A Framework for Expressing and Enforcing Purpose-Based Privacy Policies

Mohammad Jafari
jafarm@ucalgary.ca
Philip W. L. Fong
pwlfong@ucalgary.ca
Reihaneh Safavi-Naini
rei@ucalgary.ca
Ken Barker
kbarker@ucalgary.ca
Department of Computer Science, University of Calgary

Technical Report: 2013-1037-04

January 2013

Abstract

Purpose is a key concept in privacy policies and has been mentioned in major privacy laws and regulations. Although some models have been proposed for enforcing purpose-based policies, little has been done in defining formal semantics for purpose and therefore an effective enforcement mechanism for policies has remained a challenge. In this paper, we develop a framework for formalizing and enforcing *purpose-based privacy policies*. Purpose is formally defined as the dynamic situation of an action within the network of inter-related actions in the system. Accordingly, we propose a modal-logic language for formally expressing constraints about purposes of actions which can be used to model purpose-based policies. The semantics of this language are defined over an abstract model of activities in the system which is directly derivable from *business processes*.

Based on this formal framework, we discuss some properties of purpose and show how some well-known, as well as new forms of purpose constraints can be formalized using the proposed language. We also show how purpose-based constraints can be tied to other access control policies in the system. Finally, we present a model-checking algorithm for verifying whether a given state of the system complies with a given set of policies, followed by a discussion of how this can be used in an actual implementation of a *purpose reference monitor*.

Keywords: Purpose, Semantics, Purpose-Based Policies, Privacy, Modal Logic, Workflow, Petri Net

Contents

1 Introduction **4**
- 1.1 Semantics of Purpose . . . 5
- 1.2 Contributions . . . 6
- 1.3 Roadmap . . . 6

2 Related Work **7**
- 2.1 Purpose . . . 7
  - 2.1.1 Expressing Purpose-Based Policies . . . 7
  - 2.1.2 Enforcement . . . 8
- 2.2 Obligation . . . 9
- 2.3 Workflow . . . 9

3 Walk-Through: A Medical Research Institute **10**

4 Conceptual Basis **12**
- 4.1 Transitivity and Multiple Purposes . . . 13
- 4.2 Inferring and Enforcing Purpose . . . 13

5 Modelling Plans **15**
- 5.1 Petri Nets . . . 15
- 5.2 Workflow Net (WN) . . . 15
  - 5.2.1 Structured Loops . . . 16
- 5.3 Hierarchy Nets (HN) . . . 17
  - 5.3.1 Policy Components of a Hierarchy Net Definition . . . 17
- 5.4 Operational Semantics . . . 17
  - 5.4.1 Operational Semantics of a HN . . . 18
- 5.5 Structural Properties . . . 19
- 5.6 Dynamic Properties . . . 20
  - 5.6.1 Finiteness . . . 20
  - 5.6.2 Soundness . . . 20
  - 5.6.3 Persistence . . . 21
  - 5.6.4 Safety . . . 21
- 5.7 Example . . . 21
- 5.8 Composition of Sound Workflows . . . 21
  - 5.8.1 Loop (λ) . . . 22
  - 5.8.2 Refine (ρ) . . . 22
  - 5.8.3 Sequence (σ) . . . 22
  - 5.8.4 Parallel (π) . . . 22
  - 5.8.5 Choice (χ) . . . 23
- 5.9 A- and F- Relation . . . 23

6 The Modal Logic Language **24**
6.1 Types of Modality . . . 24
6.1.1 Common Types of Assertion . . . 25
6.2 Definition of the Language . . . 26
6.2.1 Formal Definition . . . 26
6.2.2 Pre-Image Functions . . . 27
6.2.3 Recursive Definition of Modal Operators . . . 27
6.2.4 The Importance of Splits . . . 28
6.3 Dynamic Semantics: Linking the Modal Operators to Purpose . . . 28
6.3.1 Formal Definition . . . 29
6.3.2 Discussion . . . 29
6.3.3 Equivalence of Dynamic and Static Semantics . . . 30

7 Purpose-Based Policies **31**
7.1 Types of Purpose Constraints . . . 31
7.1.1 Required Purposes . . . 31
7.1.2 Forbidden Purposes . . . 32
7.1.3 A- vs. F- Purposes . . . 33
7.1.4 Order-Based Constraints . . . 33
7.2 Linking to Other Access Control Policies . . . 33
7.2.1 Action-Centric Policies . . . 34
7.2.2 Subject-Centric Policies . . . 34
7.2.3 Data-Centric Policies . . . 34
7.2.4 Environment-Centric Policies . . . 35
7.2.5 Compound Policies . . . 35
7.2.6 Purpose-Based Obligations . . . 35
7.3 Policy Origins . . . 35

8 Model Checking **37**
8.1 Computing Pre-Image functions . . . 37
8.2 Recursive Model Checking of Diamond Operators . . . 38
8.3 Correctness . . . 38
8.4 Example . . . 38

9 Implementation Considerations **40**
9.1 Model Checking Configurations . . . 40
9.1.1 Workflow-Definition Reference Monitor . . . 40
9.1.2 Workflow-Instantiation Reference Monitor . . . 40
9.1.3 Task-Instantiation Reference Monitor . . . 41
9.2 System Components . . . 41
9.2.1 Workflow Definition Request . . . 41
9.2.2 Workflow Instantiation Request . . . 42
9.3 Case: A Clinical Research Institute . . . 42
9.3.1 Vocabulary . . . 43
9.3.2 Consent Policy . . . 43
9.3.3 Workflow . . . 43
9.3.4 Labelling . . . 43
9.3.5 Event Flow . . . 44
9.4 Discussion . . . 45
9.4.1 Ontologies and the Granularity Gap in the Vocabulary . . . 45
9.4.2 Translation and Harmonization . . . 45

10 Concluding Remarks and Future Work **47**
10.1 Parametrized Purposes [1]We would like to thank Professor Jörg Denzinger of University of Calgary Department of Computer Science, for an enlightening conversation about this section. 47
10.2 Modification of the Workflow 48
10.3 Purpose-Based Data Adaptation 48
10.4 Satisfiability 49

A Proof of Theorems **55**
A.1 Assumptions and Notations 55
A.2 Proofs 55

1

Introduction

> "*As he looked back upon man moving through history, he was haunted by a feeling of loss. So much had been surrendered and to such little purpose!*"
>
> *Oscar Wilde, The Picture of Dorian Gray*

As storing and processing data in electronic form becomes more and more prevalent, privacy concerns are on the rise and have become a major issue in developing nearly every electronic system. Personal and private data is growingly turned into electronic form in traditional and emerging applications and services (such as social networks, electronic health records, cloud storage services, online banking, *etc.*) thereby making data vulnerable to unauthorized usage by various parties. To address such concerns, privacy policies were introduced as a means to formulate the rules that govern the use of data and express what data can be used, by what parties, to what extent, and in what manner. Defining privacy policies and enforcing them is, therefore, a crucial step in developing nearly every electronic system nowadays to ensure that private data will remain safe and reliable.

Purpose of use is one of the core concepts in privacy policies, mentioned in major privacy laws and regulations in different jurisdictions such as Canada's Federal Privacy Act (1983) and the U.S. Privacy Act (1974). Informally, purpose refers to the user's intention for using data; for example, an employee in an online retailer may use a customer's home address *for the purpose of shipping an ordered good*, or a physician in a research institute may use some of the data in a patient's health record *for research purposes*. Many privacy policies contain rules and restrictions about purpose of data access; for example, an online customer may like to allow access to her email address only for the purpose of sending *order confirmation* and *billing information*, and not for *marketing*, or, the government may decide to prohibit using an applicants' ethnic background information for the purpose of *making hiring decisions*. A *purpose-based privacy policy* is the set of such rules that stipulate whether or not access should be allowed, based on its purpose.

Enforcing such policies requires a *purpose-based access control* system in which purpose of access is a major factor in deciding whether or not access should be allowed. This enables making different decisions based on the purpose of access, even to the same data item and by the same user. For example, the policy may allow an online store employee to use a customer's email address for the purpose of *sending the bill*, but prohibit access to the same data and by the same person/role, for other purpose such as *email marketing*. Traditional access control systems are unable to enforce such rules.

Identifying purpose of access is also important in mitigating the risk of privacy breach by insiders by preventing authorized users from abusing their authorized access rights. For example, a *bank teller* is usually given the permission to query bank customers by name and account number, which is useful in cases where a customer wants to deposit money to another customer's account –since double-checking the receiver account's information is helpful to ensure the money is transferred to the right recipient. In a traditional access control

system, this will result in granting the *teller* role *read* access to all customers' account information. A more advanced system may add more restrictions such as limiting this access right to business hours or within the premises of the branch, but none of these policies can prevent a malicious teller from taking advantage of the granted permission and pry into a friend or family member's account information. A purpose-based policy, however, can prevent this by restricting the granted access right only to the legitimate purposes (such as *money deposit*) and thus, denying access when it is not for one of the legitimate purposes. Various similar cases can be identified in other business systems where access rights need to be granted but only for certain purposes; for example access of university authorities to student academic records, physicians to medical records, and employees to customer information.

1.1 Semantics of Purpose

There are two main components in a purpose-based access control system:

- *Expression*: A mechanism for expressing access control rules based on the purpose of access.
- *Enforcement*: A mechanism for identifying the purpose of a requested access, so that it can be verified against the policy rules to decide whether to allow or deny access.

An effective mechanism for any of the above cases requires defining formal *semantics* for purpose, i.e. defining what it means *to have a purpose* in terms of the artefacts of an information system.

Without formal semantics, as it is the case in most existing models, purposes are merely opaque labels (i.e. character strings) with little or no semantics. Formal definitions are often evaded with the excuse that they are application-specific details beyond the scope of the access control system. The resulting ambiguity of the meaning of purposes makes them prone to arbitrary interpretation that may sometimes be against data subjects' interests and lead to violation of privacy. For example, the privacy policy of a company may state that data collected from customers could be used for the purpose of *web browsing*, which is informally explained as: *information may be exchanged automatically for the purpose of browsing web pages*[56]. With general and ambiguous statements like that, it remains unclear what exactly this purpose entails, how it must be enforced, and under what conditions one can say it is violated. Each company may interpret this in its own way and so, a user who is asked to consent to such a purpose cannot make an informed decision. Similarly, due to this ambiguity, an auditing authority will not be able to accurately decide whether the company has honoured its promise about the purpose of use.

A framework with formal semantics for purpose establishes a unified vocabulary and a clear and granular language among the policy makers, policy enforcers and the auditors so that all parties can have a clear understanding of what purpose-based policies mean, how they must be enforced, and what constitutes a violation.

A central issue in purpose enforcement is to identify the purpose of a requested access, so that it can be verified against the policy rules. Existing models assume that purpose is either declared by the user who requests access, or is decided based on her role or assigned task. Most of these mechanism are insufficient for identifying purpose of access as we will discuss in Section 2.1 and identifying the purpose of an action in an information system has remained a challenge. A formal definition of semantics can result in an efficient method for identifying the purpose of access based on the existing artefacts in an information system, and thereby an effective enforcement mechanism for purpose-based policies.

Once the purpose of a requested access is identified, it must be verified against the purpose-based policies. In most existing models, this comes down to simple string matching, since purposes are treated as character strings. Some of the more advanced models consider a hierarchy of purposes in such verification (see Section 2.1). Formal semantics lead to a formal mechanism for accurately defining under what conditions the purpose of an access complies with the corresponding policies.

1.2 Contributions

Our work contributes a framework for formally expressing and enforcing purpose-based policies. This includes:

- Defining formal semantics for *purpose*, based on the dynamic situation of an action within a network of other related actions in the system and discussing some of its important properties.
- Defining a modal-logic language for formally expressing purpose-based policies based on an abstract model of business processes and proving that the semantics of this language correspond to the semantics of purpose.
- Showing how well-known, as well as some new types of purpose constraints can be formulated using this language and discussing how such constraints can be linked to other access control rules to form more complex policies.
- Developing an algorithm to enforce formally-expressed purpose-based policies in a workflow-based information system and discussing how it can be used to implement different variations of of a *purpose reference monitor*.

1.3 Roadmap

Figure 1.1 shows the roadmap for the rest of this paper.

Section 2	Review of the related work.
Section 3	A walk-through of the framework and the big picture of its components and how it can be used in practice.
Section 4	The conceptual basis for defining semantics of purpose.
Section 5	A formal model of the activities and their relations in an information system based on a formal workflow definition language.
Section 6	A modal logic language for expressing purpose constraints.
Section 7	Discussing different types of purpose constraints, how they can be modelled using the modal logic language, and how they can be tied to broader access control policies.
Section 8	Discussing the algorithm for checking whether a given system complies with given purpose constraints.
Section 9	Discussing how the framework can be implemented and what issues must be considered in the course of an example case.
Sections 10	Concluding remarks and some future work.

Figure 1.1: The roadmap for the rest of this paper.

2

Related Work

2.1 Purpose

Purpose has been mentioned by major privacy legislations, such as Canada's Federal Privacy Act (1983), the U.S. Privacy Act (1974) as well as domain-specific laws and business guidelines and standards, such as the *OECD Guidelines* [55], the Canadian Personal Information Protection and Electronic Documents Act (PIPEDA, 2000), and the U.S. Health Insurance Portability and Accountability Act (HIPAA, 1996). These laws and guidelines and similar ones in other fields and jurisdictions specify restrictions about usage of data that are based on the *purpose of use*. Accordingly, purpose has been considered as a major factor in privacy-aware access control models [76, 39, 18, 54, 11, 27, 53] and policy languages such as P3P [56], EPAL [29], and XACML [72].

The existing models of purpose propose mechanisms for both expressing and enforcing purpose-based policies which we will review in this section. We argue that, comparatively, our framework is more expressive than the existing models, encompasses their features and addresses their shortcomings as mentioned below.

Most existing models simply include purpose as a concept without defining formal semantics. Purposes are often treated as labels (i.e. a character string) that are associated to an action with no or little semantic connection to other entities in the information system. Some models have proposed some form of semantic relation among purposes in the form of a generalization/specialization tree or lattice [14, 18, 67]. Also, some models have suggested semantic connections between purposes and user/roles or activities in the system which will be discussed below.

2.1.1 Expressing Purpose-Based Policies

The existing models for purpose, usually rely on either a *data-centric* or a *rule-centric* approach to expressing purpose-based policies. In a *data-centric* approach, the policy is formed around data, i.e. each data item is associated with the purpose constraints that apply to them [55, 72, 31, 18]. The form of such purpose-constraints is often simple and limited to a black/white list of unauthorized/authorized purposes. Although data-centric policies are perhaps the most commonly used form of purpose-based policies, they are not the only form as we will see in Section 7.2. Moreover, black/white lists are very limited forms of purpose constraints and there are other forms as we will discuss in Section 7.

The *rule-based* approach taken by some models [29, 33, 43] allows defining rules composed of authorized/unauthorized combinations of subjects, resources, actions and purposes. This is a more expressive form of policy which is closer to our approach, although we support more expressive rules that might contain contextual parameters, complex conditions and obligations as in a general access control language (see Section 7.2).

2.1.2 Enforcement

The method by which the purpose of an access control request is identified very much depends on how the semantics of purpose is defined. Since for the most part, formal semantics has not been defined for purposes in the existing literature, this has remained a challenging issue. The existing propositions fall under three categories: *self-declaration*, *user- or role-based*, and *task- or action-based*. We review these approaches briefly.

Self-Declaration

Several authors have proposed that the initiator of an access request declare the purpose for which she or he wishes to access the data her- or himself [32, 44, 66]. This approach is based on trusting the requester to honestly declare their purpose of access and obviously will not prevent a malicious user from circumventing the information's intended purpose by claiming a false purpose.

User- and Role-Based Approach

A greater number of authors have proposed to assign purposes to particular users [25, 77], or to the roles of a role-based access control system [46, 19, 50, 57, 61, 74, 75, 34, 30]. Thus only users with the *marketing* privilege, or who are members of the *marketing* role, for example, are permitted to request access to data for *marketing* purposes. This approach presumes a correspondence between purposes and users/roles which is not necessarily a valid assumption. As we have argued in [43], there are many examples where members of the same role may have different purposes; a user with the *bank teller* or *manager* role, for example, can have different purposes based on different activities that they perform as part of their job. So, this is not a sound approach unless the meaning of *roles* are stretched to model purposes in which case they will no longer be *roles* in the original sense as a set of permissions required in a job function.

Task- and Action-Based Approach

The conceptual link between purposes and actions, which is the basic assumption of our approach, has been observed by a number of researchers. Tschantz *et. al.* define purpose semantics based on a plan of related actions and verify a purpose of an action by how much it contributes to the realization of the purpose of the plan [63, 64]. van Sataden *et. al.* suggest that purpose names can be taken from the *verbs* in a standard dictionary [65]. Similarly, Powers *et. al.* mention that business purposes are a form of high-level action and argue that in high-level privacy policies instead of referring to low-level actions such as *read* or *write*, high-level business purposes such as *treatment* or *diagnosis* are used [59].

In the context of an object-oriented system, Yasuda *et al.* associate purpose with the *calling method* in which context the action is being performed [76]. For example, if the *housekeeping* method of an object of type *Person* calls the *withdraw* method of a *Bank Account* object, it indicates that the money withdrawal is for the purpose of *housekeeping*. This is consistent with our definition of purpose of type A which refers to a higher-level more abstract activity of which the current action is a part (see Section 4).

In their development of a formal semantics for privacy policies, Breaux and Antón propose to model purpose as an auxiliary related action [17]. For example, the policy that *data is collected for the purpose of marketing* is taken to mean the primary action of *data collection* is related to the auxiliary action of *marketing* that happens later. This is very similar to our notion of purpose as a future action. A similar approach by Al-Fedaghi relates purposes to generic data processing activities in the lifecycle of the data [10].

HL7 Reference Information Model ("RIM") specifications, a standard data model used in designing many healthcare systems, mentions the *has-reason* relation between two actions for specifying that one is the *reason* for the other [36]. In this design, *reason* is similar to *purpose* of an action, especially in its sense as a future action. More recent efforts at HL7 to expand support for privacy use this as one of the bases for defining a vocabulary for *purpose of use* [21].

There has been a few proposals [31, 33, 20, 43] that suggest associating purpose with the units of work in a system. They argue that *tasks* or *workflows*, can be used to identify the purpose of an action by looking at the higher-level unit of work in which it takes place. The higher-level unit of work is basically equivalent

to our definition of more abstract actions, so this approach is consistent with the first type of purpose as defined in Section 4. Our work extends these earlier approaches by considering both meanings of purpose as a future action and a more abstract activity, so, it is more general and encompasses these approaches. Moreover, our model also supports multiple purposes for a single action which is a one of the contributions of our work.

2.2 Obligation

A different line of research on *obligations* in access control systems is also concerned about actions and how they are related to future actions (e.g. [35, 38]). This bears some similarity to the subset of our model that deals with purpose as future actions. But our model is also concerned about other relations among actions which are not of interest in the study of obligations. Also, even in the study of future actions we do not follow a strict linear notion of time and our model also considers multiple alternative future actions. As the formal language, we use modal logic to articulate the relations among actions. Temporal logic, which is a type of modal logic, has been used previously to formalize obligations policies [35].

2.3 Workflow

We define the purpose semantics in the context of the actions and activities in an information system. This is usually modelled in the form of business processes and workflows. There are a number of open standards for workflow definition (e.g. BPEL4WS [15] and WS-BPEL [70]) as well as many proprietary workflow languages; van der Aalst and ter Hofstede have reviewed and evaluated a number of these proprietary systems [5].

Using Petri nets as a formal basis for workflows has been the focus of attention since the early nineties [28, 8], especially because of the formal semantics which facilitates simulation, validation and formal analysis of the workflows. Our model is largely based on *YAWL*, a workflow modelling language based on Petri nets [6]. For a survey of the use of Petri nets in modelling business processes see [49].

On the other hand, enforcing purpose-based policies fits in the broader category of *workflow authorization* as it deals with purpose-based constraints in workflows. Although we do not discuss the classic workflow authorization problems in this work, we briefly study the links between purpose-based policies and other workflow authorization policies such as role-based constraints. Workflow authorization is a broad research area and has been studied by many researchers (e.g. [12], [13], and [22]).

Our model of actions also bears similarities to *control flow graph* used to model the network of function calls in programs in programming languages which can be considered as low-level workflows in a program code [9]. Temporal logic, which is a type of modal logic has been used to model control flow policies in a similar way to our framework [45]. Our model also bears some similarities with the hierarchical planning in the artificial intelligence literature [60].

3

Walk-Through: A Medical Research Institute

In this section, we discuss, in the course of an example, the big picture of our proposed framework and its components and describe the steps needed to be taken to formulate and enforce purpose-based privacy policies in practice. We will eventually get back to this example at the end of this paper in Section 9 and discuss further details and implementation-level considerations.

Case: Clinical Research Institute Based on a case we studied and discussed in a previous work, we consider an institute conducting medical research on health records [42]. This organization has several research projects which use the data from the health records residing at a nearby hospital. The practice is called *re-purposing* in which the patients are asked whether or not they agree that their data, originally collected for medical treatment purposes, be used for various clinical research purposes. Records are chosen based on the patient's consent, and also their relevance to the research, depending on the medical conditions and other factors such as age, gender, lifestyle, genetic traits, and the like. Current projects are focused on liver-related diseases such as hepatitis and liver cancer.

Patient consent, which is either an existing electronic document in a local or remote database, or a submitted form upon arrival to the hospital, specifies the policies governing the use of the patient's health records, including the corresponding purpose-based policies. The research institute needs to ensure that aside from complying with the organizational policies, the use of each individual medical record in all stages of research complies with the patient-specific consent.

Step 1: Setting Up the Vocabulary To unify the use of terms across all components of the framework, it is necessary to agree on a common terminology for naming activities and their attributes. This also establishes a common language that connects the activities and their attributes to the policy language. The vocabulary must be developed with the help of domain experts but in some business domains, there are standard vocabularies that enable a common terminology across different systems. SNOMED-CT, for example, is a standard nomenclature for clinical terms which includes an extensive hierarchy of clinical procedures, ranging from *administrative procedures* such as *insurance authorization*, to detailed treatment procedures such as *home visit for intramuscular injection* [62]. Another vocabulary we have used in this paper is the NCI Thesaurus which defines the terminology used in clinical research [52].

In the case of the research institute we study, some of the standard terms are: *translational research*, *human subject research*, *correlative study*, *laboratory test*, *immunologic procedure*, and *hepatitis immunity test*. Note that these labels are not all at the same level of abstraction and, for example, *translational research* is a much broader term than *immunologic procedure*. The vocabulary contains all actions at all different levels of abstraction.

Step 2: Modelling Activities and Their Relations The next step is to define the activities in the system and their relations in the form of business processes. In many organizations, activities are already organized in the form of well-defined business processes which are often modelled and automated as *workflows*; so, this is usually an existing artefact. For modelling the relations among activities we use *hierarchy nets*, a formal model for workflows that supports modelling the hierarchical and sequential relations between activities. This will be discussed in Section 5.

The activities should also be labelled by the terms defined in the vocabulary. For example, a research procedure that performs a correlational study on hepatitis immunology test should be assigned the labels *hepatitis immunity test* and *correlative study*. These labels explain what the task does using the standard vocabulary.

In the research institute example, there are various workflows that are hierarchically organized in the form of different activities in different research projects. Figure 9.4 shows an example of these workflows.

Step 3: Formulating the Policies The policy is a set of rules that restrict purposes of actions. These rules can be expressed as formulas in the language. The atomic propositions are the terms from the vocabulary (shown in italics in the examples below) and the modal operators provide the suitable tools for expressing constraints on purposes. These operators are part of the modal logic language defined in Section 6. A more elaborate discussion of the policy is given in Section 7.

In our running example of the research institute, general policies are set by the authorities and record-specific policies are set by the patients. Some examples of such policies are:

- "*genetic tests* are only allowed for the purpose of *cancer research*."
- "No access for any *research* purposes is allowed to my record."
- "I allow access to my record for the purpose of *liver cancer study*."
- "I do not allow any access to my record for the purpose of *immunological analysis* which is for *research* purposes."

Step 4: Policy Enforcement Having modelled the system and formulated the corresponding policies, our final goal is to ensure that the system complies with the policies. Since policies are formulated using a formal language whose semantics are defined based on the formal model of system activities, the formulas in the language have a clear meaning in terms of the action in the system and their relations. The framework includes a *model-checking algorithm* that can verify whether the system satisfies given policies. This will be discussed in Section 8. More details about the implementation of a model-checking algorithm in the form of an actual *reference monitor* are discussed in Section 9.

4

Conceptual Basis

Purpose is a characteristic attributed to an intentional action and refers the rationale behind it; everyday usages of *purpose* usually presume a sense of teleology concerning the goal pursued by performing an action and its ultimate outcome. One may ask or talk, for example, about the purpose of reading a book, increasing salaries, or collecting health information. The purpose of reading a book may be to entertain, increasing salaries could be done to keep and encourage high-quality workforce, and health information may be collected to perform medical research. To diversifying our vocabulary, in this paper we use the terms, *action*, *activity* and *procedure* synonymously:

Definition 1. Action, Activity, Process: *A low-level or high-level operation in the system.* Activity *and* process *are often used when the operation is more high-level and is composed of other more fine-grained actions. We use* sub-activity *or* sub-process *to refer to such components.*

The use of the word *purpose* in the natural language and examples of purposes mentioned in privacy-related standards and guidelines show that purpose often refers to some high-level activity, for example *website and system administration*, *marketing*, and *research* [56, 21, 73, 26, 40]. The close semantic relation between purpose and action has also been observed by others in the literature as was mentioned in Section 2.1.

