The Correspondence between Automata and Logics

Jens Kerber
Programming Systems Lab
Seminar: Logical Aspects of XML
Saarland University

22.10.2003

1 Introduction

In this paper we will show that weak monadic second order logic and automata correspond to each other. Therefore we will prove two historical theorems, one of Büchi (1960) and one of Thatcher and Wright (1968). The correspondence is important, because it offers us one opportunity to prove the decidability of this logic.

For the automata we distinguish between word-automata and tree-automata. For those readers who are not familiar with tree-automata we recommend [1].

Throughout this paper we only consider the weak monadic second order logic. That means, we only treat the case of second order variables with a finite size. As a consequence of that we only work with finite word-automata and finite tree-automata respectively.

In chapter 2 we will show how a string or a tree can be represented such that a logic and an automaton can handle it. Therefore we need characteristic functions and characteristic sets. In chapter 3 both logics that we consider here will be defined. Chapter 4 is the core of this paper. Here the two theorems, mentioned above, and their proofs are presented. In chapter 5 we have a look at the complexity. In chapter 6 we will conclude.

2 Representation

A characteristic function of a set B is a function from A to $\{0,1\}$, where A is a superset of B. It returns 1 iff the element of A is also an element of B.

$$\begin{aligned} B &\subseteq A \\ f_B &= A \to \{0,1\} \\ \forall a \in A : f_B(a) &= \begin{cases} 1, if\ a \in B \\ 0, if\ a \notin B \end{cases} \end{aligned}$$

A characteristic set is a subset of a set A that contains all elements of A for which the characteristic function returns 1.

$$\begin{aligned} S_f &\subseteq A \\ S_f &= \{a \in A | f(a) = 1\} \end{aligned}$$

What we need in this paper are characteristic sets, which are subsets of the set of positions of a tree or word, and which tell us where a given functionsymbol occurs.

Given the following string: abbac over the signature $\Sigma = \{a_0, b_0, c_0, d_0\}$. This string can now be written like a matrix:

	0	1	2	3	4	5	6	$\cdots$
f_a	1	0	0	1	0	0	0	$\cdots$
f_b	0	1	1	0	0	0	0	$\cdots$
f_c	0	0	0	0	1	0	0	$\cdots$
f_d	0	0	0	0	0	0	0	$\cdots$

An entry (i, j) is 1 iff position i is marked with functionsymbol j. In this case there is exactly one functionsymbol which occurs at each position. This will not need to be if we work with second order variables.

The given string can be uniquely identified by the characteristic sets of the functionsymbols:

$$\begin{aligned} S_{f_a} &= \{0,3\} \\ S_{f_b} &= \{1,2\} \\ S_{f_c} &= \{4\} \\ S_{f_d} &= \emptyset \end{aligned}$$

(For our use the naming of the positions is not yet correct, it will be defined later)

If a tuple-word-automaton had to read this string, it would do this column by column from left to right. In every step it would read one tuple of characteristic functions. A logic would work top-down (line-by-line) only considering the characteristic set for each functionsymbol.

3 WS1S and WS2S

Both logics as we use them here only operate over second order variables (sets of first order variables). So every first order variable x is transformed into $X = \{x\}$, which describes a second order variable of size 1. WS1S describes the weak monadic second order logic with one successor relation. WS2S is the extension to WS1S and has one additional successor relation. Monadic means, that all predicates which occur in the logic, can be reduced to those predicates which have arity 1. As an exception, the predicates which define the logic may have a higher arity.

The definition of WS1S is as follows:

$$\text{Atomic formulas: } X \subseteq Y, X = Y1$$
$$\text{Logical connections: } \neg\phi, \phi_1 \vee \phi_2, \exists X.\phi$$

The definition of WS2S is almost the same but it contains one more atomic formula

$$X = Y2$$

All logical connections can be derived from those that are given, e.g. $\wedge, \Rightarrow, \forall$.

