN is true. One problem that stems from this is that an agent can receive conflicting pieces of information. For example, Alice might tag Bob in a post about sitting in a pub in Dublin with her friends, but at the same time, Bob might post a picture of himself in front of the Eiffel Tower. Which of these is true? Is any? It might worthwhile to have a subset of data one is sure of and can really claim to know, like birthdays of family members or information about events one has actually attended. This would help separate belief from actual knowledge.

- It might also be interesting to explore other *formalizations of real-time*, since here we model it discretely. It seems to work well for our purpose, but other formalizations might bring other advantages.

Chapter 10

Conclusion

Online social networks (OSNs) are an important part of people's lives worldwide and their users supply them with a large amount of personal data. The audience of this information is usually restricted by privacy policies which the users can define. However, these settings are often limited and their meaning is not clearly defined, leading to discrepancies between the users' expectations and actual flow of information in the OSN.

In this thesis we explored and formalized two concepts of real-time OSN frameworks built upon the formal privacy policy framework $\mathcal{FPPF}$ [25, 26]. The aim was to develop a time-sensitive formalization of OSNs and the ways information spreads there, ultimately leading to a fine-grained privacy policy language based on a knowledge-based logic.

The first framework we propose, $\mathcal{TFPPF}$, comprises three main components. First, there is the timed social network model TSNM, which captures the structure of an OSN at a certain point in time, along with its users and the relationships and permissions between them. Next, we introduce the temporal knowledge-based logic $\mathcal{TKBL}_{\mathcal{SN}}$, which is used to reason about knowledge in evolving OSNs. A satisfiability relation is then used to determine whether a $\mathcal{TKBL}_{\mathcal{SN}}$ formula holds in an evolving OSN. Finally, we define the timed privacy policy language $\mathcal{TPPL}_{\mathcal{SN}}$, a formal language to write privacy policies based on $\mathcal{TKBL}_{\mathcal{SN}}$, and we also formalize how to determine if an OSN violates a specific policy. $\mathcal{TPPL}_{\mathcal{SN}}$ allows the users to have their policies evaluated either from a point in time forward, or in a (number of) precisely defined time window(s).

The second framework proposed in this work, $\mathcal{RTFPPF}$, also comprises three parts. First, there is the real-time social network model RTSNM, whose structure is very similar to that of TSNM. The first major difference comes at the level of the knowledge-based logic. $\mathcal{RTFPPF}$'s logic is called real-time knowledge-based logic ($\mathcal{RTKBL}_{\mathcal{SN}}$) and contains timestamps on the level of syntax, allowing for a wide range of time-sensitive formulae. A satisfiability relation is provided as well, to determine whether a $\mathcal{RTKBL}_{\mathcal{SN}}$ formula holds in the dynamic OSN. Finally, we define the real-time privacy policy language $\mathcal{RTPPL}_{\mathcal{SN}}$, built on $\mathcal{RTKBL}_{\mathcal{SN}}$, together with a conformance relation.

Each of the frameworks allows users of OSNs to define fine-grained, time-sensitive privacy policies based on formal logic, thus addressing the problem of privacy policy ambiguity in OSNs in a dynamic context. Moreover, both $\mathcal{TKBL}_{\mathcal{SN}}$ and

$\mathcal{RTKBL}_{\mathcal{SN}}$ can be used directly to reason about knowledge spreading and evolving in an OSN.

Both $\mathcal{TFPPF}$ and $\mathcal{RTFPPF}$ constitute a step forward in reasoning about knowledge and restricting the audience of information in evolving OSNs.

Bibliography

[1] R. Alur and D. L. Dill. "A Theory of Timed Automata." In: *Theoretical Computer Science* 126 (1994), pp. 183–235.

[2] R. Alur and T. A. Henzinger. "Logics and Models of Real Time: A Survey." In: *Real Time: Theory in Practice*. Ed. by J. W. de Bakker et al. Vol. 600. Lecture Notes in Computer Science. Springer-Verlag, 1992, pp. 74–106.

[3] R. Alur and T. A. Henzinger. "Real-Time Logics: Complexity and Expressiveness." In: *Information and Computation* 104 (1993), pp. 35–77.

[4] I. Ben-Zvi and Y. Moses. *Agent-Time Epistemics and Coordination*. 2014.

[5] I. Ben-Zvi and Y. Moses. "Beyond Lamport's Happened-Before: On Time Bounds and the Ordering of Events in Distributed Systems." In: *Journal of the ACM* 61.2 (2014).

[6] I. Ben-Zvi and Y. Moses. *Known Unknowns: Time Bounds and Knowledge of Ignorance*. 2016.

[7] I. Ben-Zvi and Y. Moses. *On Interactive Knowledge with Bounded Communication*.

[8] I. Ben-Zvi and Y. Moses. *The Shape of Reactive Coordination Tasks*.

