Draft
13/11/2000

Predicate Logic and Tree Automata with Tests

Ralf Treinen[1]
Laboratoire de Recherche en Informatique
Université Paris-Sud
F-91495 Orsay cedex, France
http://www.lri.fr/~treinen

November 13, 2000

[1]Partially supported by the Esprit Working Group 22457 - CCL II

Abstract

We investigate the question whether the well-known correspondence between tree automata and the weak second order logic of two successor functions (*WS2S*) can be extended to tree automata with tests. Our first insight is that there is no generalization of tree automata with tests that has a decidable emptiness problem and that is equivalent to the full class of formulae in some extension of *WS2S*, at least not when we are asking for an conservative extension of the classical correspondence between *WS2S* and tree automata to tree automata with tests.

As a consequence we can extend the correspondence between tree automata and *WS2S* to automata with tests only when we admit a restriction of the class of formulae. We present a logic, called *WS2Sy*, and a restriction of the class of formula, called *uniform*, that is equivalent to tree automata with tests.

Contents

Draft 13/11/2000

1 Introduction

The equivalence of tree automata and weak second-order logic of 2 successor functions, short *WS2S*, is known for more then 30 years [11]. During the current decade, a lot of work has been done on classes of tree automata that are stronger than classical tree automata (see [4] and [9] for a survey). The general frame to obtain these stronger classes of tree automata is to augment the automaton model with tests. In the case of tree automata with tests between brothers [1], for instance, we can write a transition rule like $f(q_1, q_2) \rightarrow^{1=2} q$ that will accept a tree $f(t_1, t_2)$ in state q if t_1 is accepted in state q_1, t_2 is accepted in q_2, and if in addition $t_1 = t_2$. These tests increase in a considerable way the expressiveness of tree automata. Tree automata with equality tests between brothers as sketched above can for instance recognize the set of balanced trees (which is not possible with classical tree automata).

Defining new classes of tree automata is of course only interesting when the usual "nice" properties of automata are preserved. The most important of these is the decidability of the emptiness of the language recognized by such an automaton. Furthermore we expect a good class of recognizable languages to be closed under operations such as union, intersection, complement and (maybe restricted classes of) homomorphisms and inverse homomorphisms.

Several classes of tree automata with "nice" properties have been identified (see [4] for an excellent overview of tree automata and tree automata with tests). In light of the above mentioned equivalence between tree automata and the logic *WS2S* it is a natural question whether there are similar logical characterizations of tree automata with tests. Before attacking this question we will try to explain in an informal manner the logic *WS2S* and its correspondence with tree automata.

The second-order logic *WS2S* comes with a standard interpretation. There are two sorts: the sort of words over the alphabet $\{1, 2\}$ and the sort of *finite* sets of words. There are a constant for the empty word, unary functions to append a symbol 1, resp. 2 at the end of a word, and the elementship relation between words and sets. An example of a formula in this logic, expressing that the word x is a prefix of the word y is (the convention is that word variables are written in lower case and set variables in upper case):

$$\forall Y(y \in Y \wedge \forall z([z1 \in Y \vee z2 \in Y] \rightarrow z \in Y) \rightarrow x \in Y)$$

A tree automaton with k states can be translated into a formula of *WS2S* in the following sense: We represent a tree over an alphabet of n symbols by n sets, each of them representing the occurrences (addresses of nodes) of the tree marked with one the symbols. The formula has to state that these sets are pairwise disjoint (since an occurrence can carry only one symbol), that their union is closed under prefix (using the prefix predicate defined above) and that the arity of the function symbols is respected, that there is an assignment of states to the occurrences (this is expressed by an existential quantification over k sets similar to what we did to represent the term itself) respecting the transition rules of the automaton such that the root is marked with an accepting state.

