Modal Logic Foundations of Markup Structures in Annotation Systems

Marcus Kracht
Fakultät LiLi
Postfach 10 01 31
33501 Bielefeld
Germany

January 21, 2011

Abstract

In this paper I explain how modal logic is used to talk about structured documents and how this relates to markup languages, in particular XML. It will be seen that there is a tight connection between XPath and dynamic logic over ordered trees. This connection allows to get a good insight into the semantics and complexity of XPath.

1 Introduction

Markup structures have established themselves as a quasi universal tool for storing and sharing data. Deriving ultimately from attribute value systems, they have become very powerful through the use of recursive embedding. The most known format is perhaps XML, but similar formats have been used before that. What is more, typed feature structures—known mainly from computational linguistics—are quite similar (see [4] and [5], for an introduction to XML see [18]).

Formal work on markup languages has focused on the computational behaviour as well as the expressive power. On the one hand, one wants to allow for a rich query language, on the other one would like to guarantee reasonably fast algorithms considering the magnitude of the data that is being searched. The relationship between expressiveness of a language and its computational complexity is precisely the field of finite model theory ([8]). Consequently, the quickest way to establish results is to measure the strength of queries into XML documents against languages studied in finite model theory. The queries are formulated in a special language of the XML family called XPath. XPath is a language that allows to define and use

relations in a document. XPath is most directly connected with modal logic (see [2]). Research has therefore proceeded by comparing XPath with various variants of modal logic.

There is a very close connection between markup and linguistic structures. For the idea of using modal logic for analysing semistructured data derives from the research on the model theory of syntactic and phonological structures (see [12]). This was part of a research agenda now known as model theoretic syntax (MTS). Similar to finite model theory, MTS uses logical languages to describe the model structures of linguistic theories. There are two plurals here: "logical languages" means that there are alternatives. Indeed, [19] has used (weak) monadic second order logic to do this. [17] has used predicate logic with an added transitive closure operator. Similarly, "linguistic theories" means that there are several theories, not just one. Indeed, not only is there a plethora of linguistic theories, it should also be said that there are numerous other languages, each defining a different set of constraints on structures.

The research into model theoretic syntax was mainly an exercise in formalising linguistic theories. It has emerged, though, that there is a mutual benefit for markup languages. There is now quite an active research area of query languages, where techniques of finite model theory are being used to determine the strength and tractability of query languages. The results can be applied almost directly also to implementations of MTS.

The present paper serves as an introduction into the particular perspective of modal logic on markup structures, or semistructured data. Its main purpose is to explain the background and not to give a comprehensive overview over the literature and the results of the research.

Many thanks to Richard Schröder for helping me piecing the files together after some data crash and reading earlier drafts. I also thank two anonymous referees for carefully reading this paper and making many helpful suggestions. Remaining errors and omissions are solely my own responsibility.

2 Some Elements of Modal Logic

Propositional modal logic is defined as follows. The set of symbols of the language consists of

1. a set Var of propositional variables (typically Var = $\{p_i : i \in \mathbb{N}\}$);
2. a set Con of propositional constants;
3. a set MOp of modalities;

4. and the boolean connectives $\top$, $\bot$, $\neg$, $\wedge$, $\vee$, $\rightarrow$.

So, we basically distinguish modalities from modal operators. In standard terminology, when μ is a modality, the construct $[\mu]$ is a *modal operator*. The terminology here makes the notation more user friendly and aligns modal logic with dynamic logic (see the end of this section). Propositions are formed as follows.

1. Variables and constants are propositions.
2. $\top$ and $\bot$ are propositions.
3. If φ is a proposition and μ a modality then $([\mu]\varphi)$ is a proposition as well.
4. If φ and χ are propositions, so are $(\neg\varphi)$, $(\varphi \wedge \chi)$, $(\varphi \vee \chi)$ and $(\varphi \rightarrow \chi)$.

There is another modal operator associated with the modality μ, namely $\langle\mu\rangle$. It can be defined as follows.

$$\langle\mu\rangle\varphi := (\neg([\mu](\neg\varphi))) \tag{1}$$

Brackets are dropped when no confusion arises. Conjunction binds stronger than disjunction and implication. Sequences of unary operators need not be disrupted by brackets. There is another notational tool that I shall borrow from dynamic logic, namely sequencing.

$$\begin{array}{ll} [\mu_1;\mu_2;\cdots;\mu_n]\varphi & := [\mu_1][\mu_2]\cdots[\mu_n]\varphi \\ \langle\mu_1;\mu_2;\cdots;\mu_n\rangle\varphi & := \langle\mu_1\rangle\langle\mu_2\rangle\cdots\langle\mu_n\rangle\varphi \end{array} \tag{2}$$

Propositions are evaluated in so-called *pointed frames*. A *frame* is a triple $\langle W, R, U\rangle$, where W is a set (members of which are called *worlds*), R is a function assigning each modality a binary relation on W, and U is a function assigning to every constant a subset of W. A *pointed frame* is a quadruple $\langle W, R, U, w\rangle$ where $\langle W, R, U\rangle$ is a frame and $w \in W$. A *valuation* is a function $\beta : \mathrm{Var} \rightarrow \wp(W)$. A *model* is a quintuple $\langle W, R, U, \beta, w\rangle$ such that $\langle W, R, U, w\rangle$ is a pointed frame and β a valuation into it.

In the following definition p ranges over variables, c over constants and μ over modalities.

$$
\begin{array}{lll}
\langle W,R,U,\beta,w\rangle \vDash p & :\Leftrightarrow & w \in \beta(p) \\
\langle W,R,U,\beta,w\rangle \vDash c & :\Leftrightarrow & w \in U(c) \\
\langle W,R,U,\beta,w\rangle \vDash \top & :\Leftrightarrow & \text{true} \\
\langle W,R,U,\beta,w\rangle \vDash \bot & :\Leftrightarrow & \text{false} \\
\langle W,R,U,\beta,w\rangle \vDash (\neg\varphi) & :\Leftrightarrow & \langle W,R,U,\beta,w\rangle \nvDash \varphi \\
\langle W,R,U,\beta,w\rangle \vDash (\varphi \wedge \chi) & :\Leftrightarrow & \langle W,R,U,\beta,w\rangle \vDash \varphi \\
 & & \text{and } \langle W,R,U,\beta,w\rangle \vDash \chi \\
\langle W,R,U,\beta,w\rangle \vDash (\varphi \vee \chi) & :\Leftrightarrow & \langle W,R,U,\beta,w\rangle \vDash \varphi \\
 & & \text{or } \langle W,R,U,\beta,w\rangle \vDash \chi \\
\langle W,R,U,\beta,w\rangle \vDash (\varphi \to \chi) & :\Leftrightarrow & \langle W,R,U,\beta,w\rangle \nvDash \varphi \\
 & & \text{or } \langle W,R,U,\beta,w\rangle \vDash \chi \\
\langle W,R,U,\beta,w\rangle \vDash ([\mu]\varphi) & :\Leftrightarrow & \text{for all } w' \text{: if } w\ R(\mu)\ w' \text{ then} \\
 & & \langle W,R,U,\beta,w'\rangle \vDash \varphi
\end{array}
\tag{3}
$$