Now we define the predicates above more precisely and build some more complex ones out of them.

$$\begin{aligned}
X \subseteq Y &\stackrel{def}{=} \forall x.(x \in X \Rightarrow x \in Y) \\
X = \{\varepsilon\} &\stackrel{def}{=} \forall Y.X \neq Yi \\
X = \bigcup_{i=1}^{n} X_i &\stackrel{def}{=} \bigwedge_{i=1}^{n} X_i \subseteq X \wedge \forall x.(x \in X \Rightarrow \bigvee_{i=1}^{n} x \in X_i) \\
X \cap Y = Z &\stackrel{def}{=} \forall x.x \in Z \Leftrightarrow (x \in X \wedge x \in Y) \\
Partition(X, X_1, \ldots, X_n) &\stackrel{def}{=} X = \bigcup_{i=1}^{n} X_i \wedge \bigwedge_{i=1}^{n-1} \bigwedge_{j=i+1}^{n} X_i \cap X_j = \emptyset \\
PrefixClosed(X) &\stackrel{def}{=} \forall z.\forall y.(z \in X \wedge y \leq z) \Rightarrow y \in X \\
X = Y &\stackrel{def}{=} X \subseteq Y \wedge Y \subseteq X \\
X = \emptyset &\stackrel{def}{=} \forall Y.(Y \subseteq X \Rightarrow Y = X) \\
Sing(X) &\stackrel{def}{=} X \neq \emptyset \wedge \forall Y.(Y \subseteq X \Rightarrow (Y = X \vee Y = \emptyset)) \\
X = Yi &\stackrel{def}{=} Sing(X) \wedge Sing(Y) \wedge \forall x \in X.\forall y \in Y.x = yi \\
x \leq y &\stackrel{def}{=} \forall X.(y \in X \wedge (\forall z.(\bigvee_{i=1}^{k} zi \in X) \Rightarrow z \in X)) \Rightarrow x \in X
\end{aligned}$$

$x = yi$ means that x is the i-th successor of y. Therefore the positions that were named $0, 1, 2, 3, \ldots$ in the example in section 2 have to be renamed to $\varepsilon, 1, 11, 111, \ldots$. Furthermore ε indicates the root of a tree or word. Partition() is true iff X is the pairwise disjoint finite union of $X_1, \ldots, X_n$. Sing() returns true iff it has a set with exactly one element as input. Finally $x \leq y$ is the ordering for first order variables.

With the predicates mentioned above we are able to code finite words or trees in the following way:

$$
\begin{aligned}
Term(X, X_1, \ldots, X_n) = \\
& X \neq \emptyset \\
\wedge \quad & Partition(X, X_1, \ldots, X_n) \\
\wedge \quad & PrefixClosed(X) \\
\wedge \quad & \bigwedge_{i=1}^{k} \bigwedge_{arity(f_j)=i} \Big(\bigwedge_{l=1}^{i} \forall x.(x \in X_{f_j} \Rightarrow xl \in X) \\
& \wedge \bigwedge_{l=i+1}^{k} \forall y.(y \in X_{f_j} \Rightarrow yl \notin X) \Big)
\end{aligned}
$$

where X is a set of positions describing the shape of the tree or word, and X_i is the characteristic set of functionsymbol i. The shape must not be empty and must not contain any holes (prefixclosed). Finally each node has to be arity-consistent.

Here is one example coding for a tree:

Let `or(not(0),not(1))` be a tree over the signature $\Sigma = \{or_2, not_1, 0_0, 1_0, \exists_2\}$ then

$$
\begin{aligned}
S &= \{\varepsilon, 1, 2, 11, 21\} \\
S_{or} &= \{\varepsilon\} \\
S_{not} &= \{1, 2\} \\
S_0 &= \{11\} \\
S_1 &= \{21\} \\
S_\exists &= \emptyset
\end{aligned}
$$

The corresponding coding looks like this: $(S, S_{or}, S_{not}, S_0, S_1, S_\exists)$

4 The Correspondence

In 1960 Büchi proved the following

Theorem: WS1S is as expressive as finite word-automata.

In 1968 Thatcher and Wright found out that there is an analogous correspondence for the extended case.

Theorem: WS2S is as expressive as finite tree-automata.

The proof of both theorems works in two directions. First, we have to show that it is possible to create a WS2S-(WS1S)formula, which simulates a successful run of a tree-(word)automaton. For the other direction, we have to prove that for every WS2S-(WS1S)formula we can construct a corresponding tree-(word)automaton. In the next chapter the first part of the proof is explained in detail.

4.1 From Automata to WSkS

The fact that we only work in the finite case, gives us the possibility to use bottom-up-tree-automata.

