NAIST-IS-DD0361214

Doctoral Dissertation

Formal Models for XML Access Control and Aspect-Oriented Programs beyond Regular Languages

Isao Yagi

September 8, 2006

Department of Information Processing
Graduate School of Information Science
Nara Institute of Science and Technology

A Doctoral Dissertation
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
Doctor of ENGINEERING

Isao Yagi

Thesis Committee:

Professor Hiroyuki Seki	(Supervisor)
Professor Minoru Ito	(Co-supervisor)
Professor Shunsuke Uemura	(Member)
Associate Professor Yuichi Kaji	(Co-supervisor)

Formal Models for XML Access Control and Aspect-Oriented Programs beyond Regular Languages*

Isao Yagi

Abstract

Recently, information systems including computer and network systems have become quite large according to striking development of information technologies. As systems using these technologies tend to be complicated, system designers often face unintended consequences of program execution. One of the solutions of such problems is formal modeling of systems. Given a simple formal model of a system, we can understand its complicated behavior more easily, and formal analysis and verification of the system become possible. In this thesis, we endeavor to provide formal models for access control in XML databases and execution control in systems based on Aspect-Oriented Programming (AOP). XML access control and AOP are emerging areas that have received much attention from both researchers and practitioners, and the models for them have not been established yet. We model both systems based on formal language theory, which is simple and many analyzing algorithms are known.

In Chapter 2, we propose a formal model for XML database access control and define a static analysis problem for access control. Given an access control policy and a query expression, static analysis determines whether the query does not access any elements nor attributes that are prohibited by the policy. In a related work, policies and queries were modeled as regular sets of paths in trees. However, the model cannot accurately represent some policies. We model

* Doctoral Dissertation, Department of Information Processing, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DD0361214, September 8, 2006.

both a policy and a query as tree automata, and a policy is provided with two alternative semantics; AND-semantics and OR-semantics. We investigate the computational complexities of the static analysis problems in AND-semantics and OR-semantics and show that they are solvable in quadratic time and EXPTIME-complete, respectively. We show that our query model is sufficiently general by showing that the expressive power of our model is strictly stronger than Neven's query automata. We also discuss a consistency problem of policies in schema transformation of XML databases and show that the problem is decidable.

In Chapter 3, we propose A-LTS, a simple model of aspect-oriented programs, based on labeled transition systems. The model is especially concerned with the recursion of weaving advices, which is not considered in related works. We investigate the expressive power of A-LTS and show that it is strictly stronger than finite state machines and strictly weaker than pushdown automata (PDA). Then we compare in detail the expressive power of A-LTS with a few subclasses of PDA (or equivalently of context-free grammars): classes of deterministic PDA and linear grammars. We also discuss the relationship between A-LTS and the pointcuts of AspectJ, a well-known AOP language, and confirm that they can be represented by A-LTS.

Keywords:

formal language theory, formal model, XML database, access control, tree automaton, aspect-oriented programming, labeled transition system, pushdown automaton

XML アクセス制御とアスペクト指向プログラムに対する正規言語を超える表現能力をもった形式モデルの提案*

八木 勲

内容梗概

近年，情報技術の著しい発展に伴い，大規模な情報システムが構築されるようになった．このようなシステムは，高度な機能を持つ反面，動作が複雑になるため，システム開発者の予期しない振る舞いを行うことがある．この問題を解決する方法の 1 つとして，システムの形式モデル化がある．システムを簡潔な形式モデルで表すことで，複雑な動作を把握しやすくなり，システムの形式検証にも利用することができる．本論文では，XML データベースのアクセス制御，および，アスペクト指向プログラム (AOP) の実行制御の形式モデル化を試みた．これらの技術は，研究者，開発者の双方から注目されている分野であるが，そのモデルは未だ確立されていない．本研究では，簡潔で，かつ，様々な性質が知られている形式言語理論に基づいて，これらをモデル化する．

第 2 章では，XML データベースのアクセス制御に対する形式モデルを提案し，アクセス制御の静的解析問題について考察する．静的解析問題とは，アクセス制御ポリシ（以下，ポリシと呼ぶ）と問い合わせが与えられたとき，ポリシによってアクセスが禁止された要素または属性へのアクセスが発生しないかどうかを，問い合わせ実行前に調べることをいう．既存研究では，ポリシおよび問い合わせを木中のパスの正規集合でモデル化しているが，この方法では正確に表現できないポリシが存在する．本論文では，ポリシおよび問い合わせを木オートマトンでモデル化し，ポリシに 2 つの意味論（AND 意味論，OR 意味論）を与えた．そして静的解析問題の時間計算量が，AND 意味論の下では 2 乗オーダであり，OR 意味論の下では決定性指数時間完全であることを示した．また，提案した問い合わ

* 奈良先端科学技術大学院大学 情報科学研究科 情報処理学専攻 博士論文，NAIST-IS-DD0361214, 2006 年 9 月 8 日.

せモデルの表現能力が，Neven らの query automaton より真に大きいことを示した．さらに，XML データベースのスキーマ変換におけるポリシの整合性問題について議論し，この問題が決定可能であることを示した．

第 3 章では，ラベル付き遷移システムに基づいた，アスペクト指向プログラムのモデル A-LTS を提案する．A-LTS は，既存モデルでは考慮されていないアドバイスの再帰性を特に考慮している．そして，A-LTS の表現能力が有限状態機械の表現能力より真に大きく，プッシュダウンオートマトン（PDA）の表現能力より真に小さいことを示した．さらに，PDA（および，文脈自由文法）のサブクラスである，決定性 PDA と線形文法との表現能力の比較も行った．最後に，よく知られた AOP 言語である AspectJ のポイントカットと A-LTS の関係を議論し，それらが A-LTS によって表現可能であることを示した．

キーワード

形式言語理論，形式モデル，XML データベース，アクセス制御，木オートマトン，アスペクト指向プログラミング，ラベル付き遷移システム，プッシュダウンオートマトン

List of Publications

1 Journal Papers

(1) Isao Yagi, Yoshiaki Takata and Hiroyuki Seki: A Static Analysis Using Tree Automata for XML Access Control, Computer Software, Vol.23, No.3, pp.51–65, July 2006.

2 International Conferences (Reviewed)

(1) Isao Yagi, Yoshiaki Takata and Hiroyuki Seki: A Static Analysis Using Tree Automata for XML Access Control, Automated Technology for Verification and Analysis – Third International Symposium, ATVA2005, Taipei, Taiwan, October 2005, Proceedings, LNCS3707, pp.234–247, Oct. 2005.

3 Workshops

(1) Isao Yagi, Yoshiaki Takata and Hiroyuki Seki: A Formal Model of Aspect-Oriented Programs Based on Labeled Transition Systems, Technical Report of IEICE, SS2003–46, pp.1–6, Mar. 2004 (in Japanese).

(2) Isao Yagi, Yoshiaki Takata and Hiroyuki Seki: A Static Analysis Using Tree Automata for XML Access Control, The Seventh JSSST Workshop on Programming and Programming Languages (PPL2005), p.43, Mar. 2005 (in Japanese).

(3) Isao Yagi, Yoshiaki Takata and Hiroyuki Seki: A Static Analysis Using Tree Automata for XML Access Control, Technical Report of IEICE, SS2005–18, pp.1–6, June 2005.

(4) Isao Yagi, Yoshiaki Takata and Hiroyuki Seki: A Static Analysis Using Tree Automata for XML Access Control, Proceedings of the Second Symposium on Science and Technology for System Verification, AIST–PS–2005–017, pp.33–51, Oct. 2005.

Acknowledgements

During the course of this work, I have received help from many individuals. First, and foremost, I would like to thank my supervisor Professor Hiroyuki Seki for his continuous support, encouragement and guidance of the work. Also foremost, I would like to thank Professor Shunsuke Uemura and Professor Minoru Ito for their invaluable comments and helpful suggestions concerning this thesis. I also wish to thank Associate Professor Yuichi Kaji for his valuable comments.

I am very grateful to Assistant Professor Yoshiaki Takata for his valuable comments and discussions throughout the work.

Finally, I would like to thank all the members of Seki Laboratory.

Contents

List of Figures

Chapter 1

Introduction

Recently, information systems including computer and network systems have become quite large according to striking development of information technologies. As systems using these technologies tend to be complicated, system designers often face unintended consequences of program execution. For example, the behavior of a system with exception handling is much complicated, and designers seldom notice wrong control flows until the flows actually take place. In a database with a large number of users, an access control policy may become complex, and sometimes a user who should be prohibited from accessing specific data in the database may unintentionally be allowed accessing them. One of the solutions of such problems is formal modeling of systems. Given a simple formal model of a system, we can understand its complicated behavior more easily, and formal analysis and verification of the system become possible. In this thesis, we endeavor to provide formal models for access control in XML databases and execution control in systems based on Aspect-Oriented Programming (AOP). XML access control and AOP are emerging areas that have received much attention from both researchers and practitioners, and the models for them have not been established yet. We model both systems based on formal language theory, which has the following advantages:

- Models based on formal language theory are simple and easy to analyze.
- Relationship between subclasses of formal languages has widely been investigated.

- Many algorithms such as boolean operations (union, intersection, complement on languages) and emptiness test are known.

For XML access control, we provide a model based on tree automata and discuss a static analysis of an access control and its computational complexity. We also show that the expressive power of our model is more general than Neven's query automata [27]. For AOP, we provide a model based on pushdown automata (PDA) and compare the expressive power of our model with subclasses of PDA.

XML database access control XML is now becoming the *de facto* standard for data exchange format and is also widely used as a schema language for database of structured documents (XML database). Since a schema defined by XML is more complex than traditional database schema such as relational database schema, a few query languages specialized to XML database are being developed such as XPath [7] and XQuery [6]. Access control is one of the most important technologies for database security, and several models for XML database access control have been proposed [5, 10, 17, 25]. Usually, an access control policy (e.g., 'a professor can read every record of student files,' and 'a student can read the record of her/himself only.') is provided to a database management system (DBMS) in advance. When a query is issued, DBMS checks whether the query is valid for the access control policy. That is, DBMS determines whether the query is accessing only the portion that the policy permits to access. If the query is valid, then DBMS permits the access, and the query is aborted otherwise. In this kind of runtime access control process, DBMS is required to determine whether or not the access is granted by the policy whenever an element or attribute in an XML database is being accessed, and this process sometimes brings non-negligible overhead to DBMS. Static analysis, which decides whether a query is always valid for (or always against) a policy without examining an actual database, is effective in overcoming this problem. For example, Murata et al. [25] have shown an empirical results that 65% of the pairs of queries and policies does not require runtime access check.

Especially for XML databases, Murata et al. [25] discuss the static analysis problem that, given an access control policy AP, an XML schema S and a query R, decides whether the query R is always valid for (or always against) the policy AP in any XML databases conforming the schema S. In their setting, both

a policy and a query are given as XPath expressions and a schema is given as a regular tree grammar (or equivalently, a tree automaton). Then, three finite automata on strings are constructed by extracting regular expressions from these XPath expressions and the tree automaton, and the static analysis problem is reduced to the set-inclusion problem for regular string languages. They also present experimental results on static analysis of XMark queries and show their method is efficient and has enough scalability. They mentioned that they did not use tree automata because decision procedures for tree automata need more time and space complexity than string automata. However, using regular expression as approximation of XPath expression and tree automaton loses information on the structure of the original tree. For example, we cannot distinguish the first son labeled with tag 'a' and the second son labeled with the same tag 'a' in the regular expression approximation. A more concrete discussion is provided in the following chapter.

In Chapter 2, we propose a static analysis method based on tree automata theory. Both a policy and a query are modeled as tree automata, and a policy is provided with two alternative semantics; AND-semantics and OR-semantics. We investigate the computational complexity of the static analysis problem. We show that our query model is sufficiently general by showing that the expressive power of our model is strictly stronger than Neven's query automata. We also discuss a consistency problem of policies in schema transformation of XML databases and show that the problem is decidable.

Aspect-Oriented Programming AOP is a new programming paradigm addressing the shortcomings of Object-Oriented Programming (OOP). OOP is not always suitable for describing functions and operations that cannot be encapsulated within a single class of objects (e.g., logging and synchronizing). These functions and operations are called crosscutting concerns because they straddle more than one class. AOP introduces a new module unit "aspect" for describing a crosscutting concern as a single module. In AOP, any procedure describing a crosscutting concern can be inserted into a specific execution point of a program. Each execution point where a procedure can be inserted is called a **join point**, and the inserted procedure is called an **advice**. When an advice is inserted into a program, we say the advice is woven into a basic program. The set of join points

to which a specific advice should be connected is called a **pointcut**. An aspect is a pair of an advice and a pointcut.

AOP programs sometimes act contrary to a programmer's intention because aspects are tangled with a basic program. One direction to solve this problem is the formal modeling of AOP programs. Several works on formal modeling of AOP have been done; however, a simple, clear, and widely accepted formal model has not been established yet.

In Chapter 3, we propose A-LTS, a simple model of aspect-oriented programs, based on labeled transition systems. The model is especially concerned with the recursion of weaving advices, which is not considered in related works. We investigate the expressive power of A-LTS and show that it is strictly stronger than finite state machines and strictly weaker than pushdown automata (PDA). Then we compare in detail the expressive power of A-LTS with a few subclasses of PDA (or equivalently of context-free grammars): classes of deterministic PDA and linear grammars. We also discuss the relationship between A-LTS and the pointcuts of AspectJ and confirm that they can be represented by A-LTS.

Chapter 2

Static Analysis using Tree Automata for XML Access Control

2.1 Introduction

In this chapter, we propose a formal model for access control of XML databases and provide a static analysis method for XML access control based on tree automata theory. Static analysis determines whether a given query does not access any elements nor attributes that are prohibited by a given policy, for reducing the cost of runtime check made by DBMS. Following Murata et al. [25], we consider the node level (or element level) fine-grained access control.

We first model both an access control policy and a query by tree automata (TA), called a *policy TA* and a *query TA*, respectively. For this purpose, we introduce a charged alphabet to distinguish permission/denial in a policy and access/non-access in a query in a simple and uniform way. For simplicity, database schema is not considered in this thesis: A schema defined by DTD or XML schema can be represented by a tree automaton, and it is easy to incorporate a schema as a part of a problem instance in our setting. Next, a static analysis problem is defined based on the tree languages accepted by a policy TA and a query TA. We introduce two alternative semantics, AND-semantics and OR-semantics. Generally, an access control policy may contain conflicts, e.g., one rule says that

a student file is allowed to read while another rule says no [19, 16]. These two semantics provide alternative conflict resolution strategies (if any conflict occurs in a policy). Intuitively, a query is valid for a policy in AND-semantics if for every tree t, every possible run of the query on t meets *all* the individual policies for t. A query is valid for a policy in OR-semantics if for every tree t, every possible run of the query on t meets *one of* the individual policies for t. Finally, we investigate the computational complexity of the static analysis problem and show that the problem in AND-semantics is solvable in square time while the problem in OR-semantics is EXPTIME-complete. Also we discuss the generality of the proposed model of query by comparing it with Neven's two-way query automata [27] and show that the expressive power of our model is strictly stronger than Neven's one.

Next, we discuss a consistency problem of policies in schema transformation of XML databases. Let t be an instance of a schema S and t' be an instance of a schema S' obtained from t by schema transformation. Also let P and P' be a policy for S and one for S', respectively. The problem determines whether P' protects all the information in t' that P protects in t. The statement "P' protects all the information in t' that P protects in t" means that if an access instance τ for t that is prohibited by P (i.e. τ is not valid for P), then an access instance τ' for t' that is equivalent to τ in some sense is prohibited by P'. We define the consistency problem based on tree transducers [9] and show that it is decidable.

Related Works Several access control models for XML databases have been proposed [5, 10, 17, 25] but static analysis has not been discussed except Murata et al. [25]. Our model has two alternative semantics (AND-semantics and OR-semantics) for a database administrator to choose an appropriate conflict resolution strategy according to the database under consideration. For a traditional database, more sophisticated conflict resolution methods are proposed [19, 16].

The static analysis problem discussed in this chapter can also be considered as a model checking problem for infinite state systems. Model checking methods have been proposed for infinite state systems such as pushdown system (PDS), Petri Net and Process Rewrite Systems [21, 12, 30]. Most of these works are based on automata theory over strings. For example, LTL model checking for PDS can be solved by reducing it to the decision problem on the reachability set

of the given PDS, which is known to be a regular string language. The analysis method proposed in this chapter uses tree automata instead of automata on strings so that more accurate analysis can be performed by taking tree structure information into consideration.

Several formal models for XML query processing have been proposed (e.g., [14, 15, 27]). For example, Hosoya et al. [15] defines a query sublanguage (XDuce) on top of a general-purpose host language. As the whole system becomes Turing complete in such an approach, an abstract system (e.g., type system) is usually defined for static analysis. A two-way query automaton [27] is one of the well-designed automata-theoretic model for XML query. The model in Neven et al. [27] is non-Turing complete as well as our model while we show in this chapter that the expressive power of our model is strictly stronger than the one of their model.

Martens et al. consider a type-checking problem that statically verifies whether the output of a given schema transformation of any documents of a given input type always conforms to a given output type and clarify the computational complexity of the problem when a transformation is defined by a top-down unranked tree transducer [20]. The consistency problem of policies in schema transformation defined in this chapter can be considered as an extended variant of the type-checking problem.