It is an easy exercise left to the reader to check that

$$
\langle W,R,U,\beta,w\rangle \vDash (\langle\mu\rangle\varphi) \Leftrightarrow \text{there is } w' \text{: } w\ R(\mu)\ w' \text{ and } \langle W,R,U,\beta,w'\rangle \vDash \varphi \tag{4}
$$

(Since $\langle\mu\rangle\varphi$ is an abbreviation, this is not a definition. Rather, it now follows from the above definitions.) Note that only in the case of modalities does the world at which we evaluate get changed. In XPath terminology, we change the *focus* (see [10]).

Example 1. Linear Structures. Let the language L_ℓ be defined by Var := $\{p_i : i \in \mathbb{N}\}$, Con := $\varnothing$, and MOp := $\{\to, \leftarrow, \to^*, \leftarrow^*\}$. Now let $\langle W,R,U\rangle$ be such that

1. W is finite.
2. $R(\to^*)$ is the reflexive and transitive closure of $R(\to)$.
3. $R(\leftarrow^*)$ is the reflexive and transitive closure of $R(\leftarrow)$.
4. $R(\leftarrow)$ is the converse of $R(\to)$.
5. For all $v, w \in W$: either $w\ R(\to^*)\ v$ or $v\ R(\to^*)\ w$.
6. If $w\ R(\to^*)\ v$ and $v\ R(\to^*)\ w$ then $w = v$.

Recall that H is the transitive closure of $K \subseteq W \times W$ if and only if it is the smallest set that is transitive and contains K. The reflexive and transitive closure of K,

denoted by K^*, is the transitive closure of $K \cup \{\langle w, w\rangle : w \in W\}$. Also, $K^\smile := \{\langle y, x\rangle : \langle x, y\rangle \in K\}$ is called the *converse* of K. It is easy to see that $(K^\smile)^* = (K^*)^\smile$.

The above conditions on the frames say that $\langle W, R(\rightarrow^*)\rangle$ is a finite linear order. We may think, for example, of a file as an instance of such a structure. (The frame here supplies only the positions; I shall discuss below where the actual symbols of the file come in.)

Example 2. Ordered Trees. Let the language L_t be defined by Var := $\{p_i : i \in \mathbb{N}\}$, Con := $\varnothing$, and MOp := $\{\uparrow, \downarrow, \rightarrow, \leftarrow, \uparrow^*, \downarrow^*, \rightarrow^*, \leftarrow^*\}$. Now let $\langle W, R, U\rangle$ be such that

1. W is finite.
2. $R(\uparrow^*)$ ($R(\downarrow^*)$, $R(\rightarrow^*)$, $R(\leftarrow^*)$) is the reflexive and transitive closure of $R(\uparrow)$ ($R(\downarrow)$, $R(\rightarrow)$, $R(\leftarrow)$).
3. $R(\uparrow)$ is the converse of $R(\downarrow)$; $R(\leftarrow)$ is the converse of $R(\rightarrow)$.
4. There is exactly one w such that for all v: $w\ R(\downarrow^*)\ v$. (This is called the *root*.)
5. For all v, the set of w such that $w\ R(\downarrow^*)\ v$ is linearly ordered by $R(\downarrow^*)$.
6. For all v, the set of w such that $w\ R(\rightarrow^*)\ v$ or $w\ R(\leftarrow^*)\ v$ is linearly ordered by $R(\rightarrow^*)$.

This structure is therefore defined by only two of the eight relations, namely the 'vertical' $R(\downarrow)$ and the 'horizontal' $R(\rightarrow)$. For $R(\uparrow)$ is the converse of $R(\downarrow)$, and $R(\leftarrow)$ the converse of $R(\rightarrow)$; and the other relations are reflexive and transitive closures of these four.

Such a structure is easily recognised as an *ordered tree*. It will play a fundamental role in what is to follow. It is a tree because of the vertical relation $R(\downarrow)$ satisfies the typical properties of immediate dominance; it is ordered since the daughters of each node (in the tree sense) are linearly ordered with respect to each other.

We conclude this section with some remarks on PDL. Propositional Dynamic Logic or PDL, presents a substantial strengthening of modal logic. In PDL, modalities are called *programs*. As before there is a fixed set MOp of programs; however, programs can now be combined.

1. If α and β are programs, so are α^*, $\alpha;\beta$ and $\alpha \cup \beta$.
2. If φ is a proposition, $\varphi?$ is a program.

We extend the function R as follows.

$$
\begin{array}{ll}
R(\alpha;\beta) & := R(\alpha) \circ R(\beta) \\
R(\alpha \cup \beta) & := R(\alpha) \cup R(\beta) \\
R(\alpha^*) & := R(\alpha)^* \\
R(\varphi?) & := \{\langle x, x\rangle : x \vDash \varphi\}
\end{array} \tag{5}
$$

In PDL with converse we also have an operator $^\smile$ on programs. Furthermore, $R(\alpha^\smile) := R(\alpha)^\smile$. I note here the following properties of the converse, which I state as identities between programs.

$$
\begin{array}{ll}
(\alpha \cup \beta)^\smile & = \alpha^\smile \cup \beta^\smile \\
(\alpha;\beta)^\smile & = \beta^\smile;\alpha^\smile \\
(\alpha^*)^\smile & = (\alpha^\smile)^* \\
(\varphi?)^\smile & = \varphi?
\end{array} \tag{6}
$$

Using PDL we can reduce the number of basic modalities as follows. $\uparrow^*$ can be defined; in fact, the notation has now become transparent in the right way. Now we have only four basic programs, $\uparrow$, $\downarrow$, $\rightarrow$, and $\leftarrow$. With converse, just two of them suffice.