More specifically, there are two types of relations between actions that imply a purpose:

Purpose as a High-Level Activity In some contexts, *purpose* refers to a more abstract, semantically higher-level activity, and so, performing an action for some purpose actually means performing it as a part, or a sub-activity, of that higher-level action. For example, when the patient's insurance information is accessed *for the purpose of hospital admission*, it means that this action is a part of the more abstract *hospital admission* procedure which includes several other actions, such as *printing an admission wrist band.* Similarly, when a bank clerk requests to check a customer's credit history *for the purpose of evaluating a loan application*, it implies that checking the credit history is part of the evaluating process for a loan application. We refer to this type of relation as the *A-relation* ('A' for *abstract*).

Sometimes, purpose is used to refer to very general desired states of affairs; in such cases we assume that purpose actually refers to the high-level process of *reaching* that state; for example, doing something for the purpose of *happiness* can be interpreted as doing something as part of the very abstract high-level process of *pursuing happiness.*

Purpose as a Future Action Sometime, purpose indicates that an action is performed as a prerequisite of a future action. For example, when a university professor requests for cash advance *for the purpose of conference registration*, it means that receiving the cash advance is a prerequisite to the action of registering for the conference and it will follow once the cash advance is received. We refer to this relation as the *F-relation* ('F' for *future*).

Thus, the purpose of an action lies in its situation within a larger context containing other related actions which can be modelled as a network of inter-related actions. This is in line with some of the philosophical

literature on the meaning of *intention* which propose that intentions refer to future plans, and an agent's purpose is the basis to explain and predict its future behaviour [16]. On this basis and inspired by the planning literature in artificial intelligence literature, we refer to this as a *plan* of actions [60]. Based on the points mentioned above, we define purpose as follows:

Definition 2. Purpose: *Purpose refers to the attributes of either a higher-level activity of which the action is a* part*, or an activity happening in future for which the action is a* prerequisite.

Later on, in Section 5 and 6, we present a formal model for representing plans and give a formal definition for the *part-of* (A-) and *prerequisite-of* (F-) relations.

For example, in the plan of Figure 4.1, consider action T_{41}, *look up receiver's information*. It is a prerequisite, and therefore for the purpose of *confirming the receiver's identity* (T_{42}). Also, it is a part, and therefore for the purpose of *double-checking receiver's identity* (T_4). Likewise, it is for the purpose of *transfer from card* (T') and *deposit* (T''). Moreover, assuming that *transfer from card* (T') is associated with the attribute *in-branch transactions* (as opposed to an *online transaction* via the online banking system) it can also be said that action *looking up receiver's information* is for the purpose of an *in-branch transaction* since it is part of an activity with that attribute.

4.1 Transitivity and Multiple Purposes

Intuitively, the purpose of a purpose for an action is also its purpose. For example, if the purpose of *studying* is to *pass the exam*, and passing the exam is for the purpose of *getting a degree*, it follows that the purpose of *studying* is also *getting a degree*. This property, straightforwardly holds for purposes resulting from either A- or F- relations since they are both transitive, i.e. a *prerequisite* of a *prerequisite* of p is also a *prerequisite* of p, and a *part of* a *part of* p is also a *part of* p. We will later prove in Theorem 5 that combinations of A- and F-purposes can be reduced to simple A- and F-purposes, and thus, purpose, regardless of its type, is transitive.

As a result, one important feature in our framework is that an action can often be associated with several tangentially-related purposes. For example, in the money deposit example, accessing a customer's information is associated with a number of purposes including: *confirming receiver's identity*, *double-checking receiver's identity*, *transfer from card*, *deposit*, and *in-branch transaction*.

Most of the time, an action has many purposes at different levels of abstraction, since it is part of a plan that in turn leads to, or is part of, other plans. These plans can be traced up to the point that they fall out of the scope of the information system. For example, the *deposit* in the above example might itself be part of the *registration* process for a course, which is in turn a prerequisite for *passing the course*, which is in turn part of the plan for *finishing a university degree* which might be a prerequisite for *getting a well-paid job* and so on. To the best of our knowledge, this feature has been left unnoticed in the existing literature and all of the models for purpose of which we are aware assume a single purpose for each action. Capturing this broader aspect of purposes is one of the contributions of our framework.

4.2 Inferring and Enforcing Purpose

Although initially, purpose of an action is in the agent's mind, it eventually emerges as different cascaded plans and shapes the agent's behaviour and the sequence of activities. Therefore, the set of inter-related actions performed by an agent can disclose its purpose. In other words, zooming out and considering the bigger picture can reveal the *plan*, and thereby the purpose. For example, when the bank teller looks up a customer's information, considering this action alone and in isolation does not reveal the purpose, but looking at the context of the action and the sequence of actions that take place before and after it, can reveal what the purposes are. Conversely, it is possible to *enforce* a purpose by ensuring that the user is committed to a specific *plan*. The purpose of *double checking receiver's identity*, for example, can be enforced by ensuring that the teller follows the process shown in Figure 4.1.

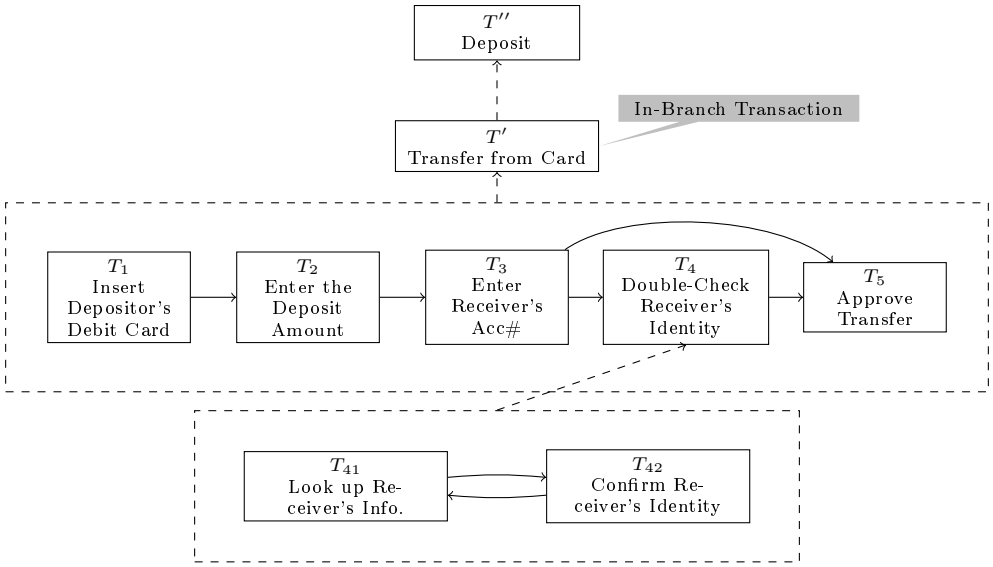


Figure 4.1: The plan for *money deposit* in a bank system. The solid arrows show the order of execution, and the dashed arrows show actions that are part of a more abstract higher-level action.

Generally, in order to enforce purposes, the system must guarantee that a) all authorized activities in the system are defined in the form of plans, and b) access to data is possible only in the course of well-defined plans and no isolated arbitrary action is allowed. This is further discussed in Section 9.

5

Modelling Plans

Organizations usually perform their functions in the form of well-defined activities called *business processes* which are often modelled and automated as *workflows*. Workflow is an automated business process in the form of a sequence of tasks to achieve a business objective [37] which is often defined using a *workflow definition language*. The workflow definitions model a group of activities in the system and their relations, and so, naturally match the plans as we discussed them in Section 4. As we discussed in Section 2, various workflow definition languages exist with different features. Because of its formal basis and wide academic literature, we will rely on a *Petri net-based* language. Our model is largely based on *Yet Another Workflow Language* ("YAWL"), a popular workflow definition language in academia as well as business [6], although we only use a selected subset of the language with direct relevance to our framework for purpose-based policies.

5.1 Petri Nets

Petri net is a formal tool for defining the structure of a network and its dynamics and is used in different fields such as concurrent programming and communications. Petri nets are also suitable for process modelling and formalizing workflows.

The structure of a Petri net is defined in the form of a bipartite graph with two sets of nodes, *places* and *transitions*, and the set of edges that connect them [58]. Places represent the passive parts of the system and therefore hold the system state while transitions indicate its active parts and model how the state may change. The *flow relation* is a set of directed arcs that connect some transitions to some places (transitions' *output places*) and some places to some transitions (transitions' *input places*). The set of input and output places/transitions of a transition/place n are respectively shown as $\bullet n$ and $n\bullet$. Graphically, places and transitions are often shown respectively as circles and squares. The structure of a Petri net is, thus, defined as follows:

Definition 3. *A Petri net is a triple $\langle P, T, F\rangle$ in which:*

- *P and T are respectively the set of* places *and* transitions*, and $P \cap T = \varnothing$.*
- *$F \subseteq (P \times T) \cup (T \times P)$ is the* flow relation.

For a given Petri net $\mathcal{P}$, the set of places, transitions, and the flow relation are respectively shown as $PLAC(\mathcal{P})$, $TRAN(\mathcal{P})$, and $FLOW(\mathcal{P})$.

5.2 Workflow Net (WN)

Petri nets corresponding to a workflow need to have some constraints and some extensions in order to fit the needs of workflow definition. This has led to the definition of *workflow nets* ("WNs"). Our definition here is based on YAWL with some minor modifications [2, 6].

A WN is a Petri net in which transitions correspond to workflow tasks. Transitions in a WN are, therefore, sometimes referred to as *tasks*. Every WN has two special places and two special transitions called *source* and *sink* place and transition, shown as p_c, t_c, p_k, and t_k which correspond to the beginning and end of a workflow. The source place does not have an input and is the only input to the source transition. The sink place does not have any output and is the only output of the sink transition. All nodes (*place* or *transition*) reside on a path from the *source place* to the *sink place*.

Note that any workflow with multiple start/end tasks can be adapted into this format by adding a unique *source*/*sink* transition that is connected respectively to all the start/end tasks via a number of intermediate places.

Every transition in a WN is assigned a *split* and a *join type* which specify its behaviour respectively when there are more than one outgoing or incoming paths [68]. An *AND-split* is used for *parallel routing* and causes all the following paths to be activated concurrently, whereas an *XOR-split* is used for *selective routing* and results in activation of only one of the following paths. Likewise, an *AND-join* causes the transition to wait for all preceding paths before activation while a transition with an *XOR-join* is activated after the conclusion of any of its predecessors. The split/join type is immaterial where there is only one outgoing/incoming path from/to the transition. As a default, we assume the AND type in such cases. We do not discuss the case of free OR splits/joins as they can be reduced to a combination of AND and XOR splits/joins. More details about run-time behaviour of joins and splits will be given in Section 5.4.

Following the graphical notation of YAWL [2, 6], we show AND-splits as  , XOR-splits as , AND-joins as  , and XOR-joins as .

On this basis, a WN is formally defined as follows:

Definition 4. *Workflow Net (WN): a triple $\langle P, S, J\rangle$ in which P is a Petri net with the following properties, and S and J are mappings from transitions to their split/join types, i.e. $TRAN(P) \mapsto \{XOR, AND\}$.*

1. *There exist two special* places, *p_c, p_k, and two special* transitions, *t_c, t_k such that: $\bullet p_c = \varnothing$, $p_c\bullet = \{t_c\}$, $\bullet t_c = \{p_c\}$, $p_k\bullet = \varnothing$, $\bullet p_k = \{t_k\}$, and $t_k\bullet = \{p_k\}$.*
2. *With the exception of source and sink places, every place has exactly one input and one output.*
3. *Every transition t and place p reside on a path from p_c to p_k.*
4. *There is no unstructured loops in P (defined below).*

Besides the *structural properties* mentioned above, a WN that correspond to a valid workflow must also have some *dynamic properties* which are discussed in Section 5.6.2.

5.2.1 Structured Loops

Loops are necessary constructs in modelling activities and thus, a workflow language must support them. On the other hand, allowing arbitrary loops leads to difficulties and complications in workflow analysis often without actually adding much practical value to the expressive power of the language.

To address this issue, we choose the practical approach of supporting only *structured loops* (also known as *structured cycles* [7]), a loop with single well-defined *entry* and *exit*. This type of loops which correspond to structured loops in programming languages (such as *while*, and *for*; as opposed to *goto*) are sufficient for modelling most workflows; moreover, arbitrary loops can almost always be converted to structured loops. An elaborate study of this conversion problem and the loops which cannot be converted has been done in [47].

As shown in Figure 5.1, a *structured loop* O is formed by adding a *loop place* p_l to a workflow net w via which the sink transition of w (t_k) is connected to its source transition (t_c), i.e. $\bullet p_l = t_k$ and $p_l\bullet = t_c$. We refer to this path as the *loop path* or *return path* of the loop, as opposed to $t_k p_k$ which is the loop's *exit path*. Also, the split type of the sink transition and the join type of the source transition are changed to XOR which enables the execution of the source transition after the sink transition. The loop can be connected to the rest of the workflow using the source and sink place.

Figure 5.1: Constructing a *structured loop* (right) based on a workflow net (left).

Note that this definition is recursive, i.e. the underlying WN on which basis the loop is constructed may contain other structured loops. Also, note that this definition simulates a *repeat-until* loop –which we have chosen for its simplicity– but can straightforwardly be used to implement a *while-* or *for-*like loop as well.

5.3 Hierarchy Nets (HN)

Workflow nets are suitable for modelling individual workflows but they cannot model the *part-of* relations where a high-level activity is broken down to its sub-activities, like the dashed lines in Figure 4.1. To cover this, we define *hierarchy nets* ("HNs"), based on the similar concepts in YAWL.

A *hierarchy net*, $\mathcal{H}$, is composed of a set of WNs, W, together with the hierarchy relation, H, which refines abstract transitions (a.k.a *composite*), in the form of lower-level workflow nets (a.k.a. *sub-nets*). For example, in Figure 4.1, the composite task T_4 is a decomposed into the sub-net containing T_{41} and T_{42}.

In each hierarchy net, there is one single most high-level WN, known as the *root*, which is not a sub-net of any other tasks and is shown as R. The hierarchy relation maps every composite task to a distinct WN (injective) and, except the root, leaves no WNs out (surjective). Moreover, it is free of circular mappings and has a tree structure. Note that for simplicity and without loss of generality, we only allow one sub-net for each composite task. Should a composite task need to be mapped to multiple sub-nets, they can be modelled as a single WN made of AND-ing or XOR-ing all of them.

The set of all transitions of a hierarchy net $\mathcal{H}$ is shown as $\mathcal{T}_{\mathcal{H}}$, or $\mathcal{T}$ where the context is clear. Similarly, the set of all places in $\mathcal{H}$ is shown as $\mathcal{P}_{\mathcal{H}}$, or $\mathcal{P}$. We also use $\mathcal{T}_{\mathcal{H}}^{AND}$ and $\mathcal{T}_{\mathcal{H}}^{XOR}$ to show to the set of all transitions with the split type AND and XOR respectively. Similarly, ${}^{AND}\mathcal{T}_{\mathcal{H}}$ and ${}^{XOR}\mathcal{T}_{\mathcal{H}}$ are used for joins.

A hierarchy net $\mathcal{H}$ is formally defined as:

Definition 5. Hierarchy Net: *A hierarchy net is a triple* (W, R, H)*, where:*

- W *is a set of WNs,*
- $R \in W$ *is the most high-level workflow or the* root*,*
- $H : \mathcal{T} \rightarrow W \backslash \{R\}$*,* hierarchy function*, is a bijective function which maps all composite transitions onto low-level workflow nets. Moreover, the following relation is a directed tree:* $\{(w_1, w_2) \in W^2 | \exists t \in TRAN(w_2).(t, w_1) \in H\}$.

5.3.1 Policy Components of a Hierarchy Net Definition

A workflow definition usually includes some constraints on activities such as authorized roles, input data type, and temporal/spatial constraints for the execution of each task. We do not include these in the workflow definition and consider them as part of the access control policy. Such constraints can be modelled as a set of *action-centric* authorization policies, which may also be linked to purpose constraints, as discussed in Section 7.2.

5.4 Operational Semantics

The state of a Petri net is defined based on assigning abstract objects called *tokens* to its *places* and is also known as a *marking*. For a Petri net with n places, $p_1 \cdots p_n$, a marking μ is shown as an n-tuple $\langle x_1 \cdots x_n \rangle$ in which x_is represent the number of tokens residing at p_i.

Operational semantics of a Petri net are defined based on *firing* of transitions. A transition can be fired when it is *enabled*, i.e. when there is tokens in its input places. When fired, a transition removes tokens from its input places and produces tokens at its output places which results in a new marking. This is shown as: $\mu_j \xrightarrow{t} \mu_{j+1}$ in which t is the fired transition and μ_j and μ_{j+1} are the marking of the Petri net respectively before and after the firing.

Firing a transition in a HN is equivalent to the execution of the corresponding task in the workflow. An XOR-join transition is enabled when there is a token in one of its input places; an AND-join transition, however, is enabled when all of its input places contain a token. When fired, an XOR-split transition creates a token in only one of its output places, whereas an AND-split transition creates a token in all of its output places. If at a given marking, multiple transitions are enabled, any one of them may be fired and the order is not deterministic.

The execution of a WN starts from the *initial marking* in which a token resides in its *source* place enabling the *source* transition. After firing the source transition, the execution continues by firing other transitions based on the semantics of AND and XOR splits and joins as defined above. A finite execution concludes at the *final marking*, when a token arrives at the *sink* place –right after firing the *sink* transition. Note that, as it will be discussed in Section 5.6.1, we only consider the finite firing sequences.

In a given execution of a WN, the sequence of transitions fired between the initial and final markings is called a *firing sequence*. The firing sequence $q = t_1 \cdots t_n$, for examples, corresponds to:

$$\mu_1 \xrightarrow{t_1} \mu_2 \xrightarrow{t_2} \cdots \mu_n \xrightarrow{t_n} \mu_{n+1}$$

in which μ_1 is the initial marking and μ_{n+1} is the final marking. We refer to the sequence $\mu_1 \cdots \mu_{n+1}$ as the *marking sequence* corresponding to q and show it as $M(q)$.

We use Q_w to show the set of all possible firing sequences of a given workflow net w. This is sometimes called the *language* of the Petri net, since it is the set of all possible strings generated by the Petri net if a symbol is assigned to each of its transitions [51]. Note that the existence of AND or XOR splits results in different possible firing sequences that in case of XOR-split, capture different choices, and in case of AND-split, different orders of execution.

5.4.1 Operational Semantics of a HN

We define the operational semantics of a HN based on its *extension*. The extension of a HN is a WN in which composite tasks are expanded to their sub-nets to reflect the run-time execution of the tasks. It can be straightforwardly shown that the extension of a hierarchy net is a workflow net. Running a HN is defined as running its extension, according to the operational semantics of workflow nets discussed above.

Given a hierarchy net $\mathcal{H}$, its extension is shown as $EXT(\mathcal{H})$ and is made by running the following steps for every composite task t that is refined to sub-net w. Figure 5.2 shows an example of a HN and its extension.

- Two new auxiliary tasks t^e and t^x and two auxiliary places p^e and p^x are added to the HN which respectively indicate *entry* to, and *exit* from t. These are similar to *call* and *return* instructions in programming languages.
- Transition t and its sub-net w are added as parallel paths between t^e and t^x, i.e. t is connected to t^e as output, and to t^x as input, via two auxiliary places. Also, the source place of w is connected as output to t^e and its sink place is connected as input to t^x. The split-type t^e of and join-type of t^x is AND.
- The network between t^e and t^x replaces t in the WN, i.e. all the input places of t are connected as input to t^e and all its output places are connected as output to t^x. Also, the join-type of t^e and the split-type of t^x is the same as t.

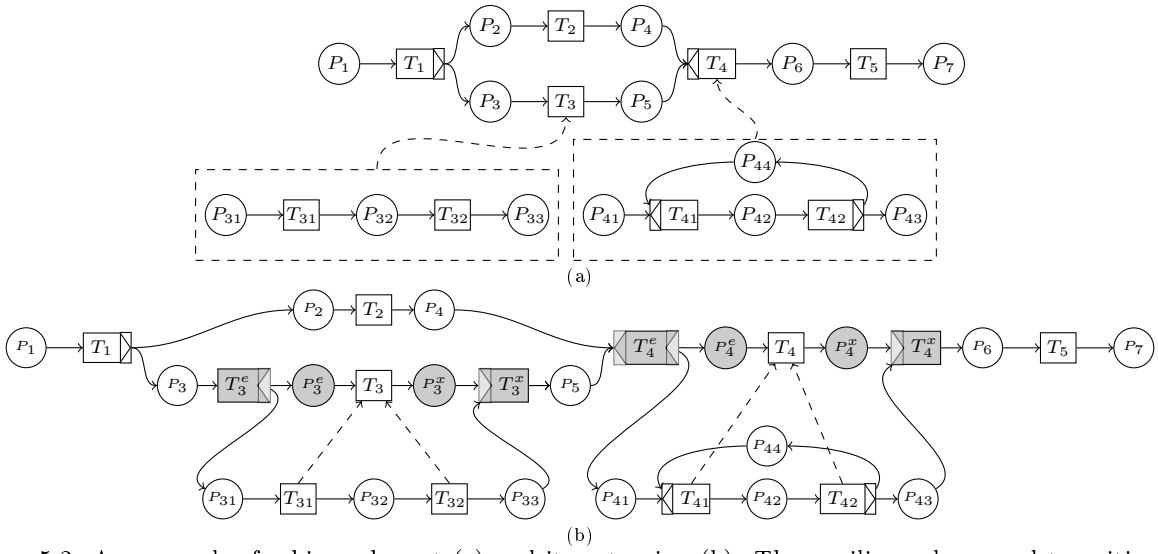


Figure 5.2: An example of a hierarchy net (a) and its extension (b). The auxiliary places and transition are shaded and the A-relation is shown with dashed lines.

Formally, if we use $C_p(w)$ and $K_p(w)$ to respectively refer to the source and sink places of w:

$$
\begin{aligned}
PLAC(EXT(\mathcal{H})) &= PLAC(\mathcal{H}) \cup \{p^e, p^x\} \\
TRAN(EXT(\mathcal{H})) &= TRAN(\mathcal{H}) \cup \{t^e, t^x\} \\
\mathcal{T}^{AND}_{EXT(\mathcal{H})} &= \mathcal{T}^{AND}_{\mathcal{H}} \cup \{t^e\} \\
{}^{AND}\mathcal{T}_{EXT(\mathcal{H})} &= {}^{AND}\mathcal{T}_{\mathcal{H}} \cup \{t^x\} \\
FLOW(EXT(\mathcal{H})) &= FLOW(\mathcal{H}) \backslash \{(u,v) | u = t \vee v = t\} \cup \\
&\quad \{(t^e, p^e), (p^e, t), (t, p^x), (p^x, t^x), (t^e, C(w)), (K(w), t^x)\} \cup \\
&\quad \{(u, t^e) | u \in {\bullet t}\} \cup \{(t^x, v) | v \in t{\bullet}\} \\
t \in \mathcal{T}^{AND}_{\mathcal{H}} &\implies t^x \in \mathcal{T}^{AND}_{EXT(\mathcal{H})} \\
t \in \mathcal{T}^{XOR}_{\mathcal{H}} &\implies t^x \in \mathcal{T}^{XOR}_{EXT(\mathcal{H})} \\
t \in {}^{AND}\mathcal{T}_{\mathcal{H}} &\implies t^e \in {}^{AND}\mathcal{T}_{EXT(\mathcal{H})} \\
t \in {}^{XOR}\mathcal{T}_{\mathcal{H}} &\implies t^e \in {}^{XOR}\mathcal{T}_{EXT(\mathcal{H})}
\end{aligned}
$$

5.5 Structural Properties

A plain Petri net does not support split- and join-types and its operational semantics are defined so that a transition is enabled if there is at least one token in each of its inputs, and after firing the transition, a token is produced in all its outputs. In other words, the operational semantics of plain Petri nets are defined as if all splits and joins are of type AND. For analysing the structural properties of WNs, and in order to be able to use the results already studied for plain Petri nets in the literature, we can convert a WN to an equivalent plain Petri net with no splits or joins of type XOR, as shown in Figure 5.3.

It is easy to see that the resulting Petri net from converting a WN according to such process is a special type of Petri net known as *free-choice* Petri net. A *free-choice* Petri net is a type of Petri net in which for any two transitions t_1 and t_2, ${\bullet t_1} \cap {\bullet t_2} \neq \emptyset$ implies ${\bullet t_1} = {\bullet t_2}$. Considering Figure 5.3, the only case where transitions may share inputs is the case for an XOR-split transition in which case all such transitions share the same set of inputs.

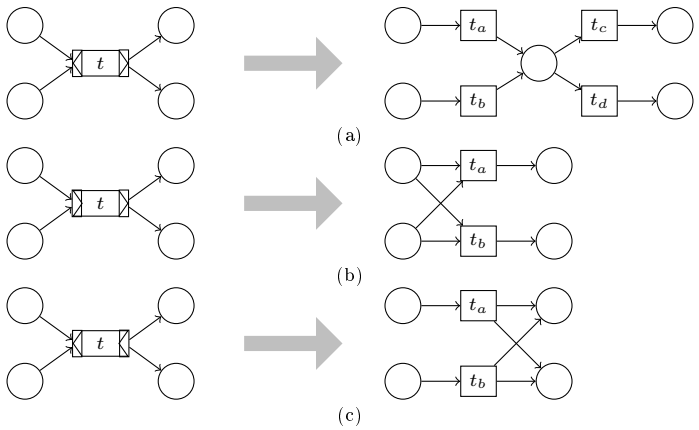


Figure 5.3: Converting (a) XOR joins and split, (b) AND join and XOR split, and (c) XOR join and AND split to plain Petri net structures (adapted from [2]). The case for n predecessor/successor is similar.

This structural property has some major results about the complexity and the dynamic properties of WNs as we will discuss below.

5.6 Dynamic Properties

Besides the structural properties mentioned in Section 5.2, there are some constraints about the dynamic behaviour of a valid WN.

5.6.1 Finiteness

A loop can potentially cause infinite firing sequences should it take the loop path for an infinite number of times and never leave it by taking the exit path. We assume, on the other hand, that *structured loop*s in WNs eventually take their exit path after a finite number of iterations. As a result, all firing sequences of a WN are finite.