The following formula serves our needs:

$$
\begin{aligned}
&\exists Y_{q_1}, \ldots, \exists Y_{q_l}. && (1)\\
&\quad Term(X, X_{f_1}, \ldots, X_{f_m}) && (2)\\
&\wedge \quad Term(X, Y_{q_1}, \ldots, Y_{q_l}) && (3)\\
&\wedge \quad \bigvee_{q \in Q_f} \varepsilon \in Y_q && (4)\\
&\wedge \quad \forall x. \bigwedge_{f \in F} \bigwedge_{q \in Q} ((x \in X_f \wedge x \in Y_q) \Rightarrow && (5)\\
&\qquad \Big(\bigvee_{f(q_1,\ldots,q_s) \to q \in \Delta} \bigwedge_{i=1}^{s} xi \in Y_{q_i} \Big))
\end{aligned}
$$

where Y_{q_i} is the characteristic set of state i, and X_{f_j} is the characteristic set of functionsymbol j.

Basically it says: there exist characteristic sets of states (1). We have a tree, which is labeled with exactly one functionsymbol at each of its positions (2). We have a tree of the same shape as the one above but labeled with states (3). The root of the tree is labeled with a final state (4). And finally all transition rules that are use during the run belong to the set Δ of the automaton. (5)

This formula has exactly $m+1$ free variables, and it is satisfied iff the coding of a tree, which is accepted by the automaton, is assigned to it. For a word the formula is exactly the same, but the word has to be processed in reverse order. This is again possible because we are in the finite case.

So far we showed that it is possible to transform a tree automaton into an equivalent WSkS formula.

4.2 From WSkS to Automata

The task is now to construct the corresponding automata for every WSkS-formula. This will be done via induction over the structure of the formula. First we will show how the automata for the atomic formulas look like. As induction step we will show how to connect them logically. All automata work over the alphabet $\{0, 1, \perp\}$.

4.2.1 The Termination Symbol

On the one hand we only work with finite words or trees, on the other hand we want an arbitrary length or depth. If we added a new symbol $\perp$ to our alphabet, we would be able to mark the end of a word or a path in a tree. The only open question is: How do we know where to put this new terminationsymbol?

To answer this, we need to introduce a new predicate.

$$
\begin{aligned}
ModifiedTerm(X, X_1, \ldots, X_n) = \\
& X \neq \emptyset \\
\wedge \quad & X = \bigcup_{i=1}^{n} X_i \\
\wedge \quad & PrefixClosed(X) \\
\wedge \quad & \bigwedge_{i=1}^{k} \bigwedge_{arity(f_j)=i} (\bigwedge_{l=1}^{i} \forall x.(x \in X_{f_j} \Rightarrow xl \in X) \\
& \wedge \bigwedge_{l=i+1}^{k} \forall y.(y \in X_{f_j} \Rightarrow yl \notin X))
\end{aligned}
$$

It works exactly like the original *Term*, but it does not have the restriction, that all sets have to be pairwise disjoint. Due to the finiteness of all sets, there exists a position in the shape after which no entry is 1 anymore. If we use our new predicate in the following way, the new variable *Shape* is 1 as long as there is one or more variable left, for which a 1 occurs at this position or later.

$$ModifiedTerm(Shape, X_1, \ldots, X_n, X_{col})$$

X_{col} is needed to collect those positions in between where no variable is 1. The characteristic function of the set *Shape* has exactly one position at which it turns from 1 to 0. After this position it stays 0. Now we add a tuple, with the new $\perp$ -Symbol in every component, at the position of the first 0 in the characteristic function of *Shape*, because it indicates the end of the string or the path. Here is an example:

	ε	1	11	111	1111	11111	111111	$\cdots$
X_1	1	0	1	1	0	0	0	$\cdots$
X_2	0	0	0	1	1	0	0	$\cdots$
X_{col}	0	1	0	0	0	0	0	$\cdots$
Shape	1	1	1	1	1	0	0	$\cdots$

Without X_{col} it could happen, that the sequence of 1's is interrupted once ore more often by 0's. In other words X_{col} keeps *Shape* prefixclosed. This new termination symbol gives us the opportunity to work without any fixed limits.

4.2.2 Base Automata

Now we will show and explain the automata which correspond to the atomic formulas.

The word-automaton for $X \subseteq Y$ can be seen in figure 1. It accepts every word that does not contain any tuple $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$. The first component in every tuple shows if a position is labeled with an element of X or not. The second component does the same for Y. $X \subseteq Y$ is satisfied as long as it does not happen that a position is labeled with an element of X but not with one of Y.

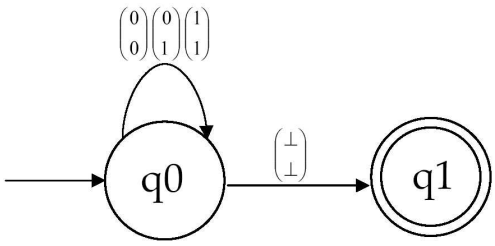

Figure 1: Word-automaton for $X \subseteq Y$.