3 Classes of Models

Given a proposition and a model we can evaluate the proposition and see whether it is true; this is known as *model checking*, since we are checking the proposition in the model. Given a proposition, we can also ask which frames allow no countermodel for it. This is used in axiomatising classes of structures. First a few definitions. We write $\langle W, R, U, w\rangle \vDash \varphi$ if for all valuations β: $\langle W, R, U, \beta, w\rangle \vDash \varphi$. In this way, a pointed frame satisfies a formula if it satisfies the formula for all valuations; with propositional quantifiers (which we do not use) we may write this as $\langle W, R, U, w\rangle \vDash (\forall \overline{p})\varphi$, where $\overline{p}$ collects all variables occurring in φ. We write $\langle W, R, U\rangle \vDash \varphi$ if for all $w \in W$: $\langle W, R, U, w\rangle \vDash \varphi$.

Definition 1 (Axiomatisable Classes) *Let $\mathcal{K}$ be a class of pointed frames. $\mathcal{K}$ is called **axiomatisable** if there is a set Δ of formulae such that $\langle W, R, U, w\rangle \in \mathcal{K}$ iff $\langle W, R, U, w\rangle \vDash \delta$ for all $\delta \in \Delta$.*

Similarly for classes of frames. I shall present here a few positive and negative results. First, let us note the following. Given w, let $\mathrm{Tr}(w)$ denote the set of worlds that can be reached from w using any of the relations. Formally, let S be the reflexive and transitive closure of the union of all the $R(\mu)$. Then

$$\mathrm{Tr}(w) := \{v : w\ S\ v\} \tag{7}$$

Furthermore, let $R'(\mu) := R(\mu) \cap S$ and $U'(c) := U(c) \cap \mathrm{Tr}(w)$. Then $\langle \mathrm{Tr}(w), R', U', w \rangle$ is called the *subframe generated by w*. Given a valuation β into W, we put $\beta'(p) := \beta(p) \cap \mathrm{Tr}(w)$. Then

(8) $\langle W, R, U, \beta, w \rangle \vDash \varphi$ iff $\langle \mathrm{Tr}(w), R', U', \beta', w \rangle \vDash \varphi$

If β' is a valuation on the generated subframe then we may put $\beta(p) := \beta'(p)$. In that case, (8) also holds.

Definition 2 *$\langle W, R, U, w \rangle$ is said to be **generated** if $W = \mathrm{Tr}(w)$. $\langle W, R, U \rangle$ is said to be **1-generated** if there is a w such that $W = \mathrm{Tr}(w)$.*

It then follows that if $\mathcal{K}$ and $\mathcal{L}$ are axiomatisable classes of frames whose 1-generated members are identical then the two classes are identical. Similarly, two axiomatisable classes of pointed frames are identical if only their generated members are the same. In view of this, it is perhaps better to axiomatise just classes of generated pointed frames (1-generated frames).

It turns out that the classes of finite linear frames is axiomatisable (as a class of 1-generated frames).

Theorem 3 *The following holds.*

1. $\langle W, R, U \rangle \vDash \langle \mu \rangle p_0 \to \langle \nu \rangle p_0$ *iff* $R(\mu) \subseteq R(\nu)$.
2. $\langle W, R, U \rangle \vDash \langle \mu; \mu \rangle p_0 \to \langle \mu \rangle p_0$ *iff* $R(\mu)$ *is transitive.*
3. $\langle W, R, U \rangle \vDash p_0 \to [\mu]\langle \nu \rangle p_0$ *iff* $R(\mu) \subseteq R(\nu)^\smile$.
4. $\langle W, R, U \rangle \vDash [\nu](p_0 \to [\mu]p_0) \wedge p_0 \to [\nu]p_0$ *iff* $R(\mu)^* \supseteq R(\nu)$.
5. $\langle W, R, U \rangle \vDash \langle \mu \rangle p_0 \to [\mu]p_0$ *iff* $R(\mu)$ *is a partial function.*
6. $\langle W, R, U \rangle \vDash [\mu]([\mu]p_0 \to p_0) \to [\mu]p_0$ *iff* $R(\mu)$ *is transitive and conversely well-founded.*

I show the second claim. Suppose that $\langle W, R, U \rangle \vDash \langle \mu; \mu \rangle p_0 \to \langle \mu \rangle p_0$. Now let $x\ R(\mu)\ y\ R(\mu)\ z$. Pick β such that $\beta(p_0) := \{z\}$. Then $\langle W, R, U, \beta, x \rangle \vDash \langle \mu; \mu \rangle p_0$. Hence, by assumption, $\langle W, R, U, \beta, x \rangle \vDash \langle \mu \rangle p_0$. So there is a u such that $x\ R(\mu)\ u$ and $\langle W, R, U, \beta, u \rangle \vDash p_0$. By choice of β this means $u = z$, and so $x\ R(\mu)\ z$. Conversely, suppose that $R(\mu)$ is transitive. Pick x and β such that $\langle W, R, U, \beta, x \rangle \vDash \langle \mu; \mu \rangle p_0$. Then there are y and z such that $x\ R(\mu)\ y\ R(\mu)\ z$ and $\langle W, R, U, \beta, z \rangle \vDash p_0$. $R(\mu)$ is transitive, and therefore $x\ R(\mu)\ z$, which means that $\langle W, R, U, \beta, x \rangle \vDash \langle \mu \rangle p_0$.

For a somewhat more difficult case, I turn to (6). Rather than proving the entire claim, let me show that the formula is valid on a transitive, conversely well-founded

frame $\langle W, R, U\rangle$. To that end, let P_0 be all the points that have no R-successor. Inductively, define P_α to be the set of all points w such that all successors are in P_β for some some $\beta < \alpha$ and for every $\beta < \alpha$ there is some $u \in P_\beta$ which is a successor of w. (This definition is over all ordinals, it does not require the frame to be finite.) By ordinal induction it is shown that $\langle W, R, U, \beta, w\rangle \models [\mu]([\mu]p_0 \to p_0) \to [\mu]p_0$ for every $w \in P_\alpha$. To that end, assume that the claim has been shown for all $\beta < \alpha$. Pick $w \in P_\alpha$. Assume that $\langle W, R, U, \beta, w\rangle \models [\mu]([\mu]p_0 \to p_0)$. We need to show that $\langle W, R, U, \beta, w\rangle \models [\mu]p_0$. To that end we pick a successor u. It is in P_β for some $\beta < \alpha$. By assumption on w, we have $\langle W, R, U, \beta, u\rangle \models [\mu]p_0 \to p_0$. By transitivity, we also have $\langle W, R, U, \beta, u\rangle \models [\mu]([\mu]p_0 \to p_0)$. Finally, by inductive hypothesis we have $\langle W, R, U, \beta, u\rangle \models [\mu]([\mu]p_0 \to p_0) \to [\mu]p_0$. This gives $\langle W, R, U, \beta, u\rangle \models [\mu]p_0$, and finally $\langle W, R, U, \beta, u\rangle \models p_0$. u has been arbitrary. This shows the claim. (Notice that we do not need to prove the case $\alpha = 0$ separately. It is however easy to see directly that the claim holds in that case.)