2.2 Preliminaries

2.2.1 Trees

Each XML document can be represented by a tree, whose internal nodes correspond to the elements and the attributes in the XML document and the leaf nodes correspond to the contents of the elements. Such a tree is an *unranked tree*, which is a tree in which the number of children of a node is not bound. In this chapter, we consider only the structure of documents and ignore the nodes corresponding to the actual values contained within the elements and the attributes; that is, we only consider trees in which every node is labeled the name of an element or an attribute.

We assume that we are given a finite alphabet Σ and each node label is chosen

from Σ. A tree in which each node is labeled a symbol in Σ is called a Σ-*tree*. The set of unranked Σ-trees is denoted by T_Σ. Formally, the unranked Σ-trees are defined as strings which represent the tree structure. T_Σ is the smallest set of strings over Σ and the parenthesis symbols '(' and ')' such that for every $\sigma \in \Sigma$ and $w \in T_\Sigma^*$, $\sigma(w)$ is in T_Σ (T_Σ^* is the Kleenean closure of T_Σ). We abbreviate $\sigma()$ to σ. The set of nodes or positions of a tree t is denoted by $\mathrm{Dom}(t)$. The root node of t is denoted by $root(t)$. For every tree t and every $u \in \mathrm{Dom}(t)$, the label of u in t is denoted by $lab^t(u)$.

2.2.2 Tree Automata

A nondeterministic tree automaton (NTA) [9, 24, 26] $M = (Q, \Sigma, \delta, F)$ is a 4-tuple where

- Q is the finite set of states,
- Σ is the alphabet,
- $\delta : Q \times \Sigma \to 2^{Q^*}$ is the transition function such that $\delta(q, a)$ is a regular language over Q, and
- $F \subseteq Q$ is the set of accepting states.

A *run* of M on a Σ-tree t is a labeling $\lambda : \mathrm{Dom}(t) \to Q$ such that for every $v \in \mathrm{Dom}(t)$ and its children $v_1, \ldots, v_n$, $\lambda(v_1) \ldots \lambda(v_n) \in \delta(\lambda(v), lab^t(v))$. A run is accepting if and only if the root is labeled with an accepting state. The set of Σ-trees accepted by M is denoted by $L(M)$ and we say that M recognizes the tree language $L(M)$. A tree language is regular if it is recognized by some NTA. Let $\|M\|$ be the description length of M.

For example, consider a schema of tree-structured documents illustrated in Figure 2.1. The set of trees conforming the schema is recognized by an NTA M_t defined as follows. Note that the value of $\delta(q, a)$ for each $q \in Q_t$ and $a \in \Sigma_t$ is denoted by a regular expression (which allows the operator '+' that means one or more repetition). The empty string is denoted by ϵ.

$M_t = (Q_t, \Sigma_t, \delta_t, F_t)$ where

- $Q_t = \{q_t, q_p, q_s, q_d\}$,

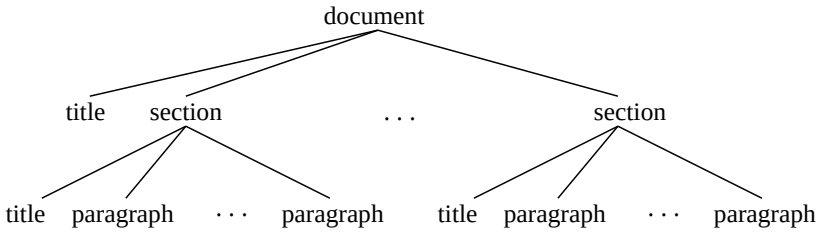

Figure 2.1. A schema of tree-structured documents

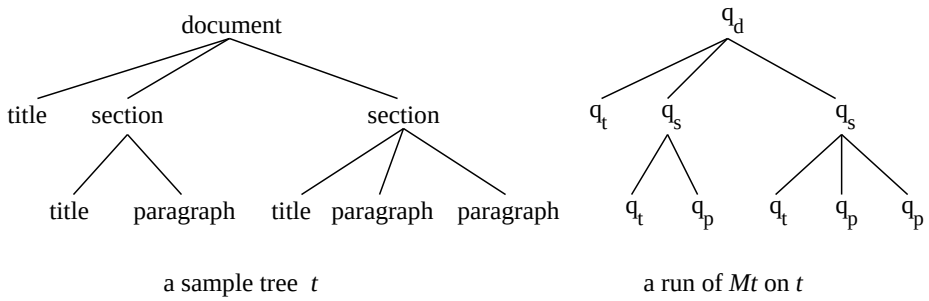

Figure 2.2. A run of M_t

- $\Sigma_t = \{document, title, section, paragraph\}$,
- $\delta_t(q_p, paragraph) = \epsilon$,
  $\delta_t(q_t, title) = \epsilon$,
  $\delta_t(q_s, section) = q_t {q_p}^+$,
  $\delta_t(q_d, document) = q_t {q_s}^+$,
  $\delta_t(q, a) = \emptyset$ for any other pair of $q \in Q_t$ and $a \in \Sigma_t$, and
- $F_t = \{q_d\}$.

By this definition, $L(M_t)$ is the set of trees whose root labeled *document* has a leaf child labeled *title* as well as one or more children labeled *section*, and each child of the root labeled *section* has a leaf child labeled *title* as well as one or more leaf children labeled *paragraph*. Figure 2.2 shows a sample tree t and a run of M_t on t.

It is known that every unranked tree can be converted into a binary tree [26]. Let t^{bin} be the binary tree obtained by this conversion from an unranked tree t. Each node v of an unranked tree t has exactly one corresponding node v_b of

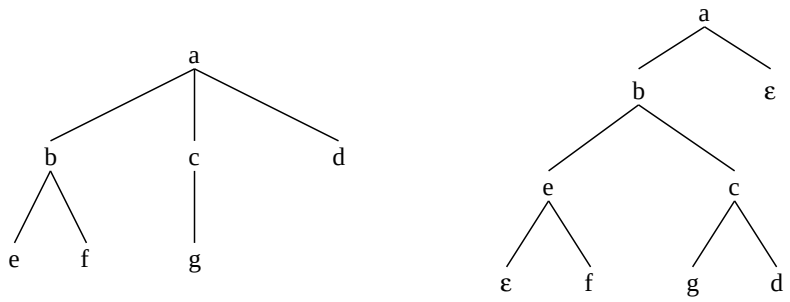

Figure 2.3. An unranked tree and its corresponding binary tree

t^{bin}. The left child and the right child of v_b represent the eldest child of v and the immediately following sibling of v, respectively. If v has no child but has a younger sibling, then the left child of v_b is labeled ϵ. If v has a child but has no younger sibling, then the right child of v_b is labeled ϵ (Figure 2.3).

We can convert an NTA $M = (Q, \Sigma, \delta, F)$ for unranked trees into an NTA M_b for binary trees such that $L(M_b) = \{t^{bin} \mid t \in L(M)\}$ and $\|M_b\|$ is at most $O(\|M\|^2)$, if $\delta(q, a)$ is given as a finite automaton over Q for any $q \in Q$ and $a \in \Sigma$. Hence, for simplicity, we consider only tree automata for binary trees. The transition function of a tree automaton for binary trees is restricted to $\delta : Q \times \Sigma \to 2^{\{\epsilon\} \cup (Q \times Q)}$. Note that we use unranked trees in examples for readability.

Similar to regular string languages, regular tree languages have the following good properties.

Lemma 1 (Sections 1.3 and 1.7 of [9])**.** *For a tree t and a regular tree language L, membership ($t \in L$?) and emptiness ($L = \emptyset$?) are decidable. The class of regular tree languages is closed under boolean operations. Thus, for regular tree languages L_1 and L_2, inclusion ($L_1 \subseteq L_2$?) is also decidable.*

2.3 Access Control Model based on Tree Automata

2.3.1 Charged Alphabet

For a given alphabet Σ, let $\Sigma^{+,-}$ be the alphabet whose elements are the symbols in Σ augmented with the positive and the negative signs ('+' and '−'); that is, $\Sigma^{+,-} = \Sigma \times \{+, -\}$. $\Sigma^{+,-}$ is called the *charged alphabet* of Σ. For readability, we

write the augmented symbol $(a, +)$ as a^+ and $(a, -)$ as a^-. Let $\Sigma^+ = \{a^+ \mid a \in \Sigma\}$ and $\Sigma^- = \{a^- \mid a \in \Sigma\}$ (i.e., $\Sigma^{+,-} = \Sigma^+ \cup \Sigma^-$). Each $\Sigma^{+,-}$-tree is called a *charged tree*. The *uncharged tree* of a charged tree τ is the tree obtained from τ by removing all $+$ and $-$ from the node labels of τ. The uncharged tree of $\tau \in T_{\Sigma^{+,-}}$ is denoted by $uc(\tau)$. An equivalence relation $\approx_{uc}$ over $T_{\Sigma^{+,-}}$ is defined as $\tau_1 \approx_{uc} \tau_2$ if and only if $uc(\tau_1) = uc(\tau_2)$. Note that we defined a Σ-tree as a string consisting of symbols in Σ and the parenthesis symbols in Section 2.2.1. In the rest of this chapter, we sometimes use the string notation of trees.

Example 1. *Let* $\tau_1 = a^+(b^-(c^+d^-)e^+)$ *and* $\tau_2 = a^+(b^+(c^-d^-)e^-)$. *Then* $uc(\tau_1) = uc(\tau_2) = a(b(c\,d)e)$ *and thus* $\tau_1 \approx_{uc} \tau_2$.

2.3.2 Query Automata

A *query tree automaton* (*query TA* for short) $M_R = (Q_R, \Sigma^{+,-}, \delta_R, F_R)$ is an NTA where $\Sigma^{+,-}$ is the charged alphabet of a given alphabet Σ.

Intuitively, a query TA M_R specifies the set of nodes accessed by the query for each XML document. For instance, assume that $\tau \in L(M_R)$ and $t = uc(\tau)$. This means that when we apply the query to the XML document represented by t, the query accesses every node u of t such that $lab^\tau(u) \in \Sigma^+$ and does not access any node v such that $lab^\tau(v) \in \Sigma^-$. For example, $a^+(b^-(c^+d^-)e^+) \in L(M_R)$ means that the query accesses exactly the nodes labeled by a, c, and e when it is applied to the tree $a(b(c\,d)e)$. Note that when we say "a query *accesses* a node," we mean that the query reads (or updates, etc.) the contents of the node, not the label of the node. In this chapter, only this kind of accesses is controlled by policies, and a query always can examine the label of each node regardless of a policy.

We can say that a charged tree τ represents a pair of a Σ-tree t and a subset of nodes of t accessed by a query, and we call τ an *access instance*. If there exist τ_1 and τ_2 in $L(M_R)$ such that $uc(\tau_1) = uc(\tau_2) = t$, then one of the accesses represented by τ_1 and τ_2 is nondeterministically chosen.

Figure 2.4 is a sample query taken from Murata et al. [25], which is written in XQuery. We model this query by a query TA described below. As the same as in Murata et al. [25], we consider only XPath location expressions occurring in the FLWR expression (which consists of a FOR, LET, WHERE, and

```
<TreatmentAnalysis>
{
  for $r in document(‘‘medical_record’’)/record
  where $r/diagnosis/pathology/@type = ‘‘Gastric Cancer’’
  return
    $r/diagnosis/pathology, $r//comment
}
</TreatmentAnalysis>
```

Figure 2.4. A sample query in Murata et al. [25]

RETURN clause) of the query. This query contains the following XPath location expressions. (Note that /record is substituted for variable $r.) We consider that the query accesses the nodes pointed by these location expressions and does not access any other nodes.

- /record
- /record/diagnosis/pathology/@type
- /record/diagnosis/pathology
- /record//comment

Let Σ be the alphabet of the XML database that is the target of the query, i.e., $\{record,\ diagnosis,\ pathology,\ comment,\ @type\} \subseteq \Sigma$. A query TA which models the query should accept any $\tau \in T_{\Sigma^{+,-}}$ such that for each node u of $uc(\tau)$, $lab^{\tau}(u) \in \Sigma^{+}$ if and only if u is pointed by one of the above location expressions. We can define such a query TA M_q. Note that for any $\tau \in T_{\Sigma^{-}}$, there exists a run λ of M_q such that $\lambda(root(\tau)) = q_F$, by the third line of the definition of δ_q. Thus M_q accepts any $\tau \in T_{\Sigma^{-}}$ such that $lab^{uc(\tau)}(root(uc(\tau))) \neq record$ [1] .

$M_q = (Q_q, \Sigma^{+,-}, \delta_q, F_q)$ where

[1] Note that a query TA does not represent the computation steps of an actual query, but represents the set of nodes that the query accesses. That is, we only consider charged trees that are accepted by a query TA and do not consider the computation steps of it.

- $Q_q = \{q_A, q_R, q_F, q_D, q_{R1}, q_P\}$,
- $\delta_q(q_A, record^+) = {q_R}^*$,
  $\delta_q(q_A, x^-) = {q_F}^*$ for $\forall x \in \Sigma - \{record\}$,
  $\delta_q(q_F, y^-) = {q_F}^*$ for $\forall y \in \Sigma$,
  $\delta_q(q_R, diagnosis^+) = {q_D}^*$,
  $\delta_q(q_R, comment^+) = {q_{R1}}^*$,
  $\delta_q(q_R, z^-) = {q_{R1}}^*$ for $\forall z \in \Sigma - \{diagnosis, comment\}$,
  $\delta_q(q_{R1}, comment^+) = {q_{R1}}^*$,
  $\delta_q(q_{R1}, u^-) = {q_{R1}}^*$ for $\forall u \in \Sigma - \{comment\}$,
  $\delta_q(q_D, pathology^+) = {q_P}^*$,
  $\delta_q(q_D, comment^+) = {q_{R1}}^*$,
  $\delta_q(q_D, v^-) = {q_{R1}}^*$ for $\forall v \in \Sigma - \{pathology, comment\}$,
  $\delta_q(q_P, @type^+) = {q_{R1}}^*$,
  $\delta_q(q_P, comment^+) = {q_{R1}}^*$,
  $\delta_q(q_P, w^-) = {q_{R1}}^*$ for $\forall w \in \Sigma - \{@type, comment\}$,
  $\delta_q(q, a) = \emptyset$ for any other pair of $q \in Q_q$ and $a \in \Sigma^{+,-}$, and
- $F_q = \{q_A\}$.

The reader might think that there is a gap between the definition of query automata and XML query in real world: A query is naturally considered as a mapping from an input tree t to a subset of nodes in t selected by the query. In Section 2.5, we will compare our model with an existing model that encodes a query in a more direct way.

2.3.3 Policy Automata

An access control policy determines the set of nodes that a user is allowed to access for a given tree. Murata et al. modeled a policy as a regular set of paths in a tree [25]. However, some policies cannot be represented by their model. For example, consider a policy that prohibits a user from accessing any subtree rooted by a node v labeled c if there exists a node labeled h among the descendants of v. This policy models a policy such that if a section of an article is labeled confidence, then a user is prohibited from accessing any node in the chapter that

includes the confidential section. This policy should include the charged trees in Figure 2.5. In the policy specification language introduced in Murata et al. [25], this policy can be specified by the following three rules.

- $(s, +\mathrm{r}, //*)$
- $(s, -\mathrm{r}, //c[.//h])$
- $(s, -\mathrm{r}, //c[.//h]//*)$

The first rule means that subject s is allowed to access the nodes pointed by the XPath location expression (which points every node in a tree)[2] . The second and third rules denote prohibition. A rule for prohibition for a node overrules any rules for permission for the same node. We show that this policy cannot be represented accurately by the model in Murata et al. [25]. In the model, each location expression in such rules is modeled as a regular expression. For example, the location expressions in the above rules are modeled as $\Sigma^*\Sigma$, Σ^*c, and $\Sigma^*c\Sigma^*\Sigma$, respectively (i.e., the predicate $[.//h]$ is conservatively approximated by 'true'). It means that a user is prohibited from accessing any node pointed by a path denoted by $\Sigma^*c\Sigma^*$ in the approximated policy. Thus any node in the subtree rooted by a node labeled c cannot be accessed, even if the node does not have a descendant labeled h.

To solve this problem, we model a policy as a tree automaton. An *access control policy tree automaton* (*policy TA* for short) $M_{AP} = (Q_A, \Sigma^{+,-}, \delta_A, F_A)$ is an NTA where $\Sigma^{+,-}$ is the charged alphabet of a given alphabet Σ. A policy TA M_{AP} specifies the set of nodes which a user is permitted to access for each XML document. When we apply the policy to a tree t and if there is a charged tree $\tau \in L(M_{AP})$ such that $uc(\tau) = t$, then the policy permits a user to access every node u of t such that $lab^\tau(u) \in \Sigma^+$ and prohibits him or her from accessing any node v of t such that $lab^\tau(v) \in \Sigma^-$.

When we provide a database with access control, the following two cases may exist.

- We would like to prohibit users from accessing some data in a database

[2] The letter 'r' in the second component means 'read'.