5.6.2 Soundness

A *sound* WN is possible to complete, terminates properly, and does not include dead tasks [2, 4]:

- *Possibility of completion:* Starting from the initial marking, there is always a firing sequence that leads to the final marking.
- *Proper termination:* When the workflow reaches its final marking (i.e. there is a token in p_e) there must not be any token in any other places.
- *No dead transition:* For every transition t, there exists a firing sequence from the initial marking that leads to a marking where t is enabled.

Note that *possibility* of completion does not guarantee completion for every possible firing sequence; an infinite firing sequence is still possible since a particular instance of a WN may never choose the path to completion. But as we mentioned above, we additionally assume that a all WNs eventually terminate, i.e. all of their firing sequences are finite.

Also, note that the structural property that all the tasks be on a path from p_c to p_k does not guarantee that there is no dead transitions since the way splits and joins are designed might cause a task to be impossible to run although it statically resides on a path from p_c to p_k.

Intuitively, soundness requires some sort of balance between the joins and splits, so that every time some parallel paths are started using an AND split, they must be synchronized with an AND-join, and likewise for the XOR splits. An imbalance between split and join types leads to violation of the *proper termination* property, for example, if an AND split is not balanced with an AND join, one of the parallel paths may get to the sink place of the workflow net while the other paths are still in progress, leading to an improper termination.

It can be shown straightforwardly by induction that if all the workflow nets in a HN are sound, its *extension* is sound.

Moreover, it has been shown that if a Petri net is *free-choice*, its *soundness* can be verified in polynomial time [1]. Further analysis and results about the *soundness* property can be found in [4].

5.6.3 Persistence

The second property of WNs, makes a workflow definition similar to a *marked graph*, a type of Petri net in which there is only one incoming and one outgoing edge to and from each place [51]. This causes WNs to have an important run-time property that if a number of transitions are enabled at a marking, firing one of them does not disable others, i.e. once a transition is enabled, it is removed from the set of enabled transitions only when it is fired. This property is called *persistence* and can be proved straightforwardly by noticing that in order that an already-enabled transition t' be disabled after firing t, firing t must remove a token from one of the inputs of t'. Since firing t can only remove tokens from the inputs of t it follows that t and t' share an input place which contradicts the second property of WNs according to their definition.

5.6.4 Safety

Safety is a dynamic property of a Petri net that ensures that in all possible markings, there is at most one token in every place. An important property of *free-choice* Petri nets is that their *soundness* implies *safety*. For a proof, see [1].

5.7 Example

Figure 5.2 shows an example of a hierarchy net with two composite tasks each of which are expanded to a sub-net and the corresponding extension. Some of the possible firing sequences for this net are as follows.

$$T_1T_2T_4^eT_4T_{41}T_{42}T_4^xT_5$$
$$T_1T_3^eT_{31}T_{32}T_3T_3^xT_4^eT_4T_{41}T_{42}T_4^xT_5$$
$$T_1T_3^eT_{31}T_3T_{32}T_3^xT_4^eT_{41}T_4T_{42}T_{41}T_{42}T_4T_4^xT_5$$

5.8 Composition of Sound Workflows

Since the dynamic properties of WNs are not defined based on its structure, in this section we discuss an inductive compositional definition for a class of Petri nets which are always guaranteed to be valid WNs, i.e. satisfy both the required structural and dynamic properties. The proof for such result will be straightforward using structural induction. We call this class $\mathcal{W}$.

Note that this process is not comprehensive, i.e. there are sound WNs that are not in $\mathcal{W}$, but the constructs provided for making workflows in $\mathcal{W}$ cover most common patterns in programming and workflow design and thus give a basic guide for designing valid WNs.

We define the *atom workflow net*, shown as w_a, as the simplest WN with a single transition t and two places p_1 and p_2 such that $FLOW(w_a) = \{(p_1, t), (t, p_2)\}$. Using C_T, K_P, C_P, and K_T to indicate the *source* and *sink* transitions and places of a WN, we have: $C_T(w_a) = K_T(w_a) = t$, $C_P(w_a) = p_1$, and $K_P(w_a) = p_2$.

The composition is based on 5 operators, defined in the rest of this section, each of which can be applied to one or two WNs to create a more complex WN. $\mathcal{W}$ is inductively defined as the minimal set such that:

$$w_a \in \mathcal{W} \quad \text{(the atom workflow net)}$$
$$w_1 \ \in \mathcal{W} \geq 2 \implies \lambda(w_1) \in \mathcal{W} \quad \text{(loop)}$$
$$\{w_1, w_2\} \subseteq \mathcal{W} \implies \rho(w_1, w_2, t) \in \mathcal{W} \ (t \in TRAN(w_1)) \quad \text{(refine)}$$
$$\{w_1, w_2\} \subseteq \mathcal{W} \implies \sigma(w_1, w_2) \in \mathcal{W} \quad \text{(sequence)}$$
$$\{w_1, \cdots, w_n\} \subseteq \mathcal{W} \implies \pi(w_1, \cdots, w_n) \in \mathcal{W} \quad \text{(parallel)}$$
$$\{w_1, \cdots, w_n\} \subseteq \mathcal{W} \implies \chi(w_1, \cdots, w_n) \in \mathcal{W} \quad \text{(choice)}$$

5.8.1 Loop (λ)

This operator makes a loop of the given WN as defined in Section 5.2.1 and then adds a new pair of source and sink transitions (t_c and t_k) and places (p_c and p_k). The reason for adding this additional nodes is to make the resulting network fit the definition of the WN (with distinct source and sink places and transitions).

$$PLAC(\lambda(w)) = PLAC(w) \cup \{p_c, p_k, p_l\}$$
$$TRAN(\lambda(w)) = TRAN(w) \cup \{t_c, t_k\}$$
$$FLOW(\lambda(w)) = FLOW(w) \cup \{(p_c, t_c), (t_c, C_P(w)), (K_P(w), t_k), (t_k, p_k), (K_T(w), p_l), (p_l, C_T(w))\}$$
$$C_T(\lambda(w)) = t_c, C_P(\lambda(w)) = p_c$$
$$K_T(\lambda(w)) = t_k, K_P(\lambda(w)) = p_k$$

5.8.2 Refine (ρ)

This operators refines the transition t in w_1 to the workflow net w_2. The structure is similar to what we discussed in Section 5.4.1.

5.8.3 Sequence (σ)

Sequencing two WNs is putting one after the other, so that the sink place of w_1 is merged with the source place of w_2.

$$PLAC(\sigma(w_1, w_2)) = (PLAC(w_1) \cup PLAC(w_2)) \backslash \{C_P(w_2)\}$$
$$FLOW(\sigma(w_1, w_2)) = (FLOW(w_1) \cup FLOW(w_2) \cup \{(K_P(w_1), C_T(w_2))\}) \backslash \{(C_P(w_2), C_T(w_2))\}$$
$$C_T(\sigma(w_1, w_2)) = C_T(w_1), C_P(\sigma(w_1, w_2)) = C_P(w_1)$$
$$K_T(\sigma(w_1, w_2)) = K_T(w_2), K_P(\sigma(w_1, w_2)) = K_P(w_2)$$

5.8.4 Parallel (π)

Paralleling a number of WNs means putting them in parallel branches enclosed between two new transitions t_c and t_k where split type of t_c and the join type of t_k is AND. The source and sink transition for the resulting WN are t_c and t_k and a new pair of places p_c and p_k are also added which will be the source and

sink places of the new WN. Let $w' = \pi(w_1, \cdots, w_n)$

$$PLAC(w') = \bigcup_{i=1}^{n} PLAC(w_i) \ \cup \{p_c, p_k\}$$
$$TRAN(w') = \bigcup_{i=1}^{n} TRAN(w_i) \ \cup \{t_c, t_k\}$$
$$FLOW(w') = \bigcup_{i=1}^{n} FLOW(w_i) \cup \bigcup_{i=1}^{n} \{(t_c, C_P(w_i))\} \cup \bigcup_{i=1}^{n} \{(K_P(w_i), t_k)\} \cup \{(p_c, t_c), (t_k, p_k)\}$$
$$C_T(w') = t_c, C_P(w') = p_c$$
$$K_T(w') = t_k, K_P(w') = p_k$$

5.8.5 Choice (χ)

The structure of the choice operator is completely similar to the parallel with the exception that the split/join type between the branches is XOR.

5.9 A- and F- Relation

The two purpose relations introduced in Section 4 can be directly recognized based on the definition of a workflow net: the F-relation corresponds to the flow relation of the Petri net that sequences tasks in a certain order, and the A-relation corresponds to the hierarchy relation that expands composite tasks to sub-nets.

One consideration is about the structured loops. The return path for a loop can hardly indicate a purpose relation; for example, consider the loop in the example of Figure 4.1: although it makes sense to assume *looking up receiver's info* is for the purpose of *confirming the receiver's identity*, the reverse, i.e. to assume that *confirming the identity* is for the purpose of *lookup*, does not seem to make much sense. Moreover, assuming that the return path of the loop indicates a *prerequisite* (F-) relation causes a conceptually counter-intuitive case of circular prerequisites where the sink transition is a prerequisite for the source transition (p_c is for the purpose of p_k) while the source transition is also a prerequisite for the sink transition (p_k is for the purpose of p_c). On this basis, we believe that the return path of a loop merely indicates a temporal precedence and not a prerequisite relation with purpose implications.

Assuming $O(\mathcal{P}_\mathcal{H})$ is the set of all loop places for all the structured loops in the HN, the F-relation is formally defined based on the extension of $\mathcal{H}$ as follows. We use $F_\mathcal{H}$ for the F-relation of the hierarchy net $\mathcal{H}$:

Definition 6. F-Relation: *the F-relation connects every pair of transitions that are connected to each other via a non-loop place in the extension of $\mathcal{H}$:*

$$F_\mathcal{H} = \big\{(t_1, t_2) \in \mathcal{T}_\mathcal{H}^2 \mid \exists p \in PLAC(EXT(\mathcal{H})).\ p \notin O(\mathcal{P}_\mathcal{H}) \ \wedge \ \{(t_1, p), (p, t_2)\} \subseteq FLOW(EXT(\mathcal{H}))\big\} \quad (5.9.1)$$

The A-relation is defined for the hierarchy net $\mathcal{H} = \langle W, R, H\rangle$ as follows. We use $A_\mathcal{H}$ to refer to the A-relation of the hierarchy net $\mathcal{H}$:

Definition 7. A-Relation: *the A-relation maps every transition of a sub-net to the composite transition it expands.*

$$A_\mathcal{H} = \{(t_1, t_2) \in \mathcal{T}_\mathcal{H}^2 \mid \exists w \in W;\ t_1 \in TRAN(w) \wedge (t_2, w) \in H\}. \quad (5.9.2)$$

Note that the direction of A is the opposite to that of H, i.e. the edges start from the low-level transitions.

The A- and F-relations can be thought of as the edges of a graph which defines the relations between the actions in a workflow. This graph, which can be extracted from workflow definitions is equivalent to the *action graph* as defined in [41].

6

The Modal Logic Language

Modal logic languages include *quantifiers* (or synonymously, *operators*) to particularize an assertion to specific modes such as the *where*, *when*, or *how* of the assertion. For example, *temporal* quantifiers modify the meaning of a proposition to specific times.

Modal logic languages are usually defined based on the *Kripke semantics* [48] which assumes the universe is composed of a number of *possible worlds* (also referred to as *states* or *nodes*), some of which are reachable from some others as defined by an *accessibility relation*.

A workflow definition happens to perfectly match this pattern: it gives a model of the world in which workflow tasks correspond to different possible worlds, and the F- and A-relations define different types of accessibility between them. Moreover, typical modal logic operators seem to be suitable for modelling reachability properties which are closely related to the meaning of purpose as discussed in Section 4. These reasons justify our decision to choose modal logic as the basis of a language for expressing rules about purposes of actions. In the rest of this section we discuss the modal logic language and its formal syntax and semantics.

6.1 Types of Modality

The language we define is based on three different *types* of modality each with two possible modes. Accordingly, every modal operator in the language has three semantic elements which specify the mode of the operator with respect to each of the three modality types (see Figure 6.1). This leads to 2^3 possible operators, however, only six of them are meaningful as we will discuss later and shown in Figure 6.2.

A- and F-Particularity This type of modality quantifies whether the assertion is based on A- or F-accessibility. An assertion based on F-relation refers to the future states while an assertion based on A-relation refers to the broader context in which an action takes place. For example, the assertion *in-branch transaction*, when quantified by F, means an in-branch transaction will follow in *future*, and when quantified by F, means we are currently *part of*, or *in the course of* an in-branch transaction.

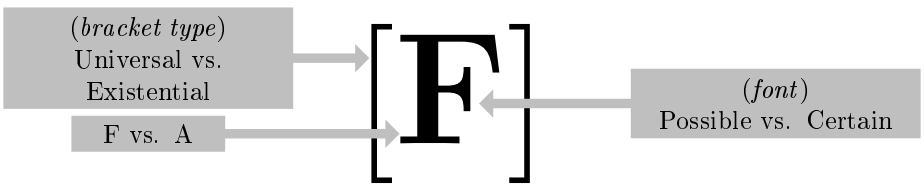


Figure 6.1: The notational convention for the modal operators.

	Possible		*Certain*	
	A	*F*	*A*	*F*
Universal	-	$[\mathbb{F}]$	$[\mathbf{A}]$	$[\mathbf{F}]$
Existential	-	$\langle\mathbb{F}\rangle$	$\langle\mathbf{A}\rangle$	$\langle\mathbf{F}\rangle$

$[\mathbf{F}]$	$\langle\mathbb{F}\rangle$
$[\mathbb{F}]$	$\langle\mathbf{F}\rangle$
$[\mathbf{A}]$	$\langle\mathbf{A}\rangle$

Figure 6.2: Summary of the modal operators and types of modalities covered in the language (left) and the dual operators (right).

Possibility vs. Certainty As a result of non-determinism in the workflow model about the future tasks, an assertion about the future can be quantified based on its certainty. A *certain* assertion is guaranteed to be true regardless of the non-deterministic nature of the future whereas a *possible* assertion may be true or false –but not guaranteed to be false. For example, in an academic journal publication workflow, eventual publication of the paper is a possibility while sending a notification email to the author is certain. Note that these two modes are the dual of one another, i.e. that an assertion is *possibly* true means its negation is not *certain*. We use bold typeface (like $\mathbf{F}$) for representing certainty and blackboard bold typeface (like $\mathbb{F}$) for possibility.

Note that in the case of F-relation, the existence of XOR-splits introduces some non-determinism in the progression of workflows, since when an XOR-split is encountered the workflow may proceed to any of the following branches thereby making the exact run-time sequence of transitions indeterminable. Therefore, F-operators can be of either *certain* or *possible* type. Since such non-determinism does not exist in the case of the A-relation, there is no *possible* A-operators.

Existentiality vs. Universality The span of the truth of an assertion is another type of modality, i.e. whether an assertion is true at all states or only in some. A *universal* assertion is true *for all* states, while an *existential* assertion is true at *some* states, i.e. *there exists* some state at which it is true. For example, in a clinical workflow for a lab test it might be true that *all* activities are done within the geographical location of the clinic, but in a clinical consultation workflow involving an external clinic, *some* activities happen outside the clinic's premises. Note that these two concepts are the dual of one another, i.e. that an assertion is *existentially* true means it is not *universally* false. We represent existential operators with angle- (like $\langle\rangle$) and universal operators with square-brackets (like $[]$).

6.1.1 Common Types of Assertion

Of all the modal quantifiers in the language shown in Figure 6.2 which are created by combining different types of modalities, some of them are more important in formulating purpose-based policies which are the focus of this paper:

Inevitably: This type of assertion says that something is *certainly* true at *some* following state. This type of assertion, which is modelled by $\langle\mathbf{A}\rangle$ and $\langle\mathbf{F}\rangle$ operators, can be used to model purpose requirement as discussed further in Section 7.1.1.

Invariably: This type of assertion says that something is *certainly* true *for all* following states. Purpose prohibitions are assertions of this type for they require that the negation of some purpose *certainly* holds true in *all* following states. This is further discussed in Section 7.1.2. The $[\mathbf{F}]$ and $[\mathbf{A}]$ operators can model this kind of assertion.

Potentially: This type of assertion says that something is *possibly* true at *some* following state. Intuitively, this is the dual of *invariably*, for when an assertion is not guaranteed to be true in all following states, it means that its negation is possible to happen at some point. So, purpose prohibitions can also be expressed as negation of this type of assertions which can be modelled by $\langle\mathbb{F}\rangle$ and $\langle\mathbf{A}\rangle$ operators.

Potentially Always: This type of assertion says that something is *possibly* true at *all* following states. Intuitively, this is the dual of *inevitably*, for when an assertion is not guaranteed to be true at some point, it means that there is the possibility that its negation be the case at all following states. For example, if *medical treatment* cannot be guaranteed to follow as a purpose, it means that $\neg$(*medical treatment*) is *possibly* true in *all* following states. The semantics of $[\mathbb{F}]$ and $[\mathbf{A}]$ match this type of assertions.

6.2 Definition of the Language

Formal semantics for the language define the truth value of every formula based on a model. We define the *model* to be a workflow definition given in the form of a hierarchy net, together with a labelling function, L that gives the truth or falsehood of atomic propositions at each transition. This is given as a function that maps each atomic proposition p to $L(p)$, the set of transitions at which it holds true.

We define the truth value of a formula ϕ by defining the set of transitions at which ϕ holds true, in the context of the hierarchy net $\mathcal{H}$ and the labelling function L; this is shown as $[\![\phi]\!]_{\mathcal{H},L}$, or $[\![\phi]\!]$ where the context is clear.

In the context of $\mathcal{H}$ and L, we say a transition t *satisfies* the formula ϕ if and only if $t \in [\![\phi]\!]_{\mathcal{H},L}$. Similarly, in the context of L, a hierarchy net $\mathcal{H}$ *satisfies* ϕ if and only if:

$$\forall t \in \mathcal{T}_{\mathcal{H}},\ t \in [\![\phi]\!]_{\mathcal{H},L}$$

In this section, we first give the formal definition of the syntax and semantics of the language and then present further discussion about the semantics. Finally, we show that the semantics for the language correspond to the semantics of purpose and the language in fact can express rules about purposes as defined in Section 4.

6.2.1 Formal Definition

The syntax is defined as the following context-free grammar, in which p is any atomic proposition:

$$\phi ::= \top \mid p \mid \neg\phi \mid \phi \wedge \phi \mid \langle\mathbf{A}\rangle\phi \mid \langle\mathbf{F}\rangle\phi \mid \langle\mathbb{F}\rangle\phi \tag{6.2.1}$$

We assume that unary operators take precedence over binary operators and we use parentheses where necessary.

In the context of the hierarchy net $\mathcal{H}$, whose set of all transitions is referred to as $\mathcal{T}$, $[\![\phi]\!]$ is defined inductively as the smallest *fixed point* satisfying the following. The *PRE* functions are defined later in this section:

$$[\![\top]\!] = \mathcal{T}. \tag{6.2.2}$$
$$[\![p]\!] = \{t \in \mathcal{T} \mid p \in L(t)\}. \tag{6.2.3}$$
$$[\![\neg\phi]\!] = \mathcal{T} \backslash [\![\phi]\!]. \tag{6.2.4}$$
$$[\![\phi_1 \wedge \phi_2]\!] = [\![\phi_1]\!] \cap [\![\phi_2]\!]. \tag{6.2.5}$$
$$[\![\langle\mathbf{A}\rangle\phi]\!] = [\![\phi]\!] \cup PRE_A([\![\langle\mathbf{A}\rangle\phi]\!]). \tag{6.2.6}$$
$$[\![\langle\mathbf{F}\rangle\phi]\!] = [\![\phi]\!] \cup PRE_F([\![\langle\mathbf{F}\rangle\phi]\!]). \tag{6.2.7}$$
$$[\![\langle\mathbb{F}\rangle\phi]\!] = [\![\phi]\!] \cup PRE_F^{\exists}([\![\langle\mathbb{F}\rangle\phi]\!]). \tag{6.2.8}$$

Note that as we will show in Section 8.3, such fixed point exists. For further discussion of fixed points see [69].

The following derived forms are defined to facilitate expressing more complex formulas:

$$\bot \stackrel{def}{=} \neg\top \tag{6.2.9}$$

$$\phi_1 \vee \phi_2 \stackrel{def}{=} \neg(\neg\phi_1 \wedge \neg\phi_2) \tag{6.2.10}$$

$$\phi_1 \rightarrow \phi_2 \stackrel{def}{=} \neg(\phi_1 \wedge \neg\phi_2) \tag{6.2.11}$$

$$[\mathbf{A}]\phi \stackrel{def}{=} \neg\langle\mathbf{A}\rangle(\neg\phi) \tag{6.2.12}$$

$$[\mathbf{F}]\phi \stackrel{def}{=} \neg\langle\mathbb{F}\rangle(\neg\phi) \tag{6.2.13}$$

$$[\mathbb{F}]\phi \stackrel{def}{=} \neg\langle\mathbf{F}\rangle(\neg\phi) \tag{6.2.14}$$

6.2.2 Pre-Image Functions

The semantics of the modal operators are defined recursively using the *pre-image functions* which enable a step-by-step way for defining the transitions that satisfy a formula.

For the A-relation, $PRE_A(S)$ is defined on S, a set of transitions, as the set of transitions whose immediate successors along the A-edges are in S. Note that in A, there is at most one immediate successor for any given transition.

$$PRE_A(S) = \{t \in \mathcal{T} \mid \exists t' \in S.(t, t') \in A\} \tag{6.2.15}$$

For the case of F-relation, we define two pre-image functions: the *existential pre-image* of S returns the set of nodes that have at least one immediate successor in S along F-edges, while the *universal pre-image* of S returns the nodes that all of their immediate successors are in S.

$$PRE_F^{\exists}(S) = \{t \in \mathcal{T} \mid \exists t' \in \mathcal{T}.(t, t') \in F \wedge t' \in S\} \tag{6.2.16}$$

$$PRE_F^{\forall}(S) = \{t \in \mathcal{T} \mid \forall t' \in \mathcal{T}.(t, t') \in F \Rightarrow t' \in S)\} \tag{6.2.17}$$

For example, in Figure 6.3 we have:

$$PRE_F^{\exists}(\{T_2, T_5\}) = \{T_1, T_3\}$$
$$PRE_F^{\forall}(\{T_2, T_3\}) = \{T_1, T_6\}$$

6.2.3 Recursive Definition of Modal Operators

The intuitive meaning of $\langle\mathbf{A}\rangle\phi$ is that ϕ is true at the current or some following transition along the A-edges, which can be rephrased recursively as: ϕ is true at the current transition, or $\langle\mathbf{A}\rangle\phi$ holds at the immediate A-successor (see Lemma 1). So, we define the set of transitions satisfying $\langle\mathbf{A}\rangle\phi$ as the union of those that satisfy ϕ and those whose A-successor satisfies $\langle\mathbf{A}\rangle\phi$ which considering the definition of PRE_A can be written as:

$$[\![\langle\mathbf{A}\rangle\phi]\!] = [\![\phi]\!] \cup PRE_A([\![\langle\mathbf{A}\rangle\phi]\!]). \tag{6.2.18}$$

Similarly, the intuitive meaning of $\langle\mathbb{F}\rangle\phi$ is that either the current transition satisfies ϕ or one of the *possible* future transition does, which can be rephrased recursively as: ϕ is true at the current transition, or one of the immediate F-successors satisfy $\langle\mathbb{F}\rangle\phi$ (see Lemma 1). So, we define the set of transitions satisfying $\langle\mathbb{F}\rangle\phi$ as the union of those that satisfy ϕ and those with at least one immediate F-successor that satisfies $\langle\mathbb{F}\rangle\phi$:

$$[\![\langle\mathbb{F}\rangle\phi]\!] = [\![\phi]\!] \cup PRE_F^{\exists}([\![\langle\mathbb{F}\rangle\phi]\!]) \tag{6.2.19}$$

The intuitive meaning of $\langle\mathbf{F}\rangle\phi$ is that either the current or some *certain* future transition along the F-edges satisfies ϕ. Considering the split-type, we know that in the case of an AND split, all of the successor

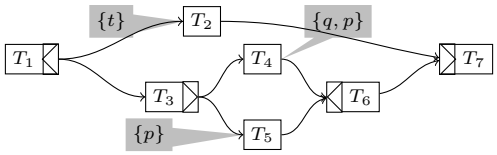


Figure 6.3: The effect of split types on the interpretation of quantifiers. The callouts show the labelling function with the atomic propositions p, q, and t.

transitions will be activated, so it is enough if at least one of them satisfies $\langle \mathbf{F} \rangle \phi$, so, the *existential pre-image function* is applicable. For XOR splits, however, it is not determined which succeeding transition will ensue, so, in order to ensure that $\langle \mathbf{F} \rangle \phi$ is *certainly* satisfied in future paths, we have to ensure *all* of the following transitions satisfy $\langle \mathbf{F} \rangle \phi$ which means we have to apply the *universal pre-image* function. Thus, the set of all transitions satisfying $\langle \mathbf{F} \rangle \phi$ can be recursively defined as the union of those that satisfy ϕ, those with split-type AND with at least one immediate F-successor that satisfies $\langle \mathbf{F} \rangle \phi$, and those with split-type XOR whose all immediate F-successors satisfy $\langle \mathbf{F} \rangle \phi$:

$$[\![\langle \mathbf{F} \rangle \phi]\!] = [\![\phi]\!] \cup (\mathcal{T}_{AND} \cap PRE_F^{\exists}([\![\langle \mathbf{F} \rangle \phi]\!])) \cup (\mathcal{T}_{XOR} \cap PRE_F^{\forall}([\![\langle \mathbf{F} \rangle \phi]\!])). \tag{6.2.20}$$

To simplify this, we define $PRE_F(S)$ which separates the nodes with AND- or XOR-split and applies the appropriate pre-image function to them:

$$PRE_F(S) = (\mathcal{T}_{AND} \cap PRE_F^{\exists}(S)) \cup (\mathcal{T}_{XOR} \cap PRE_F^{\forall}(S)) \tag{6.2.21}$$

Using this, we can simplify (6.2.20) as:

$$[\![\langle \mathbf{F} \rangle \phi]\!] = [\![\phi]\!] \cup PRE_F([\![\langle \mathbf{F} \rangle \phi]\!]) \tag{6.2.22}$$

As an example, consider the workflow of Figure 6.3 again. Since node T_3 has split-type XOR, and both of its immediate successors satisfy p, they both satisfy $\langle \mathbf{F} \rangle p$ and thus, it follows that T_3 satisfies $\langle \mathbf{F} \rangle p$. Now, since T_1 has split-type AND and hence it will proceed to T_3 in any case, and T_3 satisfies $\langle \mathbf{F} \rangle p$, we can be assured that T_1 satisfies $\langle \mathbf{F} \rangle p$. Note that, had T_1 had split-type XOR, we would have to ensure that T_2 satisfies $\langle \mathbf{F} \rangle p$ as well. On the other hand, T_3 does not satisfy $\langle \mathbf{F} \rangle q$, since the workflow may proceeds to T_5 after T_3 and T_4 may never happen. Nonetheless, T_3 satisfies $\langle \mathbb{F} \rangle q$ since T_4 may *possibly* follow.