Corollary 4 *The following holds.*

① *The class of linear orders is axiomatisable in L_ℓ (as a class of 1-generated frames).*

② *The class of (ordered) trees is axiomatisable in L_t (as a class of 1-generated frames).*

Let me also say a few words about the relationship between frames and pointed frames. We say that ξ is a *master modality* in $\langle W, R, U\rangle$ if $R(\xi)$ is reflexive, transitive, and for all $\mu \in \mathrm{MOp}$, $R(\mu) \subseteq R(\xi)$.

Proposition 5 *Let $\langle W, R, U, w\rangle$ be generated and assume that ξ is a master modality. Then $\langle W, R, U, \beta, w\rangle \models [\xi]\varphi$ iff $\langle W, R, U, \beta\rangle \models \varphi$. Also, $\langle W, R, U, w\rangle \models [\xi]\varphi$ iff $\langle W, R, U\rangle \models \varphi$.*

It is possible to axiomatise the class of frames where a given modality is guaranteed to be a master modality. In linear frames, there is no such master modality. However, it turns out that we have something that is effectively the same. Namely, if both $R(\to^*)$ and $R(\leftarrow^*)$ are linear and each others converse, then for every x, y there is a z such that: $x\ R(\to^*)\ z\ R(\leftarrow^*)\ y$. Thus, in place of the above we have

(9) $\quad \langle W, R, U, \beta, w\rangle \models [\to^*; \leftarrow^*]\varphi \ \Leftrightarrow\ \langle W, R, U, \beta\rangle \models \varphi$

4 Modal Logic and DOMs

Now we turn to more realistic models. First of all, we like to define a realistic theory of a file. To this end, all we need to do is to take the language of our first

example and supplement it with constants. There are many ways to go. Given our alphabet A of letters, we can define a constant $\underline{a}$ for each letter of A. (The alphabet is usually assumed to be finite.) Then we add the axioms $\underline{a} \to \neg\underline{b}$ for all $a, b \in A$ such that $a \neq b$; moreover we shall add $\bigvee\langle \underline{a} : a \in A\rangle$. In raw (that is, binary) format, we can take A to be just $\{0, 1\}$. The worlds are then the positions of the individual bits. But different structures can be used (say, with a constant for each alphabetical symbol of Unicode).

A given file is therefore a 1-generated frame for this logic. It has the form $\langle W, R, U\rangle$, where W is a finite set, and $R(\to)$ is a partial function with inverse $R(\leftarrow)$. Since the frame is 1-generated, every point is connected with every other, which means that we have a finite linear order in the standard sense. The worlds can thus be identified with an initial segment of the natural numbers. The function U assigns to each constant $\underline{a}$, with $a \in A$, a set $U(\underline{a})$. If $i \in U(\underline{a})$ the file is said to carry the letter a at position i. The axiom $\underline{a} \to \neg\underline{b}$ guarantees that every position carries exactly one letter.

Consider a generated pointed frame $\langle W, R, U, i\rangle$. Here, the members of W are numbers, and i is a particular number. This is equivalent to a file plus cursor position. The commands that move the cursor to the right and left can be interpreted as changing the world i to $i + 1$ and $i - 1$, respectively. More complex motions of the cursor can be defined, though the present language may be too limited for that. Without the modalities, we can only determine what symbol is present at the current position; the modalities present something of a lookahead. For example, $\langle\to\rangle\underline{a}$ is true at i if $i + 1$ carries the letter a, equivalently, if $\underline{a}$ is true at $i + 1$.

Let me now turn to XML. For readers unfamiliar with this format, I advise to get hold of a book on XML, for example [18]. An XML-document is a file, that is, in first instance a string. The data in the file combines both the primary data itself and the metadata in the form of annotation tags. The tags may carry any additional information about this data. But first and foremost they serve to structure it. They turn it into a *tree*.

(10)
```
<library>
    <book>
        <author>Hugo</author>
        <title>Les Misérables</title>
    </book>
    <book>
        <author>Flaubert</author>
        <title>Madame Bovary</title>
    </book>
</library>
```

For each opening tag, say <author> there must be a corresponding closing tag, </author>. In principle, tags can be inserted anywhere as long as they are properly nested. However, from a theoretical point of view it is best to require that text can only be inserted between deepest embedded tags. (This is the default in XSL Schema, by the way.) A slight modification of the structure is enough to get this form. Say we have the following line.

(11) This was an inspiring discussion.

This line is then transformed as follows, where <text> is a tag reserved for text input.

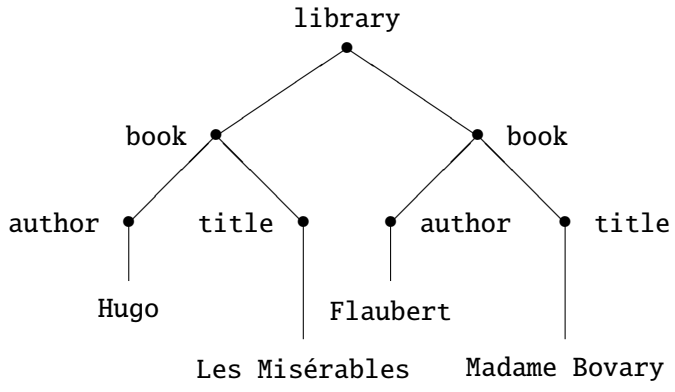
```
      <p><text>This was an</text>
(12)      <i><text>inspiring</text></i>
          <text>discussion.</text></p>
```

This is the form we shall assume here. In XML talk, we disallow the mixed type.