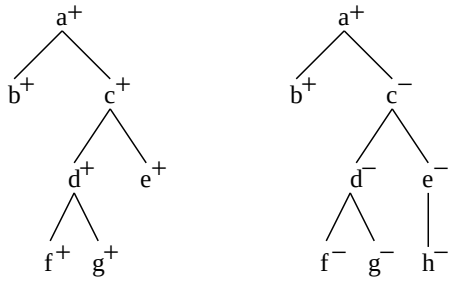

Figure 2.5. A sample policy

where most of its data are open to public. That is, we may specify a policy consisting of prohibition rules.

- We would like to permit users to access some data in a closed database. That is, we may specify a policy consisting of permission rules.

To model these cases, we provide a policy TA with two alternative semantics: AND-semantics and OR-semantics. In AND-semantics, every charged tree τ_1 in a query (i.e., $L(M_R)$) has to be valid for all charged trees τ_2 such that $\tau_1 \approx_{uc} \tau_2$ in the policy (i.e., $L(M_{AP})$). In OR-semantics, every charged tree τ_1 in a query has to be valid for at least one charged tree τ_2 such that $\tau_1 \approx_{uc} \tau_2$ in the policy. AND-semantics models the former case, while OR-semantics models the letter case. Formal definitions of AND-semantics and OR-semantics are provided in Section 2.3.5.

2.3.4 Example

A policy of a newspaper browsing system is given by a policy TA as an example. For readability, we use unranked (not binary) trees and tree automata in this section. A newspaper browsing system which distributes electronic newspapers on the Web may have the following policy.

> The system permits users to read exactly one article which they would like to read among all articles, and prohibits them from reading the other articles at the same time.

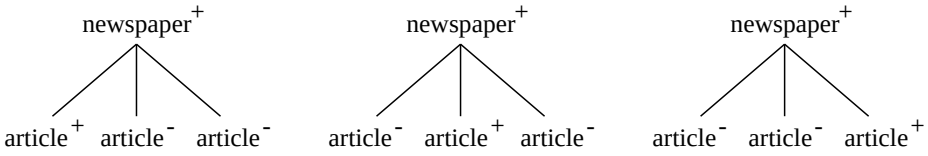

Figure 2.6. A part of the policy of a newspaper browsing system

The policy can be represented by an (infinite) set of charged trees. Figure 2.6 is a part of the policy that consists of charged trees for a newspaper with three articles. We use OR-semantics in this example, i.e., each query should be valid for at least one of these charged trees. Thus, the user is permitted to access exactly one article among all the three articles. The policy TA of this system is as follows.

$M_n = (Q_n, \Sigma_n^{+,-}, \delta_n, F_n)$ where

- $Q_n = \{q_0, q_1, q_2\}$,
- $\Sigma_n = \{newspaper, article\}$,
- $\delta_n(q_1, article^+) = \epsilon$,
  $\delta_n(q_2, article^-) = \epsilon$,
  $\delta_n(q_0, newspaper^+) = {q_2}^* q_1 {q_2}^*$,
  $\delta_n(q, a) = \emptyset$ for any other pair of $q \in Q_n$ and $a \in \Sigma_n^{+,-}$, and
- $F_n = \{q_0\}$.

2.3.5 Validity of Query to Access Control Policy

In Section 2.3.3, we stated two alternative intuitive semantics of a policy TA. In this section, we define them formally. For the rest of this chapter, we fix an alphabet Σ. First we define a partial order $\preceq$ over $T_{\Sigma^{+,-}}$, and then define the two semantics of a policy TA using $\preceq$ in Definition 3.

Definition 1. *For charged trees τ_1 and τ_2 in $T_{\Sigma^{+,-}}$, $\tau_1 \preceq \tau_2$ if and only if the following two properties hold.*

- $uc(\tau_1) = uc(\tau_2)$.

- *For every node u of τ_1, if $lab^{\tau_2}(u) \in \Sigma^+$, then $lab^{\tau_1}(u) \in \Sigma^+$.*

Proposition 1. *The relation $\preceq$ is a partial order over $T_{\Sigma^{+,-}}$.*

Intuitively, $\tau_1 \preceq \tau_2$ means that τ_2 is obtained from τ_1 by changing the sign $+$ of zero or more nodes into $-$. If τ_1 is in the language of a policy TA and τ_2 is in the one of a query TA, then $\tau_1 \preceq \tau_2$ means that all the accessed nodes represented by τ_2 is permitted by τ_1.

Example 2. *If $\tau_1 = a^+(b^+c^+)$ and $\tau_2 = a^+(b^-c^+)$, then $\tau_1 \preceq \tau_2$. However, if $\tau_1 = a^+(b^+c^-)$ and $\tau_2 = a^+(b^-c^+)$, then $\tau_1 \not\preceq \tau_2$, because the sign of the node labeled by c in τ_1 is $-$ while the sign of the corresponding node in τ_2 is $+$. Similarly, $\tau_2 \not\preceq \tau_1$ for the latter example.*

We overload the symbol $\preceq$ for a partial order over $\{+,-\}$ for convenience.

Definition 2. *A partial order $\preceq$ over $\{+,-\}$ is defined as $+ \preceq -$ and $- \not\preceq +$.*

AND-semantics and OR-semantics of a policy TA are defined as follows using $\preceq$.

Definition 3.
1. *M_R is valid for M_{AP} in AND-semantics if and only if $\forall \tau_1 \in L(M_R), \forall \tau_2 \in L(M_{AP}), \tau_1 \approx_{uc} \tau_2 \Rightarrow \tau_2 \preceq \tau_1$.*
2. *M_R is valid for M_{AP} in OR-semantics if and only if $\forall \tau_1 \in L(M_R), \exists \tau_2 \in L(M_{AP})$, $\tau_2 \preceq \tau_1$.*

Example 3. *Let $\tau_1 = a^+(b^-c^+)$ and $\tau_2 = a^+(b^+c^-)$ as in the latter case of example 2. We consider each combination of three policies ($\emptyset$, $\{\tau_1\}$, $\{\tau_1, \tau_2\}$) and two queries ($\{\tau_1\}$, $\{\tau_1, \tau_2\}$). The validity of M_R to M_{AP} in AND-semantics is summarized in the following table.*

		$L(M_{AP})$		
		$\emptyset$	$\{\tau_1\}$	$\{\tau_1, \tau_2\}$
$L(M_R)$	$\{\tau_1\}$	*valid*	*valid*	*not valid*
	$\{\tau_1, \tau_2\}$	*valid*	*not valid*	*not valid*

In a similar way, the validity of M_R to M_{AP} in OR-semantics is summarized in the following table.

		$L(M_{AP})$		
		$\emptyset$	$\{\tau_1\}$	$\{\tau_1, \tau_2\}$
$L(M_R)$	$\{\tau_1\}$	*not valid*	*valid*	*valid*
	$\{\tau_1, \tau_2\}$	*not valid*	*not valid*	*valid*

For convenience, we define the following relations and operations over the subsets of $T_{\Sigma^{+,-}}$. As stated in Lemma 2, $\sqsubseteq$ and $\sqsubseteq_{\mathrm{A}}$ characterize OR-semantics and AND-semantics, respectively. $L\uparrow$ is intuitively the set which contains all upper-bounds (with respect to $\preceq$) of each $\tau' \in L$, while $L\uparrow^{\mathrm{A}}$ is the set which contains all upper-bounds of each equivalent class defined by $\approx_{uc}$ in L. $L^{\downarrow}$ is the set of the minimal elements (with respect to $\preceq$) in L. The minimum (resp. maximum) of the cardinality of each equivalent class defined by $\approx_{uc}$ in L is denoted by $\min(L)$ (resp. $\max(L)$).

Definition 4. *Let L, L_1, and L_2 be subsets of $T_{\Sigma^{+,-}}$. We define the following relations and operations.*

- $L_1 \sqsubseteq L_2 \quad \Leftrightarrow \forall \tau_2 \in L_2, \exists \tau_1 \in L_1,\ \tau_1 \preceq \tau_2$.
- $L_1 \sqsubseteq_{\mathrm{A}} L_2 \Leftrightarrow \forall \tau_2 \in L_2, \forall \tau_1 \in L_1, \tau_1 \approx_{uc} \tau_2 \Rightarrow \tau_1 \preceq \tau_2$.
- $L\uparrow \quad = \{\tau \mid \tau' \preceq \tau \text{ for } \exists \tau' \in L\}$.
- $L\uparrow^{\mathrm{A}} = \{\tau \mid \tau' \preceq \tau \text{ for } \forall \tau' \in L \text{ such that } \tau \approx_{uc} \tau'\}$.
- $L^{\downarrow} = \{\tau \in L \mid \text{there is no } \tau' \in L \text{ such that } \tau' \precneqq \tau\}$.
- $\min(L) = \min_{t \in T_\Sigma} |\{\tau \in L \mid uc(\tau) = t\}|$.
- $\max(L) = \max_{t \in T_\Sigma} |\{\tau \in L \mid uc(\tau) = t\}|$.

By the definition, the following lemma holds obviously. This lemma characterizes the two semantics by $\sqsubseteq$ and $\sqsubseteq_A$.

Lemma 2. *M_R is valid for M_{AP} in OR-semantics if and only if $L(M_{AP}) \sqsubseteq L(M_R)$. M_R is valid for M_{AP} in AND-semantics if and only if $L(M_{AP}) \sqsubseteq_{\mathrm{A}} L(M_R)$.*

In the following, we discuss the relationship between the two semantics. As stated in Theorem 1, Theorem 2, and Corollary 1, the expressive power of OR-semantics is properly stronger than AND-semantics. Theorem 2 shows a necessary and sufficient condition for a given policy TA in OR-semantics to have an equivalent policy TA in AND-semantics. However, the problem to determine whether this condition holds is EXPTIME-complete, as shown in Theorem 3. Moreover, Corollary 2 and Theorem 7 in Section 2.4 imply that constructing a policy TA in AND-semantics that is equivalent to a given policy TA M_{AP} in OR-semantics requires exponential time to $\|M_{AP}\|$ in general.

Lemma 3. *$L{\uparrow^{\mathrm{A}}} \sqsubseteq L'$ if and only if $L \sqsubseteq_{\mathrm{A}} L'$.*

Proof. It is sufficient to show that the following two statements are equivalent.

1. $\forall \tau_2 \in L'$, $\exists \tau_1 \in L{\uparrow^{\mathrm{A}}}$, $\tau_1 \preceq \tau_2$.
2. $\forall \tau_2 \in L'$, $\forall \tau_3 \in L$, $\tau_3 \approx_{uc} \tau_2 \Rightarrow \tau_3 \preceq \tau_2$.

($1 \Rightarrow 2$) Assume that statement 1 holds and let $\tau_2 \in L'$. By statement 1, $\exists \tau_1 \in L{\uparrow^{\mathrm{A}}}$, $\tau_1 \preceq \tau_2$ (and thus $\tau_1 \approx_{uc} \tau_2$). By the definition of $L{\uparrow^{\mathrm{A}}}$, $\tau_3 \approx_{uc} \tau_1 \Rightarrow \tau_3 \preceq \tau_1$ for $\forall \tau_3 \in L$. Assume that $\tau_3 \in L$ and $\tau_3 \approx_{uc} \tau_2$. Then, $\tau_3 \approx_{uc} \tau_1$ and $\tau_3 \preceq \tau_1 \preceq \tau_2$. ($2 \Rightarrow 1$) Assume that statement 2 holds. By the definition of $L{\uparrow^{\mathrm{A}}}$, $L' \subseteqq L{\uparrow^{\mathrm{A}}}$. Thus, statement 1 holds by letting $\tau_1 = \tau_2$. □

Lemma 4. *If L is regular, then $L{\uparrow^{\mathrm{A}}}$ is also regular.*

Proof. Let $M = (Q, \Sigma^{+,-}, \delta, F)$ be an NTA such that $L = L(M)$. We construct an NTA $M_A = (Q_A, \Sigma^{+,-}, \delta_A, F_A)$ such that $L(M_A) = \overline{L{\uparrow^{\mathrm{A}}}}$ as follows.

- $Q_A = Q \times \{0, 1\}$,
- $F_A = F \times \{1\}$,
- δ_A is the function such that for each $q_A \in Q_A$ and $a_A \in \Sigma^{+,-}$, $\delta_A(q_A, a_A)$ is the smallest set satisfying the followings for any $q, q_1, q_2 \in Q$, $a \in \Sigma$, $s_1, s_2 \in \{+, -\}$, and $d_1, d_2 \in \{0, 1\}$.
  - $\delta_A((q, 0), a^{s_1}) \ni \epsilon$ if $\delta(q, a^{s_2}) \ni \epsilon$ and $s_2 \preceq s_1$.
  - $\delta_A((q, 1), a^{+}) \ni \epsilon$ if $\delta(q, a^{-}) \ni \epsilon$.

– $\delta_A((q, max(d_1, d_2)), a^{s_1}) \ni (q_1, d_1)(q_2, d_2)$ if $\delta(q, a^{s_2}) \ni q_1 q_2$ and $s_2 \preceq s_1$.

– $\delta_A((q, 1), a^+) \ni (q_1, d_1)(q_2, d_2)$ if $\delta(q, a^-) \ni q_1 q_2$.

Note that $max(d_1, d_2)$ denotes the larger value of $d_1, d_2 \in \{0, 1\}$.

Intuitively, M_A behaves as follows. For any $\tau_1 \in T_{\Sigma^{+,-}}$, $\tau_1 \in \overline{L\uparrow^{\mathrm{A}}}$ if and only if there exists a tree $\tau_2 \in L$ such that $\tau_2 \approx_{uc} \tau_1$ and $\tau_2 \not\preceq \tau_1$; that is, there exists a node v of τ_1 such that $lab^{\tau_1}(v) \in \Sigma^+$ and $lab^{\tau_2}(v) \in \Sigma^-$. To accept such τ_1, M_A simulates a run of M on a tree τ_2, by ignoring the positive and negative signs of the node labels, and by the second component of each state, M_A indicates for each node v of τ_2 that $lab^{\tau_1}(v) \in \Sigma^+$ and $lab^{\tau_2}(v) \in \Sigma^-$ or a descendant of v fulfills this property. When this property holds on v, there exists a run λ of M_A such that the second component of $\lambda(v)$ is 1. Thus, M_A exactly accepts $\tau_2 \in \overline{L\uparrow^{\mathrm{A}}}$. By Lemma 1, $L\uparrow^{\mathrm{A}}$ is regular, since $\overline{L\uparrow^{\mathrm{A}}}$ is regular. □

Theorem 1. *For any policy TA M_{AP} in AND-semantics, we can convert it into an equivalent policy TA M'_{AP} in OR-semantics.*

Proof. By Lemma 3, $L(M_{AP}) \sqsubseteq_{\mathrm{A}} L(M_R)$ if and only if $L(M_{AP})\uparrow^{\mathrm{A}} \sqsubseteq L(M_R)$ for any query TA M_R. By Lemma 4, we can obtain a policy TA M'_{AP} such that $L(M'_{AP}) = L(M_{AP})\uparrow^{\mathrm{A}}$. □

Lemma 5. *If L is regular, then $L^{\downarrow}$ is also regular.*

Proof. Let $L\uparrow^P = \{\tau \in T_{\Sigma^{+,-}} \mid \tau' \precneqq \tau \text{ for } \exists \tau' \in L\}$. We can construct an NTA that recognizes $L\uparrow^P$ in a similar way to the NTA that recognizes $\overline{L\uparrow^{\mathrm{A}}}$ in the proof of Lemma 4. Since $L^{\downarrow} = L - L\uparrow^P$, $L^{\downarrow}$ is also regular if L is regular. □

Theorem 2. *For any policy TA M_{AP} in OR-semantics, $\max(L(M_{AP})^{\downarrow}) \leq 1$ if and only if there is a policy TA M'_{AP} in AND-semantics that is equivalent to M_{AP}.*

Proof. ($\Rightarrow$) Let $L(M'_{AP}) = L(M_{AP})^{\downarrow} \cup (T_{\Sigma^-} - h_-(L(M_{AP})))$, where h_- is a homomorphism such that $h_-(a^+) = h_-(a^-) = a^-$ for every $a \in \Sigma$.

($\Leftarrow$) Assume that $\max(L(M_{AP})^{\downarrow}) > 1$. Then there are $\tau_1, \tau_2 \in L(M_{AP})$ such that $\tau_1 \approx_{uc} \tau_2$, $\tau_1 \neq \tau_2$, and there is no $\tau \in L(M_{AP})$ that satisfies $\tau \precneqq \tau_1$ or $\tau \precneqq \tau_2$. Let $L(M_{R_1}) = \{\tau_1, \tau_2\}$ and $L(M_{R_2}) = \{\tau_1 \sqcap \tau_2\}$, where $\tau_1 \sqcap \tau_2$ is the

greatest lower bound[3] (with respect to $\preceq$) of $\{\tau_1, \tau_2\}$. Then $L(M_{AP}) \sqsubseteq L(M_{R_1})$ and $L(M_{AP}) \not\sqsubseteq L(M_{R_2})$, since there is no $\tau \in L(M_{AP})$ such that $\tau \preceq \tau_1 \sqcap \tau_2 \not\preceq \tau_1$. However, for any $L' \subseteq T_{\Sigma^{+,-}}$, $L' \sqsubseteq_A L(M_{R_1})$ implies $L' \sqsubseteq_A L(M_{R_2})$. Therefore there is no policy in AND-semantics that is equivalent to M_{AP}. □

Corollary 1. *There is a policy TA in OR-semantics such that there is no equivalent policy TA in AND-semantics.*

Proof. The policy TA M_n in Section 2.3.4 is an example. Let $\tau_1 = n^+(a^+a^-)$ and $\tau_2 = n^+(a^-a^+)$ in the proof of Theorem 2 (*newspaper* and *article* are abbreviated to n and a, respectively). □

Lemma 6. *The problem to determine whether a given NTA M_{AP} over $\Sigma^{+,-}$ satisfies* $\max(L(M_{AP})) \leq 1$ *is solvable in polynomial time.*

Proof. We construct an NTA M over Σ that recognizes the following language.

$$L(M) = \{\, t \in T_\Sigma \mid \text{There exist some } \tau_1, \tau_2 \in L(M_{AP}) \text{ such that } t = uc(\tau_1) = uc(\tau_2) \text{ and } \tau_1 \neq \tau_2 \,\}$$

$L(M) = \emptyset$ if and only if $\max(L(M_{AP})) \leq 1$.