Note that any transition with no successor along F-edges (i.e. the sink transition in the root WN of $\mathcal{H}$) is (vacuously) a member of $PRE_F^{\forall}$, but since we assumed in Section 5 that the split-type of such transitions is always AND, it is filtered out by the intersection in (6.2.20).

6.2.4 The Importance of Splits

Split types have an important effect in evaluating the future states and thus purposes. A model such as our previous model of *action graphs* [41] which does not consider the split types could at times be too conservative and block innocuous workflows that do not actually violate the policy. Consider the example of Figure 6.3 and the policy that t is a required purpose for T_1, i.e. T_1 is not allowed unless in the case that t is one of its purposes. With a model in which the split-types are abstracted away and hence not considered in the evaluation, the safe decision is to reject this workflow, since it might violate the policy (the split type for T_1 could be XOR and after executing T_1 the workflow might take the path to T_3 and never reach a transition where t is true). Considering the split types, therefore, is important for an accurate evaluation.

6.3 Dynamic Semantics: Linking the Modal Operators to Purpose

So far, we have defined a modal logic language based on the static structure of the workflows. But in order to be able to model purpose-based policies with this language, we have to show that the semantics of this language correspond to the the meaning of purpose.

Our intuitive understanding of purpose is based on the order of occurrence for actions in the system as we defined in Section 4: a possible/certain F-type purpose p means that there is a possible/certain action in future which has the property p, and an A-type purpose means there is an activity with the property p of which the current action is a part. These can be formally defined over the set of all possible firing sequences of a hierarchy net and result in a set of semantics definition which we call the *dynamic semantics*.

In this section, we define and discuss the dynamic semantics and show that they are equivalent to the semantics of the modal operators and thus, the modal operators do model purpose-based constraints as we understand them.

The following notational conventions are defined to facilitate these definitions:

- $Q_{\mathcal{H}}$: the set of all possible firing sequences for the hierarchy net $\mathcal{H}$.
- q_i (where q is a firing sequence): the transition at the ith location of q.
- $M(q)$: the marking sequence corresponding to the firing sequence q.
- $M_i(q)$: the ith marking in $M(q)$. Note that for any i we have: $M_i(q) \xrightarrow{q_i} M_{i+1}(q)$.
- $LOOP(t, t')$: a Boolean functions that specifies whether tt' is the return path of a loop, i.e. whether $\exists p_l \in O(\mathcal{P}_{\mathcal{H}}).\ \{(t, p_l), (p_l, t')\} \subset FLOW(EXT(\mathcal{H}))$.
- $E(\mu)$: the set of transitions that are enabled at the marking μ.

6.3.1 Formal Definition

We define the dynamic semantics as $[\![\phi]\!]_{\mathcal{H},L}$ which represents the transitions satisfying ϕ in the context of the hierarchy net $\mathcal{H}$ and the labelling function L. We drop the subscripts where the context is clear. For convenience, we assume that non-composite tasks are also expanded to a small net including an *entry*, *main* and *exit* task, similar to the way explained in Section 5.4. Note that this assumption does not cause any loss of generality and is only for the purpose of keeping the definitions easy to express.

$$[\![\top]\!] = \mathcal{T}. \tag{6.3.1}$$
$$[\![p]\!] = \{t \in \mathcal{T} \mid p \in L(t)\}. \tag{6.3.2}$$
$$[\![\neg\phi]\!] = \mathcal{T} \setminus [\![\phi]\!]. \tag{6.3.3}$$
$$[\![\phi_1 \wedge \phi_2]\!] = [\![\phi_1]\!] \cap [\![\phi_2]\!]. \tag{6.3.4}$$
$$[\![\langle \mathbf{A} \rangle \phi]\!] = \{t \mid \forall q \in Q (q_i = t \rightarrow\ \exists u \in [\![\phi]\!]\ \exists k > i\ \exists j < i\ (q_j = u^e \wedge q_k = u^x))\}. \tag{6.3.5}$$
$$[\![\langle \mathbf{F} \rangle \phi]\!] = \{t \mid \forall q \in Q\ (q_i = t \rightarrow \exists j \geq i\ (q_j \in [\![\phi]\!]))\}. \tag{6.3.6}$$
$$[\![\langle \mathbb{F} \rangle \phi]\!] = \{t \mid \exists t'.\ t' \in [\![\phi]\!] \wedge t \rightsquigarrow^* t'\}. \tag{6.3.7}$$

The definition of $\rightsquigarrow^*$ is given later below.

6.3.2 Discussion

The dynamic semantics for $\top$, atomic propositions and the logical connectives are defined to be identical to the static semantics.

The intuitive meaning of an A-purpose, shown as $\langle \mathbf{A} \rangle \phi$, is that the action performed as *part* of an activity that satisfies ϕ.

Definition 8. Part-of: *An action is said to be* part *of an activity if and only if it always happens between that activity's start and end.*

Thus, we define $[\![\langle \mathbf{A} \rangle \phi]\!]$ as the set of transitions that in every possible firing sequence occur between the start and end of a transition that satisfies ϕ.

The intuitive meaning for a possible F-purpose, shown as $\langle \mathbb{F} \rangle \phi$ is that the action is *possibly* a prerequisite to another action that satisfies ϕ. Thus, we define $[\![\langle \mathbb{F} \rangle \phi]\!]$ as the set of all transitions t that, in at least one possible firing sequence, are the *prerequisite* of a transition that satisfies ϕ. Prerequisite is defined as:

Definition 9. Prerequisite-of: *Action t' is said to be a* prerequisite *of action t if and only if there exists a firing sequence in which enabling t depends on the firing of t', i.e. if t' did not occur, t would not be enabled, except in the case of the return path of a loop.*

The return path of a loop does not imply a prerequisite relation as we discussed in Section 5.9. On this basis, the *direct* prerequisite of t in q is the transition whose firing enables t. This is shown as $t' \rightsquigarrow_{\mathcal{H}} t$ and formally defined below. The reflexive transitive closure of $\rightsquigarrow_{\mathcal{H}}$, shown as $\rightsquigarrow^*_{\mathcal{H}}$ captures the broader meaning of *prerequisite-of* as defined above which includes direct and indirect prerequisites.

$$t' \rightsquigarrow_{\mathcal{H}} t \text{ iff } \exists q \in Q_{\mathcal{H}} \exists i.\ q_i = t' \wedge t \notin E(M_i(q)) \wedge t \in E(M_{i+1}(q)) \ \wedge \ \neg LOOP(t, t'). \tag{6.3.8}$$

Theorem 1 states that the dynamic definition of *prerequisite-of* is equivalent to the formal definition of F-relation given in Section 5; i.e. the dynamic and static definition of *prerequisite-of* are equivalent:

Theorem 1. *The dynamic and static definitions of* prerequisite-of *are equivalent, i.e., for a given hierarchy net $\mathcal{H}$:* $(t' \rightsquigarrow^*_{\mathcal{H}} t) \Longleftrightarrow (t', t) \in F^*_{\mathcal{H}}$

Proof. See Appendix A. □

The intuitive meaning for a *certain* F-purpose, shown as $\langle \mathbf{F} \rangle \phi$ is that the action will *certainly* lead to another action that satisfies ϕ. Thus, we define $[\![\langle \mathbf{F} \rangle \phi]\!]$ as the set of transitions t that, in every possible firing sequence lead to a transition that satisfies ϕ.

6.3.3 Equivalence of Dynamic and Static Semantics

Theorem 2 states that the dynamic and static semantics are equivalent, and so, the static semantics of the modal operators match the intuitive meaning of purposes defined by the dynamic semantics.

Theorem 2. *For any formula ϕ, any hierarchy net $\mathcal{H}$ and labelling function L, we have:* $[\phi]_{\mathcal{H},L} = [\![\phi]\!]_{\mathcal{H},L}$.

Proof. See Appendix A. □

7

Purpose-Based Policies

Traditional access control policies are a collection of rules that stipulate what *action* can be performed by what *subjects* on what *data resources* and under what contextual (a.k.a *environment*) conditions [71]. Purpose-based policies add *purpose* as an additional decision factor in such rules so that the policy can additionally specify constraints about the purposes of access. We define a *purpose constraint*, or synonymously a *plain purpose policy* as:

Definition 10. ***Purpose Constraint (Plain Purpose Policy):*** *a restriction about the purposes of access which can be modelled as a formula belonging to the modal logic language defined in Section 6.*

A purpose-based access control policy is defined as:

Definition 11. ***Purpose-Based Access Control Policy:*** *a set of access control rules including restrictions on actions, subjects, data resources, environment, and their attributes that additionally, include some purpose-based constraints.*

In other words, a purpose-based policy is created by linking a purpose constraint to a traditional access control policy. Note that we do not discuss the formal semantics of access control policies in this paper and our discussion of formal semantics is restricted to plain purpose-based policies (see Section 6).

In this section we discuss purpose-based policies by introducing different types of purpose constraints and how they can be modelled using our modal logic language. Subsequently, we discuss the different ways in which purpose constraints can be tied to access control rules to create a general purpose-based access control policy.

7.1 Types of Purpose Constraints

In this section we discuss some typical forms of purpose constraints and how they can be expressed using the modal logic language. Some of these forms have been discussed in the literature before, but we also propose some new forms that are the contribution of this work.

7.1.1 Required Purposes

One of the most common types of purpose constraints is to require a certain purpose, i.e. access is allowed only if it is for a specific purpose. For example:

- A patient may require the purpose of *medical treatment* for access to her or his health record, so, access will not be allowed for any other purposes, such as *research* or *training*.
- An online store's customer may restrict access to her or his home address to the purpose of *product delivery*, so, access is not allowed for any other purposes such as *marketing*.

A required purpose p can be modelled using the *certain* operators, so, depending on whether F- or A-purpose is intended, it can be modelled using $\langle \mathbf{A} \rangle p$ or $\langle \mathbf{F} \rangle p$, and if the type of purpose does not matter it can be modelled as:

$$\langle \mathbf{A} \rangle p \vee \langle \mathbf{F} \rangle p$$

Theorem 3 states that these two operators can cover any type of certain purposes, i.e. any other combination of A- and certain F-purposes is covered by $\langle \mathbf{A} \rangle p \vee \langle \mathbf{F} \rangle p$.

Theorem 3. *Any combination of* certain *F- and A- purposes which includes both operators, is equivalent to the disjunction of a* certain *F- or an A-purpose, i.e. for any $n > 1$ we have:*

$$[\![\langle X_1 \rangle \cdots \langle X_n \rangle \phi]\!] = [\![\langle \mathbf{A} \rangle \phi \vee \langle \mathbf{F} \rangle \phi]\!] \ (X_i \in \{\mathbf{A}, \mathbf{F}\})$$

given that $\exists i, j.\ X_i = \mathbf{F} \wedge X_j = \mathbf{A}$.

Proof. See Appendix A. □

Since our model supports multiple purposes for an action (see Section 4), it is possible to require more than one purpose. For example, requiring both the purpose of *order-processing* and *delivery* can be formulated as:

$$(\langle \mathbf{A} \rangle \text{order-processing} \vee \langle \mathbf{F} \rangle \text{order-processing}) \wedge (\langle \mathbf{A} \rangle \text{delivery} \vee \langle \mathbf{F} \rangle \text{delivery})$$

7.1.2 Forbidden Purposes

Another common type of constraint is to forbid some purposes, i.e. access is denied if it is for a certain purpose. For example:

- An online store's customer may wish to forbid access to her or his home address for the purpose of *marketing*.
- A patient may wish to forbid access to her or his psychological health information for the purpose of *research*.
- A hospital may prohibit access to any patients' health record for the purpose of *medical treatment* by any user in the role of *admin assistant*.

Forbidding p is basically equivalent to ruling out the possibility of p, so it can be modelled as the negation of a *possible* operator. So, depending on whether F- or A-purpose is intended, it can be modelled using $\neg\langle \mathbf{A} \rangle p$ or $\neg\langle \mathbb{F} \rangle p$, and if the type of purpose does not matter it can be modelled as:

$$\neg(\langle \mathbf{A} \rangle p \vee \neg\langle \mathbb{F} \rangle p)$$

Theorem 4 states that these two operators can cover any type of possible purposes, i.e. any other combination of A- and possible F-purposes is covered by $\langle \mathbf{A} \rangle p \vee \langle \mathbb{F} \rangle p$.

Theorem 4. *Any combination of* possible *F- and A- purposes which includes both operators, is equivalent to the disjunction of a* possible *F- and an A-purpose, i.e. for any $n > 1$ we have:*

$$[\![\langle X_1 \rangle \cdots \langle X_n \rangle \phi]\!] = [\![\langle \mathbf{A} \rangle \phi \vee \langle \mathbb{F} \rangle \phi]\!] \ (X_i \in \{\mathbf{A}, \mathbb{F}\})$$

given that $\exists i, j.\ X_i = \mathbb{F} \wedge X_j = \mathbf{A}$.

Proof. See Appendix A. □

Theorem 5 says that any combination of certain F-, possible F-, and A-type operators can be covered by their disjunction.

Theorem 5. *Any combination of A-,* certain *F- and* possible *F- purposes that include all the three operators, is equivalent to the disjunction of a* possible *F- and an A-purpose, i.e. for any $n > 2$ we have:*

$$[\![\langle X_1\rangle \cdots \langle X_n\rangle \phi]\!] = [\![\langle \mathbf{A}\rangle \phi \vee \langle \mathbb{F}\rangle \phi]\!] \ (X_i \in \{\mathbf{A}, \mathbf{F}, \mathbb{F}\})$$

given that $\exists i, j, k.\ X_i = \mathbb{F} \wedge X_j = \mathbf{F} \wedge X_k = \mathbf{A}$.

Proof. See Appendix A. □

Logical combinations of forbidden and required purposes can model more complex constrains.

7.1.3 A- vs. F- Purposes

The type of purpose (A- vs. F-) often does not matter in the purpose constraints and they usually apply equally to both F- and A-purposes. But there are cases where such distinction matters. For example, consider the case of an *admin assistant* in a hospital. The policy may want to prohibit this role from partaking in activities that are directly *part of* a treatment procedure, and yet allow activities that *lead to* some treatment. Therefore, this role is allowed to have a treatment purpose of type F but not of type A, which can be modelled as $\neg\langle \mathbf{A}\rangle$treatment.

As another example, a patient, or an organizational policy may wish to restrict access to the sensitive parts of her or his record to the activities that are directly *part of* a treatment procedure rather than activities that *lead to* such procedures such as *hospital admission*. This can be modelled as $\langle \mathbf{A}\rangle$treatment.

7.1.4 Order-Based Constraints

The order of the purposes of an action can sometimes be important in a purpose-based policy. For example, suppose that an insurance company pays for different portions of the expenses for the purpose of *surgery*, depending on whether the *surgery* is in turn for the purpose of *treatment*, *birth control*, or *cosmetics*. In such cases, not only is it important that these purposes be present, but they also must be in a specific order. In the above example, suppose that the policy requires that in order to file a claim for full reimbursement, the purpose of *surgery* must be present and be in its turn for the purpose of *treatment*, which can be formulated as:

$$\langle \mathbf{F}\rangle(\text{surgery} \wedge (\langle \mathbf{F}\rangle\text{treatment} \vee \langle \mathbf{A}\rangle\text{treatment}))\vee$$
$$\langle \mathbf{A}\rangle(\text{surgery} \wedge (\langle \mathbf{F}\rangle\text{treatment} \vee \langle \mathbf{A}\rangle\text{treatment}))$$

Note that a simpler rule requiring both *surgery* and *treatment* does not necessarily describe the same requirement since it can be satisfied even when *surgery* and *treatment* both appear but in the reverse order. Consider the treatment activities that might be performed to control the blood pressure of a hypertensive patient to prepare her or him for a cosmetic surgery. In this case, both *treatment* and *surgery* are among the purposes of these activities but the order is not as desired by the policy. A more complex example of order-based constraints which deals with *possible* orders is given in Section 9.

7.2 Linking to Other Access Control Policies

By tying a purpose constraint to other access control rules, a purpose-based policy can be created. The purpose constraints can be formulated using the modal logic language developed in our framework and an existing access control policy language can be extended so that the purpose constraints can be formulated alongside, and in connection with other access control rules. Some of the most common forms of such policies are discussed in this section. We use the *eXtensible Access Control Markup Language* ("XACML") [71].

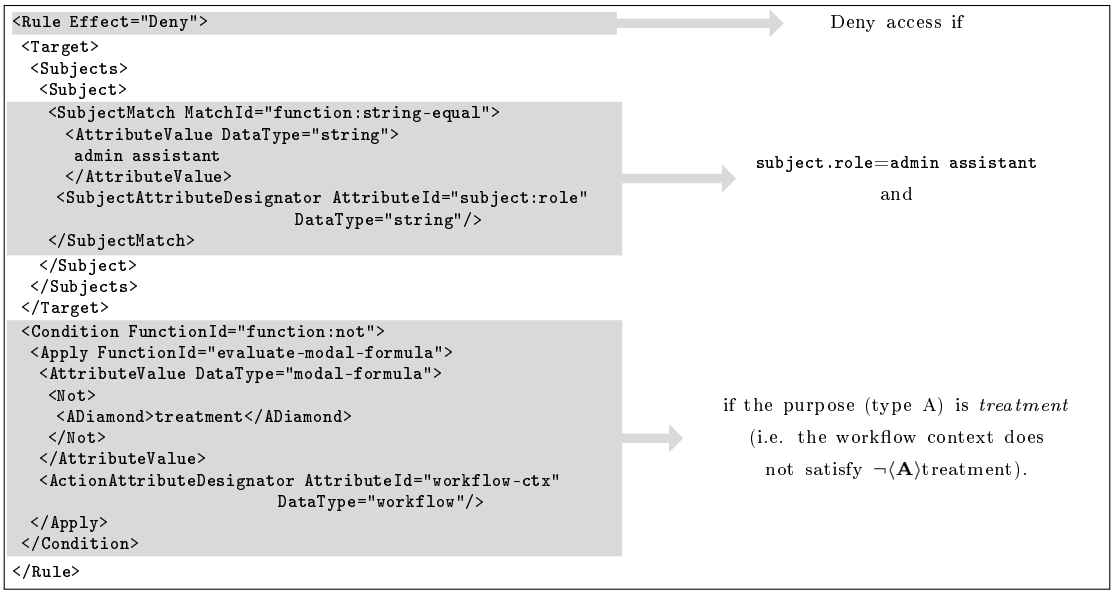


Figure 7.1: An example of a subject-centric policy modelled in XACML which forbids purpose of treatment for users with the role *admin assistant*.

7.2.1 Action-Centric Policies

Action-centric policies are the result of tying a purpose constraint to a specific (type of) action. The general form of such policies is to require a certain purpose constraint to hold as a condition for allowing an action. For example, the policy for using a printer in an office might require that printing is only allowed for the purpose of *billing confirmation* –thereby deny using the printer for other purposes. Or, an organizational policy may prohibit any *remote* action (i.e. over the Internet) for the purpose of *research* but allow this purpose for *local* actions.

Since actions and their attributes are already modelled in our language, this form of policy can be modelled within the modal logic language in the form of an implication like the following:

$$\text{action id/attribute} \rightarrow \phi.$$

It is still possible, however, to use the facilities of the access control language to model this kind of policies in case a particular action's id or its attributes are not captured in the modal logic vocabulary but are available in the broader vocabulary used by the access control policy language.

7.2.2 Subject-Centric Policies

A subject-centric purpose-based policy expresses some constraints about the purposes that certain subjects can claim. A black/whit list of unauthorized/authorized purposes for roles is a simple form of this type of policy. For example, a hospital's policy may stipulate that *admin assistant* role cannot perform any actions for the purpose (type A) of *treatment* as shown in Figure 7.1. More complex policies can be expressed using other attributes, for example, a policy may require that in order to access data for the purpose of *genetic analysis* the subject should have the *physician* role and at least 5 years of experience in the practice.

7.2.3 Data-Centric Policies

Data-centric policies are composed of linking some purpose constraints to certain data resource, so that any action on the data by any subject must satisfy the purpose constraints. This is a very common approach for

purpose-based policies and has been adopted by many models and standards [72, 18]. A black/white list of unauthorized/authorized purposes (a.k.a *intended purposes* [18]) attached to a data resource is the simplest form of this kind of policies. For example, a patient consent can require that the medical record shall not be used for the purpose of *marketing*.

More complex policies can be formed by considering other attributes of the data resource, for example, a patient can stipulate that if the data item in her/his record is labelled with *reproductive health* it cannot be accessed for any purposes other than *medical treatment*, or a jurisdictional authority may prohibit the purpose of *research* for the health record of patients younger than 18 years, unless the patient's legal guardian consents otherwise. An example of a data-centric policy modelled in XACML is shown in Figure 9.3.

7.2.4 Environment-Centric Policies

Likewise, environment-centric policies tie a purpose constraint to some environmental attributes such as time or location. For example, a policy may require that the purpose of *marketing* is not authorized for access outside business hours, or the purpose of *public health* is only authorized when access is requested within a particular country or province.

7.2.5 Compound Policies

Complex policies can be formed by combining action-, subject-, data- and environment-centric rules. For example, a subject-data-environment-based policy may restrict purpose of *research* to users with the role of *physician* and only on *anonymized* data and within the location of the organization premises.

In the most general form, a policy specifies a constraint on subject, data, action, environment, and a purpose rule which is the approach taken by some privacy policy languages such as the Enterprise Privacy Authorization Language ("EPAL") [29]. All the other kinds of policies discussed above are especial cases for this general form.

However, modelling policies in more specific forms such as the ones discussed earlier is an efficient way of organizing them and facilitates finding the corresponding policies to an access request. For example, data-centric policies can be checked easier since the applicable policies for a data item can be fetched and checked whenever the data item is about to be used.

7.2.6 Purpose-Based Obligations

Obligations are commitments that should be incurred as a result of granting access [35]. A purpose-based obligation is an obligation that is triggered based on some purpose constraints. For example, an obligation may require the record to be *anonymized* before access for *research* purposes is allowed. Or, a fee might be charged, or notification/audit information produced, every time some personal data is used for the purpose of *marketing*. This type of policies can be modelled using an access control policy language that supports obligations (e.g. XACML) in similar ways to the above cases.

7.3 Policy Origins

Policies may have different origins and set by different policy makers. The decision as to who has the authority to make what kind of policies and in what domain belongs to the purview of *administrative policies* which also decide how different policies from different origins need to be *combined*, and in case of any conflicts, *reconciled*.

Jurisdictional policies are very high-level policies that apply to all systems in a certain jurisdiction. For example, a federal or provincial policy may stipulate that a patient's health record information must not be used for the purpose of *marketing* without her or his prior consent.

Organizational policies are policies that apply to all the activities within an organization. For example, to strengthen privacy, a hospital may decide that data shall not be accessed for the purpose of *research* outside the hospital premises.

Another important source of purpose-based policies is the *business rules*. For example, in a visit to optometrist for getting new eye glasses, the doctor or the nurse do not *need* to check a patient's blood test results. Thus, the principle of least-privilege requires that access to blood test results be prohibited for the purpose of *optometric procedures*.

In many systems, the *subject* or the *owner* of the data, has the authority to make some policy decisions about its usage. For example, a patient usually has the right to make certain decisions about her or his health data. Or in a social network application, a user who appears in a photo has some rights to make access control decisions about it, even if she or he is not its owner. Such policies are only applicable to an individual data item.

8

Model Checking

The model-checking algorithm answers the question whether or not a given *model* satisfies a policy. The model is given as a hierarchy net $\mathcal{H}$ together with a labelling function L, and the policy comes in the form of a formula ϕ belonging to the modal logic language. The output of the algorithm is a yes-or-no answer indicating whether the given model complies with the given policy. The algorithm is also capable of a more fine-grained verification by specifying whether a particular transition satisfies the policy, i.e. for every $t \in \mathcal{T}_H$, the model-checking algorithm can answer if $t \in [\![\phi]\!]_{\mathcal{H},L}$ is true.

This answer can then be used by a *reference monitor* to make an access control decision to allow/deny a particular action. In this section, we only discuss the model-checking algorithm in the abstract and leave a discussion of the practical configuration for packaging this algorithm in the form of a *purpose reference monitor* for Section 9.

A straightforward algorithm can compute the extension of $\mathcal{H}$ and thereby the F- and A-relations. We assume that A and F are represented in a suitable data structure so that computing F, F^{-1}, A and A^{-1} for any given transition take constant time. The labelling function is given as L where $L(t)$ returns the set of atomic propositions that are true at transition t, and $L^{-1}(p)$ returns the set of transitions at which the atomic proposition p is true –both in constant time.