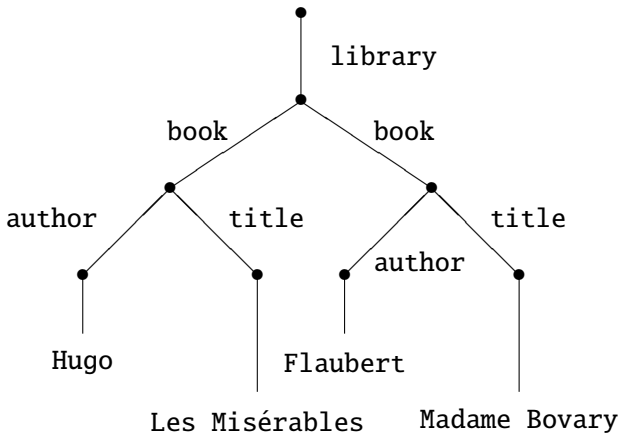
The linear structure (10) is converted into a tree structure in the following way. A tag <τ> together with the next corresponding closing tag </τ> define a constituent of type τ. In this way, a linguistic representation of the above structure might look like this:

(13) $[_{\texttt{library}}[_{\texttt{book}}[_{\texttt{author}}\texttt{Hugo}][_{\texttt{title}}\texttt{Les Misérables}]]$
$[_{\texttt{book}}[_{\texttt{author}}\texttt{Flaubert}][_{\texttt{title}}\texttt{Madame Bovary}]]]$

However, this is not entirely adequate. For disregarding order the structure is a node labelled directed graph. These are triples $\langle N, E, \ell \rangle$, where $E \subseteq N \times N$ and $\ell : N \to L$, where L is a labelling domain.



However, in semistructured data we like to think of these structures as edge labelled graphs ([1]). These are triples $\langle N, E, \ell \rangle$, where $E \subseteq N \times N$ and $\ell : E \to L$. These graphs are called the **DOM**s (document object models).



This necessitates the introduction of a root node since we need an edge with label `library`. Now, in this structure the tags correspond to edge labels. This does not apply to the attributes, though. Furthermore, types and content are properties of the nodes, not the edges. Thus the following tag represents a mixture of edge and node labels:

(14)

<	author	ID="VH07"	nationality="French"	>
	↑	↑	↑	
	edge label	node label	node label	

In fact, as noted in [1], the notation is not fully transparent with respect to the structure that it represents. First, `ID` and `IDREF` should be treated separately; they deal with so-called oids (object identifiers). Attributes on the other hand should also be seen as edge labels, so that they are basically equivalent to tags. However, one difference remains: attributes are not recursive; and they are not ordered. I remark briefly that in a tree there is a simple correspondence between edge labels and labels on nodes other than the root. Observe that every node has a unique parent, so if we make the edge label a label of the endpoint of the edge, the edge labelling can be recovered except of the root node. The document root however is added on top of the root node of the tree; and this makes the correspondence exact. Also note that in practice, XPath seems to treat DOMs rather as node labeled trees.

I shall first present a formalism for fixed tag sets (for example, HTML). In the next section I shall return to XML. The tags are at present undecorated (no attributes, no identifiers). Hence, for each tag τ we take a separate modal operator with the same name. So, we have, in the case of HTML, modal operators `h1`, `p`, and so on. The following is assumed: the union over all $R(\tau)$, where τ is a tag, corresponds to the relation $R(\downarrow)$. Second, for different tags τ and τ', $R(\tau) \cap R(\tau') = \varnothing$ (though, surprisingly, this is not modally axiomatisable unless all daughters of a node are linearly ordered with respect to each other). Third, $x\ R(\rightarrow)\ y$ only if there is z and a tag τ such that $z\ R(\tau)\ x, y$. This is a known situation: only the siblings are ordered with respect to each other. (Thus $R(\rightarrow)$ is the sibling ordering, not the linear precedence in the file defining the model.) A different proposal is to assume that $x\ R(\rightarrow)\ y$ if x and y have ancestors that are neighbouring siblings. (So, using the previous definition, we take the ordering to be $R(\uparrow^*) \circ R(\rightarrow) \circ R(\downarrow^*)$.) The sibling order is derived from the linear precedence in the XML-document (the file), which specifies a linear order on the entire set of nodes.

5 XPath

In XML the tagset is not fixed. Yet we still can encode XML structures with a finite set of modalities. The trick is as follows. We return to the language of ordered trees. The edge label no longer defines a modality in its own right. Rather, we make the edge label a property of the node to which the edge points. This is exactly how it is done in XML. (And it is the reason why in the DOM we need an extra root node.) We introduce a new modality, τ, which relates a node with its tag. XPath has a function called `name()` to return the name of a node. However, notice that there is no limit on the number of tags, so we need to convert tags into structures as well. This we can do by representing the tag in the model as a string. Thus we arrive at a new sort of language and structures. They extend the ordered trees. The additional postulates are as follows. Say that z is a *tag node* if there are x and y such that $x\ R(\tau)\ y$, and $z \in \mathrm{Tr}(y)$. Then we add the condition that if z is a tag node, there is no y such that $z\ R(\tau)\ y$ or $z\ R(\uparrow)\ y$ or $z\ R(\downarrow)\ y$. This makes $\mathrm{Tr}(y)$ in effect a linear structure, for any tag node y. It is important to note that if z is not a tag node and $z\ R(\mu)\ y$ for $\mu \in \{\uparrow, \downarrow, \rightarrow, \leftarrow\}$, then y also is not a tag node. Thus, the only way to enter tag nodes is via the relation τ. Finally, the tag nodes are treated as coding a string, so the letters are basically introduced via constants. It must then be specified that the constants are false at ordinary tree nodes.

With this in mind we shall now turn to the analysis of XPath. XPath is a language for selecting nodes from a tree, based on various properties. These properties need not be local to the node (like the tag), mostly they involve ways in which the

node is embedded in a structure. We can define the set of nodes that have a particular parent node, for example. We can also find nodes based on their linear position in the DOM. The expressive power of XPath is therefore quite rich; too rich to receive a comprehensive logical treatment. (In fact, [21] show that adding numeric comparisons quickly leads to undecidability.) [9] have therefore proposed to define a subset, called Core XPath, where only the relational properties are studied. It is this language that we shall look at below. XPath itself nowadays has two versions: there is XPath 1.0 and XPath 2.0. Consequently, there is Core XPath 1.0, and Core XPath 2.0. The discussion below treats Core XPath 1.0, and takes only a brief look at Core XPath 2.0.