M with $\|M\| = O(\|M_{AP}\|^2)$ can be constructed as follows. Let $M_{AP} = (Q, \Sigma^{+,-}, \delta, F)$. M is defined as $M = (Q \cup (Q \times Q), \Sigma, \delta', F \times F)$, where δ' is the function such that for each $q \in Q \cup (Q \times Q)$ and $a \in \Sigma$, $\delta'(q, a)$ is the smallest set satisfying the followings for $q_1, q_2, \ldots \in Q$ and $s, s' \in \{+, -\}$.

- $q_1q_2 \in \delta'(q_3, a)$ if $q_1q_2 \in \delta(q_3, a^s)$.
- $q_1q_2 \in \delta'(\langle q_3, q_4 \rangle, a)$ if $q_1q_2 \in \delta(q_3, a^+)$ and $q_1q_2 \in \delta(q_4, a^-)$.
- $\langle q_1, q_2 \rangle\, q_3 \in \delta'(\langle q_4, q_5 \rangle, a)$ if $q_1q_3 \in \delta(q_4, a^s)$ and $q_2q_3 \in \delta(q_5, a^{s'})$.
- $q_1\, \langle q_2, q_3 \rangle \in \delta'(\langle q_4, q_5 \rangle, a)$ if $q_1q_2 \in \delta(q_4, a^s)$ and $q_1q_3 \in \delta(q_5, a^{s'})$.

Since the emptiness problem is solvable in polynomial time, this lemma holds. □

Theorem 3. *The problem to determine whether a given policy TA M_{AP} in OR-semantics has an equivalent policy TA in AND-semantics is EXPTIME-complete.*

[3] $\tau_1 \sqcap \tau_2$ is a charged tree such that $uc(\tau_1 \sqcap \tau_2) = uc(\tau_1) = uc(\tau_2)$ and for every node u in $\tau_1 \sqcap \tau_2$, $lab^{\tau_1 \sqcap \tau_2}(u) \in \Sigma^-$ if and only if $lab^{\tau_1}(u) \in \Sigma^-$ and $lab^{\tau_2}(u) \in \Sigma^-$

Proof. We can construct an NTA that recognizes $L(M_{AP})\uparrow^P$ (defined in the proof of Lemma 5) in polynomial time to $\|M_{AP}\|$. Thus, an NTA $M^{\downarrow}$ that recognizes $L(M_{AP})^{\downarrow} = L(M_{AP}) - L(M_{AP})\uparrow^P$ can be constructed in exponential time to $\|M_{AP}\|$. By Lemma 6, the problem to determine whether $\max(L(M_{AP})^{\downarrow}) \leq 1$ is solvable in polynomial time to $\|M^{\downarrow}\|$, that is exponential to $\|M_{AP}\|$.

This problem is EXPTIME-hard because the following problem known as EXPTIME-complete (Theorem 14 in [9]) can be transformed to the problem.

> Regular Tree Language Non-Universality (RTLNU)
> **Input:** An NTA M over a finite alphabet Σ.
> **Output:** $L(M) \neq T_{\Sigma}$?

Let M be an instance of RTLNU and Σ be the alphabet of M. We choose an element a in Σ and fix it. For M and a, we construct an NTA M_{AP} over $\Sigma^{+,-}$ that recognizes the following tree language.

$$\begin{aligned} L(M_{AP}) = &\{\, a^+(a^+\,\tau) \mid \tau \in T_{\Sigma^+} \text{ and } uc(\tau) \in L(M) \,\} \\ &\cup \{\, a^-(a^+\,\tau) \mid \tau \in T_{\Sigma^+} \,\} \\ &\cup \{\, a^+(a^-\,\tau) \mid \tau \in T_{\Sigma^+} \,\} \end{aligned}$$

M_{AP} can be constructed in polynomial time to $\|M\|$, and $\max(L(M_{AP})^{\downarrow}) \leq 1$ if and only if $L(M) = T_{\Sigma}$. Thus this theorem holds. □

2.4 Static Analysis

2.4.1 Problem Statement

The static analysis problem in AND-semantics (resp. OR-semantics) for M_R and M_{AP} is defined as follows.

Input: A query TA M_R and a policy TA M_{AP} over the same charged alphabet $\Sigma^{+,-}$.

Output: "YES" if M_R is valid for M_{AP} in AND-semantics (resp. OR-semantics) and "NO" otherwise.

By Theorem 1, it is sufficient to give an algorithm for the problem in OR-semantics. We propose such an algorithm and discuss the time complexity of it.

2.4.2 Decision Algorithm

We show that the static analysis problem in OR-semantics is decidable (Theorem 4), using the following Lemmas 7 and 8. By these lemmas, we can reduce the static analysis problem in OR-semantics to the set-inclusion problem of regular tree languages.

Lemma 7. $L_1 \sqsubseteq L_2 \;\Leftrightarrow\; L_2 \subseteqq L_1\!\uparrow$.

Proof. $L_2 \subseteqq L_1\!\uparrow$ implies $\tau_2 \in L_1\!\uparrow$ for $\forall \tau_2 \in L_2$. By the definition of $L_1\!\uparrow$, $\tau_1 \preceq \tau_2$ for some $\tau_1 \in L_1$, and thus $L_1 \sqsubseteq L_2$. The converse can be shown by the reverse way. □

Lemma 8. *If L is regular, then $L\!\uparrow$ is also regular.*

Proof. Let $M = (Q, \Sigma^{+,-}, \delta, F)$ be an NTA such that $L = L(M)$. We define $M^{\uparrow} = (Q, \Sigma^{+,-}, \delta^{\uparrow}, F)$ as follows. For each $q \in Q$ and $a \in \Sigma$,

- $\delta^{\uparrow}(q, a^-) = \delta(q, a^+) \cup \delta(q, a^-)$, and
- $\delta^{\uparrow}(q, a^+) = \delta(q, a^+)$.

We can easily show that $L(M^{\uparrow}) = L\!\uparrow$. □

Theorem 4. *The static analysis problem for M_R and M_{AP} in OR-semantics is decidable.*

Proof. $L(M_{AP})\!\uparrow$ is regular by Lemma 8 and thus $L(M_R) \subseteqq L(M_{AP})\!\uparrow$ is decidable by Lemma 1. Therefore, this theorem holds by Lemma 2 and Lemma 7. □

From the proof of Theorem 4, we obtain the following algorithm for solving the static analysis problem in OR-semantics.

Algorithm 1 Perform the following two steps in this order.

1. Construct an NTA M_b such that $L(M_b) = L(M_R) \cap \overline{L(M_{AP})\!\uparrow}$.
2. Decide whether $L(M_b) = \emptyset$ or not.

Next, we consider the time complexity of the problem. By the following two lemmas, Algorithm 1 can be performed in $O(\|M_R\| \cdot \|N\|)$ time where N is an NTA such that $L(N) = \overline{L(M_{AP})\!\uparrow}$.

Lemma 9 (Section 1.3 of [9]). *For NTAs M_1 and M_2, we can construct an NTA M such that $L(M) = L(M_1) \cap L(M_2)$ and $\|M\| = O(\|M_1\| \cdot \|M_2\|)$.*

Lemma 10 (Theorem 11 in [9]). *For an NTA M, emptiness of $L(M)$ is decidable in $O(\|M\|)$ time.*

On the other hand, Step 1 of Algorithm 1 would require the construction of an NTA N such that $L(N) = \overline{L(M_{AP})\uparrow}$; however, its size would be exponential to $\|M_{AP}\|$ in general[4] . In fact, the problem is EXPTIME-complete in general as shown below.

Theorem 5. *The static analysis problem in OR-semantics is EXPTIME-complete.*

Proof. We can construct an NTA N such that $L(N) = \overline{L(M_{AP})\uparrow}$ and $\|N\| = O(c^{\|M_{AP}\|})$ for some constant $c > 1$. Thus the problem is in EXPTIME by Lemma 9 and Lemma 10. EXPTIME-hardness can be shown by transforming RTLNU problem (in the proof of Theorem 3) to the static analysis problem. From a given instance M of RTLNU, we construct M_{AP} as the same as M except that its alphabet is Σ^+ and it uses $a^+ \in \Sigma^+$ instead of each $a \in \Sigma$. We let M_R be an NTA such that $L(M_R) = T_{\Sigma^+}$. Obviously, M_R is valid for M_{AP} in OR-semantics if and only if $L(M_{AP}) = T_{\Sigma^+}$, i.e., $L(M) = T_\Sigma$. □

Theorem 5 can be strengthened as follows.

Theorem 6. *The static analysis problem in OR-semantics is EXPTIME-complete even if the policy TA M_{AP} is required to satisfy either*

1. $\max(L(M_{AP})) \leq 1$, *or*
2. $1 \leq \min(L(M_{AP}))$.

Proof. 1. The theorem in this case can be proved as the same as Theorem 5, since M_{AP} in the proof of Theorem 5 satisfies $\max(L(M_{AP})) \leq 1$.

[4] If $M_{AP}^{\uparrow}$ constructed from M_{AP} in the proof of Lemma 8 is bottom-up deterministic (Section 1.1 of [9]), then the size of N is the same order of $\|M_{AP}\|$. However, $M_{AP}^{\uparrow}$ is not bottom-up deterministic in general even if M_{AP} is so.

2. After constructing M_R and M_{AP} according to the proof of Theorem 5, we construct an NTA M'_{AP} such that $L(M'_{AP}) = L(M_{AP}) \cup T_{\Sigma^-}$ and $\|M'_{AP}\| = O(\|M_{AP}\|)$. M'_{AP} is equivalent to M_{AP} in OR-semantics and satisfies $1 \leq \min(L(M'_{AP}))$.

□

Corollary 2. *The static analysis problem in OR-semantics is EXPTIME-complete even if the policy TA M_{AP} is required to have an equivalent policy TA in AND-semantics.*

In contrast to OR-semantics, the static analysis problem in AND-semantics can be solved in polynomial time, as stated in Theorem 7.

Lemma 11. $(L\uparrow^A)\uparrow = L\uparrow^A$.

Proof. It is obvious by the definition of $L\uparrow^A$. □

Theorem 7. *The time complexity of the static analysis problem for M_R and M_{AP} in AND-semantics is $O(\|M_R\| \cdot \|M_{AP}\|)$.*

Proof. By Theorem 1, this problem is equivalent to deciding whether $L(M_{AP})\uparrow^{\mathrm{A}} \sqsubseteq L(M_R)$, which is equivalent to $L(M_R) \subseteqq (L(M_{AP})\uparrow^{\mathrm{A}})\uparrow = L(M_{AP})\uparrow^{\mathrm{A}}$ by Lemma 7 and Lemma 11. Thus, in Step 1 of Algorithm 1, it is sufficient to construct an NTA M_b such that $L(M_b) = L(M_R) \cap \overline{L(M_{AP})\uparrow^{\mathrm{A}}}$. As is the way of constructing $L\uparrow^{\mathrm{A}}$ in Lemma 4, we can directly construct an NTA M_A from M_{AP} (without determinization and explicit complementation) such that $L(M_A) = \overline{L(M_{AP})\uparrow^{\mathrm{A}}}$ and $\|M_A\| = O(\|M_{AP}\|)$. Therefore, this theorem holds by Lemma 9 and Lemma 10. □

As stated in Theorem 8, the static analysis problem in OR-semantics under a particular condition is solvable in polynomial time by reducing the problem to the one in AND-semantics. However, the problem to determine whether this condition holds is EXPTIME-complete, as shown in Theorem 9.

Theorem 8. *The static analysis problem in OR-semantics is solvable in polynomial time if the policy TA M_{AP} is required to satisfy* $\min(L(M_{AP})) = \max(L(M_{AP})) = 1$.

Proof. When $\min(L(M_{AP})) = \max(L(M_{AP})) = 1$, $L(M_{AP}) \sqsubseteq L$ if and only if $L(M_{AP}) \sqsubseteq_{\mathrm{A}} L$ for any $L \subsetneqq T_{\Sigma^{+,-}}$. Therefore the answer to the problem in OR-semantics is the same as the one in AND-semantics and is solvable in polynomial time by Theorem 7. □

Theorem 9. *The problem to determine whether an NTA M_{AP} over $\Sigma^{+,-}$ satisfies $\min(L(M_{AP})) = \max(L(M_{AP})) = 1$ is EXPTIME-complete.*

Proof. According to the proof of Theorem 5, construct M_{AP} from an instance M of RTLNU problem. Then $L(M) = T_\Sigma$ if and only if $\min(L(M_{AP})) = \max(L(M_{AP})) = 1$. □

2.5 Discussion on query model

Formal models for XML query processing can be divided roughly into two approaches. The first approach defines a query sublanguage on top of a general-purpose host language. The whole system becomes Turing complete and an abstract system (e.g., type system) is usually defined for static analysis (e.g. [15]). The other approach provides a non-Turing complete model based on tree automata theory, mainly by neglecting data manipulation. Among the latter approach, we take Neven's two-way query automaton [27], which is one of the well-designed automata-theoretic models for XML query: Decidability and complexity of fundamental problems such as emptiness, containment as well as their expressive power compared to formal logic have been clarified for the model. Here, we compare our model with Neven's model and show that the expressive power of our model is strictly stronger than Neven's one. Note that in this section we only consider mechanisms for selecting a subset of nodes (e.g., by labeling '+') in each tree, and do not consider the difference between AND-semantics and OR-semantics.

The vocabulary of monadic second-order logic (MSO) over an unranked alphabet Σ consists of first-order variables $x, y, z, \ldots$ (possibly with subscripts) representing a node, second-order variables $X, Y, Z, \ldots$ representing a set of nodes, constant *root*, successor function representing a child of a node (for a node u, ui represents the i-th child of u), O_σ (the set of nodes of which labels are σ), logical

connectives $\vee, \wedge, \neg, \Rightarrow$, and quantifiers $\exists, \forall$. For details of monadic second-order logic, see Section 3.3 of [9] and [34] for example.

Let $MSO(X_1, \ldots, X_n)$ denote the set of MSO formulas $\psi(X_1, \ldots, X_n)$ with at most n distinct second-order free variables $X_1, \ldots, X_n$. An $MSO(X_1, \ldots, X_n)$ formula $\psi = \psi(X_1, \ldots, X_n)$ is interpreted in a tree t with n designated subsets $U_1, \ldots, U_n$ of nodes; the satisfaction relation $(t, U_1, \ldots, U_n) \models \psi(X_1, \ldots, X_n)$ holds if ψ evaluates to true in tree t when substituting U_i into X_i $(1 \leq i \leq n)$. The structure $(t, U_1, \ldots, U_n)$ can alternatively be represented by a tree t' over the extended alphabet $\Sigma \times \{+, -\}^n$ such that $lab^{t'}(u) = (\sigma, c_1, \ldots, c_n)$ indicates that $lab^t(u) = \sigma$ as well as $u \in U_i$ if $c_i = +$, and $u \notin U_i$ if $c_i = -$. For example, $(f(a, b), \{\varepsilon\}, \{1, 2\})$ is represented by $f^{+-}(a^{-+}, b^{-+})$. Note that $\Sigma \times \{+, -\} = \Sigma^{+,-}$ in our notation.

For an $MSO(X_1, \ldots, X_n)$ formula ψ, let $L(\psi) = \{t' \mid t' \models \psi\} \subseteqq T_{\Sigma \times \{+,-\}^n}$, which is called the tree language defined by ψ. A tree language $L \subseteqq T_{\Sigma \times \{+,-\}^n}$ is *definable* in $MSO(X_1, \ldots, X_n)$ if there exists an $MSO(X_1, \ldots, X_n)$ formula ψ such that $L = L(\psi)$. The following is a well-known equivalence property between tree automata and MSO.

Theorem 10 (Theorem 3.6 in [34])**.** *A tree language over $\Sigma \times \{+, -\}^n$ is regular if and only if it is definable in $MSO(X_1, \ldots, X_n)$ over Σ.*

As an immediate corollary, we have:

Corollary 3. *A tree language over $\Sigma^{+,-}$ is regular if and only if it is definable in $MSO(X)$ over Σ.*

In Neven et al. [27], a two-way deterministic query automaton (2DQA) is defined as a formal model for XML query. Although both Neven et al. and this chapter incidentally use the same term 'query automata,' a query automaton in this chapter is merely a tree automaton while Neven et al.'s query automaton is a tree transducer which emits a subset of nodes of an input tree rather than a simple acceptor.