The model-checking algorithm should compute $[\![\phi]\!]$, the set of nodes that satisfy formula ϕ; the pair of $\mathcal{H}$ and L satisfy the formula if all the transitions in $\mathcal{H}$ satisfy ϕ, i.e. $\mathcal{T}_{\mathcal{H}} = [\![\phi]\!]$. Algorithm 1 shows the overall model-checking algorithm discussed in the remainder of this section.

The core idea of the algorithm is to recursively compute $[\![\phi]\!]$ in a bottom-up fashion by computing $[\![\psi_i]\!]$ for ψ_is, the subformulas of ϕ, and combining the result for smaller subformulas to compute $[\![\psi_i]\!]$ for larger subformulas. The post-order traversal of ϕ's abstract syntax tree provides a suitable ordering of subformulas for this purpose because all the constituents of a formula appear before itself. Figure 8.2 shows an example.

Based on the formal definitions of the semantics in Section 6.2, the base cases are $[\![\top]\!]$ and $[\![p]\!]$, and the cases for the logical operators, $[\![\neg\psi]\!]$ and $[\![\psi_1 \vee \psi_2]\!]$ are straightforward. So, we only need to discuss the case of modal operators. Note that we only give the algorithm for the main operators; the case for derived operators can be reduced to the main operators.

8.1 Computing Pre-Image functions

By finding F^{-1} for all the members of a given set, $PRE_F^{\exists}$ can be computed. Given that F is represented in a suitable data structure, computing F^{-1} for each member takes constant time, and thus, computing $PRE_F^{\exists}(S)$ takes $|S|$ steps. The case for $PRE_A(S)$ is similar.

For computing $PRE_F^{\forall}(S)$, we can compute $PRE_F^{\exists}(S)$ (in $|S|$ steps) and then loop over all x_i's in $PRE_F^{\exists}(S)$ to test whether $F(x_i) \subseteq PRE_F^{\exists}(S)$. The latter loop takes at most $|F|$ steps, yielding an overall complexity of $|S| + |F|$. Overall, since $|S|$ is bound by the number of transitions ($|\mathcal{T}|$), the complexity of computing pre-image functions is bound by $|\mathcal{T}| + |F|$.

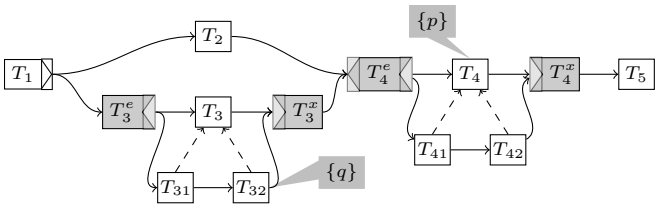
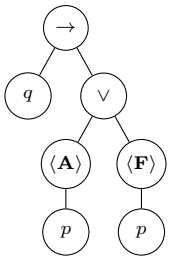


Figure 8.1: A- and F- relations based on the extension of the hierarchy net from Figure 5.2. The callouts show the labelling function.



			$[\![\psi_i]\!]$
ψ_1		q	$\{T_{32}\}$
ψ_2		p	$\{T_4\}$
ψ_3	$\langle \mathbf{A} \rangle \psi_2$	$\langle \mathbf{A} \rangle p$	$\{T_4, T_{41}, T_{42}\}$
ψ_4		p	$\{T_4\}$
ψ_5	$\langle \mathbf{F} \rangle \psi_2$	$\langle \mathbf{F} \rangle p$	$\{T_4, T_4^e, T_3^x, T_2, T_3, T_{32}, T_{31}, T_3^e, T_1\}$
ψ_6	$\psi_3 \vee \psi_5$	$\langle \mathbf{A} \rangle p \vee \langle \mathbf{F} \rangle p$	$\{T_4, T_{41}, T_{42}, T_4^e, T_3^x, T_2, T_3, T_{32}, T_{31}, T_3^e, T_1\}$
ψ_7	$\psi_1 \rightarrow \psi_6$	$q \rightarrow (\langle \mathbf{A} \rangle p \vee \langle \mathbf{F} \rangle p)$	$\mathcal{T}$

Figure 8.2: The abstract syntax tree of the formula $q \rightarrow (\langle \mathbf{A} \rangle p \vee \langle \mathbf{F} \rangle p)$ and the ordered set of its subformulas based on its post-order traversal. The right-most column shows the steps of model-checking as explained in Section 8.4.

8.2 Recursive Model Checking of Diamond Operators

Based on the definition of $[\![\langle \mathbf{F} \rangle \phi]\!]$, it can be computed in a bottom-up fashion by the following steps:

$$
\begin{aligned}
X_1 &= [\![\phi]\!] \\
X_2 &= X_1 \cup PRE_F(X_1) \\
&\vdots \\
X_n &= X_{n-1} \cup PRE_F(X_{n-1})
\end{aligned}
$$

Figure 2 depicts the algorithm for this computation. The right-hand side grows monotonically, $X_{i-1} \subseteq X_i$ for every i, so, X_i grows at each iteration until it settles to $\mathcal{T}$ or a subset of it. The loop, thus, halts after at most $n = |\mathcal{T}|$ iterations. And, since the most expensive computation at each iteration is computing the pre-image function (taking at most $|\mathcal{T}| + |F|$ steps), the overall complexity is bound by $|\mathcal{T}|(|\mathcal{T}| + |F|)$.

The cases for $[\![\langle \mathbb{F} \rangle \phi]\!]$ and $[\![\langle \mathbf{A} \rangle]\!]$ are similar, although a bit simpler since they use the $PRE_F^{\exists}$ and PRE_A functions which are less costly to compute.

8.3 Correctness

For all non-recursive operators, the algorithm straightforwardly follows the semantics definitions. For the recursive operators, since they are monotonic functions defined over the power set of $\mathcal{T}$, and $(2^{\mathcal{T}}, \subseteq)$ is a complete lattice, we can apply the constructive version of *Knaster-Tarski fixed-point theorem* [69] which states the smallest fixed-point for such a function can be computed as: $\bigcup_{i \in \mathbb{N}} f^i(\emptyset)$. Note that the theorem also states that the least fixed-point exists, so, there exists an $l \in \mathbb{N}$ such that $i \geq l \implies f^i(\emptyset) = f^{i-1}(\emptyset)$.

8.4 Example

Based on the model given in Figure 8.1 and the formula of Figure 8.2, we show an example of running the algorithm. The formula has 7 subformulas labelled as ψ_1 to ψ_7, and the partial results of the computation

ALGORITHM 1: The overall model-checking algorithm.

Input: ϕ, the labelling function L, the model
Output: whether or not the model satisfies ϕ

Subformulas : An array containing all ϕ's subformulas resulting from the post-order traversal of its abstract syntax tree)
Results : A map storing the result of $[\![\psi]\!]$ for each subformula ψ

forall the $\psi \in Subformulas$ **do**
  switch ψ **do**
    case $\top$
      $Results[\psi] \leftarrow \mathcal{T}$
    case *is an atomic proposition*
      $Results[\psi] \leftarrow L^{-1}(\psi)$
    case *has the form* $\neg\psi_1$
      $Results[\psi] \leftarrow \mathcal{T} \backslash Results[\psi_1]$
    case *has the form* $\psi_1 \wedge \psi_2$
      $Results[\psi] \leftarrow Results[\psi_1] \cap Results[\psi_2]$
    case *has the form* $\langle \mathbf{A} \rangle \psi_1$
      $Results[\psi] \leftarrow$ Result of the algorithm discussed in Section 8.2
    case *has the form* $\langle \mathbf{F} \rangle \psi_1$
      $Results[\psi] \leftarrow$ Result of the algorithm discussed in Section 8.2
    case *has the form* $\langle \mathbb{F} \rangle \psi_1$
      $Results[\psi] \leftarrow$ Result of the algorithm discussed in Section 8.2
  endsw
end
return $(Results[\phi] = \mathcal{T})$

ALGORITHM 2: Recursive computation of $[\![\langle \mathbf{F} \rangle \psi]\!]$ based on $[\![\psi]\!]$.

Input: $[\![\psi]\!]$
Output: $[\![\langle \mathbf{F} \rangle \psi]\!]$

$NewResult \leftarrow [\![\psi]\!]$;
$OldResult \leftarrow \{\}$;
while $NewResult \neq X_{Old}$ **do**
  $OldResult \leftarrow X_{New}$;
  $NewResult \leftarrow NewResult \cup PRE_F(NewResult)$;
end
return $NewResult$

are shown in the last column on the table in Figure 8.2.

Beginning from $[\![\psi_1]\!]$, and $[\![\psi_2]\!]$, they are directly given by the labelling function. For $[\![\psi_3]\!]$, we follow the Algorithm 2 involving the following steps:

$$
\begin{aligned}
X_1 &= [\![\psi_1]\!] = \{T_4\} \\
X_2 &= X_1 \cup PRE_A(X_1) = \{T_4\} \cup \{T_{41}, T_{41}\} = \{T_4, T_{41}, T_{41}\} \\
X_3 &= X_2 \cup PRE_A(X_2) = \{T_4, T_{41}, T_{41}\} = X_2
\end{aligned}
$$

Similarly, $[\![\psi_5]\!]$ is computed by the following steps:

$$
\begin{aligned}
X_1 &= [\![\psi_1]\!] = \{T_4\} \\
X_2 &= X_1 \cup PRE_F(X_1) = \{T_4\} \cup \{T_4^e\} \\
X_3 &= X_2 \cup PRE_F(X_2) = \{T_4, T_4^e\} \cup \{T_3^x, T_2\} \\
X_4 &= X_3 \cup PRE_F(X_3) = \{T_4, T_4^e, T_3^x, T_2\} \cup \{T_3, T_{32}\} \\
X_5 &= X_4 \cup PRE_F(X_4) = \{T_4, T_4^e, T_3^x, T_2, T_3, T_{32}\} \cup \{T_3^e, T_{31}\} \\
X_6 &= X_5 \cup PRE_F(X_5) = \{T_4, T_4^e, T_3^x, T_2, T_3, T_{32}, T_3^e, T_{31}\} \cup \{T_1\} \\
X_7 &= X_6 \cup PRE_F(X_6) = \{T_4, T_4^e, T_3^x, T_2, T_3, T_{32}, T_3^e, T_{31}, T_1\} \cup \{\} = X_6.
\end{aligned}
$$

For ψ_6, and ψ_7 we use the definitions for the logical connectives which eventually leads to $[\![\phi]\!] = \mathcal{T}$, i.e. the model satisfies the formula.

9

Implementation Considerations

In this section we will discuss different configurations for linking the model checking algorithm to an actual implementation of a reference monitor for access control. Subsequently, we will discuss the system components and an example case of how the framework can be implemented to enforce purpose-based policies in the medical research institute discussed in Section 3.

We assume a workflow-based information system is in place, i.e. a system in which all authorized activities are defined in the form of workflows and no isolated or arbitrary access to data outside the context of workflows is allowed.

9.1 Model Checking Configurations

The enforcement core of the framework is the model-checking algorithm which sits at the heart of a reference monitor to control access based on the answer from the model-checking algorithm as discussed in Section 8. This can be implemented in different ways based on the application context. We discuss some of the possible configurations below. To keep our discussion more concise, we only focus on the first two cases for the remainder of this section. Implementing the third case is very similar to the second.

9.1.1 Workflow-Definition Reference Monitor

Some high-level overarching policies that govern the use of any data in the system can be checked statically, and regardless of the data processing context of any particular data item. In such cases, the reference monitor interferes when a new workflow definition is submitted to the system and verifies it against all such policies; violating workflow definitions are rejected. For example, the policy that *printing* is allowed only for the purpose of *billing confirmation* can be checked statistically by ensuring that in a given workflow definition, all printing actions are for the said purpose.

9.1.2 Workflow-Instantiation Reference Monitor

Some more specific policies depend on a particular instance of data processing, such as a specific data item, user, or *parametrized purposes* (see Section 10.1). In such cases, static verification cannot ensure compliance with the policies because some of the parameters of the policy are not known at workflow definition time. So, the reference monitor must interfere when these parameters are known, which is typically at the time of workflow instantiation. For example, in a data-centric information system where each data item is associated with its specific privacy preference, model-checking should be performed at workflow instantiation time, when it is known what data will be used and in what workflow. This is referred to as the *static workflow execution model* (not to be confused with *static model checking* discussed above) in which data and users assigned to each task of the workflow are determined at the workflow instantiation time [24].

9.1.3 Task-Instantiation Reference Monitor

There are cases where the specific parameters such as the specific data item or user are not known even at the time of workflow instantiation and will not be determined until later, once a specific activity in the workflow instance is about to be performed, i.e. *task-instantiation time* when the actual parameters of an instance of a task are determined. This is known as *dynamic workflow execution model* [24] in which data and users assigned to a task can only be determined at the task-instantiation time, a very likely case in larger workflows, such as cross-organizational workflows, or those that span over a long period time.

In such cases, the reference monitor must interfere before running the specific task when all the parameters (e.g. the user IDs and data items) are known, and ensure all corresponding policies are followed.

For example, consider the case of a clinical consultation where Alice's medical record is to be sent to another doctor for a second opinion. On the other hand, as part of her consent, she has decided that Dr. Bob, who happens to be her neighbour, is not allowed to access her medical record unless for the purpose of *emergency treatment*. Since the workflow is cross-organizational, it is very likely that the identity of the consulting doctor is not known at workflow instantiation time, so, the assignment of the consulting doctor will be delayed until later in the workflow when the actual consultation task is instantiated. If it happens that Dr. Bob is selected for consultation, the reference monitor interferes at the time of task instantiation and rejects the assignment since it violates Alice's consent policy, in which case a different doctor should be selected to proceed.

Note that when the assignment of task parameters such as users and data items is delayed until task instantiation time, the workflow management system must provide the flexibility for alternative assignments should one particular assignment fail for access control reasons.

9.2 System Components

Figure 9.1 shows the system components and their interactions. The greyed-out components are those that are not directly part of the implementation of our framework but have important interactions with it.

The *data repository* stores collected data and the *policy database* keeps all general and data-specific policies. Each data item in the repository is linked to its corresponding policies in the policy database. The *policy authoring interface* enables policy makers and patients (or their authorized delegates) to formulate the policies governing the use of data. Depending on the type of system and policy authors' level of expertise, a suitable graphical or textual interface is used to design policies, but the eventual output of this component is in the form of the modal-logic language defined by our framework and will be stored in the policy database.

Since we presume a workflow-based system, there exists a *workflow definitions catalogue* that stores the workflows defined by business process experts. A *workflow management system* manages and runs the *workflow instances*. The access control reference monitor also resides within the workflow management system. We assume that the system guarantees that all access to data takes place via some workflow and no arbitrary access is possible in the system.

The policies and the workflow definitions use a common vocabulary which is defined and stored at the *vocabulary database*. The vocabulary is defined by domain experts and administrators of the system and can be based on international or business standards where applicable.

The framework is configured to be used for implementing a *purpose reference monitor* ("PRM") that monitors two types of operations: *workflow definition requests* and *workflow instantiation requests*. These are shown in bold typeface in Figure 9.1. Note that the PRM is part of the broader reference monitor of the system which handles all access control policies and uses PRM whenever a purpose-based constraint needs to be evaluated.

9.2.1 Workflow Definition Request

A *workflow definition request* is made by a business process administrator and includes the definition of a workflow as discussed in Section 5 which includes a hierarchy net and possibly other properties such as

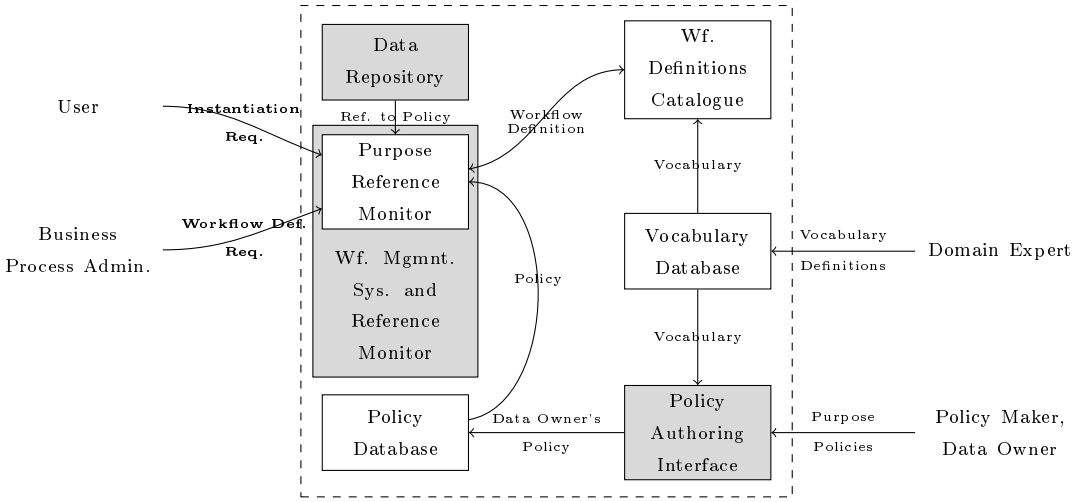


Figure 9.1: System view of the framework: the components and their interaction.

authorized roles, input data type or temporal and spatial constraints for each activity. The PRM, upon receiving this type of request, checks the workflow definition against overarching policies and rejects it in case of any non-compliance. Aside from the workflow definition, the request may contain information such as the user ID of the administrator who is submitting the definition and other similar information which are used by other parts of the access control policy than the purpose constraints.

9.2.2 Workflow Instantiation Request

A workflow instantiation request includes the reference to the requested workflow, together with instance-specific information such as the assignment of users and data items to workflow activities. As we discussed in Section 9.1, we assume that all data and user assignments are given at instantiation time, although this assumption can be easily relaxed in a more extensive implementation.

Upon receiving this type of request, the PRM checks the workflow definition against the instance-specific policies and rejects it in case of any non-compliance. Examples of instance-specific policies are the patient's consent and any parametrized purpose-based policies (see Section 10.1) that rely on runtime parameters such as user ID or data item ID.

The instantiation request may also include a range of other information such as the identity of requester, active role and other contextual information such as time and location, which are needed for evaluating the request against other parts of the access control policy than the purpose constraints.

9.3 Case: A Clinical Research Institute

Back to the case of a clinical research institute of Section 3: assume Alice –a researcher in the *hepatitis research program*– needs to perform a correlation analysis between certain demographic information and immunity to hepatitis –an activity defined as the workflow WF-01 in the system. On the other hand, the medical record of a patient pseudonymized as PA-01 is stored in the data repository together with the corresponding consent directive that defines the rules governing its use. We discuss the details of the vocabulary, the patient's consent, the workflow and labelling function, and the event flow of the authorization that takes place in this case.

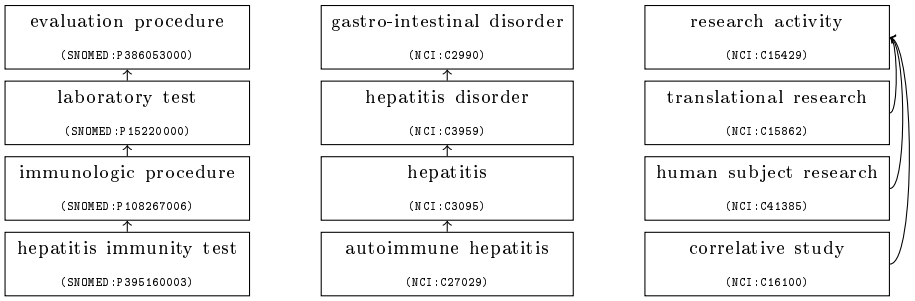


Figure 9.2: Some SNOMED and NCI terms used in our example with their hierarchical relations.

9.3.1 Vocabulary

We assume the standard vocabulary in use is based on SNOMED-CT and NCI Thesaurus. SNOMED-CT is a standard vocabulary for clinical terms [62], and NCI Thesaurus is another vocabulary that provides the standard terms related to translational and basic medical research [52]. Both vocabularies come in the form of an ontology and include relations between the terms in the vocabulary. Here, we only consider the hierarchical generalization-specification or *is-a* relations. Figure 9.2 shows some of the terms and their conceptual relations which are used in our example.

9.3.2 Consent Policy

The patient's consent contains a simple *data-centric* policy in which an *order-based* purpose-based constraint (see Section 7.1.4) is linked to a data item. Since the patient is concerned about any kind of analyses that could lead to revealing the HIV-related information, she or he has decided to prohibit access to the record for the purpose of *immunologic analysis* that is in its turn for the purpose of *clinical research*. Note that this policy does not prohibit access to the record for other *research* purposes than *immunologic analysis*. For example, the institute can still use the record in other research activities as long as it does not involve immunologic analyses; for example, in *correlation of alcohol/substance abuse and liver cancer*. On the other hand, it does not prohibit access for the purpose of *immunologic analysis* altogether; the record can still be accessed for that purpose if it is in turn for other purposes such as *treatment*.

Figure 9.3 shows the patient's consent directive codified as an XACML rule. The atomic propositions for *immunologic procedure* and *research activity* are shown using their standard codes and the modal logic formula is encoded in a straightforward XML format.

9.3.3 Workflow

The workflow WF-01 is shown in Figure 9.4. It starts by setting up the test instance (T_0) and then simply looping over a number of patients and enrolling each patient in the research procedure (T_1), reading and registering age, gender and ethnicity (T_2, T_3, and T_4) and then proceeding to check and register the immunity of the patient to hepatitis (T_5). When there is enough samples, the workflow performs some mathematical correlation analyses (T_6) and concludes by saving the results (T_7).

The workflow WF-01 is one of the realizations of the activity *testing demographic factors* (T') which is in turn one realization of *autoimmune hepatitis study process* (T'').

9.3.4 Labelling

The designer of the workflow assigns some atomic propositions from the vocabulary to each workflow task based on its semantics. In this case, for example, the task *check hepatitis immunity* is mapped to atomic proposition *hepatitis immunity test* from SNOMED-CT. Moreover, the parents of this proposition based on the ontological relations (shown in Figure 9.2) are also assigned to the task. Similar assignments leads to


```
<Rule Effect="Deny">
 <Target>
  <Resources>
   <Resource>
    <ResourceMatch MatchId="function:string-equal">
     <AttributeValue DataType="string">
      PA-01
     </AttributeValue>
     <ResourceAttributeDesignator
       AttributeId="resource:id" DataType="string"/>
    </ResourceMatch>
   </Resource>
  </Resources>
 </Target>
 <Condition FunctionId="function:not">
  <Apply FunctionId="evaluate-modal-formula">
   <AttributeValue DataType="modal-formula">
    <And>
     <Not>
      <FDiamondMaybe>
       <And>
        SNOMED:P108267006
        <Or>
         <ADiamond>NCI:C15429</ADiamond>
         <FDiamondMaybe>NCI:C15429</FDiamondMaybe>
        </Or>
       </And>
      </FDiamondMaybe>
     </Not>
     <Not>
      <ADiamond>
       <And>
        SNOMED:P108267006
        <Or>
         <ADiamond>NCI:C15429</ADiamond>
         <FDiamondMaybe>NCI:C15429</FDiamondMaybe>
        </Or>
       </And>
      </ADiamond>
     </Not>
    </And>
   </AttributeValue>
   <ActionAttributeDesignator
     AttributeId="workflow-ctx" DataType="workflow"/>
  </Apply>
 </Condition>
</Rule>
```

Figure 9.3: A sample consent directive formulated in XACML 2.0; the policy says that access to the specific record data is denied if it is *possibly* for *immunologic procedure* which is in turn *possibly* for the purpose of *research activity*. Note that the XACML namespaces are removed/summarized for better readability.

the labelling function shown by the callouts in Figure 9.4. Note that since the ontological relations between labels can be captured in electronic form, propagation of the labelling to the parent labels can be done automatically by the workflow definition tool.

9.3.5 Event Flow

Alice makes a workflow instantiation request to run the workflow WF-01. This includes: the workflow definition ID, the identifier of the data items that will be used in the workflow including that of the patient pseudonymized as PA-01, as well as the user ID of Alice as the assigned user to all the activities in the workflow (which are not important in this particular case since the policy is not based on them).

Upon receiving this request, the PRM fetches the definition of WF-01 and also the consent policy of all the patients whose records are going to be used in this workflow. This includes PA-01's consent policy, shown in Figure 9.3 according to which the PRM checks the following modal-logic formula against WF-01

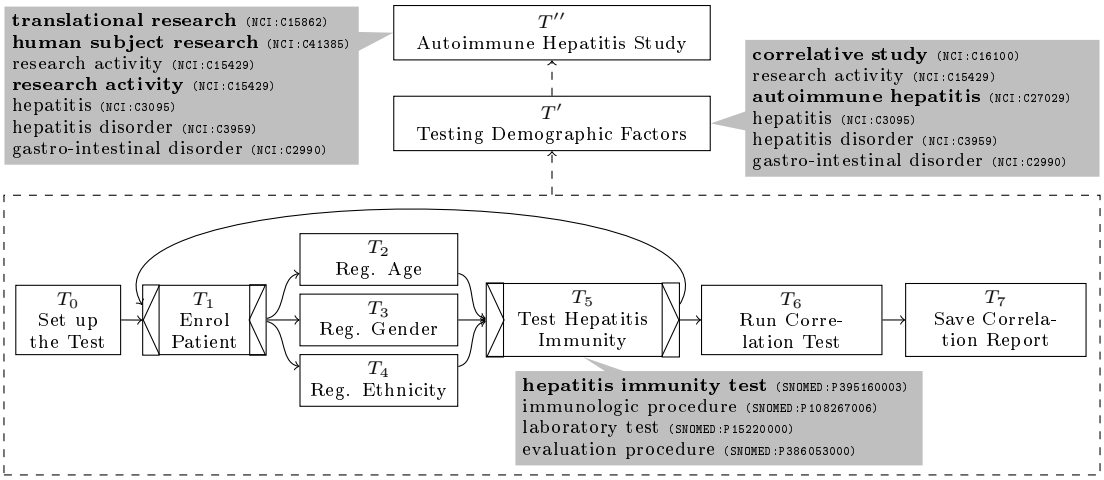


Figure 9.4: The definition of the workflow WF-01 and the labelling function. The boldfaced labels are assigned by the workflow designers and the normal-font labels result from the ontological relations in the vocabulary.

by running the model-checking algorithm:

$$\neg\langle\mathbf{A}\rangle(\text{immunologic procedure} \wedge (\langle\mathbf{A}\rangle\text{research} \vee \langle\mathbb{F}\rangle\text{research}))\wedge$$
$$\neg\langle\mathbb{F}\rangle(\text{immunologic procedure} \wedge (\langle\mathbf{A}\rangle\text{research} \vee \langle\mathbb{F}\rangle\text{research}))$$

The result is negative since PA-01's record is accessed in T_2 through T_5 and given the labelling function as shown in Figure 9.4, the formula does not hold true in those tasks.