[9] N. Catano, V. Kostakos, and I. Oakley. "Poporo: A Formal Framework for Social Networking." In: *Proceedings of the Third International Workshop on Formal Methods for Interactive Systems (FMIS 2009)*. 2009, pp. 79–82.

[10] N. B. Ellison and danah boyd. "Sociality through Social Network Sites." In: *The Oxford Handbook of Internet Studies*. Ed. by W. H. Dutton. Oxford University Press, 2013, pp. 151–172.

[11] R. Fagin et al. *Reasoning About Knowledge*. MIT Press, 1995.

[12] P. W. L. Fong, M. Anwar, and Z. Zhao. *A Privacy Preservation Model for Facebook-like Social Network Systems*. Tech. rep. University of Regina, 2009.

[13] Y. A. Gonczarowski and Y. Moses. "Timely Common Knowledge. Characterising Asymmetric Distributed Coordination via Vectorial Fixed Points." In: *TARK 2013*. 2013.

[14] J. Y. Halpern. "Reasoning About Knowledge: A Survey." In: *Handbook of Logic in Artificial Intelligence and Logic Programming*. Ed. by D. Gabbay, C. J. Hogger, and J. A. Robinson. Vol. 4. Oxford University Press, 1995, pp. 1–34.

[15] J. Y. Halpern, R. van der Meyden, and M. Y. Vardi. "Complete Axiomatizations for Reasoning About Knowledge and Time." In: *SIAM Journal on Computing* 33 (3 2004), pp. 674–703.

[16] J. Hintikka. *Knowledge and Belief*. Cornell University Press, 1962.

[17] M. Huth and M. Ryan. *Logic in Computer Science. Modelling and Reasoning about Systems*. Cambridge University Press, 2004.

[18] *Internet Seen as Positive Influence on Education but Negative on Morality in Emerging and Developing Nations. Internet Usage More Common Among the Young, Well-Educated and English Speakers*. Report. Pew Research Center, 2015. URL: `http://www.pewglobal.org/2015/03/19/internet-seen-as-positive-influence-on-education-but-negative-influence-on-morality-in-emerging-and-developing-nations/` (visited on 05/18/2016).

[19] *ISO 8601:2004. Data elements and interchange formats – Information interchange – Representation of dates and times*. URL: `http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=40874` (visited on 05/25/2016).

[20] S. A. Kripke. "Semantical Analysis of Modal Logic I: Normal Modal Propositional Calculi." In: *Zeitschrift für mathematische Logik und Grundlagen der Mathematik* 9 (1963), pp. 67–96.

[21] Y. Liu et al. "Analyzing Facebook Privacy Settings: User Expectations vs. Reality." In: *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*. IMC '11. ACM, 2011, pp. 61–70.

[22] M. Madden and L. Rainie. *Americans' Attitudes About Privacy, Security and Surveillance*. Report. Pew Research Center, 2015. URL: `http://www.pewinternet.org/2015/05/20/americans-attitudes-about-privacy-security-and-surveillance/` (visited on 05/17/2016).

[23] Y. Moses. "Reasoning About Knowledge and Belief." In: *Handbook of Knowledge Representation*. Ed. by F. van Harmelen, V. Lifschitz, and B. Porter. Elsevier, 2008, pp. 621–647.

[24] E. Pacuit. "Dynamic Epistemic Logic I: Modeling Knowledge and Belief." In: *Philosophy Compass* (2013), pp. 1–17.

[25] R. Pardo. "Formalising Privacy Policies for Social Networks." Licentiate thesis. Chalmers University of Technology and University of Gothenburg, 2015.

[26] R. Pardo and G. Schneider. "A Formal Privacy Policy Framework for Social Networks." In: *SEFM '14, LNCS 8702*. 2014, pp. 378–392.

[27] A. Perrin. *Social Networking Usage: 2005-2015. 65% of Adults Now Use Social Networking Sites: A Nearly Tenfold Jump in the Past Decade*. Report. Pew Research Center, 2015. URL: `http://www.pewinternet.org/2015/10/08/2015/Social-Networking-Usage-2005-2015/` (visited on 05/18/2016).

[28] A. Pnueli. "The Temporal Logic of Programs." In: *Proceedings of the 18th Annual Symposium on Foundations of Computer Science* (1977), pp. 46–57.

[29] *The Universal Declaration of Human Rights*. URL: `http://www.un.org/en/universal-declaration-human-rights/` (visited on 05/18/2016).

[30] R. van der Mayden and K.-s. Wong. "Complete Axiomatizations for Reasoning About Knowledge and Branching Time." In: *Studia Logica* 75 (1993), pp. 93–123.

[31] B. Woźna and A. Lomuscio. *A Logic for Knowledge, Correctness, and Real Time*. Tech. rep. University College London.