A 2DQA is a tuple $A = (Q, \Sigma, F, s, \delta, \lambda)$ where Q is the set of states, Σ is the input alphabet, $F \subseteqq Q$ is the set of accepting states, $s \in Q$ is the initial state, δ is the transition function, and λ is the selection function from $Q \times \Sigma$ to $\{+, -\}$. 2DQA A starts at the root of an input tree t with the initial state

s and deterministically traverses t either downward or upward according to δ. More precisely, δ consists of down transitions $\delta_\downarrow : Q \times \Sigma \to Q^*$, up transitions $\delta_\uparrow : (Q \times \Sigma)^* \to Q$, and stay transitions $\delta_- : (Q \times \Sigma)^* \to Q^*$. Note that when A moves downward from a node u, states are assigned to all children of u according to $\delta_\downarrow$ (i.e., the control forks to every child of the current node). Likewise, an upward transition to a node v is possible only when states are assigned to all children of v. A node may be visited more than once with possibly different states. We say that A selects a node u if during the traverse u is visited with a state q such that $lab^t(u) = \sigma$ and $\lambda(q, \sigma) = +$. Let $L(A) = \{(t, U) \mid U \text{ is the set of nodes of } t \text{ selected by } A \text{ for input } t\}$, which is called the query (or tree language) computed by A.

In Neven et al. [27] the expressive power of 2DQA is compared with the following subclass of MSO formulas: Let $MSO(x)$ denote the set of MSO formulas $\psi(x)$ with at most one first-order free variable x. For a tree t and a node u in t, we write $(t, u) \models \psi(x)$ if ψ evaluates to true in tree t when substituting u for x. Also, let $L(\psi) = \{(t, U) \mid U = \{u \mid (t, u) \models \psi(x)\}\}$, which is called the language defined by ψ. A tree language $L \subseteqq T_{\Sigma^{+,-}}$ is *definable* in $MSO(x)$ if there exists an $MSO(x)$ formula $\psi(x)$ such that $L = L(\psi)$.

Theorem 11 (Theorem 5.18 in [27])**.** *A tree language over $\Sigma^{+,-}$ is computed by a 2DQA if and only if it is definable in $MSO(x)$ over Σ.*

It is also known that a first-order variable can be simulated by a second-order variable [34]. For the converse direction, observe that for every 2DQA A over Σ and every $t \in T_\Sigma$, $L(A)$ contains exactly one (t, U) by definition. This property does not always hold for a language defined in $MSO(X)$. For example, $\{(a, \{\varepsilon\}), (a, \emptyset)\}$ for a constant $a \in \Sigma$, which is denoted as $\{a^+, a^-\}$ in our notation, is definable in $MSO(X)$ but is not definable in $MSO(x)$. Summarizing, the following strict inclusion holds.

Theorem 12. *The class of tree languages over $\Sigma^{+,-}$ computed by 2DQA is properly included in the class of regular tree languages over $\Sigma^{+,-}$.*

This theorem implies that our characterization of XML queries by regular tree languages over a charged alphabet ($\Sigma^{+,-}$) is strong enough to subsume Neven's natural formalization of queries based on two-way tree automata with outputs.

2.6 Consistency Problem of Policies in Schema Transformation

In this section, we discuss a consistency problem of policies in schema transformation of XML databases. Let Φ be a transducer that nondeterministically translates any instance t of a schema S into an instance t' of another schema S', and let P and P' be a policy for schema S and one for S', respectively. Intuitively, this problem determines whether P' protects all the information in t' that P protects in t. The statement "P' protects all the information in t' that P protects in t" means that if an access instance τ for t (i.e. $uc(\tau) = t$) is prohibited by P (i.e. τ is not valid for P), then an access instance τ' for t' which is equivalent to τ in some sense is prohibited by P' as well. Assuming that Φ defines a correspondence between nodes in t and nodes in t', τ' is derived from τ according to the correspondence; that is, every node u of t' that corresponds to a node of t labeled + (resp. $-$) in τ is also labeled + (resp. $-$) in τ'. Let $\Phi^{+,-}$ be a transducer that translates τ into such τ'. We consider that Φ is nondeterministic, and thus $\Phi^{+,-}$ is also nondeterministic. We model the nondeterministic function $\Phi^{+,-}$ as a mapping that maps an access instance τ into the set of all the access instances nondeterministically obtained by applying $\Phi^{+,-}$ to τ. Note that, this nondeterminism is necessary since we replace conditionals depending on data values with nondeterminism when we introduce a formal model Φ for real world XML transducers such as XSLT. This situation is similar to the fact that we need nondeterminism when we model XML queries by tree automata in Section 2.3.2. When τ is not valid for P, any elements of $\Phi^{+,-}(\tau)$ should not be valid for P', because any nondeterministically chosen access instance in $\Phi^{+,-}(\tau)$ should be prohibited by P'. Therefore, this problem is defined as the problem that determines whether any elements of $\Phi^{+,-}(\tau)$ are not valid for policy P', for every access instance τ that is not valid for policy P.

Since XML documents are represented by tree structures, we model a transducer by a tree transducer. A tree transducer is an NTA extended by a function for translating an input tree $s \in T_\Sigma$ into another tree $s' \in T_\Sigma$. In this thesis, since XML documents are represented by binary trees, we use transducers mapping a binary tree to binary trees.

Before we define tree transducers, some words are defined. A *term* is a tree that consists of symbols in Σ and variables. A term is *linear* if each variable occurs at most once in it. A linear term is also called a *context*. The set of contexts over Σ containing n variables $x_1, \ldots, x_n$ is denoted by $\mathcal{C}^n(\Sigma)$. $\mathcal{C}^1(\Sigma)$ is also denoted by $\mathcal{C}(\Sigma)$. For a context $C \in \mathcal{C}^n(\Sigma)$, the expression $C[t_1, \ldots, t_n]$ means a term obtained by substituting t_i for x_i in C for $1 \leq i \leq n$.

Definition 5 (Section 6.4.2 of [9])**.** *A (nondeterministic top-down) tree transducer over an alphabet Σ is a 4-tuple $\Phi = (Q, \Sigma, Q_i, \Delta)$ where*

- *Q is the finite set of states,*
- *Σ is the alphabet. Note that the rank of each input symbol in Σ is either 0 or 2 and each input symbol except ϵ is overloaded for rank 0 and 2,*
- *$Q_i \subseteq Q$ is the set of initial states,*
- *Δ is a set of transduction rules in the following form:*
  - *$q(f(x_1, x_2)) \to u[q_1(x_{i_1}), \ldots, q_p(x_{i_p})]$ where $f \in \Sigma$, $u \in \mathcal{C}^p(\Sigma)$, $q, q_1, \ldots, q_p \in Q$, and $x_{i_1}, \ldots, x_{i_p} \in \{x_1, x_2\}$.*
  - *$q(f) \to u$ where $f \in \Sigma, u \in \mathcal{C}^0(\Sigma), q \in Q$.*

Let $t, t' \in T_{\Sigma \cup Q}$. The move relation $\to$ is defined as follows:

$$t \to t' \Leftrightarrow \begin{cases} \exists q(f(x_1, x_2)) \ \to u[q_1(x_{i_1}), \ldots, q_p(x_{i_p})] \in \Delta, \\ \exists C \in \mathcal{C}(\Sigma \cup Q), \\ \exists u_1, u_2 \in T_\Sigma, \\ t = C[q(f(u_1, u_2))], \\ t' = C[u[q_1(v_1), \ldots, q_p(v_p)]] \text{ where } v_j = u_k \text{ if } x_{i_j} = x_k. \end{cases}$$

$\to^*$ is the reflexive and transitive closure of $\to$.

We consider that Φ defines the function that returns the following set for given Σ-tree t. (This function is also denoted by Φ.)

- $\Phi(t) = \{u \in T_\Sigma \mid \exists q \in Q_i, q(t) \to^* u\}$.

For a given tree transducer Φ over Σ, we define a tree transducer $\Phi^{+,-}$ over $\Sigma^{+,-}$ which represents a correspondence relation between an access instance for a Σ-tree t and access instances for Σ-trees in $\Phi(t)$.

Definition 6. *For a tree transducer* $\Phi = (Q, \Sigma, Q_i, \Delta)$*, a tree transducer* $\Phi^{+,-} = (Q, \Sigma^{+,-}, Q_i, \Delta')$ *is defined as follows.*

- *For a context* $u \in \mathcal{C}^p(\Sigma)$ *and* $s \in \{+, -\}$*,* u^s *is a context such that* $\mathrm{Dom}(u) = \mathrm{Dom}(u^s)$ *and for all node* v *in* u*,*
  - *If* $lab^u(v) = g \in \Sigma$*, then* $lab^{u^s}(v) = g^s$*.*
  - *If* $lab^u(v)$ *is a variable, then* $lab^{u^s}(v) = lab^u(v)$*.*

  Intuitively, u^s *is the context obtained from* u *by labeling* s *to every node except a variable.*
- Δ' *is the smallest set satisfying the followings:*
  - *For a rule* $q(f(x_1, x_2)) \to u[q_1(x_{i_1}), \ldots, q_p(x_{i_p})] \in \Delta$ *and* $s \in \{+, -\}$*,* $q(f^s(x_1, x_2)) \to u^s[q_1(x_{i_1}), \ldots, q_p(x_{i_p})] \in \Delta'$*.*
  - *For a rule* $q(f) \to u \in \Delta$ *and* $s \in \{+, -\}$*,* $q(f^s) \to u^s \in \Delta'$*.*

Example 4. *Let* $\Phi_1 = (Q^1, \Sigma^1, Q^1_i, \Delta^1)$ *where* $Q^1 = \{q_0, q_1, q_2, q_3, q_4\}$*,* $\Sigma^1 = \{doc, title, chap, sec, par\}$*,* $Q^1_i = \{q_0\}$*, and* Δ^1 *consists of the following rules.*

- $q_0(doc(x_1, x_2)) \to doc(title(\epsilon, q_1(x_1)), q_2(x_2))$*,*
- $q_1(chap(x_1, x_2)) \to sec(q_3(x_1), q_1(x_2))$*,*
- $q_1(\epsilon) \to \epsilon$*,*
- $q_2(\epsilon) \to \epsilon$*,*
- $q_3(par(x_1, x_2)) \to par(q_2(x_1), q_3(x_2))$*,*
- $q_3(par) \to par$*, and*
- $q_3(\epsilon) \to \epsilon$*.*

Intuitively, Φ_1 is a transducer that adds a *title* node as a child of a *doc* node and translates a *chap* node in an input tree into a *sec* node in the output tree.

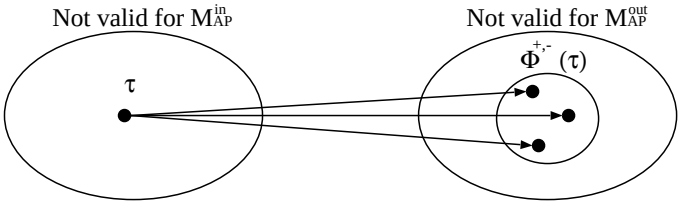

Figure 2.7. Consistency problem of policies

Given an unranked charged tree
$\tau = doc^+(chap^+(par^+par^-)chap^-(par^-))$,
$\Phi_1^{+,-}$ obtained from Φ_1 outputs
$\Phi_1^{+,-}(\tau) = doc^+(title^+sec^+(par^+par^-)sec^-(par^-))$.
(For readability, unranked (not binary) trees are used here.)

Below we formally define the consistency problem and show that it is decidable.

2.6.1 Problem Statement

Here we define the consistency problem of policies formally. First, we define validity of an access instance τ to policy TA M_{AP} in OR-semantics in a similar way to the validity of a query TA in Definition 7, and then define the consistency problem in Definition 8.

Definition 7. *Access instance $\tau \in T_{\Sigma^{+,-}}$ is valid for policy TA M_{AP} in OR-semantics if and only if $\tau' \preceq \tau$ for some τ' in $L(M_{AP})$.*

Definition 8. *The consistency problem of policies is defined as follows(Figure 2.7).*

Input: *A policy TA M_{AP}^{in} , a policy TA M_{AP}^{out}, and a tree transducer Φ.*

Output: *"YES" if the following condition holds, and "NO" otherwise. For any charged tree τ that is not valid for M_{AP}^{in} in OR-semantics, any elements of $\Phi^{+,-}(\tau)$ are not valid for M_{AP}^{out} in OR-semantics.*

The following is an instance of the problem that results in "NO". Let $L(M_{AP}^{in}) = \{a^+(b^+c^+d^-), a^+(b^+c^-d^+)\}$ and $L(M_{AP}^{out}) = \{a^+(b^+c^+e^+)\}$. Let $\tau_1 = a^+(b^-c^+d^+)$ and $\Phi^{+,-}$ be a tree transducer obtained from an input Φ such that $\Phi^{+,-}(\tau_1) =$

$\{a^+(b^-c^+e^+)\}$. Then the answer for this problem is "NO", because τ_1 is not valid for M^{in}_{AP} and $a^+(b^-c^+e^+) \in \Phi^{+,-}(\tau_1)$ is valid for M^{out}_{AP}.

The consistency problem in AND-semantics can be defined similarly and can be reduced to the one in OR-semantics by Theorem 1. Thus, we focus on the problem in OR-semantics.

Instead of the consistency problem, it would be more desirable if we could solve a policy transformation problem to construct, given a policy TA M^{in}_{AP} and a tree transducer Φ, a policy TA M^{out}_{AP} that (at least) satisfies the "YES" condition described in Definition 8. However, solving the policy transformation problem seems difficult since it is known that a tree transducer does not always preserve regularity [22] and thus $\Phi^{+,-}(M^{in}_{AP})$ is not always regular. Of course, the empty tree language is a conservative approximation of $\Phi^{+,-}(M^{in}_{AP})$, but such a policy is meaningless in real world. Hence, we have to construct a sufficiently large (hopefully maximal) regular subset of $\Phi^{+,-}(M^{in}_{AP})$ to solve the policy transformation problem in a meaningful way.

2.6.2 Decidability of the Problem

In this section, we show that the consistency problem is decidable. First, we state a few properties of a mapping (and its inverse) that maps a tree to a set of trees.

Definition 9. *Let Σ be an arbitrary alphabet and $\Phi : T_\Sigma \to 2^{T_\Sigma}$ be a mapping that maps a Σ-tree to a set of Σ-trees. Then we extend the domain of Φ to the power set 2^{T_Σ} of Σ-trees, and we define the inverse mapping Φ^{-1} of Φ, as follows:*

- $\Phi(L) = \bigcup_{t \in L} \Phi(t)$.
- $\Phi^{-1}(L) = \{t \mid \Phi(t) \subseteqq L\}$.

Lemma 12. *Φ and Φ^{-1} that are defined by Definition 9 satisfy the following properties, where L_1 and L_2 are subsets of T_Σ.*

- $\Phi(L_1) \subseteqq L_2 \Leftrightarrow L_1 \subseteqq \Phi^{-1}(L_2)$.

Proof. It is known that the pair (Φ, Φ^{-1}) is a Galois connection [33]. Therefore the following properties hold.

1. $L_1 \subseteqq L_2 \Rightarrow \Phi(L_1) \subseteqq \Phi(L_2)$. (Monotonicity of Φ)
2. $L_1 \subseteqq L_2 \Rightarrow \Phi^{-1}(L_1) \subseteqq \Phi^{-1}(L_2)$. (Monotonicity of Φ^{-1})
3. $L_1 \subseteqq \Phi^{-1}(\Phi(L_1))$.
4. $\Phi(\Phi^{-1}(L_2)) \subseteqq L_2$.

($\Rightarrow$) By 2, $\Phi(L_1) \subseteqq L_2 \Rightarrow \Phi^{-1}(\Phi(L_1)) \subseteqq \Phi^{-1}(L_2)$. By 3, $\Phi^{-1}(\Phi(L_1)) \subseteqq \Phi^{-1}(L_2) \Rightarrow L_1 \subseteqq \Phi^{-1}(L_2)$.
($\Leftarrow$) By 1, $L_1 \subseteqq \Phi^{-1}(L_2) \Rightarrow \Phi(L_1) \subseteqq \Phi(\Phi^{-1}(L_2))$. By 4, $\Phi(L_1) \subseteqq \Phi(\Phi^{-1}(L_2)) \Rightarrow \Phi(L_1) \subseteqq L_2$. □

We can reduce the problem into the set-inclusion problem for tree languages as follows.

Lemma 13. *The following two statements are equivalent.*

1. *For any charged tree τ that is not valid for M_{AP}^{in} in OR-semantics, any elements of $\Phi^{+,-}(\tau)$ are not valid for M_{AP}^{out} in OR-semantics.*
2. $\Phi^{+,-}(\overline{L(M_{AP}^{in})\uparrow}) \subseteqq \overline{L(M_{AP}^{out})\uparrow}$.

Proof. 1. $\Leftrightarrow \forall\tau(\tau \notin L(M_{AP}^{in})\uparrow \Rightarrow \forall\tau' \in \Phi^{+,-}(\tau), \tau' \notin L(M_{AP}^{out})\uparrow)$
$\Leftrightarrow \forall\tau(\tau \in \overline{L(M_{AP}^{in})\uparrow} \Rightarrow \Phi^{+,-}(\tau) \subseteqq \overline{L(M_{AP}^{out})\uparrow})$
$\Leftrightarrow \Phi^{+,-}(\overline{L(M_{AP}^{in})\uparrow}) \subseteqq \overline{L(M_{AP}^{out})\uparrow}$ □

It is known that the tree transducers have the following good property.