The tree-automaton for $X \subseteq Y$ works in the same way and is depicted in figure 2. It works bottom up and its only final state is q. Again every combination but $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ is allowed.

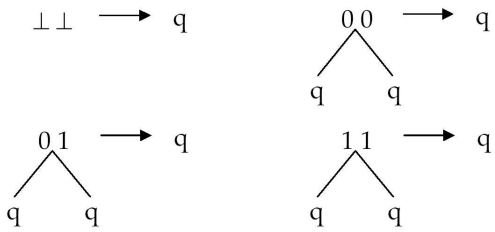

Figure 2: Tree-automaton for $X \subseteq Y$.

Figure 3 shows how the word-automaton for $X = Y1$ looks like. It accepts iff the one and only occurrence of y is directly followed by the one and only occurrence of x. Thats exactly what the definition of $X = Y1$ says.

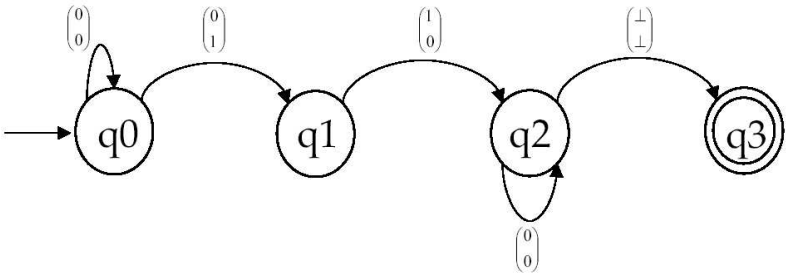

Figure 3: Word-automaton for $X = Y1$.

The tree-automaton for $X = Y1$ can be seen in figure 4. The only final state is q''. It accepts iff a node labeled with y has a left child labeled with x. For the case of $X = Y2$ just one rule has to be changed (q and q' have to be interchanged in the upper right rule).

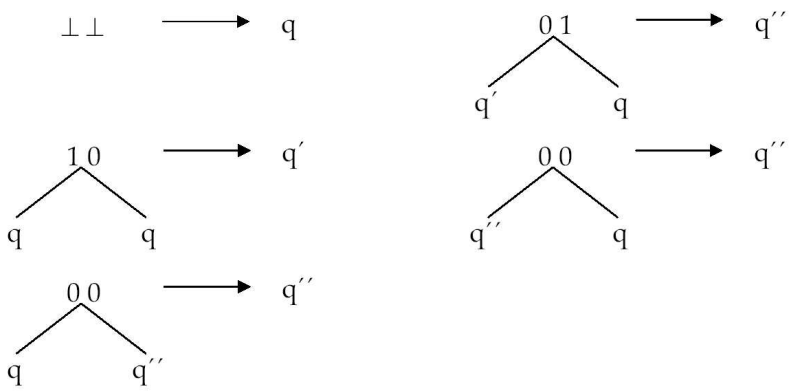

Figure 4: Tree-automaton for $X = Y1$.

So far we have constructed automata for every atomic formula in WS1S and WS2S.

4.2.3 Logical Connections

In this section we have to treat the three cases of logical connections.

Let A be the automaton corresponding to the formula ϕ. The automaton A' for the formula $\neg\phi$ can be derived from A. First A has to be completed. After that it has to be determinized, and finally we have to transform all regular states into final states and vice versa. Then A' accepts exactly the complement of A.

Let A be the automaton for ϕ_1 and B the one for ϕ_2. The automaton C for the formula $\phi_1 \vee \phi_2$ is found as follows: First we have to cylindrify. This means that C has to work over the alphabet $\Sigma_A \cup \Sigma_B$. Now the number of components in every tuple of C is $Size(\Sigma_C)$. Those Symbols who did not occur in A before are marked with a wildcard symbol. The same is done for B. Now C is the automaton which we get after adding a new starting state and lead it to the former starting states of A and B by ε-edges (after their alphabets have been extended).

Let A be the automaton corresponding to ϕ. The automaton A' for the formula $\exists X.\phi$ is derived from A by projection. This means the alphabet of A' has to be one element smaller than the alphabet of A. In every tuple of A the X-component is cut out, so that its size is decreased by 1. The rest of the automaton stays the same. A' may be nondeterministic even if A was not.