XPath contains so-called *axes*. These are relations between nodes in a tree. Here are the main axis relations:

(15)

XPath	PDL
`parent`	$\uparrow$
`ancestor-or-self`	$\uparrow^*$
`ancestor`	$\uparrow;\uparrow^*$
`child`	$\downarrow$
`descendant-or-self`	$\downarrow^*$
`descendant`	$\downarrow^*;\downarrow$
`following-sibling`	$\rightarrow^*;\rightarrow$
`following`	$\uparrow^*;\rightarrow;\rightarrow^*;\downarrow^*$
`preceding-sibling`	$\leftarrow^*;\leftarrow$
`preceding`	$\uparrow^*;\leftarrow;\leftarrow^*;\downarrow^*$

There are three more: `self`, `namespace` and `attribute`. The actual surface syntax of XPath is somewhat different. There are first of all two types of path expressions: *relative* and *absolute*. We deal with the relative pattern first. The relative pattern is composed from so-called **step patterns** by means of `/` and `//`. A step pattern in turn is a sequence consisting of (a) an axis specifier, (b) a node test, and (c) a sequence of predicates. The axis specifier says whether the node test specifies the value of an attribute or the label. The predicates are properties of nodes. They are enclosed in square brackets. These properties can be even numeric, but it is customary to restrict them to what is expressible to the relational language described here. The absolute paths are obtained from the relative paths by prefixing them with `/` or `//` (as is customary in Unix). This makes `/` and `//` both unary and binary symbols. Finally, it is important to realise that path expressions can have a short form and a long form. What appears in short as `author` is in long form `self::node()/child::author`. In the long form the symbol `/` is interpreted as relational composition while in the short form it effectively takes the meaning

"compose with child-of", as it does in Unix. (More on the precise syntax and the relationship to modal logic and relational algebras can be found in [21].)

One big problem area in the theory of markup language is fast algorithms for the path containment problem. This is the problem to determine, given two path descriptions p and q, whether or not in every tree all the p-paths are included in the q-paths. One of the reasons to be interested in this problem is in reformulating queries either to speed them up or to discover if thery are consistent ([7], [16]).

Since the full language is quite difficult to tackle one is therefore interested in fragments of it. A popular fragment is one where the horizontal axes are eliminated. One can then in effect only talk about hierarchy, not about linear order. An XP-expression is an expression generated by the following grammar, taken from [16]. It is based on the short forms and does not use upward directed axes.

$$
\begin{array}{lll}
p := & p_1 \mid p_2 & \text{(disjunction)} \\
 & \mid /p & \text{(root)} \\
 & \mid //p & \text{(descendant)} \\
 & \mid p_1/p_2 & \text{(child)} \\
 & \mid p_1//p_2 & \text{(descendant)} \\
 & \mid p_1[p_2] & \text{(filter)} \\
 & \mid \sigma & \text{(element test)} \\
 & \mid * & \text{(wildcard)}
\end{array}
\tag{16}
$$

Here, σ is a basic boolean constant. The queries are evaluated as relations in Kripke-models. Let $\mathfrak{T}$ be a tree with root r and $\langle W, R, U, \beta, w\rangle$ a Kripke-model.

$$
\begin{array}{ll}
[\![p_1 \mid p_2]\!]_{\mathfrak{T}} & = [\![p_1]\!]_{\mathfrak{T}} \cup [\![p_2]\!]_{\mathfrak{T}} \\
[\![/p]\!]_{\mathfrak{T}} & = [\![p]\!]_{\mathfrak{T}} \cap \{r\} \times W \\
[\![//p]\!]_{\mathfrak{T}} & = \{\langle r, w\rangle : \exists u : \langle u, w\rangle \in [\![p]\!]_{\mathfrak{T}}\} \\
[\![p_1/p_2]\!]_{\mathfrak{T}} & = [\![p_1]\!]_{\mathfrak{T}} \circ R(\downarrow) \circ [\![p_2]\!]_{\mathfrak{T}} \\
[\![p_1//p_2]\!]_{\mathfrak{T}} & = [\![p_1]\!]_{\mathfrak{T}} \circ R(\downarrow^+) \circ [\![p_2]\!]_{\mathfrak{T}} \\
[\![p_1[p_2]]\!]_{\mathfrak{T}} & = \{\langle v, w\rangle \in [\![p_1]\!]_{\mathfrak{T}} : \exists u : \langle w, u\rangle \in [\![p_2]\!]_{\mathfrak{T}}\} \\
[\![\sigma]\!]_{\mathfrak{T}} & = \{\langle w, w\rangle : w \in \beta(\sigma)\} \\
[\![*]\!]_{\mathfrak{T}} & = \{\langle w, w\rangle : w \in W\}
\end{array}
\tag{17}
$$

Based on work by [15], [16] establish the following. The first, standard, case is when the set L of labels is infinite:

Theorem 6 *The following holds.*

1. *Containment of XP(/, //, [], $*$, |) is in* CONP.
2. *Containment of XP(/, |) is* CONP*-hard.*

3. *Containment of XP(//, |) is* CONP*-hard.*

A problem P is in CONP if it can be verified in nondeterminstic polynomial time (= NP) whether a given structure is a counterexample to P.

Theorem 7 *Let L be finite.*

1. *Containment of XP(/, //, [], $*$, |) is in* PSPACE.
2. *Containment of XP(/, //, |) is* PSPACE*-hard.*

[16] contains many more results. Additional complexity comes from the addition of DTDs, which describe the structure of documents. The problem becomes the following: given a DTD d and two path expressions p, q say whether $[\![p]\!]_{\mathfrak{T}} \subseteq [\![q]\!]_{\mathfrak{T}}$ for all $\mathfrak{T}$ satisfying d. Effectively, DTDs are some kind of axioms. Thus adding a DTD may in fact increase the complexity of the problem.

Theorem 8 *The following holds.*

1. *Containment of XP(DTD, /, []) is in* CONP.
2. *Containment of XP(DTD, /, []) is* CONP*-hard.*
3. *Containment of XP(DTD, //, []) is* CONP*-hard.*
4. *Containment of XP(/, //, [], $*$) is* EXPTIME*-complete.*
5. *Containment of XP(/, //, |) is* EXPTIME*-complete.*

6 Paths in Dynamic Logic

Based on the results of Sections 2 and 3 we can conclude.