9.4 Discussion

9.4.1 Ontologies and the Granularity Gap in the Vocabulary

As more power is given to data subjects to make policies about the usage of their data, it is often the case that the policy maker is not a domain experts and does not know the details of fine-grained technical terms. For example, an average patient is seldom able to understand the meaning of all medical terms and consent directives are normally expressed using a high-level vocabulary. As we saw in the example of this section, the ontological relations in a standard vocabulary can close this granularity gap by linking low-level technical terms to high-level vocabulary understandable by non-expert policy makers.

Moreover, existence of very granular terms in the vocabulary (such as *home visit for intramuscular injection*) makes the assignment of atomic propositions to workflow tasks less subjective. Note that if there is only general terms in the vocabulary (such as *marketing* or *research*), such assignment will very much depend on the personal opinion of the workflow designer and in some cases it may be a challenging and hard-to-audit decision whether a term corresponds to the semantics of a task. This might in turn lead to privacy breaches if an organization maliciously stretches the meaning of general terms to apply them to controversial tasks. Therefore, there can be requirements for the minimum level of granularity of the terms used by the workflow designers to avoid hiding behind broad terms and provide transparency and facilitate auditability.

9.4.2 Translation and Harmonization

As we saw in the case of Section 9, the standard vocabulary plays a crucial role in connecting the policies to workflow definitions and thereby to their semantic interpretations in a system. Harmonizing vocabularies

or translating terms from one vocabulary to another is important in case the policies are written using a different vocabulary than that of the workflow definitions and can enable understanding and enforcing policies across different domains.

10

Concluding Remarks and Future Work

In this paper, we developed a framework for expressing and enforcing purpose-based privacy policies. We defined the semantics of the purpose of an action in terms of its dynamic situation among other related actions. A modal logic was developed to enable expressing constraints about purposes. We justified the static semantics of this language with the dynamic definition of purpose, showed that the two definitions are equivalent and discussed some of the properties of purposes and proved them. We also justified the language by showing its expressive power in modelling common types of purpose-based policies. A model-checking algorithm was described to check the compliance of a system with purpose-based policies and its configuration in the implementation of a practical access control system was discussed. There are a number of topics that have the potential for future work which will be discussed in the remainder of this section.

10.1 Parametrized Purposes *

Consider the simple case that the policy allows using patient's record for *treatment* purposes. What is naturally meant by this policy is that the purpose must be the treatment of *the same patient* whose record is being used. If Alice's record, for example, is used by a physician to help with Bob's treatment (for some genetic analysis if they are siblings), this does not match the intuitive intention of the policy although access is still for the purpose of *treatment*.

This example shows that in reality, there is not one single *treatment* purpose; rather, there is a family of different purposes of the form *treatment-of*$<x>$ where x represents a patient. In other words, the simple purpose of treatment can be parametrized to differentiate between treatments of different patients. Alice may wish to restrict access to her record only to her own treatment, or perhaps to her children and close family members. There are various other examples for parametrized purposes in different domains; for example, a customer can specify that her/his home address can only be used for *delivery* of the ordered product as opposed to delivery of other things, such as marketing material.

In the general case, supporting parametrized purposes requires introducing variable in the language and leads to the extension of the language to a modal logic over first-order predicate logic which brings about serious computational problems. However, since in access control we usually deal with simple forms of query, and often, the number of entities involved is limited, this can be handled by some practical techniques. Consider the following policy, for example, that requires the purpose of treatment of the same patient for access to her/his record, and forbids the purpose of anyone else's treatment:

$$\forall x.\ \text{access}(x) \rightarrow (\langle \mathbf{A} \rangle \text{treatment}(x) \vee \langle \mathbf{F} \rangle \text{treatment}(x))$$
$$\forall x, y.\ (\text{access}(x) \wedge y \neq x) \rightarrow \neg(\langle \mathbf{A} \rangle \text{treatment}(y) \vee \langle \mathbb{F} \rangle \text{treatment}(y))$$

*We would like to thank Professor Jörg Denzinger of University of Calgary Department of Computer Science, for an enlightening conversation about this section.

In such a system, instead of atomic propositions, workflow activities are assigned to predicates such as *access(x)* or *treatment(x)* in which x is a variable in the workflow such as the input data.

In the real access control case, we have a workflow instance with a limited number, and most often only a few patient's records at stake, say Alice's and Bob's. By limiting the scope of the domain of discourse to the this single workflow instance, we can instantiate the policy by replacing the variables and generate atomic propositions for the specific instance:

$$\begin{aligned}
\text{access_Alice} &\rightarrow (\langle \mathbf{A} \rangle \text{treatment_Alice} \vee \langle \mathbf{F} \rangle \text{treatment_Alice}) \\
\text{access_Bob} &\rightarrow (\langle \mathbf{A} \rangle \text{treatment_Bob} \vee \langle \mathbf{F} \rangle \text{treatment_Bob}) \\
\text{access_Alice} &\rightarrow \neg(\langle \mathbf{A} \rangle \text{treatment_Bob} \vee \langle \mathbb{F} \rangle \text{treatment_Bob}) \\
\text{access_Bob} &\rightarrow \neg(\langle \mathbf{A} \rangle \text{treatment_Alice} \vee \langle \mathbb{F} \rangle \text{treatment_Alice})
\end{aligned}$$

Likewise, the predicates assigned to the workflow activities can also be instantiated and turned into atomic propositions such as *access_ Alice* or *treatment_ Bob* by replacing the variable with the exact value in the particular workflow instance. Thus, model checking can be performed as usual using the atomic propositions and the modal logic formulas.

Note that as the number of patients involved in the workflow increases, the number of atomic propositions and the policy instances grow towards explosion, but since we often deal with cases with a limited number of data items in the database and only a few that get involved in a single instance of the workflow, we can find a practical solution in most real cases. This assumption is not far from reality since most workflow systems assume a *case-driven* model in which each workflow instance focuses on one or a few *cases* as its input [3].

Moreover, handling variables in our language is only limited to the cases where they affect some parametrized purposes and we assume that other variables that do not affect purposes are handled outside our language as discussed in Section 7.2. Thus, we can assume that most purposes are not parametrized and do not need variables.

10.2 Modification of the Workflow

There are two types of XOR splits; those that are decided automatically at run-time based on some deterministic variable, and those that are controlled by users. An example of the first type, is a *reimbursement approval* workflow with a split that takes two different paths depending on the amount claimed in order to take higher management's approval should the amount claimed be larger than a certain threshold. An example of a user-controlled split is the choice of the service by a user at an ATM machine or the type of tests by a researcher in the course of research workflow.

Limiting the choices at a user-controlled splits can enable the reference monitor to make an effort to make it possible to run a modified version of the workflow in which the potentially violating paths are eliminated from the user-controlled splits. For example, if Alice prohibits the purpose of marketing for her home address, the reference monitor can disable a path leading to such violation at a user-controlled split, such as an optional choice to add some marketing material to the package in a *delivery* workflow, and allow the workflow to instantiate with this modification.

In general, instead of simply giving a yes-or-no answer to the model checking question, a smarter reference monitor can consider trimming the user-controlled splits in order to make the workflow compliant to the policies. We leave this as a topic of future work.

10.3 Purpose-Based Data Adaptation

The normal scenario for access control is to receive a request for a unit of data such as a medical record and then decide with an authorization answer as to whether or not the access is allowed. This can be slightly extended to a scenario where data can be modified to make it authorized for the access. For example, when the policy stipulates that access to *blood test* information is not allowed for the purpose of *research*, the

reference monitor can adapt the data by removing the blood test from the medical record so that the subject be able to access it. In other words, instead of giving a negative answer to the request, the reference monitor makes a best effort to make the access possible by trying to adapt the data to the polices.

In a slightly different case, the subject can provide the workflow in the access request and the reference monitor make an effort to find data items (or parts thereof) whose policies allow to be used in the given workflow. This is particularly helpful in scenarios such as medical research where candidates are sought to be enrolled in a specific workflow. The reference monitor in such cases can look for the records with a matching consent directive, and also for records which can be adapted so that the policy matches the workflow, by removing the non-matching fields.

10.4 Satisfiability

Workflow satisfiability refers to the problem of checking whether it is possible to instantiate a workflow, given the constraints related to user-task assignments and other constraints such as separation of duty [23]. It is possible to redefine this problem with regard to the purpose constraints, particularly the subject- and data-centric purpose-based policies, and verify whether a workflow is possible to instantiate given all the purpose-based policies that apply to it. We leave this as a future work.

Bibliography

[1] W. M. P. Aalst, *Structural Characterizations of Sound Workflow Nets*, Tech. report, Eindhoven University of Technology, Computing Science Reports 96/23, 1996.

[2] ———, *The Application of Petri Nets to Workflow Management*, Journal of Circuits, Systems, and Computers **8** (1998), no. 1, 21–66.

[3] ———, *Business Process Management Demystified: A Tutorial on Models, Systems and Standards for Workflow Management*, Lectures on Concurrency and Petri Nets, vol. 3098, Springer Berlin Heidelberg, 2004, pp. 1–65.

[4] W. M. P. Aalst, K. M. Hee, A. H. M. Hofstede, N. Sidorova, H. M. W. Verbeek, M. Voorhoeve, and M. T. Wynn, *Soundness of Workflow Nets: Classification, Decidability, and Analysis*, Formal Aspects of Computing **23** (2011), 333–363.

[5] W. M. P. Aalst and A. H. M. Hofstede, *Workflow Patterns: On the Expressive Power of (Petri-Net Based) Workflow Languages*, CPN2002: Proceedings of the 4th Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (Aarhus, Denmark), 2002, pp. 1–20.

[6] ———, *YAWL: Yet Another Workflow Language*, Information Systems **30** (2005), no. 4, 245–275.

[7] W. M. P. Aalst, A. H. M. Hofstede, B. Kiepuszewski, and A. P. Barros, *Workflow Patterns*, Distributed and Parallel Databases **14** (2003), 5–51.

[8] W. M. P. Aalst, K. M. van Hee, and G. J. Houben, *Modelling Workflow Management Systems with High-Level Petri Nets*, Proceedings of the 2nd Workshop on Computer-Supported Cooperative Work, Petri Nets and Related Formalisms (G. de Michelis, C. Ellis, and G. Memmi, eds.), 1994, pp. 31–50.

[9] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: Principles, Techniques, and Tools, 2nd Edition*, Addison-Wesley, 2006.

[10] S. S. Al-Fedaghi, *Beyond Purpose-Based Privacy Access Control*, ADC 2007: Proceedings of the 18th Australasian Database Conference (Ballarat, Australia) (James Bailey and Alan Fekete, eds.), 2007, pp. 23–32.

[11] C. A. Ardagna, S. De Capitani di Vimercati, and P. Samarati, *Enhancing User Privacy Through Data Handling Policies*, Data and Applications Security (Sophia Antipolis, France), 2006, pp. 224–236.

[12] V. Atluri and W. K. Huang, *An Authorization Model for Workflows*, ESORICS 96: Proceedings of the 4th European Symposium on Research in Computer Security (Rome, Italy), Springer Berlin / Heidelberg, 1996, pp. 44–64.

[13] E. Bertino, E. Ferrari, and V. Atluri, *The Specification and Enforcement of Authorization Constraints in Workflow Management Systems*, ACM Trans. Inf. Syst. Secur. **2** (1999), no. 1, 65–104.

[14] P. A. Bonatti, E. Damiani, S. de Capitani di Vimercati, and P. Samarati, *A Component-Based Architecture for Secure Data Publication*, ACSAC'01: Proceedings of the 17th Annual Computer Security Applications Conference, 2001, pp. 309–318.

[15] *Business Process Execution Language for Web Services*, `http://public.dhe.ibm.com/software/dw/specs/ws-bpel/ws-bpel.pdf`, 2002.

[16] M. Bratman, *Intention, Plans, and Practical Reason*, Harvard University Press, 1987.

[17] T. D. Breaux and A. I. Antón, *Deriving Semantic Models from Privacy Policies*, IEEE POLICY'05: Proceedings of the 2005 IEEE International Symposium on Policies for Distributed Systems and Networks (Stockholm, Sweden), 2005, pp. 67–76.

[18] J. W. Byun, E. Bertino, and N. Li, *Purpose-Based Access Control of Complex Data for Privacy Protection*, SACMAT '05: Proceedings of the 10th ACM Symposium on Access Control Models and Technologies, ACM, 2005, pp. 102–110.

[19] J. W. Byun and N. Li, *Purpose Based Access Control for Privacy Protection in Relational Database Systems*, The VLDB Journal **17** (2008), 603–619.

[20] W. Cheung and Y. Gil, *Towards Privacy Aware Data Analysis Workflows for e-Science*, Proceedings of the 2007 Workshop on Semantic e-Science (SeS2007), July 2007, pp. 17–25.

[21] K. Connor, *HL7 Harmonization Proposal July 2012 Security WG Purpose of Use*, 2012, available at `http://wiki.hl7.org/index.php?title=HL7_Security_Document_Library`.

[22] J. Crampton, *A Reference Monitor for Workflow Systems with Constrained Task Execution*, SACMAT'05: Proceedings of the 10th ACM Symposium on Access Control Models and Technologies, 2005, pp. 38–47.

[23] J. Crampton and H. Khambhammettu, *Delegation and Satisfiability in Workflow Systems*, SACMAT'08: Proceedings of the 13th ACM Symposium on Access Control Models and Technologies, 2008, pp. 31–40.

[24] ———, *On delegation and workflow execution models*, SAC'08: Proceedings of the 2008 ACM Symposium on Applied Computing, 2008, pp. 2137–2144.

[25] C. Desmarais, X. Shen, S. Shirmohammadi, A. Cameron, N. D. Georganas, and I. Kerr, *PLUTO – a privacy control protocol for e-commerce communities*, IEEE International Conference on E-Commerce Technology and IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (Washington, DC, USA), 2007, pp. 349–256.

[26] L. L. Dimitropoulos, *Privacy and Security Solutions for Interoperable Health Information Exchange*, `http://www.rti.org/pubs/nationwide_summary.pdf`, 2006.

[27] A. C. Duta and K. Barker, *P4A: A New Privacy Model for XML*, Annual IFIP WG 11.3 Working Conference on Data and Applications Security (London, UK), 2008, pp. 65–80.

[28] C. A. Ellis and G. J. Nutt, *Modeling and Enactment of Workflow Systems*, Application and Theory of Petri Nets 1993, Lecture Notes in Computer Science, vol. 691, Springer Berlin Heidelberg, 1993, pp. 1–16.

[29] *The Enterprise Privacy Authorization Language (EPAL 1.1)*, `http://www.zurich.ibm.com/security/enterprise-privacy/epal`, 2003.

[30] J. Fan, K. Barker, B. Porter, and P. Clark, *Representing Roles and Purpose*, K-CAP '01: Proceedings of the 1st International Conference on Knowledge Capture (New York, NY, USA), ACM, 2001, pp. 38–43.

[31] S. Fischer-Hübner, *IT-security and privacy: Design and use of privacy-enhancing security mechanisms, chapter 5: A task-based privacy model*, Springer, Berlin, Germany, 2001.

[32] H. Haygood, Q. He, S. Smith, and J. Snare, *A Privacy-Aware Database Interface*, Tech. Report TR-2003-05, North Carolina State University, 2003.

[33] Q. He, *Privacy Enforcement with an Extended Role-Based Access Control Model*, Tech. Report TR-2003-09, North Carolina State University, 2003.

[34] Q. He and A. I. Antón, *A Framework for Modeling Privacy Requirements in Role Engineering*, International Workshop on Requirements Engineering – Foundation for Software Quality (Klagenfurt/Velden, Austria), 2003, pp. 115–124.

[35] M. Hilty, D. Basin, and A. Pretschner, *On Obligations*, ESORICS 2005: Proceedings of the 10th European Symposium On Research in Computer Security (Milan, Italy), September 2005, pp. 98–117.

[36] *HL7 Reference Information Model, ANSI/HL7 V3 RIM, R1-2003*, 2003.

[37] D. Hollingsworth, *The Workflow Reference Model*, Tech. Report TC00-1003, Workflow Management Coalition, 1995.

[38] K. Irwin, T. Yu, and W. H. Winsborough, *On the Modeling and Analysis of Obligations*, CCS '06: Proceedings of the 13th ACM Conference on Computer and Communications Security (Alexandria, Virginia, USA), 2006, pp. 134–143.

[39] T. Ishida, M. Yasuda, H. Higaki, and M. Takizawa, *A Transaction-Based Purpose-Oriented Access Control Model for Information Flow Management*, Joho Shori Gakkai Zenkoku Taikai Koen Ronbunshu **58** (1999), no. 3, 3.331–3.332.

[40] *ISO/TS 14265:2011 Health Informatics - classification of purposes for processing personal health information*, 2011.

[41] M. Jafari, P. W. L. Fong, R. Safavi-Naini, K. Barker, and N. P. Sheppard, *Towards Defining Semantic Foundations for Purpose-Based Privacy Policies*, Proceedings of the 1st ACM Conference on Data and Application Security and Privacy, CODASPY '11, ACM, 2011, pp. 213–224.

[42] M. Jafari, R. Safavi-Naini, C. Saunders, and N. P. Sheppard, *Using Digital Rights Management for Securing Data in a Medical Research Environment*, DRM'10: Proceedings of the 10th Annual ACM Workshop on Digital Rights Management, ACM, 2010, pp. 55–60.

[43] M. Jafari, R. Safavi-Naini, and N. P. Sheppard, *Enforcing Purpose of Use via Workflows*, WPES '09: Proceedings of the 8th ACM Workshop on Privacy in the Electronic Society, 2009, pp. 113–116.

[44] M. Jawad, P. S. Alvaredo, and P. Valduriez, *Design of PriServ, a Privacy Service for DHTs*, International Workshop on Privacy and Anonymity in the Information Society (Nantes, France), 2008, pp. 21–26.

[45] T. Jensen, D. Le Metayer, and T. Thorn, *Verification of control flow based security properties*, Proceedings of the 1999 IEEE Symposium on Security and Privacy (Oakland, CA, USA), May 1999, pp. 89–103.

[46] M. E. Kabir, H. Wang, and E. Bertino, *A role-involved purpose-based access control model*, Information Systems Frontiers **14** (2012), 809–822.

[47] B. Kiepuszewski, A. H. M. Hofstede, and C. J. Bussler, *On Structured Workflow Modelling*, CAiSE'2000: Proceedings 12th International Conference on Advanced Information Systems Engineering, volume=1789, Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2000, pp. 431–445.

[48] S. Kripke, *Semantical Considerations On Modal Logic*, Acta Philosophica Fennica **16** (1963), no. 1963, 83–94.

[49] N. Lohmann, E. Verbeek, and R. Dijkman, *Petri Net Transformations for Business Processes âĂŞ A Survey*, Transactions on Petri Nets and Other Models of Concurrency II (Kurt Jensen and WilM.P. Aalst, eds.), Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2009, pp. 46–63.

[50] A. Masoumzadeh and J. B. D. Joshi, *PuRBAC: Purpose-Aware Role-Based Access Control*, On the Move to Meaningful Internet Systems, Part II (Monterrey, Mexico), 2008, pp. 1104–1121.

[51] T. Murata, *Petri nets: Properties, Analysis and Applications*, Proceedings of the IEEE, vol. 77, April 1989, pp. 541–580.

[52] *NCI Thesaurus v.12.04e*, 2012.

[53] Q. Ni, E. Bertino, J. Lobo, and S. B. Calo, *Privacy-Aware Role-Based Access Control*, IEEE Security and Privacy **7** (2009), no. 4, 35–43.

[54] Q. Ni, A. Trombetta, E. Bertino, and J. Lobo, *Privacy-Aware Role Based Access Control*, SACMAT'07: Proceedings of the 12th ACM Symposium on Access Control Models and Technologies (Sophia Antipolis, France), 2007, pp. 41–50.

[55] *OECD Guidelines on the Protection of Privacy and Transborder Flows of Personal Data*, 1980.

[56] *The Platform for Privacy Preferences 1.1 (P3P1.1) Specification*, 2006.

[57] H. Peng, J. Gu, and X. Ye, *Dynamic Purpose-Based Access Control*, International Symposum on Parallel and Distributed Processing with Applications (Sydney, Australia), 2008, pp. 695–700.

[58] J. L. Peterson, *Petri Nets*, ACM Comput. Surv. **9** (1977), no. 3, 223–252.

[59] C. S. Powers, P. Ashley, and M. Schunter, *Privacy Promises, Access Control, and Privacy Management*, ISEC'02: Proceedings of the 3rd International Symposium on Electronic Commerce (Research Triangle Park, North Carolina, US), IEEE Computer Society, 2002, pp. 13–21.

[60] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach (3rd Edition)*, Prentice Hall, 2009.

[61] F. Salim, N. P. Sheppard, and R. Safavi-Naini, *Enforcing P3P Policies Using a Digital Rights Management System*, Privacy Enhancing Technologies Symposium (Ottawa, Canada), 2007, pp. 200–217.

[62] *SNOMED CT, Systematized Nomenclature of Medicine-Clinical Terms*, `http://www.ihtsdo.org/snomed-ct`, 2012.

[63] M. C. Tschantz, A. Datta, and J. M. Wing, *Formalizing and Enforcing Purpose Restrictions in Privacy Policies*, Proceedings of the 2012 IEEE Symposium on Security and Privacy, IEEE, 2012, pp. 176–190.

[64] ———, *Formalizing and Enforcing Purpose Restrictions in Privacy Policies (Full Version)*, Tech. Report CMU-CS-12-106, Carnegie Mellon State University, 2012.

[65] W. van Staden and M. S. Olivier, *Purpose Organisation*, ISSA2005: Proceedings of the 5th Annual Information Security South Africa Conference (Sandton, South Africa), 2005.

[66] ———, *Extending SQL to Allow the Active Usage of Purposes*, International Conference on Trust, Privacy and Security in Digital Business (Krakow, Poland), 2006, pp. 123–131.

[67] ———, *Using purpose lattices to facilitate customisation of privacy agreements*, Trust, Privacy and Security in Digital Business, Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2007, pp. 201–209.

[68] *Workflow Management Coalition Terminology and Glossary*, Tech. Report WFMC-TC-1011, 1999.

[69] G. Winskel, *Formal Semantics of Programming Languages*, The MIT Press, 1993.

[70] *Web Services Business Process Execution Language Version 2.0*, `http://docs.oasis-open.org/wsbpel/2.0/Primer/wsbpel-v2.0-Primer.pdf`, 2007.

[71] *eXtensible Access Control Markup Language (XACML) Version 2.0*, `http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf`, 2005.

[72] *Privacy policy profile of XACML v2.0*, `http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-privacy_profile-spec-os.pdf`, 2005.

[73] *Cross-Enterprise Security and Privacy Authorization (XSPA) Profile of Security Assertion Markup Language (SAML) for Healthcare, Version 1.0*, 2009.

[74] C. Yang and C. N. Zhang, *A Privacy Enhanced Role-Based Access Control Model for Enterprises*, International Conference on Computer Networks and Mobile Computing (Zhangjiajie, China), 2005, pp. 1012–1021.

[75] N. Yang, H. Barringer, and N. Zhang, *A Purpose-Based Access Control Model*, International Symposium on Information Assurance and Security (Manchester, UK), 2007, pp. 143–148.

[76] M. Yasuda, T. Tachikawa, and M. Takizawa, *A purpose-oriented access control model*, Proceedings of 12th International Conference on Information Networking, Jan. 1998, pp. 168–173.

[77] G. Zhan, Z. Li, X. Ye, and J. Wang, *Privacy Preservation and Protection by Extending Generalized Partial Indices*, British National Conference on Databases (Belfast, UK), 2006, pp. 102–114.

Appendix A

Proof of Theorems

A.1 Assumptions and Notations

This section briefly reviews the assumptions that have been made through the paper in order to refresh the reader's memory before reading the proofs.

- Soundness: all the workflow nets are assumed to be sound, i.e. are possible to complete, terminate properly, and do not include any dead tasks (see Section 5.6.2).
- Marked-Graph Structure: all the workflow nets are assumed to have the structure similar to a marked graph, i.e. there is at most one input and one output to/from each place (see Section 5.2).
- Persistence: all workflow nets are assumed to be persistent, i.e. an enabled transition will remain enabled until it is fired (see Section 5.6.3). This is a direct result of the marked-graph structure.
- Safety: all the workflow nets are assumed to be safe, i.e. there will be no more than one token at any given place in any reachable marking from the initial marking (see Section 5.6.4). This is a direct result of the soundness and the marked-graph structure.

Moreover, we use the following notations:

- $Q_{\mathcal{H}}$: the set of all possible firing sequences for the hierarchy net $\mathcal{H}$.
- q_i (where q is a firing sequence): the transition at the ith location of q.
- $M(q)$: the marking sequence corresponding to the firing sequence q.
- $LOOP(t,t')$: a Boolean functions that specifies whether tt' is the return path of a loop, i.e. whether $\exists p_l \in O(\mathcal{P}_{\mathcal{H}}).\ \{(t,p_l),(p_l,t')\} \subset FLOW(EXT(\mathcal{H}))$.
- $M_i(q)$: the ith marking in $M(q)$. Note that for any i we have: $M_i(q) \xrightarrow{q_i} M_{i+1}(q)$.
- $E(\mu)$: the set of transitions that are enabled at the marking μ.