Given a formula ϕ with n free variables (by a slight variation of the logic we can assume that only set variables are involved) we first have to say how to represent an n-tuple of sets as a tree: We take as tree signature the n-tuples over $\{0,1\}$, a tree representing a n-tuple of sets has at occurrence π the symbol 1 in the i-th component if and only if π belongs to the i-th set. Since all sets in *WS2S* are finite a finite tree is sufficient (there is also a technically *much* more involved variant for infinite sets and infinite trees due to Rabin [8]). Now, the idea to get an automaton that corresponds to a given formula is to first construct automata for the atomic formula of the logic, and then to apply the closure of tree automata by union, complement and projection (corresponding to disjunction, negation and existential quantification).

In fact there is a problem with the translation of formulae into automata that is easily overseen when sketched as we did above: For this translation to work we need an additional closure property of tree automata (which, luckily, does hold for classical tree automata): closure by *cylindrification*. The problem is apparent when we have constructed a tree automaton for a formula $\phi(X_1, X_2)$ and another one for a formula $\psi(X_2, X_3)$ and when we now want to construct from these the automaton for $\phi(X_1, X_2) \wedge \psi(X_2, X_3)$. We cannot just take the intersection of these two automata since the two "components" of the trees recognized by the respective automata do not correspond to the same variables. In fact we have to cylindrify both automata, that is add to the first automaton a third component and to the second automaton a new first component that are in fact completely ignored by the rules of the automata (the origin of the name "cylindrification" is the intuition of a two-dimensional geometrical figure that is slided along the third dimension through space). Finally we can take the intersection of the two cylindrified automata.

Here lies the problem when we ask for a logical characterization of tree automata with tests: The closure properties of the logic, including the "implicit" closure by cylindrification that simply stems from the presence of variables in the logic, require any class of automata that corresponds to the *full* class of formulae of some logic to be closed under cylindrification.

If we need cylindrification for our class of automata to correspond to some logic then we could simply extend our automaton model accordingly and try to show that the "nice" properties of automata still hold. The automata model that we get from *Tree Automata with Tests (TAT)* when we close under cylindrification are *Tree Automata with Component-wise Tests (TACT)* that will be formally defined in Section 3. Unfortunately, emptiness of TACT-automata is undecidable as we will show in Section 4, that is, our original program simply has to fail.

Hence we propose to ask for a *subset* of the set of formulae of some suitable logic that is equivalent to our class of formulae. We define in Section 7 such a logic *WS2Sy* and its two subclasses of so-called *restricted* and *uniform* formulae. We state the exact correspondence between TACT and TAT automata and the subclasses of restricted, resp. uniform formulae of *WS2Sy* in Section 8 and prove this correspondence in Section 9 and 10. The technical part of the paper only considers automata with tests between brothers, we briefly discuss extension to other classes of automata with tests in Section 11.

This report is an extended and corrected version of [12].

2 Preliminaries

A *signature* Σ consists of a set of symbols (also denoted Σ) and an *arity function* $\alpha\colon \Sigma \to \mathbf{N}_0$. Often one writes Σ_n for $\{f \in \Sigma \mid \alpha(f) = n\}$. For a signature Σ the set $T(\Sigma)$ of Σ *ground terms* is the smallest set such that if $f \in \Sigma_n$ and $t_1, \ldots, t_n \in T(\Sigma)$ then $f(t_1, \ldots, t_n) \in T(\Sigma)$. The set of *occurrences* $O(t)$ of a tree $t = f(t_1, \ldots, t_n)$ is $\{\epsilon\} \cup \bigcup_{i=1}^{n} \{iw | w \in O(t_i)\}$. If $\pi \in O(t)$ then the *subtree* $t \mid_\pi$ of t at π is defined by $t \mid_\epsilon = t$ and $f(t_1, \ldots, t_n) \mid_{i\pi} = t_i \mid_\pi$. Furthermore, if $\pi \in O(t)$ then $t(\pi)$ denotes the symbol of t at position π, that is the root symbol of $t \mid_\pi$.