These are all cases which we had to study for our induction step. Now we can build a corresponding automaton for an arbitrary WS2S-(WS1S)formula. Together with the result of the previous section we are able to transform an automaton into a corresponding formula, and vice versa. Due to this result, the same expressive power of tree-(word)automata and WS2S (WS1S) is proved. □

5 Complexity

Lemma:WSkS is decidable.

The proof of this lemma is easy. It should not be discussed here in detail, but the main idea is to reduce it to the emptiness problem of tree-(word)automata. This is known to be decidable, and we know that any formula of WSkS can be transformed into a corresponding automaton.

Both logics are decidable, but neither WS1S nor WS2S can be decided in elementary time. That is the case because the construction of a complement-automaton requires a determinization. So whenever we have a universal quantification in a formula it has to be transformed into $\neg\exists.\neg\phi$. The inner negation forces a determinization which may be destroyed by the existence quantification. After that it has to be determinized again. For every determinization it may happen that the set of states of the automaton grows to its own power set. If N is the number of quantor alternations in ϕ then the size of the corresponding automaton will be a tower of exponentials whose height is limited linearly.

$$\left.\begin{matrix} 2^{2^{2^{\vdots^{2}}}} \end{matrix}\right\} O(N)$$

Meyer and Stockmeyer proved in 1973 that this cannot be done better [4]. This leads to the fact that even WS1S is not decidable in elementary time.

The reader may ask why it is enough to talk about WS2S and not about logics in which the functionsymbols have higher arities. This is enough because it is possible to reduce every n-arry tree to a binary one. We simply have to mark an edge that leads further in depth with a 0 and those that lead further in breadth with a 1, as it can be seen in figure 5.

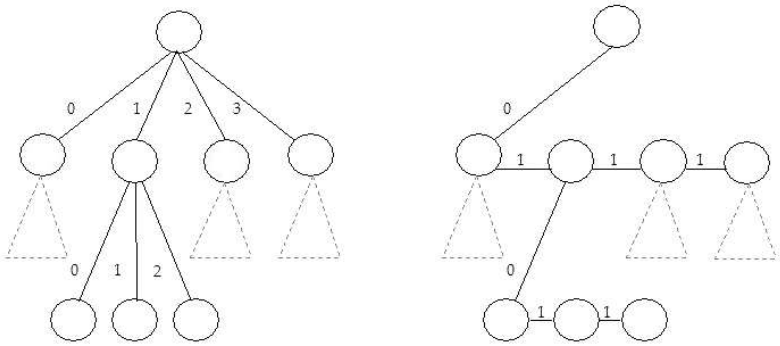

Figure 5: WSnS $\rightarrow$ WS2S.

6 Conclusions and Extensions

In this paper it has been proved that WS2S (WS1s) and tree-(word)automata correspond to each other. Therefore we introduced characteristic functions and characteristic sets. We showed how the logic can simulate an automaton and vice versa. Due to the correspondence we could conclude the decidability of WSnS, where n is arbitrary.

There exist two analogous theorems for the extended case. They compare S1S and S2S with ω-word-automata and ω-tree-automata. In the case of SkS there is restriction that all variables have to be finite. ω-automata can process infinite words or trees. The proofs require several new automata models and parts of game theory. But this is far out of the scope of this paper.

7 Acknowledgments

I would like to thank all members of the Programming Systems Lab and all the other participants of the seminar ”Logical Aspects of XML” who guided me through this research field. Special thanks to Tim Priesnitz whose remarks on this subject were very helpful. Finally i would like to thank Mathias Pohl who supported me with his Latex-skills.

References

[1] Hubert Comon, Max Dauchet, Remi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, Marc Tommasi. *Tree Automata Techniques and Applications*. Online Publication 2002.

[2] Wolfgang Thomas. *Languages, Automata and Logic*. Handbook Article 1996.

[3] Erich Gr”adel, Wolfgang Thomas, Thomas Wilke. *Automata, Logics, and Infinite Games: A Guide to Current Research*. Springer 2002 (LNCS 2500).

[4] Larry J. Stockmeyer and Albert R.Meyer. *Word Problems Requiring Exponential Time*. 5th Annual ACM Symposium on the Theory of Computing 1973.

[5] Bakhadyr Khoussainov, Anil Nerode. *Automata Theory and Its Applications*. Birkhuser 2001.

[6] Frank Neven. *Automata, Logic, and XML*. CSL 2002.

[7] Frank Neven, Thomas Schwentick. *Query Automata on finite trees*. Theoretical Computer Science 2002.

[8] Frank Neven. *Automata theory for XML researchers*. to appear in Sigmod Record.