Theorem 9 *The class of ordered forests is axiomatisable in* PDL *over* $\uparrow$, $\downarrow$, $\rightarrow$ *and* $\leftarrow$. *The logic is denoted by* $\mathsf{PDL}_{\mathsf{t}}$.

It is possible to extend this to edge labelled forests. Just add one more operator, node, and the axioms

(18) $[\mathsf{node};\mathsf{node}]\bot, \qquad \langle\mathsf{node}\rangle p \rightarrow [\mathsf{node}]p$

This makes the interpretation of node a relation R such that if $x\ R\ y$ then y has no R-successor, and if $x\ R\ z$ then $y = z$. The value at this node is any text (the edge label). If there are only finitely many tags available, then we can mimic the edge

labels by a fixed set of boolean constants. Otherwise, we need to encode the strings over an alphabet.

We can offer an analysis of path expressions by translating them into PDL expressions. Recall that in XPath expressions are evaluated into node sets. The set contains the nodes satisfying the expression. Since the syntax of path expressions is somewhat different from PDL, we must first translate them. This must be done with care. For in the previous section we have just translated them as relations. Here we must reduce them to formulae. Given an expression `book/author` we must decide whether we look at the set of nodes where the path originates or whether we look at the set of nodes where the path ends. In a template, for example, we look at the nodes where the path ends. So, for each path we have two translations, $\{\cdot\}^o$ (looking at nodes where the paths originate) and $\{\cdot\}^g$ (looking at nodes where the paths end). These can be derived, though, from a direct translation $\{\cdot\}^\delta$ into relations, which runs as follows (based on the long form).

$$
\begin{array}{ll}
\{p_1 \mid p_2\}^\delta & := \{p_1\}^\delta \cup \{p_2\}^\delta \\
\{p_1/p_2\}^\delta & := \{p_1\}^\delta \circ \{p_2\}^\delta \\
\{p_1[p_2]\}^\delta & := \{p_1\}^\delta \circ \{\langle x, x\rangle : (\exists y)(\langle x, y\rangle \in \{p_2\}^\delta)\} \\
\{\rho :: \tau\}^\delta & := \rho \circ \tau?
\end{array} \tag{19}
$$

Now $\{\cdot\}^o$ can be obtained, translating $\circ$ by ; and $\cup$ by $\cup$:

$$
\begin{array}{ll}
\{p_1 \mid p_2\}^o & := \{p_1\}^o \cup \{p_2\}^o \\
\{p_1/p_2\}^o & := \{p_1\}^o; \{p_2\}^o \\
\{p_1[p_2]\}^o & := \{p_1\}^o; (\langle \{p_2\}^o \rangle \top)? \\
\{\rho :: \tau\}^o & := \rho; \tau?
\end{array} \tag{20}
$$

To see the rationale behind this translation note that $\langle\alpha\rangle\top$ is true at a point iff there is an α-path starting at that point. Also, $\langle\alpha;\varphi?\rangle\top$ is true iff there is an α-path ending in a node satisfying φ. With the converse operator we can now write as follows. If $x \vDash \langle\alpha\rangle\top$ selects the nodes at which α can successfully start, $\{x : x \vDash \langle\alpha^\smile\rangle\top\}$ selects the nodes at which α ends. Thus we have

$$\{\pi\}^g = (\{\pi\}^o)^\smile \tag{21}$$

Applying this to path expressions gives the following.

$$
\begin{array}{ll}
\{p_1 \mid p_2\}^g & := \{p_1\}^g \cup \{p_2\}^g \\
\{p_1/p_2\}^g & := \{p_2\}^g; \{p_1\}^g \\
\{p_1[p_2]\}^g & := \langle \{p_2\}^o \rangle \top?; \{p_1\}^g \\
\{\rho :: \tau\}^g & := \tau?; \rho^\smile
\end{array} \tag{22}
$$

The third line is noteworthy (the derivation makes use of the equations (6)).

$$
\begin{aligned}
\{p_1[p_2]\}^g &= (\{p_1[p_2]\}^o)^\smile \\
&= (\{p_1\}^o; \langle\{p_2\}^o\rangle\top?)^\smile \\
&= (\langle\{p_2\}^o\rangle\top?)^\smile; (\{p_1\}^o)^\smile \\
&= \langle\{p_2\}^o\rangle\top?; \{p_1\}^g
\end{aligned}
\tag{23}
$$

Absolute paths can be defined as follows.

$$\{/p\}^o := \neg\langle\uparrow\rangle\top?; \{p\}^o \tag{24}$$

The program $\varphi?;\alpha$ effectively restricts the set of nodes to those where φ is true.

Here is a classical result.

Theorem 10 ([22]) *The satisfiability problem for* PDL *is in* EXPTIME.

By reducing it to the original result, [2] show that the same complexity holds for the logic of trees.

Theorem 11 ([2]) *The satisfiability problem of* $\mathsf{PDL_t}$ *is in* EXPTIME.

Path inclusion can be reduced to the negation of a satisfiability problem. Namely, $\pi \not\subseteq \rho$ in *some* structure iff $[\{\pi\}^o]p \wedge \langle\{\rho\}^o\rangle\neg p$ is satisfiable in $\mathsf{PDL_t}$. Formulated differently, $\pi \subseteq \rho$ in *every* structure iff $[\{\rho\}^o]p \rightarrow [\{\pi\}^o]p \in \mathsf{PDL_t}$. According to Theorem 11 the problem is decidable in EXPTIME. Notice that the actual queries that can be issued in XPath are a proper subset of the queries that are definable in $\mathsf{PDL_t}$ ([13]).

An interesting property of $\mathsf{PDL_t}$ is that we can define nominals. Nominals are special kinds of propositional variables that may be instantiated to a single point ([3]). That is, if i is a nominal and $\langle W, R, U, \beta, w\rangle$ a model then $\beta(i) = \{v\}$ for some $v \in W$. Given a formula $\varphi(i)$ which contains such a nominal, we can replace the nominal by a standard variable p as follows. We put

$$\neq := \downarrow^+ \cup(\uparrow^*; (\rightarrow^+ \cup \leftarrow^+); \downarrow^*) \tag{25}$$

It is not hard to see that $R(\neq) = \{\langle v, w\rangle : v \neq w\}$. Then suppose that the following is true at w:

$$\langle\uparrow^*; \downarrow^*\rangle(p \wedge [\neq]\neg p) \wedge \varphi(p) \tag{26}$$

Then at some point v, $v \vDash p \wedge [\neq]\neg p$, which is to say that $\beta(p) = \{v\}$, as required. Since an added nominal can be recoded at constant expense, the complexity does not rise in $\mathsf{PDL_t}$ if we add nominals.