Given signatures Σ, Γ, a *renaming* ρ is a function $\rho\colon \Sigma \to \Gamma$ such that $\alpha(\rho(f)) \leq \alpha(f)$ for all $f \in \Sigma$. A renaming $\rho\colon \Sigma \to \Gamma$ extends to a tree homomorphism $\rho\colon T(\Sigma) \to T(\Gamma)$ by $\rho(f(t_1, \ldots, t_n)) = (\rho(f))(\rho(t_1), \ldots, \rho(t_{\alpha(\rho(f))}))$ (hence, arguments not needed in Γ are simply dropped).

For a word w we write $|w|$ its length and $w(i)$, $1 \leq i \leq |w|$ its i-th symbol. An instance of the *Post Correspondence Problem (short: PCP)* is a finite sequence $P = ((p_i, q_i))_{i=1,\ldots,m}$ of pairs of words from $\{a, b\}^*$. A *solution* of such an instance of PCP is a nonempty sequence $(i_1, \ldots, i_n)$, $1 \leq i_j \leq m$, such that $p_{i_1} \cdots p_{i_n} = q_{i_1} \cdots q_{i_n}$. According to a classical result due to Post [7], it is undecidable whether an instance of the PCP has a solution or not.

3 Tree Automata with Component-Wise Tests

Definition 1 (Tuple Signature) *For any finite alphabet Γ and $n \geq 0$, the signature $\Gamma^{(n)}$ consists of the set of function symbols $(\Gamma \cup \{\bot\})^n$, where $\bot$ is a new symbol. The symbol $(\bot \ldots \bot)$ is a constant, all other symbols are binary.*

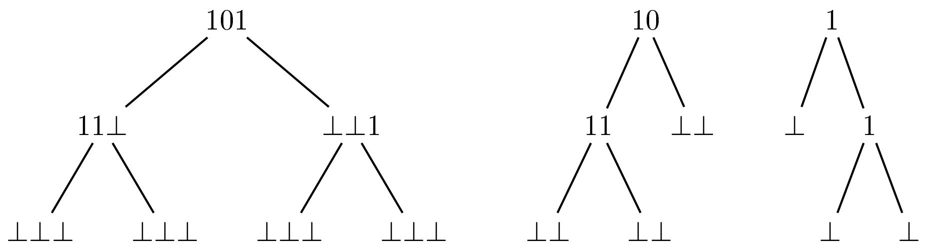
Example 1 *Let $\Gamma = \{0, 1\}$. The signature $\Gamma^{(2)}$ contains the binary symbols* $00, 01, 0\bot, 10, 11, 1\bot, \bot 0, \bot 1$ *and the constant* $\bot\bot$.

Definition 2 (Projection and Component of a Tree) *Let τ be a $\Gamma^{(n)}$-tree and $1 \leq i \leq n$.*

1. *The i-th* component *of τ, denoted τ^i, is the $\Gamma^{(1)}$-tree given by the renaming $(f_1 \ldots f_i \ldots f_n) \mapsto f_i$ for all $f_j \in \Gamma \cup \{\bot\}$.*
2. *The i-th* projection *of τ, denoted $\tau^{\langle i \rangle}$, is the $\Gamma^{(n-1)}$-tree given by the renaming $(f_1 \ldots f_{i-1} f_i f_{i+1} \ldots f_n) \mapsto (f_1 \ldots f_{i-1} f_{i+1} \ldots f_n)$ for all $f_j \in \Gamma \cup \{\bot\}$.*

Note that both projection and selection of a component can change the set of occurrences of a tree since a binary symbol may become a constant, as is illustrated in the example given on Figure 1.

Definition 3 (Tree Automata) *A* tree automaton with component-wise tests between brothers (short: TACT) *is a quintuplet $(\Gamma, n, Q, Q_f, \Delta)$ where*



(a) Tree τ (b) $\tau^{\langle 3 \rangle}$ (c) τ^3

Figure 1: Example of projection and component.