A.2 Proofs

Lemma 1. *The A- and* possible *F-purposes, are respectively equivalent to an A- or F-path.*

$$(\exists t' \in \mathcal{T}.(t,t') \in A^* \wedge t' \in [\![\phi]\!]) \Leftrightarrow (t \in [\![\langle \mathbf{A} \rangle \phi]\!])$$
$$(\exists t' \in \mathcal{T}.(t,t') \in F^* \wedge t' \in [\![\phi]\!]) \Leftrightarrow (t \in [\![\langle \mathbb{F} \rangle \phi]\!])$$

Proof. We prove this for F; the case for A is straightforwardly similar.

($\Leftarrow$):

Starting from $t \in [\![\langle \mathbb{F} \rangle \phi]\!]$ and by expanding the recursive definition of $\langle \mathbb{F} \rangle$ we have the following. Note that the definition of $PRE_F^{\exists}$ implies that $PRE_F^{\exists}(S \cup S') = PRE_F^{\exists}(S) \cup PRE_F^{\exists}(S')$:

$$\begin{aligned} & t \in [\![\phi]\!] \vee \\ & t \in PRE_F^{\exists}([\![\phi]\!]) \vee \\ & t \in {PRE_F^{\exists}}^2([\![\phi]\!]) \vee \\ & \vdots \\ & t \in {PRE_F^{\exists}}^n([\![\phi]\!]) \end{aligned} \tag{A.2.1}$$

for some minimal n where ${PRE_F^{\exists}}^n([\![\phi]\!]) = {PRE_F^{\exists}}^{n+1}([\![\phi]\!])$. Note that as we argued in Section 8.3, since $PRE_F^{\exists}(X)$ is monotonic function over the power set of $\mathcal{T}$ which is a complete lattice, n exists. Now, if we extend the definition of $PRE_F^{\exists}$ we will have:

$$\begin{aligned} & (t \in [\![\phi]\!]) \vee \\ & (\exists t_1 \in \mathcal{T}.\ (t, t_1) \in F\ \wedge t_1 \in [\![\phi]\!]) \vee \\ & (\exists t_1, t_2 \in \mathcal{T}.\ (t, t_1) \in F \wedge (t_1, t_2) \in F\ \wedge t_2 \in [\![\phi]\!]) \vee \\ & \vdots \\ & (\exists t_1, \cdots, t_n \in \mathcal{T}. (t, t_1) \in F \wedge \cdots \wedge (t_{n-1}, t_n) \in F \wedge t_n \in [\![\phi]\!]). \end{aligned}$$

in which each line implies:

$$\exists t' \in \mathcal{T}.\ (t, t') \in F^*\ \wedge t' \in [\![\phi]\!]$$

■

($\Rightarrow$):

Replacing the definition of F^* in $\exists t' \in \mathcal{T}. (t, t') \in F^* \wedge t' \in [\![\phi]\!]$

$$\exists t_1, \cdots, t_n \in \mathcal{T}. (t, t_1) \in F \wedge \cdots \wedge (t_{n-1}, t_n) \in F \wedge t_n \in [\![\phi]\!]$$

which implies $t_{n-1} \in PRE_F^{\exists}([\![\phi]\!])$, and in turn implies: $t_{n-2} \in {PRE_F^{\exists}}^2([\![\phi]\!])$, all the way to $t \in {PRE_F^{\exists}}^n([\![\phi]\!])$. Thereby, according to the expanded definition of $\langle \mathbf{F} \rangle$ above (in Formula A.2.1): $t \in [\![\langle \mathbf{F} \rangle \phi]\!]$.

□

Lemma 2. *If in all possible firing sequences of $\mathcal{H}$, t_n always fires after t_1, then t_1 is a prerequisite of t_n ($n > 1$):*

$$\forall q \in Q_{\mathcal{H}}\ (q_i = t_1 \wedge q_j = t_n) \rightarrow i \leq j\ \Longrightarrow\ t_1 \rightsquigarrow^* t_n$$

Proof. We prove this by induction over the distance between t_1 and t_n. If t_n appears right after t_1 in all possible firing sequences, then firing t_1 enables t_n, otherwise, t_n would be enabled before t_1 and thus there would be some possible firing sequence in which t_n would fire first. Note that $t_1 t_n$ cannot be the return path of any loop, since in that case, t_n and t_1 would be the source and sink of the underlying workflow net of the loop which means there is no way in which t_1 can be enabled before t_n fires.

Now as the induction hypothesis, we assume that if in all possible firing sequence t_n appears after t_1 within the distance of n, then $t_1 \rightsquigarrow^* t_n$. Now, if t_n appears within the range of $n+1$, consider the two cases: b) t_n belongs to a structured loop that does not contain t_1, a) otherwise; i.e. t_n does not belong to a structured loop, or if it does, t_1 is in the same loop.

Case a: Note that t_1 and t_n cannot be part of the same loop, otherwise there would be some possible sequence where t_n would fire before t_1 after some iteration of the loop. So, t_1 does not belong to any loops. Now, consider that in every possible sequence, there is some transaction u_j that enables t_n. Since u_j enables t_n, and t_n is not part of a loop, then u_j can never fire after t_n since that would require a return path from t_n back to u_j. Consider $u_1, \cdots, u_m$ which enable t_n in different sequences. The join type of t_n is either 1) XOR or 2) AND:

Case 1: If t_n's join type is XOR, then each individual u_j can enable it, and hence, no u_j can appear before t_1, otherwise t_n would be possible enable before t_1 and there would be a sequence in which t_n preceded t_1. On the other hand, u_j always appear before t_n. Thus, u_j always happens within the range of n transactions after t_1 and according to the induction hypothesis $t_1 \rightsquigarrow^* u_j$, and since u_j enables t_n and is not in a loop with it, we also have $u_j \rightsquigarrow^* t_n$, thereby: $t_1 \rightsquigarrow^* t_n$.

Case 2: If t_n's join type is AND, then there is at least one u_j that never appears before t_1, otherwise, if all u_js could appear before t_1, then t_n would be possible to enable before t_1 and there would be a sequence in which t_n preceded t_1. So, there exists some u_j that always fire after t_1 and before t_n, thus within a distance of n from t_1, which according to the induction hypothesis implies $t_1 \rightsquigarrow^* u_j$. On the other hand u_j enables t_n and is not in a loop with it, so $u_j \rightsquigarrow^* t_n$, and thereby: $t_1 \rightsquigarrow^* t_n$.

Case b: Consider the outermost loop of containing t_n. Note that such loop cannot contain t_1, otherwise there would be some possible sequence in which t_n would fire before t_1 after some iteration of the loop. Since every structure loop is made of an underlying WN, consider the source task of the underlying WN for the outermost loop and call it t'_n and the set transitions $u_1, \cdots, u_m$ which enable t'_n in all different possible sequences (except the the sink transaction of the loop). Also, consider that the join type of t'_n is necessarily XOR according to the definition of structured loops. So, following a similar argument as Case 1 above proves that $t_1 \rightsquigarrow^* t'_n$.

On the other hand, since t_n belongs to a loop where t'_n is the source, t_n will not be enabled unless t'_n fires (this can be proved straightforwardly by contradiction), thus $t'_n \rightsquigarrow^* t_n$ and thereby $t_1 \rightsquigarrow^* t_n$. □

Theorem 1. *The dynamic and static definitions of* prerequisite-of *are equivalent, i.e., for any given hierarchy net* $\mathcal{H}$*:* $(t' \rightsquigarrow^*_{\mathcal{H}} t) \Longleftrightarrow (t', t) \in F^*_{\mathcal{H}}$

Proof. ($\Rightarrow$):

Consider the definition of $t' \rightsquigarrow t$ from Section 6.3:

$$\exists q \in Q_{\mathcal{H}} \exists i.\ q_i = t' \wedge t \notin E(M_i(q)) \wedge t \in E(M_{i+1}(q)) \ \wedge \ \neg LOOP(t, t').$$

Considering the operational semantics, if t is enabled in $M_{i+1}(q)$ but not in $M_i(q)$, a token must have been added to one of its input places in $M_{i+1}(q)$ as a result of firing t'. Since firing t' can only add tokens to its output places, this implies that one of the output places of t' is an input place to t:

$$\exists p \in PLAC(EXT(\mathcal{H})).\ (t', p) \in FLOW(EXT(\mathcal{H})) \wedge (p, t) \in FLOW(EXT(\mathcal{H})) \wedge p \notin O(\mathcal{P}_{\mathcal{H}})$$

which according to the definition of F implies $(t, t') \in F_{\mathcal{H}}$.

$$t' \rightsquigarrow t \Longleftrightarrow (t', t) \in F_{\mathcal{H}}.$$

Now, we can write $t' \rightsquigarrow^* t$ as:

$$t' \rightsquigarrow t'_1 \rightsquigarrow \cdots \rightsquigarrow t'_n \rightsquigarrow t$$

which implies:

$$(t', t'_1) \in F_{\mathcal{H}} \wedge \cdots \wedge (t'_n, t) \in F_{\mathcal{H}}$$

thereby $(t', t) \in F^*_{\mathcal{H}}$. ■

($\Leftarrow$): The assumption that $(t', t) \in F^*_{\mathcal{H}}$ can be rewritten as the existence if the following path in $FLOW(EXT(\mathcal{H}))$:

$$t_1 p_1 \cdots p_{n-1} t_n$$

in which $t_1 = t'$, $t_n = t$, and none of the p_is are in $O(\mathcal{P}_{\mathcal{H}})$. Now consider the following discussion for any $t_j p_j t_{j+1}$:

Since there is no dead transition, there is at least one firing sequence that includes t_j and there is at least one such firing sequence in which firing t_j produces a token in p_j (if the split type of t_j is AND this is always the case whereas if the split type is XOR, this happens in at least one of the firing sequences). Now, t_{j+1} is either (a) an XOR-join or (b) an AND-join transition:

Case a: Since the join type of t_{j+1} is XOR, a token in p_j enables t_{j+1}, unless it has already been enabled, i.e. there is other tokens in some of its other inputs. Now note that this latter situation contradicts the *safety* of the WN since having more than one token behind an XOR join leads to a possible unsafe marking where more than one token might appear in its output, following numerous firings. In other words, in a sound WN as we defined it, this cannot happen.

Case b: Since the join type of t_{j+1} is AND, and since firing t_j produces a token in p_j, t_{j+1} cannot be already enabled at $M_i(q)$ because that would imply there is already a token in p_j which would be two tokens after firing $t't_j$ contradicting the *safety* of the WN. Now, if firing t_j never enables t_{j+1}, and because t_{j+1} will eventually fire (otherwise the stuck token behind t_{j+1} would violate the *soundness* properties), in every firing sequence there must be some other transition u_j that eventually enables t_{j+1}. Consider the set of $u_1 \cdots u_m$ and note that $u_j t_{j+1}$ cannot be the return path of any loops because the split type of $t_j i + 1$ is AND (see the definition of *structured loops* in Section 5.2.1), so, for all js, $u_j \rightsquigarrow^* t_{j+1}$. Now notice that if all u_js can fire before t_j there would be a sequence where all of them fire, producing a token in all other inputs of t_{j+1} so that when t_j fires, it enables t_{j+1} which proves our point. If there is at least one u_j that never fires before t_j then according to Lemma 2 $t_j \rightsquigarrow^* u_j$ and since $u_j \rightsquigarrow^* t_{j+1}$, $t_j \rightsquigarrow^* t_{j+1}$.

Now, since we proved that for every $t_j t_{i+1}$ in the path, $t_j \rightsquigarrow^* t_{j+1}$, we have $t_1 \rightsquigarrow^* t_n$, or in other words $t' \rightsquigarrow^* t$. □

Theorem 2. *For any formula ϕ, any hierarchy net $\mathcal{H}$ and labelling function L, we have: $[\![\phi]\!]_{\mathcal{H},L} = [\![\phi]\!]_{\mathcal{H},L}$.*

Proof. We prove this by structural induction. Given $[\![\phi]\!] = [\![\phi]\!]$ as the induction hypothesis, we have to prove: $[\![\langle \mathbf{A} \rangle \phi]\!] = [\![\langle \mathbf{A} \rangle \phi]\!]$, $[\![\langle \mathbf{F} \rangle \phi]\!] = [\![\langle \mathbf{F} \rangle \phi]\!]$, and $[\![\langle \mathbb{F} \rangle \phi]\!] = [\![\langle \mathbb{F} \rangle \phi]\!]$ (the case for logical connectives is straightforward).

($\langle \mathbf{A} \rangle, \subseteq$):

According to Lemma 1 $t \in [\![\langle \mathbf{A} \rangle \phi]\!]$ implies $\exists u_n \in \mathcal{T}.\ (t, u_n) \in A^* \wedge u_n \in [\![\phi]\!]$, thereby:

$$\exists u_1, \cdots, u_n \in \mathcal{T}.\ (t, u_1) \in A \wedge (u_1, u_2) \in A \wedge \cdots \wedge (u_{n-1}, u_n) \in A \wedge u_n \in [\![\phi]\!]$$

If $t = u_n$, our point is proven already. Otherwise, since $(u_{i-1}, u_i) \in A$, and according to the definition of A, we have: $\exists w \in W.\ u_{i-1} \in TRAN(w).(u_i, w) \in H$ (where W is the set of all WNs in $\mathcal{H}$ and H is its hierarchy relation). Now consider the definition of $EXT(\mathcal{H})$. Since all the possible firing sequences of w start with its source transition (t_c) and end in its sink transition (t_k), all other transitions of w including u_{i-1} appear between t_c and t_k. Now, the way we defined $EXT(\mathcal{H})$, u^e_i is the only predecessor of t_c, and thus its only prerequisite. Therefore, in all possible firing sequences if t_c appears, it is always after an occurrence of u^e_i, and since occurrence of every transition in w is always after t_c, it is also always after u^e_i. Likewise, since u^x_i is the only non-loop successor of t_k, it eventually follows every occurrence of t_k, and since occurrence of any other transition in w, including u_{i-1} is always before t_k, it is also always before u^x_n.

Repeating this argument for other u_is eventually leads to proving that in all possible firing sequences that contain t, it appears between u^e_j and u^x_j for some $1 \leq j \leq n$ such that $u_j \in [\![\phi]\!]$. ■

($\langle \mathbf{A} \rangle, \supseteq$):

According to the definition there exists u such that in all firing sequences t always occur after u^e and u^x always occurs after t. Now consider all such u_is ($1 \leq i \leq m$) and let $u = u_1$ for notational convenience. Note that according to the construction of the $EXT(\mathcal{H})$ as discussed in Section 5.4.1, these are well-formed, i.e. if u^e_i falls between u^e_j and u^x_j, then u^x_i does too.

Now consider the inner-most u_i, which is u_m. Since in all firing sequences t always occur after u_m^e and u_m^x always occurs after t, according to Lemma 2 we have: $u_m^e \rightsquigarrow^* t$ and $t \rightsquigarrow^* u_m^x$, which in turn according to Theorem 1 imply $(u_m^e, t) \in F^*$ and $(t, u_m^x) \in F^*$. From the structural definition of $EXT(\mathcal{H})$, it follows that t belongs to a WN which is a subnet of u_m, i.e. $\exists w.t \in TRAN(w) \wedge (u_m, w) \in H$ where H is the hierarchy relation of $\mathcal{H}$. This in turn implies $(t, u_m) \in A$, according to the definition of A.

Now the same argument can be repeated for u_{m-1} all the way to u, based on which we will have:

$$(t, u_m) \in A \wedge (u_m, u_{m-1}) \in A \wedge \cdots \wedge (u_2, u_1) \in A$$

which means $(t, u_1) \in A^*$. On the other hand we have $u_1 \in [\![\phi]\!]$, thereby, according to Lemma 1: $t \in [\![\langle \mathbf{A} \rangle \phi]\!]$. ■

$(\langle \mathbb{F} \rangle, \subseteq)$:
Beginning from $t \in [\![\langle \mathbb{F} \rangle \phi]\!]$:

$$\begin{aligned}
&\exists t' \in \mathcal{T}.\ (t, t') \in F^* \ \wedge t' \in [\![\phi]\!] && \text{(Lemma 1)}\\
&\exists q \in Q_w \exists t' \in \mathcal{T}.\ t \rightsquigarrow^* t' \wedge t' \in [\![\phi]\!] && \text{(Theorem 1)}\\
&t \in [\![\langle \mathbb{F} \rangle \phi]\!] && \text{(definition of } [\![\langle \mathbb{F} \rangle \phi]\!])
\end{aligned}$$

■

$(\langle \mathbb{F} \rangle, \supseteq)$:
Beginning from $t \in [\![\langle \mathbb{F} \rangle \phi]\!]$:

$$\begin{aligned}
&\exists t' \in \mathcal{T}.\ t' \in [\![\phi]\!] \wedge t \rightsquigarrow^* t'\\
&\exists t' \in \mathcal{T}.\ t' \in [\![\phi]\!] \wedge (t, t') in F^* && \text{(Theorem 1)}\\
&t \in [\![\langle \mathbb{F} \rangle \phi]\!] && \text{(Lemma 1)}
\end{aligned}$$

■

$(\langle \mathbf{F} \rangle, \subseteq)$:
Consider the expansion of the recursive definition of $[\![\langle \mathbf{F} \rangle \phi]\!]$:

$$\begin{aligned}
&X_1 = [\![\phi]\!]\\
&X_2 = X_1 \cup PRE_F(X_1)\\
&\quad \vdots\\
&X_n = X_{n-1} \cup PRE_F(X_{n-1}) && \text{(A.2.2)}
\end{aligned}$$

So, from $t \in [\![\langle \mathbf{F} \rangle \phi]\!]$ it follows that $t \in X_n$ for some minimal n for which $X_n = X_{n+1}$. Note that as we argued in Section 8.3 such n exists. Now, we prove by induction over n that $t \in [\![\langle \mathbf{F} \rangle \phi]\!]$ implies $t \in [\![\langle \mathbf{F} \rangle \phi]\!]$. The base case of $n = 1$ is straightforward. Now, as the induction hypothesis, assume $t \in X_n \rightarrow t \in [\![\phi]\!]$ and we prove $t \in X_{n+1} \rightarrow t \in [\![\phi]\!]$. If $t \in X_{n+1}$ then X_{n-1} or $PRE_F(X_{n-1})$. The former case proves our point. For the latter case, based on the definition of PRE_F we have:

$$\begin{aligned}
\forall t' \in \mathcal{T}, (t, t') \in F \rightarrow t' \in [\![\langle \mathbf{F} \rangle \phi]\!] \text{ if the split type of } t \text{ is XOR.}\\
\exists t' \in \mathcal{T}, (t, t') \in F \wedge t' \in [\![\langle \mathbf{F} \rangle \phi]\!] \text{ if the split type of } t \text{ is AND.}
\end{aligned}$$

Now, note that all t's are in X_n, so, based on the induction hypothesis: $t' \in [\![\phi]\!]$:

$$\forall q \in Q\,(q_i = t' \rightarrow \exists j \geq i\,(q_j \in [\![\phi]\!]))$$

If the split type of t is XOR, there exists a t' following t in F which according to the operational semantics, will receive a token in its input, and since this token cannot remain stuck behind t' (as it would violate the *soundness* properties), in every possible firing sequence, if t is present, it will be followed by t', and since $t' \in [\![\phi]\!]$, it follows that $t \in [\![\langle \mathbf{F} \rangle \phi]\!]$.

Similarly, if the split type of t is AND, according to the operational semantics, all the t's following t in F receive a token in their inputs, and since this token cannot remain stuck behind t' (as it would violate the *soundness* properties), in every possible firing sequence, if t is present, it will be followed by one of the t's which in any case belongs to $[\![\langle \mathbf{F} \rangle \phi]\!]$. Thus, $t \in [\![\langle \mathbf{F} \rangle \phi]\!]$. ■

$(\langle \mathbf{F} \rangle, \supseteq)$:

We prove the contrapositive, so, let us assume there exists some t so that $t \notin [\![\langle \mathbf{F} \rangle \phi]\!]$, and we show $t \notin [\![\langle \mathbf{F} \rangle \phi]\!]$. Consider the sequence A.2.2 again. On its basis, $t \notin [\![\langle \mathbf{F} \rangle \phi]\!]$ implies $t \notin X_n$. Now we show that beginning from t, a firing sequence exists in which none of the transitions satisfy ϕ.

Note that since for all X_is, $[\![\phi]\!] \subseteq X_i$, and thus, for any j and i, $t_j \notin X_i$ implies $t_j \notin [\![\phi]\!]$.

We start from t_m and add it to the firing sequence. We have $t \notin X_n$, which implies:

$$t \notin X_{n-1} \wedge t \notin PRE_F(X_{n-1})$$

in which the first term implies $t \notin [\![\phi]\!]$.

Case 1: If the split type of t_m is AND, based on the definition of PRE_F we have:

$$\forall u \in \mathcal{T}.\ (t, u) \in F \rightarrow u \notin X_{n-1}$$

which implies:

$$\forall u \in \mathcal{T}.\ (t, u) \in F \rightarrow u \notin [\![\phi]\!].$$

If u does not exist, i.e. t is the sink transition of the WN, the firing sequence in complete. Otherwise, according to the operational semantics, since the split type is AND, all such us appear in the firing sequence after t and as we see, none of them satisfies ϕ.

Case 2: If the split type of t is XOR, based on the definition of PRE_F we have:

$$\exists u \in \mathcal{T}.\ (t, u) \in F \wedge u \notin X_{n-1}$$

which implies

$$\exists u \in \mathcal{T}.\ (t, u) \in F \wedge u \notin [\![\phi]\!].$$

According to the operational semantics, since the split type is XOR, in the possible firing sequence we are building, we can add only on of the successors of t, so we add u which does not satisfy ϕ.

Now, consider that $u \notin X_{n-1}$ implies $u \notin X_{n-2} \wedge u \notin PRE_F(X_{n-2})$, we can repeat the above process for u and grow the firing sequence.

Since us at each iteration are one step ahead along the edges of F and there is no loops or infinite paths in F, this process is bound to get to the sink transition of the WN at some point when the firing sequence concludes (note that there is at least one path from each transition to the sink transition according to the definition of WN). Now, since there is no dead transition in the WN, t is bound to fire and we have built the rest of the firing sequence from t to the sink. Thus, there exists a firing sequence that includes t but no following transitions that is enabled by t satisfies ϕ. □

Lemma 3. *The three purpose operators are distributive over disjunction, i.e.*

$$[\![\langle \mathbf{A} \rangle (\phi_1 \vee \phi_2)]\!] = [\![\langle \mathbf{A} \rangle \phi_1 \vee \langle \mathbf{A} \rangle \phi_2]\!]$$
$$[\![\langle \mathbb{F} \rangle (\phi_1 \vee \phi_2)]\!] = [\![\langle \mathbb{F} \rangle \phi_1 \vee \langle \mathbb{F} \rangle \phi_2]\!]$$
$$[\![\langle \mathbf{F} \rangle (\phi_1 \vee \phi_2)]\!] = [\![\langle \mathbf{F} \rangle \phi_1 \vee \langle \mathbf{F} \rangle \phi_2]\!]$$

Proof. From $t \in [\![\langle \mathbb{F} \rangle (\phi_1 \vee \phi_2)]\!]$, and based on Lemma 1, it follows that

$$\exists t' \in \mathcal{T}.\ (t, t') \in F^* \wedge t' \in [\![\phi_1 \vee \phi_2]\!]$$

or equivalently (based on the definition of the semantics for $\vee$):

$$\exists t' \in \mathcal{T}.\ (t,t') \in F^* \wedge t' \in [\![\phi_1]\!] \vee t' \in [\![\phi_2]\!]$$

thereby: $t \in [\![\langle \mathbb{F} \rangle \phi_1]\!] \cup t \in [\![\langle \mathbb{F} \rangle \phi_2)]\!]$, which, again, based on the definition of $\vee$ is equivalent to $t \in [\![\langle \mathbb{F} \rangle \phi_1 \vee \langle \mathbb{F} \rangle \phi_2)]\!]$. The reverse argument holds straightforwardly and the case for $\langle \mathbf{A} \rangle$ is also straightforwardly similar. ■

Based on the dynamic definition, from $t \in [\![\langle \mathbf{F} \rangle (\phi_1 \vee \phi_2)]\!]$ follows that:

$$\forall q \in Q.\ q_i = t \rightarrow (\exists j.\ q_j = t' \wedge t \rightsquigarrow^* t' \wedge t' \in [\![\phi_1 \vee \phi_2]\!])$$

or equivalently (based on the definition of the semantics for $\vee$):

$$\forall q \in Q.\ q_i = t \rightarrow (\exists j.\ q_j = t' \wedge (t' \in [\![\phi_1]\!] \vee t' \in [\![\phi_2]\!]))$$

or equivalently:

$$\begin{aligned}(\forall q \in Q.\ q_i = t \rightarrow (\exists j.\ q_j = t' \wedge t' \in [\![\phi_1]\!])) \vee \\ (\forall q \in Q.\ q_i = t \rightarrow (\exists j.\ q_j = t' \wedge t' \in [\![\phi_2]\!]))\end{aligned}$$

thereby: $t \in [\![\langle \mathbf{F} \rangle \phi_1]\!] \cup t \in [\![\langle \mathbf{F} \rangle \phi_2)]\!]$, which, again, based on the definition of $\vee$ is equivalent to $t \in [\![\langle \mathbf{F} \rangle \phi_1 \vee \langle \mathbf{F} \rangle \phi_2)]\!]$. The reverse argument holds straightforwardly. □

Lemma 4. *Any* certain *F-purpose is a* possible *F-purpose.* $[\![\langle \mathbf{F} \rangle \phi]\!] \subseteq [\![\langle \mathbb{F} \rangle \phi]\!]$

Proof. Based on Formula A.2.2 and considering the definition of PRE_F, from $t \in [\![\langle \mathbf{F} \rangle \phi]\!]$, it follows that:

$$\exists t' \in \mathcal{T}.\ (t,t') \in F^* \wedge t' \in [\![\phi]\!]$$

which according to Lemma 1, implies $t \in [\![\langle \mathbb{F} \rangle \phi]\!]$. □

Lemma 5.