Lemma 14. *$\Phi(L_1) \subseteqq L_2$ is decidable for any regular tree languages L_1 and L_2 and a tree transducer Φ.*

Proof. It is known that $\Phi^{-1}(L_2)$ is regular if L_2 is regular (Theorem 49 in [9], [20]). Thus, by Lemma 12, $\Phi(L_1) \subseteqq L_2$ is decidable. □

We obtain the following theorem by Lemma 13 and Lemma 14.

Theorem 13. *The consistency problem is decidable.*

2.7 Conclusion of Chapter 2

In this chapter, we proposed a formal model for XML database access control based on tree automata and defined a static analysis problem for access control. By introducing the notion of charged alphabet, we can concisely and uniformly formalize the distinction of permission/denial in a policy and access/non-access in a query. Also, we provided two alternative semantics, AND-semantics and OR-semantics, and showed that the static analysis problems in AND-semantics and OR-semantics are solvable in square time and EXPTIME-complete, respectively. Our query model was compared with Neven's query automata [27] and the expressive power of our model was shown to be strictly stronger than Neven's one. We also proposed a consistency problem of policies in schema transformation and showed that the problem is decidable.

Chapter 3

Formal Model of Aspect-Oriented Programs and Its Expressive Power

3.1 Introduction

In this chapter, we propose a simple and general model of aspect-oriented programs called **A-LTS** and analyze its expressive power. The proposed model is very simple and is specified as a set of finite state machines (FSMs). The model can represent recursive weaving of state machines, which is not considered in related works, and therefore the class of systems obtained by weaving is a proper superset of the class of FSMs. We show that the expressive power of A-LTS is strictly stronger than FSM and strictly weaker than pushdown automaton (PDA) under language equivalence, bisimulation, and isomorphism. Then we compare in detail the expressive power of A-LTS with a few subclasses of PDA (or equivalently of context-free grammars): classes of deterministic PDA and linear grammars. Finally, we state the relationship between the pointcuts of A-LTS and AspectJ[3], which is one of the well-known AOP languages.

In Section 3.3, we mention the design principle of A-LTS, followed by a formal definition of A-LTS in Section 3.4. In Section 3.5, we compare the expressive power of A-LTS with the FSM and PDA under language equivalence, bisimulation, and isomorphism. In Section 3.6, we state the relationship between pointcuts

of A-LTS and AspectJ. Finally, we give a conclusion and future works in Section 3.7.

Related Works Douence et al. [11] proposed a formal model of aspect-oriented programs based on a functional language Haskell. Their framework is based on the following simple principles:

- Points of interest of program execution are modeled as events.
- Each pointcut is specified as a pattern of event sequences.
- When an execution trace of a program matches a pointcut, the advice associated with the pointcut is executed.

However, the formal semantics of the pattern language, which is defined by the number of equations, is not simple, so it is not easy to use in formal verifications and other applications. Moreover, only a mechanism for selecting advice at each execution step is proposed, and one for weaving advices into a basic program is not described. Nakajima et al. [28, 29] proposed an aspect-oriented extension of UML State Diagrams. Their framework follows the above three principles and inherits the clarity of State Diagrams. Moreover, the constructs of pointcuts are simple but powerful: one can specify a pointcut such as "any configuration where a component state machine M is in a specified state s and when an event e just occurs." Switching between the basic program and a woven advice is represented by general-purpose control primitives, which can stop and resume any component state machine. However, the recursion of the suspension of state machines is not concerned, and thus the state space of the whole model is still finite.

Other areas of computer science such as database and security investigate technologies resemble AOP, especially active database and history-based access control.

An active database [31] consists of a set of active rules and a database instance. An active rule usually has the form of ECA (event-condition-action), which means that 'when a specified event occurs, a specified action should be performed on the current database instance if a specified condition is satisfied.'

The access control technology most related to AOP is history-based access control (HBAC). The origin of HBAC is stack inspection, which is now broadly

used as dynamic access control infrastructure in Java and C♯. However, stack inspection has a problem: the stack does not retain security information on the called methods with which execution is finished. To solve this problem, a few access control methods have been proposed [1, 13, 32]. The common features of these works include that execution history is not always forgotten, even if the surrounding method execution is completely finished. Schneider [32] defines an enforceable security policy as a prefix-closed nonempty set of event sequences. He also defines security automata, which exactly recognize enforceable policies. However, the expressive power of security automaton is Turing powerful and thus too large. Fong [13] introduces several subclasses of security automata and compares their expressive power. In particular, [13] defines shallow history automata with finite state space and shows that the class of policies recognized by shallow history automata is incomparable with stack inspection.

Both security automata and our proposed model take automata-theoretic approaches. The main difference between them is that in HBAC, once the execution history of a controlled program does not match a given security policy (i.e., a given pattern), the execution of the program is aborted. Hence, it is impossible for a finite-state security automaton including a shallow history automaton to simulate recursive weaving.

3.2 Framework of AOP

An aspect-oriented program consists of one **basic program** (a program to which advices are woven) and zero or more aspects. In AOP, any procedure describing a crosscutting concern can be inserted into a specific execution point of a program. Each execution point where a procedure can be inserted is called a join point, and the inserted procedure is called an advice. An aspect is a pair of an advice and a pointcut, which is a specification of the set of join points to which the advice should be connected. Figure 3.1 is an example of aspect-oriented program written in AspectJ. The pointcut "call(bp.m())" in the aspect "asp" represents join points where a method call to a method "void m()" of class "bp" in the basic program is made. The advice of "asp," which is given the prefix "before," will be inserted before the join points specified by the above pointcut. Thus the

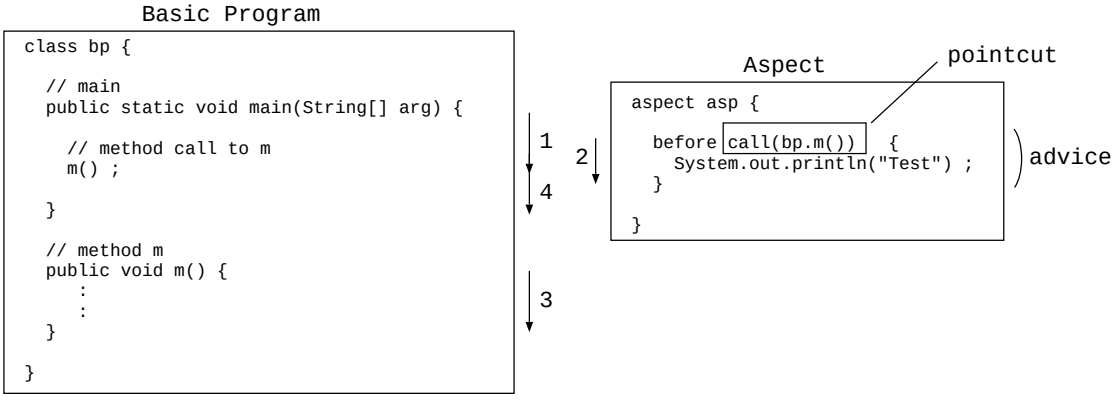

Figure 3.1. Sample aspect-oriented program

string "Test" is outputted before the method call to "m()." The execution order is $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$ in Figure 3.1. Other primitives used for pointcuts in AspectJ will be described in Section 3.6.1.

3.3 Basic design of program model

A program (or a fragment of a program) is modeled as a **labeled transition system** (LTS), which defines a set of possible sequences of atomic actions (called **events**). Both a basic program and each advice are modeled as finite LTSs, i.e., finite state machines (FSMs).

A join point where an advice is connected is determined by the following mechanism, which is similar to the one in [11]. There are two parallel synchronized virtual machines: **main thread** and **monitor**. The main thread is used for the execution of a basic program and advices, and at first it invokes the basic program. The monitor observes the event sequence performed by the main thread, deciding whether the sequence matches each pointcut. When it matches pointcut P_i, the monitor tells the main thread to join advice A_i, which is associated with P_i. After the execution of A_i is finished, the main thread resumes the execution of the program that was running just before A_i was invoked.

Each pointcut is defined as a pattern of event sequences (or equivalently, a

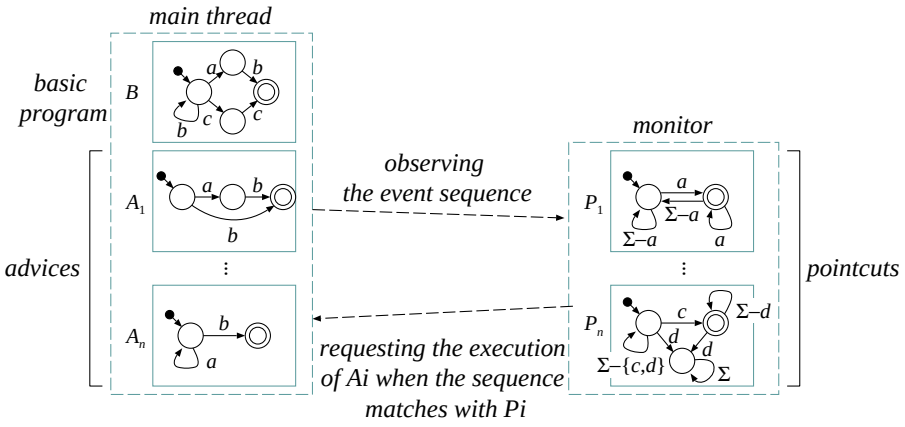

Figure 3.2. A-LTS Framework

set of event sequences). When an event sequence starting at the beginning of a whole program matches a pointcut, the advice associated with the pointcut is invoked. In our model, a pointcut is defined by a deterministic finite automaton.

The execution of each advice terminates when control reaches a specified final state. Since a finite automaton can be regarded as an LTS with specified final states, we model each basic program, pointcuts, and advices as an LTS with final states.

3.4 Program model A-LTS

An **A-LTS** is a tuple of a basic program, n pointcuts, and n advices. An A-LTS specifies a single infinite LTS.

3.4.1 Labeled transition system

In this chapter, we use LTSs with final states as the fundamental constructs. Each basic program, pointcuts, and advices is modeled as a finite LTS with final states. In a basic program and an advice, a final state is regarded as a terminating point of execution. A pointcut is used as a language acceptor. That is, for pointcut P_i, every event sequence from the initial state to a final state represents a join point

specified by P_i.

Definition 10. *A* **labeled transition system with final states** *on alphabet Σ is a 5-tuple*

$$L = (\Sigma, Q_L, \rightarrow_L, I_L, F_L),$$

where Q_L is a finite or an infinite set of states, $\rightarrow_L$ $(\subseteqq Q_L \times \Sigma \times Q_L)$ is a transition relation, $I_L (\in Q_L)$ is an initial state, and $F_L (\subseteqq Q_L)$ is a set of final states.

We denote $(q_1, a, q_2) \in \rightarrow_L$ as $q_1 \stackrel{a}{\rightarrow}_L q_2$.

Let Q_L, $\rightarrow_L$, I_L, and F_L denote the set of states, the transition relation, the initial state, and the set of final states of an LTS L, respectively. We assume that the alphabets of all LTSs are the same and denoted by Σ.

Definition 11. *An LTS L is* **deterministic** *if for all $q \in Q_L$ and $a \in \Sigma$, there exists exactly one $q' \in Q_L$ such that $q \stackrel{a}{\rightarrow}_L q'$.*

3.4.2 A-LTS

Definition 12. A-LTS *is a $(2n+1)$-tuple of finite LTSs*

$$PR = (B, P_1, A_1, P_2, A_2, \ldots, P_n, A_n),$$

where $n \geq 0$. B is a **basic program**, *$P_1, \ldots, P_n$ are* **pointcuts**, *and $A_1, \ldots, A_n$ are* **advices**. *They should satisfy the following constraints:*

- *Each pointcut is deterministic.*
- *$Q_B, Q_{A_1}, \ldots, Q_{A_n}$ are pairwise disjoint.*
- *The initial states are not final states for $B, P_1, \ldots, P_n, A_1, \ldots, A_n$.*

An intuitive semantics of an A-LTS is as follows. First, the execution of B starts. When the event sequence starting at the beginning of B (time 0) matches pointcut P_i; that is, the sequence is accepted by P_i, the execution of B is suspended and advice A_i is invoked. After that, when the event sequence from time 0 grows according to the execution and matches pointcut P_j, the execution of A_i

is suspended, and advice A_j is invoked. In this way, executions of advices are inserted recursively. When control reaches a final state of an advice, the suspended execution of the basic program or an advice is resumed. A-LTS terminates when control reaches a final state of the basic program.

When an event sequence simultaneously matches more than one pointcut, all advices associated with them are executed in a specific order. This order is defined by the indexes of pointcuts and advices. When a sequence simultaneously matches both P_i and P_j for $i < j$, A_i is invoked first, and A_j is invoked just after A_i terminates.

3.4.3 Formal semantics of A-LTS

First we define some terminologies. The formal semantics of A-LTS is given in Definition 13. In the following, we fix an A-LTS $PR = (B, P_1, A_1, \ldots, P_n, A_n)$.

- For arbitrary set X, let X^* be the set of all finite sequences of elements in X. Let ϵ be the empty sequence. The singleton sequence that consists of element x is denoted by x itself. $\xi : \nu$ denotes the concatenation of two sequences, ξ and ν.

- Let $Q = Q_B \cup Q_{A_1} \cup \cdots \cup Q_{A_n}$. Note that by Definition 12, $Q_B, Q_{A_1}, \ldots, Q_{A_n}$ are pairwise disjoint. Let $M : Q \to \{B, A_1, \ldots, A_n\}$ be a mapping that maps $q \in Q$ to the LTS to which q belongs. For example, $\mathrm{M}(q) = B$ if $q \in Q_B$.

- A mapping $AD : Q_{P_1} \times \cdots \times Q_{P_n} \to Q^*$ is defined as follows.

$$AD(q_1, \ldots, q_n) = I_{A_{i_1}} : I_{A_{i_2}} : \ldots : I_{A_{i_m}},$$

  where $1 \leq i_1 < i_2 < \cdots < i_m \leq n$ and $\{i_1, i_2, \ldots, i_m\} = \{i \mid q_i \in F_{P_i}\}$. Intuitively, $AD(q_1, \ldots, q_n)$ represents the list of the initial states of advices that should be started when each pointcut P_i goes to q_i. The order of the initial states in $AD(q_1, \ldots, q_n)$ corresponds to the execution order of the advices.

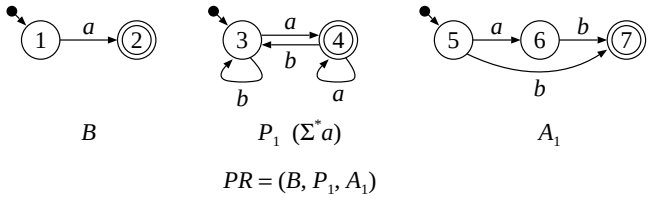

Figure 3.3. A-LTS recognizing $\{a^m b^m \mid 0 < m\}$.

- A mapping $EF : Q \to Q^*$ is defined as follows.

$$EF(q) = \begin{cases} \epsilon & \text{if } q \in F_{\mathrm{M}(q)}, \\ q & \text{otherwise.} \end{cases}$$

Definition 13. *The formal semantics of an A-LTS PR is defined as the following LTS* TS_{PR}.

$$TS_{PR} \;=\; (\Sigma, Q^* \times Q_{P_1} \times \cdots \times Q_{P_n}, \to_{PR}, (I_B, I_{P_1}, \ldots, I_{P_n}), F_{PR}),$$

where $F_{PR} = \{(\epsilon, q_1, \ldots, q_n) \mid q_i \in Q_{P_i} \text{ for } 1 \le i \le n\}$ *and* $\to_{PR}$ *is defined by the following inference rule.*

$$\frac{s \xrightarrow{a}_{\mathrm{M}(s)} s' \qquad q_i \xrightarrow{a}_{P_i} q_i' \quad (1 \le i \le n)}{(s : \xi,\ q_1, \ldots, q_n) \xrightarrow{a}_{PR} (AD(q_1', \ldots, q_n') : EF(s') : \xi,\ q_1', \ldots, q_n')}$$

Figure 3.3 shows an A-LTS PR recognizing $\{a^m b^m \mid 0 < m\}$. An A-LTS PR recognizes L if L is the set of sequences each of which brings T_{PR} to a final state (cf. Definition 17). Figure 3.4 is the TS_{PR} for the A-LTS PR in Figure 3.3. A double circle denotes a final state. When PR in the initial configuration reads a, B and P_1 enter final states. Thus B is terminated and A_1 starts. Next, A_1 can read either a or b. If A_1 reads a, then P_1 is entering a final state again, and thus A_1 is newly invoked. A_1 is recursively invoked m times just after PR reads a^m. When a newly invoked A_1 reads b, it simply terminates. Since P_1 enters a non-final state whenever PR reads b, A_1 is not invoked at that time. Each suspended A_1 can only read b, which terminates A_1. Just after PR reads $a^m b^m$, all the suspended A_1 terminates.