Formally, the argument runs as follows. Let NPDL_t be the extension of PDL_t by nominals. Then one can show that for every formula φ there is a formula $\varphi^\sharp$ such that $\varphi \in \mathsf{NPDL}_t$ iff $\varphi^\sharp \in \mathsf{PDL}_t$. The complexity of NPDL_t can be bounded from that of PDL_t using the properties of the map $\varphi \mapsto \varphi^\sharp$. Since in this case $|\varphi^\sharp| \leq c|\varphi|$ for some constant c, if PDL_t is decidable in $f(x)$ time, NPDL_t is decidable in $f(cx)$ time. The same argument can be used to show that the logic with an added constant axiom ξ has the same complexity. Here we take $\varphi \to \varphi \wedge [\uparrow^*;\downarrow^*]\xi$. The function is $f(d + x)$, where d is a constant (1 plus the length of $[\uparrow^*;\downarrow^*]\xi$. Likewise, making a program deterministic or adding the converse typically have no effect on the complexity (see [11]). Let us close this section with a quick look at Core XPath 2.0. This language extends Core XPath 1.0 by operators on paths. There are new constructs `union`, `intersect`, and `except` to form the union, intersection and difference of relations or path sets. Paths can be combined using these expressions. This exceeds the syntactic means of PDL, so [21] turn to relation algebras instead. This language is decidable, and expressively complete for first-order logic. This means that if a property of nodes is first-order definable it is also definable in Core XPath 2.0 ([14]). Since the paths definable in Core XPath 1.0 are not closed under negation ([6]), this language is therefore stronger. [21] give a complete axiomatisation of path equivalence.

7 Conclusion

A logical analysis of computer languages is important in many respects: it gives us a clear idea of what is expressible and what is not; it also gives us a clear notion of the sort of structures we are using; and third, it allows to prove precise results about the complexity of algorithms. The analysis of XML, in particular Core XPath, in terms of modal logic is a good example of this. The relational character of the models for modal logic make it very useful in studying DOMs from an abstract point of view. Also, expressive and computational properties of Core XPath can be addressed succinctly. Although quite different in detail, the two languages share a large enough core to allow for useful results. Since the model theory of PDL is well understood, results can be transfered almost immediately.

References

[1] Serge Abiteboul, Peter Bunemann, and Dan Suciu. *Data on the Web. From Relations to Semistructured Data and XML.* Morgan Kaufmann, San Francisco, 2000.

[2] Loredana Afanasiev, Patrick Blackburn, Ioanna Dimitriou, Bertrand Gaiffe, Evan Goris, Maarten Marx, and Maarten de Rijke. PDL for ordered trees. *Journal of Applied Non-Classical Logics*, 15:115–135, 2005.

[3] Patrick Blackburn. Nominal tense logic. *Notre Dame Journal of Formal Logic*, 39:56–83, 1993.

[4] Bob Carpenter. *The Logic of Typed Feature Structures*. Cambridge Tracts in Theoretical Computer Science 32. Cambridge University Press, 1992.

[5] Anne Copestake. *Implementing Typed Feature Structure Grammars*. CSLI, 2000.

[6] Maarten de Rijke and Maarten Marx. Semantic characterisation of navigational XPath. *Transactions of the ACM*, 34:41–46, 2005.

[7] A. Deutsch and V. Tannen. Containment and integrity constraints for XPath. In Maurizio Lenzerini, Daniele Nardi, Werner Nutt, and Dan Suciu, editors, *Proceedings of the 8th International Workshop on Knowledge Representation Meets Databases (KRDB 2001)*, 2001.

[8] Hans-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Perspectives in Mathematical Logic. Springer, 1995.

[9] G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. In *VLDB'02*, pages 95–102, 2002.

[10] Michael Kay. *XPath 2.0. Programmer's Reference*. Wiley Publishing, Indianapolis, 2004.

[11] Marcus Kracht. *Tools and Techniques in Modal Logic*. Number 142 in Studies in Logic. Elsevier, Amsterdam, 1999.

[12] Marcus Kracht. *Mathematics of Language*. Mouton de Gruyter, Berlin, 2003.

[13] Maarten Marx. XPath and Modal Logic of DAGs. In M. Cialdea Mayer and F. Pirri, editors, *Automated Reasoning with Analytic Tableaux and Related Methods*, number 2796 in SpringerLecture Notes in Computer Science, pages 150–164, 2003.

[14] Maarten Marx. Conditional XPath. *ACM Transactions on Database Systems*, 30:929–959, 2005.

[15] G. Miklau and Dan Suciu. Containment and equivalence for an XPath fragment. In *Proceedings of the 21st Symposium on Database Systems*, pages 65–76, 2002.

[16] Frank Neven and Thomas Schwentick. XPath containment in the presence of disjunction, DTDs and variables. In Diego Calvanese, Maurizio Lenzerini, and Rajeev Motwani, editors, *9th International Conference on Database Theory (ICDT)*, number 2572 in Springer Lectures in Computer Science, pages 312–326, 2003.

[17] Adi Palm. *Tranforming Tree Constraints into Formal Grammars. The Expressivity of Tree Languages*. PhD thesis, Universität Passau, 1997.

[18] Eric T. Ray. *Learning XML*. O'Reilly, Sebastopol, CA, 2003.

[19] James Rogers. *Studies in the Logic of Trees with Applications to Grammar Formalisms*. PhD thesis, University of Delaware, Department of Computer & Information Sciences, 1994.

[20] W. W. Stead, W. E. Hammond, and M. J. Straube. A Chartless Record–Is It Adequate? In *Proceedings of the Annual Symposium on Computer Application in Medical Care*, pages 89–94, 1982.

[21] Balder ten Cate and Maarten Marx. Axiomatizing the Logical Core of XPath 2.0. In *Database Theory - ICDT 2007*, number 4353 in SpringerLecture Notes in Computer Science, pages 134–148, 2006.

[22] Moshe Vardi and P. Wolper. Automata theoretic techniques for modal logics of programs. *Journal of Computer and Systems Sciences*, 32:183–221, 1986.