1. $[\![\langle \mathbf{A} \rangle \langle \mathbf{F} \rangle \phi]\!] = [\![\langle \mathbf{A} \rangle \phi \vee \langle \mathbf{F} \rangle \phi]\!]$
2. $[\![\langle \mathbf{F} \rangle \langle \mathbf{A} \rangle \phi]\!] = [\![\langle \mathbf{A} \rangle \phi \vee \langle \mathbf{F} \rangle \phi]\!]$
3. $[\![\langle \mathbb{F} \rangle \langle \mathbf{A} \rangle \phi]\!] = [\![\langle \mathbf{A} \rangle \phi \vee \langle \mathbb{F} \rangle \phi]\!]$
4. $[\![\langle \mathbf{A} \rangle \langle \mathbb{F} \rangle \phi]\!] = [\![\langle \mathbf{A} \rangle \phi \vee \langle \mathbb{F} \rangle \phi]\!]$
5. $[\![\langle \mathbf{F} \rangle \langle \mathbb{F} \rangle \phi]\!] = [\![\langle \mathbb{F} \rangle \phi]\!]$
6. $[\![\langle \mathbb{F} \rangle \langle \mathbf{F} \rangle \phi]\!] = [\![\langle \mathbb{F} \rangle \phi]\!]$
7. $[\![\langle \mathbf{A} \rangle \langle \mathbf{A} \rangle \phi]\!] = [\![\langle \mathbf{A} \rangle \phi]\!]$
8. $[\![\langle \mathbb{F} \rangle \langle \mathbb{F} \rangle \phi]\!] = [\![\langle \mathbb{F} \rangle \phi]\!]$
9. $[\![\langle \mathbf{F} \rangle \langle \mathbf{F} \rangle \phi]\!] = [\![\langle \mathbf{F} \rangle \phi]\!]$

Proof. (1,⊆):
From the dynamic definition of $t \in [\![\langle \mathbf{A} \rangle \langle \mathbf{F} \rangle \phi]\!]$ we have:

$$\forall q \in Q.\ q_i = t \rightarrow (\exists u \in \mathcal{T} \exists j > i \exists k < i.\ q_k = u^e \wedge q_j = u^x \wedge u \in [\![\langle \mathbf{F} \rangle \phi]\!])$$

And by extending dynamic definition of $\langle \mathbf{F} \rangle$:

$$\begin{aligned}\forall q \in Q.\ q_i = t \rightarrow (\exists u \in \mathcal{T} \exists j > i \exists k < i.\ q_k = u^e \wedge q_j = u^x \wedge \\ (\forall q' \in Q.\ q'_l = u \rightarrow \exists m \geq l (q_m = u' \wedge u' \in [\![\phi]\!])))\end{aligned}$$

which implies:

$$\forall q \in Q.\ q_i = t \rightarrow (\exists u \in \mathcal{T} \exists j > i \exists k < i.\ q_k = u^e \wedge q_j = u^x \wedge (q'_l = u \rightarrow \exists m \geq l (q_m = u' \wedge u' \in [\![\phi]\!])))$$

Note that if $u = u'$ the above formula already implies $t \in [\![\langle \mathbf{A} \rangle \phi]\!]$. Otherwise, consider that any firing sequence that includes t, also includes the sink transition of its subnet (since the subnet must eventually conclude). On the other hand, u^x is the sole successor of the sink transition of the subnet (according to the definition of the extension of a HN from Section 5.4.1), so, it always follows it. Thus, u^x always follows t, and since there is always some u' following u^x so that $u' \in [\![\phi]\!]$, and such u' always follows t, we have $t \in [\![\langle \mathbf{F} \rangle \phi]\!]$. ■

(1,$\supseteq$):

Based on the static semantics definitions, $[\![\phi']\!] \subseteq [\![\langle \mathbf{A} \rangle \phi']\!]$. Letting $\phi' = \langle \mathbf{F} \rangle \phi$ proves that $[\![\langle \mathbf{F} \rangle \phi]\!] \subseteq [\![\langle \mathbf{A} \rangle \langle \mathbf{F} \rangle \phi]\!]$.

On the other hand, $t \in [\![\langle \mathbf{A} \rangle \phi]\!]$ implies that $\exists t'.\ (t, t') \in A^* \wedge t' \in [\![\phi]\!]$. Since, according to the static definitions, $[\![\phi]\!] \subseteq [\![\langle \mathbf{F} \rangle \phi]\!]$, it follows that $t' \in [\![\langle \mathbf{F} \rangle \phi]\!]$ which proves that $[\![\langle \mathbf{A} \rangle \phi]\!] \subseteq [\![\langle \mathbf{A} \rangle \langle \mathbf{F} \rangle \phi]\!]$. Thus, both terms on the right-hand side are subsets of the left-hand side. ■

(2, $\subseteq$):

From the dynamic definition of $\langle \mathbf{F} \rangle$, it follows from $t \in [\![\langle \mathbf{F} \rangle \langle \mathbf{A} \rangle \phi]\!]$ that:

$$\forall q \in Q.\ q_i = t \rightarrow (\exists j \geq i.\ q_j = t' \wedge t' \in [\![\langle \mathbf{A} \rangle \phi]\!])$$

Replacing the dynamic definition of $\langle \mathbf{A} \rangle$ yields:

$$\forall q \in Q.\ q_i = t \rightarrow (\exists j \geq i.\ q_j = t' \wedge (\forall q' \in Q.\ q_k = t' \rightarrow (\exists u \in \mathcal{T} \exists l > k \exists m < k.\ q_m = u^e \wedge q_l = u^x \wedge u \in [\![\phi]\!])))$$

which implies:

$$\forall q \in Q.\ q_i = t \rightarrow (\exists j \geq i.\ q_j = t' \wedge (\exists u \in \mathcal{T} \exists l > j \exists m < j.\ q_m = u^e \wedge q_l = u^x \wedge u \in [\![\phi]\!]))$$

If t is within the same subnet as t', then it always fires after the source transition of the subnet which in turn always fires after u^e, thus $i \geq m$. On the other hand, since for all $q \in Q$, $i \leq j$, and j is always between l and m, i is always smaller than l. Thus, i is always between l and m which implies $t \in [\![\langle \mathbf{A} \rangle \phi]\!]$.

If t is not on the same subnet, every firing sequence that contain both t to t' must also contain the source transition of t''s subnet (t_c) and thereby u^e, since every possible way to get to t' must fire t_c first, and the only way to get to t_c is via u^e. Thus, every firing sequence that include t and t' also includes u^e. On the other hand, since the split type of u^e is AND and it always leads to firing u, every possible firing sequence including u^e also includes u; therefore, all the firing sequences including t also include u and because $u \in [\![\phi]\!]$, it follows that $t \in [\![\langle \mathbf{F} \rangle \phi]\!]$. ■

(2, $\supseteq$):

Based on the static definition $[\![\phi']\!] \subseteq [\![\langle \mathbf{F} \rangle \phi']\!]$. Letting $\phi' = \langle \mathbf{A} \rangle \phi$ proves that $[\![\langle \mathbf{A} \rangle \phi]\!] \subseteq [\![\langle \mathbf{F} \rangle \langle \mathbf{A} \rangle \phi]\!]$.

On the other hand, according to the dynamic definitions, it follows from $t \in [\![\langle \mathbf{F} \rangle \phi]\!]$ that:

$$\forall q \in Q.\ q_i = t \rightarrow (\exists j.\ \geq i q_j = t' \wedge t' \in [\![\phi]\!])$$

Since $[\![\phi]\!] \subseteq [\![\langle \mathbf{A} \rangle \phi]\!]$, it follows that $t' \in [\![\langle \mathbf{A} \rangle \phi]\!]$, thereby $t \in [\![\langle \mathbf{F} \rangle \langle \mathbf{A} \rangle \phi]\!]$. Thus, both terms on the right-hand side are subsets of the left-hand side. ■

(3,$\subseteq$):

Based on the result of Lemma 1, $[\![\langle \mathbb{F} \rangle \langle \mathbf{A} \rangle \phi]\!]$ implies:

$$\exists u, v \in \mathcal{T}.\ (t, u) \in F^* \wedge (u, v) \in A^* \wedge v \in [\![\phi]\!]$$

Now, if t belongs to the same subnet as u, it implies that $(t, v) \in A^*$ and thereby $t \in [\![\langle \mathbf{A} \rangle \phi]\!]$. If t does not belong to the same subnet, we consider the facts that any path to u from outside the subnet contains the source transition of the subnet and the only predecessor of the source transition in F is v^e, thus, there is a path from t to v^e in F. On the other hand, from the definition of $EXT(\mathcal{T})$, we have $(v^e, v) \in F$. So, $(t, v) \in F^*$, thereby $t \in [\![\langle \mathbb{F} \rangle \phi]\!]$. ■

(3,$\supseteq$):

Based on the static definition $[\![\phi']\!] \subseteq [\![\langle \mathbb{F} \rangle \phi']\!]$. Letting $\phi' = \langle \mathbf{A} \rangle \phi$ proves that $[\![\langle \mathbf{A} \rangle \phi]\!] \subseteq [\![\langle \mathbb{F} \rangle \langle \mathbf{A} \rangle \phi]\!]$.

On the other hand, $t \in [\![\langle \mathbb{F} \rangle \phi]\!]$ implies that $\exists t'.\ (t, t') \in F^* \wedge t' \in [\![\phi]\!]$. Since, according to the static definitions, $[\![\phi]\!] \subseteq [\![\langle \mathbf{A} \rangle \phi]\!]$, it follows that $t' \in [\![\langle \mathbf{A} \rangle \phi]\!]$ which proves that $[\![\langle \mathbb{F} \rangle \phi]\!] \subseteq [\![\langle \mathbb{F} \rangle \langle \mathbf{A} \rangle \phi]\!]$. Thus, both terms on the right-hand side are subsets of the left-hand side. ■

(4,$\subseteq$):

Based on the result of Lemma 1, $[\![\langle \mathbf{A} \rangle \langle \mathbb{F} \rangle \phi]\!]$ implies:

$$\exists u, v \in \mathcal{T}.\ (t, u) \in A^* \wedge (u, v) \in F^* \wedge v \in [\![\phi]\!]$$

Now if $u = v$, it follows that $t \in [\![\langle \mathbf{A} \rangle \phi]\!]$, otherwise, since u^x is the only successor of u in F, $(u, v) \in F^*$ implies $(u^x, v) \in F^*$. On the other hand, since there is a path from t to the sink transition in its subnet, and based on the definition of $EXT(\mathcal{T})$ we know that the sink transition is connected to u^x, we have $(t, u^x) \in F^*$. Thus, $(t, v) \in F^*$, which according to Lemma 1 implies $t \in [\![\langle \mathbf{F} \rangle \phi]\!]$. ■

(4,$\supseteq$):

Straightforwardly similar to (3,$\supseteq$). ■

(5,$\subseteq$):

Based on Formula A.2.2, and considering the definition of PRE_F, it follows from $t \in [\![\langle \mathbf{F} \rangle \langle \mathbb{F} \rangle \phi]\!]$ that:

$$\exists t' \in \mathcal{T}.\ (t, t') \in F^* \wedge t' \in [\![\langle \mathbb{F} \rangle \phi]\!]$$

And following Lemma 1 $t' \in [\![\langle \mathbb{F} \rangle \phi]\!]$ implies:

$$\exists t'' \in \mathcal{T}.\ (t', t'') \in F^* \wedge t'' \in [\![\phi]\!]$$

which, together, and by applying Lemma 1 again imply $t \in [\![\langle \mathbb{F} \rangle \phi]\!]$. ■

(5,$\supseteq$):

Based on the static definition $[\![\phi']\!] \subseteq [\![\langle \mathbf{F} \rangle \phi']\!]$. Let $\phi' = \langle \mathbb{F} \rangle \phi$. ■

(6,$\subseteq$):

Based on Lemma 1, $\langle \mathbb{F} \rangle$, implies:

$$\exists t' \in \mathcal{T}.\ (t, t') \in F^* \wedge t' \in [\![\langle \mathbf{F} \rangle \phi]\!]$$

Now, based on Formula A.2.2, and considering the definition of PRE_F, it follows from $t' \in [\![\langle \mathbf{F} \rangle \phi]\!]$ that

$$\exists t'' \in \mathcal{T}.\ (t', t'') \in F^* \wedge t'' \in [\![\phi]\!]$$

which, together, and by applying Lemma 1 again imply $t \in [\![\langle \mathbb{F} \rangle \phi]\!]$. ■

(6,$\supseteq$):

Based on the dynamic definition of $\langle \mathbb{F} \rangle$, it follows from $t \in [\![\langle \mathbb{F} \rangle \phi]\!]$ that:

$$\exists t' \in \mathcal{T}.\ t \rightsquigarrow^* t' \wedge t' \in [\![\phi]\!]$$

From the static definitions follows that $[\![\phi]\!] \subseteq [\![\langle \mathbf{F} \rangle \phi]\!]$, so, $t' \in [\![\langle \mathbf{F} \rangle \phi]\!]$ which implies $t \in [\![\langle \mathbb{F} \rangle \langle \mathbf{F} \rangle \phi]\!]$. ■

(7,$\subseteq$):

According to Lemma 1, $t \in [\![\langle \mathbf{A} \rangle \langle \mathbf{A} \rangle \phi]\!]$ leads to:

$$\exists t' \in \mathcal{T}.\ t' \in [\![\langle \mathbf{A} \rangle \phi]\!] \wedge (t, t') \in A^*.$$

and thus:

$$\exists t', t'' \in \mathcal{T}.\ t'' \in [\![\langle \mathbf{A} \rangle \phi]\!] \wedge (t, t') \in A^* \wedge (t', t'') \in A^*$$

which implies $t \in [\![\langle \mathbf{A} \rangle \phi]\!]$. ■

(7,⊇):

Based on the static definition $[\![\phi']\!] \subseteq [\![\langle \mathbf{A} \rangle \phi']\!]$. Let $\phi' = \langle \mathbf{A} \rangle \phi$. ■

(8,⊆):

Straightforwardly similar to that of 7. ■

(8,⊇):

Based on the static definition $[\![\phi']\!] \subseteq [\![\langle \mathbb{F} \rangle \phi']\!]$. Let $\phi' = \langle \mathbb{F} \rangle \phi$. ■

(9,⊆):

Using the dynamic definition of $\langle \mathbf{F} \rangle$, it follows from $t \in [\![\langle \mathbf{F} \rangle \langle \mathbf{F} \rangle \phi]\!]$ that:

$$\forall q \in Q\,(q_i{=}t \rightarrow \exists j{\geq}i\,(q_j{=}t' \wedge t' \in [\![\langle \mathbf{F} \rangle \phi]\!]))$$

Expanding the definition of $\langle \mathbf{F} \rangle$ again:

$$\forall q \in Q\,(q_i{=}t \rightarrow \exists j{\geq}i\,(q_j{=}t' \wedge (\forall q' \in Q\,(q'_k{=}t' \rightarrow \exists l{\geq}k\,(q'_l{=}t'' \wedge t' \in [\![\phi]\!])))))$$

which implies:

$$\forall q \in Q\,(q_i{=}t \rightarrow \exists j{\geq}i\,(q_j{=}t' \wedge (\exists l{\geq}j\,(q_l{=}t'' \wedge t'' \in [\![\phi]\!]))))$$

or equivalently:

$$\forall q \in Q\,(q_i{=}t \rightarrow \exists l{\geq}i\,(q_l{=}t'' \wedge t'' \in [\![\phi]\!]))$$

thereby $t \in [\![\langle \mathbf{F} \rangle \phi]\!]$. ■

(9,⊇):

Based on the static definition $[\![\phi']\!] \subseteq [\![\langle \mathbf{F} \rangle \phi']\!]$. Let $\phi' = \langle \mathbf{F} \rangle \phi$. □

Theorem 3. *Any combination of* certain *F- and A- purposes which includes both operators, is equivalent to the disjunction of a* certain *F- and an A-purpose, i.e. for any $n > 1$ we have:*

$$[\![\langle X_1 \rangle \cdots \langle X_n \rangle \phi]\!] = [\![\langle \mathbf{A} \rangle \phi \vee \langle \mathbf{F} \rangle \phi]\!]\ (X_i \in \{\mathbf{A}, \mathbf{F}\})$$

given that $\exists i, j.\ X_i = \mathbf{F} \wedge X_j = \mathbf{A}$.

Proof. The proof is straightforwardly similar to that of Theorem 5. □

Theorem 4. *Any combination of* possible *F- and A- purposes which includes both operators, is equivalent to the disjunction of a* possible *F- and an A-purpose, i.e. for any $n > 1$ we have:*

$$[\![\langle X_1 \rangle \cdots \langle X_n \rangle \phi]\!] = [\![\langle \mathbf{A} \rangle \phi \vee \langle \mathbb{F} \rangle \phi]\!]\ (X_i \in \{\mathbf{A}, \mathbb{F}\})$$

given that $\exists i, j.\ X_i = \mathbb{F} \wedge X_j = \mathbf{A}$.

Proof. The proof is straightforwardly similar to that of Theorem 5. □

Theorem 5. *Any combination of A-,* certain *F- and* possible *F- purposes that include all the three operators, is equivalent to the disjunction of a* possible *F- and an A-purpose, i.e. for any $n > 2$ we have:*

$$[\![\langle X_1 \rangle \cdots \langle X_n \rangle \phi]\!] = [\![\langle \mathbf{A} \rangle \phi \vee \langle \mathbb{F} \rangle \phi]\!]\ (X_i \in \{\mathbf{A}, \mathbf{F}, \mathbb{F}\})$$

given that $\exists i, j, k.\ X_i = \mathbb{F} \wedge X_j = \mathbf{F} \wedge X_k = \mathbf{A}$.

Proof. We prove this inductively over n, the number of the operators where $n \geq 3$. The base cases are:

1. $\phi = \langle \mathbf{A} \rangle \langle \mathbb{F} \rangle \langle \mathbf{F} \rangle \psi$
2. $\phi = \langle \mathbf{A} \rangle \langle \mathbf{F} \rangle \langle \mathbb{F} \rangle \psi$
3. $\phi = \langle \mathbb{F} \rangle \langle \mathbf{A} \rangle \langle \mathbf{F} \rangle \psi$
4. $\phi = \langle \mathbb{F} \rangle \langle \mathbf{F} \rangle \langle \mathbf{A} \rangle \psi$
5. $\phi = \langle \mathbf{F} \rangle \langle \mathbf{A} \rangle \langle \mathbb{F} \rangle \psi$
6. $\phi = \langle \mathbf{F} \rangle \langle \mathbb{F} \rangle \langle \mathbf{A} \rangle \psi$

Case 1:

$$\begin{aligned} [\![\langle \mathbf{A} \rangle \langle \mathbb{F} \rangle \langle \mathbf{F} \rangle \psi]\!] &= [\![\langle \mathbf{A} \rangle \langle \mathbb{F} \rangle \psi]\!] && \text{(Lemma 5)} \\ &= [\![\langle \mathbf{A} \rangle \psi \vee \langle \mathbb{F} \rangle \psi]\!] && \text{(Lemma 5)} \end{aligned}$$

■

Case 2:

$$\begin{aligned} [\![\langle \mathbf{A} \rangle \langle \mathbf{F} \rangle \langle \mathbb{F} \rangle \psi]\!] &= [\![\langle \mathbf{A} \rangle \langle \mathbb{F} \rangle \psi]\!] && \text{(Lemma 5)} \\ &= [\![\langle \mathbf{A} \rangle \psi \vee \langle \mathbb{F} \rangle \psi]\!] && \text{(Lemma 5)} \end{aligned}$$

■

Case 3:

$$\begin{aligned} [\![\langle \mathbb{F} \rangle \langle \mathbf{A} \rangle \langle \mathbf{F} \rangle \psi]\!] &= [\![\langle \mathbb{F} \rangle (\langle \mathbf{A} \rangle \psi \vee \langle \mathbf{F} \rangle \psi)]\!] && \text{(Lemma 5)} \\ &= [\![\langle \mathbb{F} \rangle \langle \mathbf{A} \rangle \psi \vee \langle \mathbb{F} \rangle \langle \mathbf{F} \rangle \psi)]\!] && \text{(Lemma 3)} \\ &= [\![\langle \mathbb{F} \rangle \psi \vee \langle \mathbf{A} \rangle \psi]\!] && \text{(Lemma 5)} \end{aligned}$$

■

Case 4:

$$\begin{aligned} [\![\langle \mathbb{F} \rangle \langle \mathbf{F} \rangle \langle \mathbf{A} \rangle \psi]\!] &= [\![\langle \mathbb{F} \rangle (\langle \mathbf{A} \rangle \psi \vee \langle \mathbf{F} \rangle \psi)]\!] && \text{(Lemma 5)} \\ &= [\![\langle \mathbb{F} \rangle \psi \vee \langle \mathbf{A} \rangle \psi]\!] && \text{(similar to case 3)} \end{aligned}$$

■

Case 5:

$$\begin{aligned} [\![\langle \mathbf{F} \rangle \langle \mathbf{A} \rangle \langle \mathbb{F} \rangle \psi]\!] &= [\![\langle \mathbf{F} \rangle (\langle \mathbf{A} \rangle \psi \vee \langle \mathbb{F} \rangle \psi)]\!] && \text{(Lemma 5)} \\ &= [\![\langle \mathbf{F} \rangle \langle \mathbf{A} \rangle \psi \vee \langle \mathbf{F} \rangle \langle \mathbb{F} \rangle \psi)]\!] && \text{(Lemma 3)} \\ &= [\![\langle \mathbf{F} \rangle \psi \vee \langle \mathbb{F} \rangle \psi \vee \langle \mathbf{A} \rangle \psi]\!] && \text{(Lemma 5)} \\ &= [\![\langle \mathbb{F} \rangle \psi \vee \langle \mathbf{A} \rangle \psi]\!] && \text{(Lemma 4)} \end{aligned}$$

■

Case 6:

$$\begin{aligned} [\![\langle \mathbf{F} \rangle \langle \mathbb{F} \rangle \langle \mathbf{A} \rangle \psi]\!] &= [\![\langle \mathbf{F} \rangle (\langle \mathbf{A} \rangle \psi \vee \langle \mathbb{F} \rangle \psi)]\!] && \text{(Lemma 5)} \\ &= [\![\langle \mathbb{F} \rangle \psi \vee \langle \mathbf{A} \rangle \psi]\!] && \text{(similar to case 5)} \end{aligned}$$

■

Now consider a formula ϕ with $n+1$ consecutive operators applied to ψ ($n \geq 3$). It is either the case that:

1. $\phi = \langle \mathbf{A} \rangle \phi'$
2. $\phi = \langle \mathbb{F} \rangle \phi'$
3. $\phi = \langle \mathbf{F} \rangle \phi'$

According to the induction hypothesis, $[\![\phi']\!] = [\![\langle \mathbf{A}\rangle\psi \vee \langle \mathbb{F}\rangle\psi]\!]$. Now we prove that $[\![\phi]\!] = [\![\langle \mathbf{A}\rangle\psi \vee \langle \mathbb{F}\rangle\psi]\!]$.

Case 1:

$$\begin{aligned}
[\![\phi]\!] = [\![\langle \mathbf{A}\rangle\phi']\!] &= [\![\langle \mathbf{A}\rangle(\langle \mathbf{A}\rangle\psi \vee \langle \mathbb{F}\rangle\psi)]\!] && \text{(induction hypothesis)}\\
&= [\![\langle \mathbf{A}\rangle\langle \mathbf{A}\rangle\psi \vee \langle \mathbf{A}\rangle\langle \mathbb{F}\rangle\psi)]\!] && \text{(Lemma 3)}\\
&= [\![\langle \mathbf{A}\rangle\psi \vee \langle \mathbb{F}\rangle\psi]\!] && \text{(Lemma 5)}
\end{aligned}$$

■

Case 2:

$$\begin{aligned}
[\![\phi]\!] = [\![\langle \mathbb{F}\rangle\phi']\!] &= [\![\langle \mathbb{F}\rangle(\langle \mathbf{A}\rangle\psi \vee \langle \mathbb{F}\rangle\psi)]\!] && \text{(induction hypothesis)}\\
&= [\![\langle \mathbb{F}\rangle\langle \mathbf{A}\rangle\psi \vee \langle \mathbb{F}\rangle\langle \mathbb{F}\rangle\psi)]\!] && \text{(Lemma 3)}\\
&= [\![\langle \mathbf{A}\rangle\psi \vee \langle \mathbb{F}\rangle\psi]\!] && \text{(Lemma 5)}
\end{aligned}$$

■

Case 3:

$$\begin{aligned}
[\![\phi]\!] = [\![\langle \mathbf{F}\rangle\phi']\!] &= [\![\langle \mathbf{F}\rangle(\langle \mathbf{A}\rangle\psi \vee \langle \mathbb{F}\rangle\psi)]\!] && \text{(induction hypothesis)}\\
&= [\![\langle \mathbf{F}\rangle\langle \mathbf{A}\rangle\psi \vee \langle \mathbf{F}\rangle\langle \mathbb{F}\rangle\psi)]\!] && \text{(Lemma 3)}\\
&= [\![\langle \mathbf{F}\rangle\psi \vee \langle \mathbf{A}\rangle\psi \vee \langle \mathbb{F}\rangle\psi]\!] && \text{(Lemma 5)}\\
&= [\![\langle \mathbf{A}\rangle\psi \vee \langle \mathbb{F}\rangle\psi]\!] && \text{(Lemma 4)}
\end{aligned}$$

□