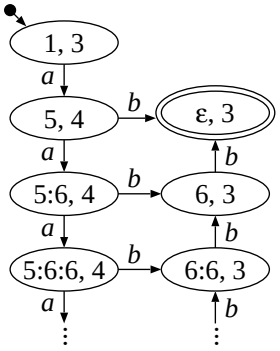

Figure 3.4. TS_{PR} for PR in Figure 3.3

3.5 Expressive power of A-LTS

In this section, we compare the expressive power of A-LTS with the other state transition models, such as FSM and pushdown automaton (PDA). For two classes C_1 and C_2 of state transition models, "C_2 **includes** C_1" ($C_1 \subseteqq C_2$) if for any model M_1 in C_1, there exists some model M_2 in C_2 that is equivalent to M_1. "C_1 **is equivalent to** C_2"($C_1 = C_2$) if $C_1 \subseteqq C_2$ and $C_2 \subseteqq C_1$.

There are a few different definitions of equivalence between the two models [23]. We use three different definitions of equivalence: isomorphism, bisimulation, and language equivalence. These equivalences have the following properties:

- Two models are bisimilar if they are isomorphic.
- Two models are language equivalent if they are bisimilar.

3.5.1 Equivalence of models

We define the three equivalences (isomorphism, bisimulation, and language equivalence) between two LTSs as follows.

Definition 14 (Isomorphism)**.** *LTSs L_1 and L_2 are isomorphic if there exists a bijection $\mathcal{R} : Q_{L_1} \to Q_{L_2}$ with the following properties.*

(a) For any states $s_1, s_1' \in Q_{L_1}$ and any event $a \in \Sigma$, $s_1 \xrightarrow{a}_{L_1} s_1'$ if and only if $\mathcal{R}(s_1) \xrightarrow{a}_{L_2} \mathcal{R}(s_1')$.

(b) For any $s \in Q_{L_1}$, $s \in F_{L_1}$ if and only if $\mathcal{R}(s) \in F_{L_2}$.

Definition 15 (bisimulation relation)**.** *For any pair of LTSs (L_1, L_2), a relation $\mathcal{R} \subseteqq Q_{L_1} \times Q_{L_2}$ is a* **bisimulation relation** *on (L_1, L_2) if for every $(s_1, s_2) \in \mathcal{R}$ and $a \in \Sigma$, $\mathcal{R}$ satisfies the following properties.*

(a) If $s_1 \xrightarrow{a}_{L_1} s_1'$, then there exists some $s_2' \in Q_{L_2}$ such that $(s_1', s_2') \in \mathcal{R}$ and $s_2 \xrightarrow{a}_{L_2} s_2'$.

(b) If $s_2 \xrightarrow{a}_{L_2} s_2'$, then there exists some $s_1' \in Q_{L_1}$ such that $(s_1', s_2') \in \mathcal{R}$ and $s_1 \xrightarrow{a}_{L_1} s_1'$.

(c) $s_1 \in F_{L_1}$ if and only if $s_2 \in F_{L_2}$.

Definition 16 (bisimulation)**.** *LTSs L_1 and L_2 are* **bisimilar** *if a bisimulation relation exists $\mathcal{R}$ on (L_1, L_2) such that $(I_{L_1}, I_{L_2}) \in \mathcal{R}$.*

The behavior of two models are identical if they are bisimilar. Note that the usual definition of bisimulation relation does not require property (c) of Definition 15, because the relation is defined on LTSs without final states. However, Definition 15 coincides with the usual definition if L_1 and L_2 satisfy the following properties.

(1) At least one transition exists from every non-final state. (In the case of A-LTS, there exists at least one transition from every non-final state of a basic program and advices.)

(2) There is no transition from final states. (Every A-LTS satisfies this property by Definition 13.)

Definition 17. *For an LTS L, $Lang(L) \subseteqq \Sigma^*$ is defined as follows.*

$$\begin{aligned} Lang(L) \;=\; \{a_1 a_2 \ldots a_n \in \Sigma^* \mid\; & \textit{There exist } s_0, \ldots, s_n \in Q_L \\ & \textit{such that } I_L = s_0 \textit{ and } s_{i-1} \xrightarrow{a_i}_L s_i \; (1 \le i \le n) \textit{ and } s_n \in F_L\}. \end{aligned}$$

$Lang(L)$ is called the **language of** *L. We say that L* **recognizes** *a set $S \subseteqq \Sigma^*$ if and only if $S = Lang(L)$. For any sequence $w \in Lang(L)$, we say L* **accepts** *w.*

Definition 18 (Language equivalence)**.** *LTSs L_1 and L_2 are* **language equivalent** *if $Lang(L_1) = Lang(L_2)$.*

3.5.2 Comparisons with FSM and PDA

An FSM is an LTS with a finite number of states. The class of languages recognized by FSMs equals the class of regular languages. The class of languages recognized by pushdown automata (PDA) equals the class of context-free languages.

Now we discuss the expressive power of A-LTS. We denote the classes of A-LTSs, FSMs, and PDAs as A-LTS, FSM, and PDA, respectively. Below we will show that A-LTS $\subsetneqq$ PDA and FSM $\subsetneqq$ A-LTS under isomorphism and PDA $\not\subseteqq$ A-LTS and A-LTS $\not\subseteqq$ FSM under language equivalence (Theorem 15). Let $\mathcal{L}_{\text{A-LTS}}$, REG, and CFL be the classes of languages recognized by A-LTSs, FSMs, and PDAs, respectively. We will show that CFL $\not\subseteqq \mathcal{L}_{\text{A-LTS}}$ and $\mathcal{L}_{\text{A-LTS}} \not\subseteqq$ REG, which imply PDA $\not\subseteqq$ A-LTS and A-LTS $\not\subseteqq$ FSM under language equivalence.

We also discuss a relation among $\mathcal{L}_{\text{A-LTS}}$ and two subclasses of CFL: the classes of deterministic context-free and linear languages. A PDA is **deterministic** if no more than one transition exists from every configuration reachable from the initial configuration. A language recognized by a deterministic PDA is a **deterministic context-free language**. A **linear context-free grammar** (or a **linear grammar**) is a context-free grammar in which at most one non-terminal symbol can occur on the right-hand side of every production. A **linear language** is a language generated by a linear grammar. Let DCFL and $\mathcal{L}_{\text{linear}}$ be the classes of deterministic context-free and linear languages, respectively.

Now we define the following eight context-free languages to discuss the inclusion relation between classes of languages.

- $L_1 = \{a^m b^m \mid 0 < m\}$
- $L_2 = \{a^m b^m c^n d^n \mid 0 < m,\ 0 < n\}$
- $L_3 = \{a^m b^n \mid 0 < m \leq n \leq 2m\}$
- $L_4 = \{a^m b^n \mid 0 \leq n \leq m,\ 0 < m\}$
- $L_5 = \{a^k b^k c^m d^n \mid 0 < m \leq n \leq 2m,\ 0 < k\}$
- $L_6 = \{a^m b^n \mid 0 < m,\ n \in \{0, m, 2m\}\}$

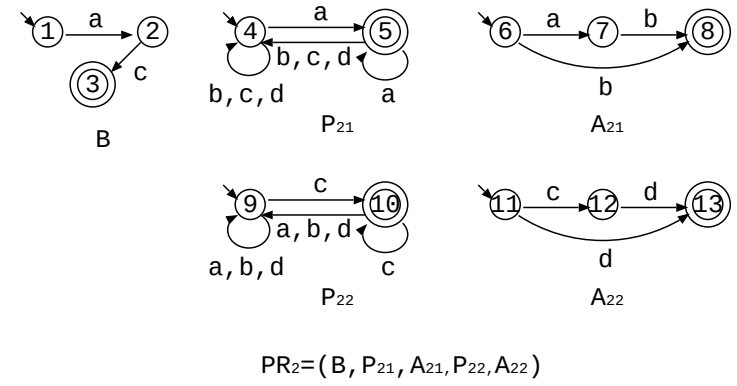

Figure 3.5. A-LTS recognizing L_2

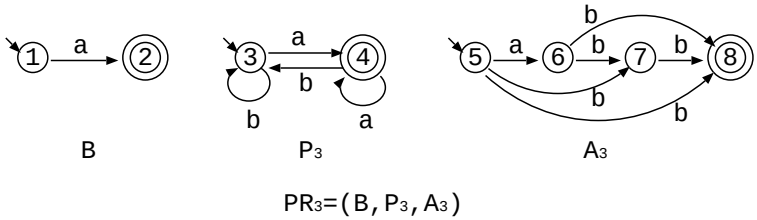

Figure 3.6. A-LTS recognizing L_3

- $L_7 = \{a^m b^n c^k d^k \mid 0 \leq n \leq m,\ 0 < m,\ 0 \leq k\}$
- $L_8 = \{a^m b^n c^k d^k \mid n \in \{0, m, 2m\},\ 0 < m,\ 0 \leq k\}$

Lemma 15. $L_1, L_2, L_3, L_5 \in \mathcal{L}_{\text{A-LTS}}$.

Proof. As mentioned in Section 3.4.3, the A-LTS in Figure 3.3 recognizes L_1. Figures 3.5, 3.6, and 3.7 show A-LTSs recognizing L_2, L_3, and L_5, respectively. A-LTS PR_2 in Figure 3.5 is obtained from PR in Figure 3.3 by adding P_{22} and A_{22}, which resemble P_{21} and A_{21} and guarantee that the numbers of cs and ds are identical. PR_3 in Figure 3.6 is obtained from PR in Figure 3.3 by replacing advice A_1 with A_3, which nondeterministically consumes one or two bs for each a. A_3 thus guarantees that PR_3 exactly accepts $a^m b^n$ such that $m \leq n \leq 2m$. PR_5 in Figure 3.7 is a combination of PR_2 and PR_3. □

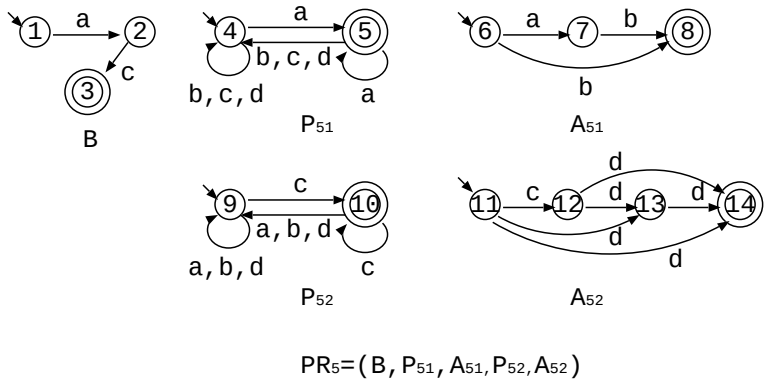

Figure 3.7. A-LTS recognizing L_5

To show that some languages are not in $\mathcal{L}_{\text{A-LTS}}$, we use the following lemma.

Lemma 16. *If A-LTS PR accepts sequence w, then no pointcuts of PR accept w.*

Proof. We show the contraposition. Let w be a sequence accepted by some pointcut of PR. When the event sequence starting at time 0 becomes w, the advice corresponding to the pointcut that accepts w is invoked. Since any state precisely when an advice is invoked is not a final state of PR by Definition 13, PR does not accept w. □

Lemma 17. *Let L be the language of an A-LTS PR such that $L' - \{\epsilon\} \subseteqq L$ for some prefix-closed language L'. Then a constant m exists that satisfies the following condition. If $uw \in L$, $u \in L'$, $|u| \geq m$ and $w \in \Sigma^*$, then u can be decomposed into $u = xyz$ such that $|y| > 0$ and $|xy| \leq m$ and xy^kz for any $k \geq 0$ also belongs to L.*

Proof. By Lemma 16, no pointcuts of PR accept any $u \in L' - \{\epsilon\}$. Therefore, since L' is prefix-closed, no advices of PR are invoked while PR reads a fixed sequence $u \in L' - \{\epsilon\}$. Let m be the cardinality of $Q_B \times Q_{P_1} \times \cdots \times Q_{P_n}$. Fix a sequence $uw \in L$ such that $u \in L'$ and $|u| \geq m$, and also fix an accepting execution of PR while reading uw. Then for the first part of the execution of PR while reading u, at least one configuration exists of PR that the execution

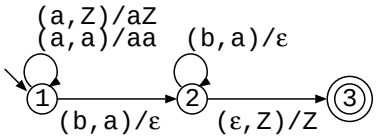

Figure 3.8. Deterministic PDA recognizing L_1

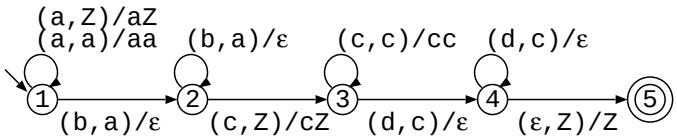

Figure 3.9. Deterministic PDA recognizing L_2

visits twice or more, and an execution obtained by removing or repeating the part between the occurrences of the same configuration is also a valid accepting execution of PR. Letting y be the fragment corresponding to the pumped part, we obtain this lemma. □

Lemma 18. $L_4, L_6, L_7, L_8 \notin \mathcal{L}_{\text{A-LTS}}$.

Proof. Note that each of these languages includes a^+, which equals $a^* - \{\epsilon\}$ and a^* is prefix-closed. Assuming that each of the languages is recognized by an A-LTS, then we can show a contradiction to Lemma 17 by selecting $uw = a^m b^m$ for L_4 and L_7 and $uw = a^m b^{2m}$ for L_6 and L_8 for the constant m in Lemma 17. □

Lemma 19. $L_1, L_2, L_4, L_7 \in$ DCFL *and* $L_3, L_5, L_6, L_8 \notin$ DCFL.

Proof. Figures 3.8, 3.9, 3.10, and 3.11 are deterministic PDAs that recognize L_1, L_2, L_4, and L_7, respectively. In these figures, each circle denotes a control state, and each double circle denotes a final state. The label on each transition specifies a triple $(a, g)/w$ where a is either an event or ϵ, g is a stack symbol at the top of the stack, and w is a sequence of stack symbols to which the top of the stack will be replaced [18]. Z is the start symbol of the stack. A deterministic PDA M accepts sequence w if M enters a final state just after reading w.

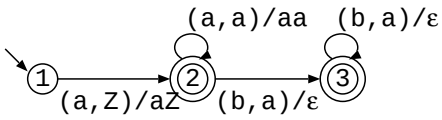

Figure 3.10. Deterministic PDA recognizing L_4

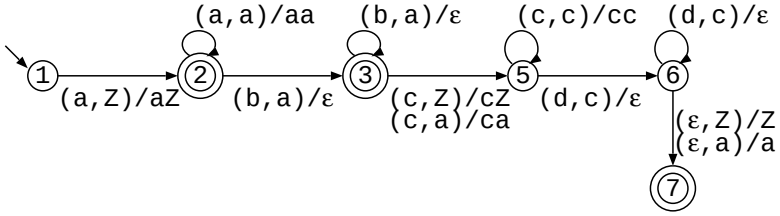

Figure 3.11. Deterministic PDA recognizing L_7

We prove $L_3 \notin$ DCFL by contradiction. Assume that a deterministic PDA exists that recognizes L_3. Then we can construct a PDA that recognizes the following L_3' using a technique shown in [18, p.196].

$$\begin{aligned} L_3' &= \{a^m b^n c^k \mid a^m b^n \in L_3,\ a^m b^{n+k} \in L_3\} \\ &= \{a^m b^n c^k \mid 0 < m \le n \le n+k \le 2m\}. \end{aligned}$$

However, $L_3' \notin$ CFL by the pumping lemma for context-free languages. Therefore, $L_3 \notin$ DCFL.

In a similar way, if a deterministic PDA exists that recognizes L_5, then we can construct a PDA that recognizes $L_5' = \{a^k b^k c^m d^n e^l \mid a^k b^k c^m d^n \in L_5,\ a^k b^k c^m d^{n+l} \in L_5\} = \{a^k b^k c^m d^n e^l \mid 0 < k,\ 0 < m \le n \le n+l \le 2m\}$. Let h be a homomorphism such that $h(a) = h(b) = \epsilon,\ h(c) = a,\ h(d) = b,$ and $h(e) = c$. Then $h(L_5') = L_3'$. Since CFL is closed under homomorphism, L_3' must be in CFL, contradicting the above fact that $L_3' \notin$ CFL. Therefore, $L_5 \notin$ DCFL.

We prove $L_6 \notin$ DCFL by contradiction. Assume that $L_6 \in$ DCFL. Since DCFL is closed under intersection with a regular language, $L_6' = L_6 \cap a^* b^+ = \{a^m b^n \mid n \in \{m, 2m\}, 0 < m\} \in$ DCFL. From a deterministic PDA recognizing L_6', we can construct a nondeterministic PDA that recognizes $L_6'' = \{a^m b^n c^k \mid$

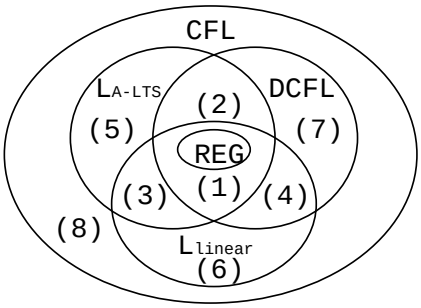

Figure 3.12. Relationship between $\mathcal{L}_{\text{A-LTS}}$ and well-known classes of languages

$m > 0,\ n \in \{m, 2m\},\ n + k \in \{m, 2m\}\}$. However, $L_6'' \notin$ CFL by the pumping lemma for context-free languages. Therefore, $L_6 \notin$ DCFL.

Since $L_8 \cap a^*b^+ = L_6'$ and DCFL is closed under intersection with a regular language, $L_8 \notin$ DCFL. □

Lemma 20. *$L_1, L_3, L_4, L_6 \in \mathcal{L}_{\text{linear}}$ and $L_2, L_5, L_7, L_8 \notin \mathcal{L}_{\text{linear}}$.*

Proof. Following linear grammars G_1, G_3, G_4, and G_6 generate L_1, L_3, L_4, and L_6, respectively.

- $G_1 = (\{S\}, \{a, b\}, \{S \to ab \mid aSb\}, S)$
- $G_3 = (\{S\}, \{a, b\}, \{S \to ab \mid abb \mid aSb \mid aSbb\}, S)$
- $G_4 = (\{S\}, \{a, b\}, \{S \to a \mid ab \mid aS \mid aSb\}, S)$
- $G_6 = (\{S, A, B, C\}, \{a, b\}, \{S \to A \mid B \mid C,\ A \to aA \mid a,\ B \to aBb \mid ab,\ C \to aCbb \mid abb\}, S)$

We can easily show that L_2, L_5, L_7, $L_8 \notin \mathcal{L}_{\text{linear}}$ by the pumping lemma for linear languages [18]. □

Lemma 21. *$L_1 \notin$ REG.*

Proof. We can show this lemma by the pumping lemma for regular languages. □

Lemma 22. *FSM* $\subsetneqq$ *A-LTS under isomorphism and REG* $\subsetneqq \mathcal{L}_{\text{A-LTS}}$.

Proof. An FSM is an A-LTS without pointcuts and advices. Therefore, FSM $\subsetneqq$ A-LTS under isomorphism, which implies REG $\subsetneqq \mathcal{L}_{\text{A-LTS}}$. □

Figure 3.12 shows the relationship between $\mathcal{L}_{\text{A-LTS}}$ and the classes discussed above. The following Theorem 14 states that each subset (1)–(8) in Figure 3.12 is not empty.

Theorem 14. *The following sets of languages are not empty.*

1. $(\mathcal{L}_{\text{A-LTS}} \cap DCFL \cap \mathcal{L}_{\text{linear}}) - REG$
2. $(\mathcal{L}_{\text{A-LTS}} \cap DCFL) - \mathcal{L}_{\text{linear}}$
3. $(\mathcal{L}_{\text{A-LTS}} \cap \mathcal{L}_{\text{linear}}) - DCFL$
4. $(DCFL \cap \mathcal{L}_{\text{linear}}) - \mathcal{L}_{\text{A-LTS}}$
5. $\mathcal{L}_{\text{A-LTS}} - (DCFL \cup \mathcal{L}_{\text{linear}})$
6. $DCFL - (\mathcal{L}_{\text{A-LTS}} \cup \mathcal{L}_{\text{linear}})$
7. $\mathcal{L}_{\text{linear}} - \mathcal{L}_{\text{A-LTS}} \cup DCFL$
8. $CFL - (\mathcal{L}_{\text{linear}} \cup \mathcal{L}_{\text{A-LTS}} \cup DCFL)$

Proof. By Lemmas 15, 18, 19, 20, and 21, L_1, L_2, ..., L_8 belong to set 1, 2, ..., 8, respectively. □

The following theorem is the main result of this section.

Theorem 15. *FSM* $\subsetneqq$ *A-LTS* $\subsetneqq$ *PDA under language equivalence, bisimulation, and isomorphism.*

Proof. By Lemma 22, FSM $\subsetneqq$ A-LTS under isomorphism. By Theorem 14, none of the subsets (1), (2), (3), and (5) is empty. Therefore, A-LTS $\not\subseteqq$ FSM under language equivalence. For any A-LTS PR, there exists a PDA isomorphic to PR, whose set of control states is $Q_{P_1} \times \cdots \times Q_{P_n}$ and the set of stack symbols is Q. Therefore, A-LTS $\subsetneqq$ PDA under isomorphism. By Theorem 14, none of the subsets (4), (6), (7), and (8) is empty. Therefore, PDA $\not\subseteqq$ A-LTS under language equivalence. □

3.6 A-LTS and AspectJ

In this section, we discuss the relationship between the pointcuts of A-LTS and AspectJ, which is one typical AOP language.

3.6.1 AspectJ

AspectJ is an AOP language based on Java. A program in AspectJ consists of a set of classes and aspects. An aspect consists of pointcuts and advices. The main constructs of pointcuts are as follows.

- call(m) – the set of method calls to m.
- execute(m) – the execution of the body of method m.
- cflow(p) – the set of all join points subsequent to any join point j_p specified by pointcut p.
- get(f) – the set of execution points at which the value of data field f is used.
- set(f) – the set of execution points at which a value is assigned to data field f.

Note that for call and execute pointcuts, the execution of the whole body of method m is regarded as a single join point. There are operators to make a pointcut from other pointcuts. For example, a pointcut $P_1 \,||\, P_2$ specifies join points that matches either P_1 or P_2. A pointcut $P_1 \,\&\&\, P_2$ specifies join points that matches both P_1 and P_2.

For each advice, one of the three keywords {**before**, **after**, **around**} as well as a pointcut is given. Before/after denotes that the advice should be inserted before/after each join point specified by the pointcut. Around denotes that each join point specified by the pointcut should be replaced with the advice. For example, when we want to execute an advice before every method call to a method *proc*, we specify pointcut "before call(*proc*)" for the advice.

3.6.2 Discussion

First, we model a basic program and advices written in AspectJ as follows. Event set of a program is the set of join points and other related actions. For example, call to a method and assignment to a data field are regarded as events. A basic program and advices are defined as processes executing events in a specific order. As stated above, in AspectJ, the execution of the whole body of a method is regarded as a single join point. However, if we consider this join point as an atomic event, then we cannot represent the recursion of method call. Therefore, we consider the start and end points of the execution of a method to be distinct events.

Next, we describe the correspondence between each pointcut in AspectJ and a pointcut in A-LTS. Let $call_m$ be an event that represents the call to method m and $return_m$ be an event that represents the return from m. Pointcut "before call(m)" of AspectJ is denoted as a regular language $P_{call_m} = \Sigma^* call_m$ in A-LTS. Pointcut "after call(m)" is represented as $\Sigma^* return_m$. For P_{call_m}, an advice is inserted just after the event $call_m$ occurs, i.e., the method call has just processed and the body of m has just started. Thus P_{call_m} does not exactly correspond to "before call(m)." To solve this problem, we introduce an event $before_{call_m}$ that represents the execution point just before the execution of the method call to m, and we define the pointcut as $\Sigma^* before_{call_m}$.

We can consider pointcuts that execute, get, and set in a similar way. If one wants to exactly express "before execute/get/set," then she needs to define events that represent the points just before execute/get/set similarly to before call.

Next we consider pointcut "cflow(pc)." Let P_{pc} be a pointcut of A-LTS represented by a regular language that represents pc. Then "after cflow(pc)" is represented as $P_{pc} + P_{pc}\Sigma^*\Sigma_{jp}$, where Σ_{jp} is the set of all events that correspond to join points. Similarly, "before cflow(pc)" is represented as $P_{pc} + P_{pc}\Sigma^*\Sigma_{b_{jp}}$ where $\Sigma_{b_{jp}}$ is the set of all events just before each event that corresponds to a join point.

Using cflow and the combining operators || and &&, we can describe complicated pointcuts, e.g., shown in the following table.

pointcuts	regular expressions
call(m) || call(n)	$\Sigma^*(call_m + call_n)$
cflow(call(m) || call(n)) && get(v)	$\Sigma^*(call_m + call_n)\Sigma^* get_v$
cflow(cflow(call(m)) && get(v)) && set(v)	$\Sigma^* call_m \Sigma^* get_v \Sigma^* set_v$

Since these complicated pointcuts in AspectJ can be represented by regular expressions, we consider that modeling a pointcut as a regular language is appropriate.

3.7 Conclusion of Chapter 3

In this chapter, we proposed a simple formal model of AOP **A-LTS**. We compared the expressive power of A-LTS with FSM and PDA under language equivalence, bisimulation, and isomorphism. As a result, we showed the relationship among a few subclasses of context-free languages and the class of the languages of A-LTSs, shown in Figure 3.12, and FSM $\subsetneqq$ A-LTS $\subsetneqq$ PDA under language equivalence, bisimulation, and isomorphism. Finally, we stated the relationship between pointcuts in A-LTS and in AspectJ, one typical AOP language.

Chapter 4

Conclusion

In this thesis, we proposed language-based formal models for XML access control and Aspect-Oriented programming to understand complicated behavior of them more easily.

In Chapter 2, a formal model for XML database access control based on tree automata was proposed and a static analysis problem for access control was defined. By introducing the notion of charged alphabet, we can concisely and uniformly formalize the distinction of permission/denial in a policy and access/non-access in a query. Also, we provided two alternative semantics, AND-semantics and OR-semantics, and showed that the static analysis problems in AND-semantics and OR-semantics are solvable in square time and EXPTIME-complete, respectively. Our query model was compared with Neven's query automata [27] and the expressive power of our model was shown to be strictly greater than Neven's one. We also proposed a consistency problem of policies in schema transformation and showed that the problem is decidable. Implementation of an analysis tool and empirical evaluation of the proposed method are left as future studies.

In Chapter 3, we proposed a simple formal model of AOP **A-LTS**. We compared the expressive power of A-LTS with FSM and PDA under language equivalence, bisimulation, and isomorphism. As a result, we showed the relationship among a few subclasses of context-free languages and the class of the languages of A-LTSs, shown in Figure 3.12, and FSM $\subsetneqq$ A-LTS $\subsetneqq$ PDA under language equivalence, bisimulation, and isomorphism. Finally, we stated the relationship

between pointcuts in A-LTS and in AspectJ, one typical AOP language. As future works, we will compare the expressive power of A-LTS with other models; e.g., context free processes [35] and recursive state machines (RSM) [4, 2]. Moreover, we will discuss a verification of A-LTS using model checking proposed by [4, 2], because we conjecture that A-LTS is a subclass of RSM. Benedikt et al. [4] discussed the complexity of the verification of a few subclass of RSM. Following their results, we will try to find an upper and a lower bound of the complexity of the verification of A-LTS. Finally, since A-LTS only models single-threaded programs, we will extend it to multithreaded programs. If we model multithreaded programs as a set of PDAs that can communicate their states to one another, then it can simulate Turing machines, i.e., it is too powerful and most problems for the model are undecidable. However, it is not known whether a set of A-LTSs that can communicate their state to one another can simulate Turing machines. On the other hand, if we model a multithreaded program as a set of PDAs that do not communicate with one another, then its language is a shuffle of CFLs, which is not a CFL in general but most decision problems for it is still decidable. The language of a set of A-LTSs that do not communicate with one another is also not a CFL in general. Further investigation into the expressive power of a set of A-LTSs that do or do not communicate with one another is needed.

References

[1] M. Abadi and C. Fournet: Access control based on execution history, Network & Distributed System Security Symposium, 107–121, 2003.

[2] R. Alur, K. Etessami and M. Yannakakis: Analysis of recursive state machines, 13th Conference on Computer Aided Verification (CAV 2001), LNCS 2102, 207–220, 2001.

[3] AspectJ Team, http://aspectj.org/

[4] M. Benedikt, P. Godefroid and T. Reps: Model checking of unrestricted hierarchical state machine, 28th International Colloquium on Automata, Languages and Programming (ICALP 2001), LNCS 2076, 652–666, 2001.

[5] E. Bertino, S. Castano, E. Ferrari and M. Mesiti: Author-X: A Java-based system for XML data protection, IFIP WG 11.3 Working Conf on Database Security, 2000.

[6] S. Boag, D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie and J. Simeon: XQuery 1.0: An XML query language. W3C working draft 16 august 2002, http://www.w3.org/TR/xquery/, 2002.

[7] J. Clark and S. DeRose: XML Path Language (XPath) version 1.0. W3C Recommendation, http://www.w3.org/TR/xpath, 1999.

[8] E. M. Clarke, O. Grumberg and D. Peled: *Model Checking,* MIT Press, 2000.

[9] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, D. S. Tison and M. Tommasi: Tree automata techniques and applications, http://www.grappa.univ-lille3.fr/tata, 1997.

[10] E. Damiani, S. D. C. di Vimercati, S. Paraboschi and P. Samarati: Securing XML documents, 7th International Conference on Extending Database Technology (EDBT 2000), LNCS 1777, Springer-Verlag, 121–135, 2000.

[11] R. Douence, O. Motelet and M. Sudholt: A formal definition of crosscuts, The 3rd International Conference on Metalevel Architectures and Separation of Crosscutting Concerns (REFLECTION 2001), LNCS 2192, 170–186, 2001.

[12] J. Esparza, D. Hansel, P. Rossmanith and S. Schwoon: Efficient algorithms for model-checking pushdown systems, Computer Aided Verification, 12th International Conference (CAV2000), LNCS 1855, Springer-Verlag, 232–247, 2000.

[13] P. W. Fong: Access control by tracking shallow execution history, IEEE Security & Privacy, pp.43–55, 2004.

[14] G. Gottlob, C. Koch, R. Pichler and L. Segoufin: The complexity of XPath query evaluation and XML typing, JACM, 52(2), 284–335, 2005.

[15] H. Hosoya and B. C. Pierce: XDuce: A typed XML processing language, ACM Transactions on Internet Technology, 3(2), 117–148, 2003.

[16] M. Koch, L. Mancini and F. Parisi-Presicce: Conflict detection and resolution in access control policy specifications, Foundation of Software Science and Computation Structures (FOSSACS 2002), LNCS 2303, Springer-Verlag, 223–237, 2002.

[17] M. Kudo and S. Harada: XML document security based on provisional authorization, 7th ACM Conference on Computer and Communication Security (CCS 2000), ACM Press, 87–96, 2000.

[18] P. Linz: *An Introduction to Formal Languages and Automata*, Jones and Bartlett Publishers, 196–198, 2001.

[19] E. C. Lupu and M. Sloman: Conflicts in policy-based distributed systems management, IEEE Transactions on Software Engineering, 25(6), 852–869, 1999.

[20] W. Martens and F. Neven: Frontiers of tractability for typechecking simple XML transformations, Proceedings of the 23rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database System (PODS 2004), ACM Press, 23–34, 2004.

[21] R. Mayr: Process rewrite system, Information & Computation, 156, 264–286, 1999.

[22] T. Milo, D. Suciu and V. Vianu: Typechecking for XML transformers, Proceedings of the 20th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database System (PODS 2000), ACM Press, 11–22, 2000.

[23] M. Mukund: From global specifications to distributed implementations, in B. Caillaud et al., editors, Synthesis and Control of Discrete Event Systems, Kluwer Academic Publishers, 19–35, 2002.

[24] M. Murata, D. Lee and M. Mani: Taxonomy of XML schema languages using formal language theory, ACM Transactions on Internet Technology, 5(4), 2005, http://www.cs.wpi.edu/~mmani/publications.html.

[25] M. Murata, A. Tozawa and M. Kudo: XML access control using static analysis, 10th ACM Conference on Computer and Communication Security (CCS2003), ACM Press, 73–84, 2003.

[26] F. Neven: Automata theory for XML researchers, SIGMOD Record, 31(3), 39–46, 2002.

[27] F. Neven and T. Schwentick: Query automata over finite trees, Theoretical Computer Science, 275, 633–674, 2002.

[28] S. Nakajima and T. Tamai: A proposal of aspect-oriented state diagram, IPSJ SIG Technical Report, 2005-SE-149, 1–8, 2005, in Japanese.

[29] S. Nakajima and T. Tamai: Aspect-oriented design using UML state diagram and its verification, 22nd JSSST Conference, 2005, in Japanese.

[30] N. Nitta, Y. Takata and H. Seki: An efficient security verification method for programs with stack inspection, 8th ACM Conference on Computer and Communication Security (CCS2001), ACM Press, 68–77, 2001.

[31] N. W. Paton and O. Diaz: Active database systems, ACM Computing Surveys, 31(1), 63–103, 1999.

[32] F. B. Schneider: Enforceable security policies, ACM Transactions on Information & System Security, 3(1), 30–50, 2000.

[33] Y. Tanabe, T. Takai and K. Takahashi: A survey of verification tools using abstraction, Technical Report, PS-2003-007, Research Center for Verification and Semantics, National Institute of Advanced Industrial Science and Technology, Dec., 2003, in Japanese.

[34] W. Thomas: *Languages, Automata, and Logic*, *Handbook of Formal Languages,* 3, Springer-Verlag, 389–455, 1997.

[35] I. Walukiewicz: Pushdown processes: games and model checking, 8th Conference on Computer Aided Verification (CAV '96), LNCS 1102, 62–74, 1996.

[36] I. Yagi, Y. Takata and H. Seki: A static analysis using tree automata for XML access control, 3rd International Symposium on Automated Technology for Verification and Analysis (ATVA2005), LNCS 3707, Springer-Verlag, 234–247, 2005.