- Γ *is a finite alphabet and* $n \geq 1$,
- Q *is a finite set of states,*
- $Q_f \subseteq Q$ *is the set of accepting states,*
- Δ *is a set of transition rules of one of the following forms:*
  - $(\perp \ldots \perp) \rightarrow q$, $q \in Q$.
  - $f(q_1, q_2) \rightarrow^c q$ *where* c *is a boolean combination of tests* $1.i = 2.i$ *with* $1 \leq i \leq n$, $f \in \Gamma^{(n)} - (\perp \ldots \perp)$, $q, q_1, q_2 \in Q$.

A tree automaton with tests between brothers (short: TAT) *is a* $TACT$ *automaton* $(\Gamma, n, Q, Q_f, \Delta)$ *where all tests of rules in* Δ *are either* **`True`**, *or* $1.1 = 2.1 \wedge \ldots \wedge 1.n = 2.n$, *or* $1.1 \neq 2.1 \vee \ldots \vee 1.n \neq 2.n$.

A tree automaton (short: TA) *is a* $TACT$ $(\Gamma, n, Q, Q_f, \Delta)$ *where all tests of rules in* Δ *are* **`True`**.

Note that in a TAT all tests operate uniformly on *all* components. Hence TAT's are exactly the tree automata with tests between brothers introduced in [1], and TA are classical tree automata. In the examples we will simply drop a test `True` on a transition rule and write it as the rule of a classical tree automaton.

Definition 4 *The* $TACT$ A tests only on the set of components I *if whenever* A *contains a transition rule with a test* c *and* $1.i = 2.i$, *resp.* $1.i \neq 2.i$ *is an atomic test in* c, *then* $i \in I$.

Definition 5 (Acceptance by a TACT) *A tree* t satisfies *the test* $1.i = 2.i$ *iff* $(t\mid_1)^i = (t\mid_2)^i$, *this extends in the canonical way to boolean combinations of tests.*

The rewriting relation of a $TACT$ $A = (\Gamma, n, Q, Q_f, \Delta)$ *is the smallest binary relation* $\rightarrow_A \subseteq T(\Gamma^{(n)}) \times Q$ *with*

- $a \rightarrow_A q$ *if* $a \rightarrow q \in \Delta$
- $f(t_1, t_2) \rightarrow_A q$ *if* $f(q_1, q_2) \rightarrow^c q \in \Delta$, $f(t_1, t_2)$ *satisfies* c, $t_1 \rightarrow_A q_1$ *and* $t_2 \rightarrow_A q_2$.

The automaton A accepts *a tree t if $t \to_A q$ for some accepting state $q \in Q_f$. The set L_A* recognized *by A is the set of all trees accepted by A.*

Definition 6 (Deterministic and Complete TACT) *A TACT A is* deterministic *if for every tree t there is at most one state q such that $t \to_A q$, and* complete *if for every tree t there is at least one state q with $t \to_A q$.*

The following proposition is immediate from our definitions:

Proposition 1 *Every TA-recognizable set is TAT-recognizable, and every TAT-recognizable set is TACT-recognizable.*

4 The Emptiness Problem for TACT

Theorem 1 *The emptiness-problem for TACT automata (even when restricted to the class of TACT without negative constraints) is undecidable.*

Proof: We reduce the Post Correspondence Problem to the emptiness problem of TACT automata. Let $P = ((p_i, q_i))_{i=1,\ldots,m}$ be an instance of PCP.

First we have to explain how n-tupes of words over an alphabet Γ can be encoded into trees from $T(\Gamma^{(n)})$. We define an operation that allows us to extract a n-tuple of Γ-words from any $\Gamma^{(n)}$-tree. The extraction function will not be one-to-one, however, for every n-tuple of Γ-words there will be at least one $\Gamma^{(n)}$-tree encoding it.

The extraction operation will be defined by its components. For a tree $t \in T(\Gamma^{(n)})$ and $1 \leq i \leq n$, the *i-th word coded by t* is the word $t\downarrow_i \in \Gamma^*$ defined by

$$t\downarrow_i = \begin{cases} \epsilon & \text{if } t^i = \perp \\ c \cdot t\mid_2\downarrow_i & \text{if } t^i = c(\_, t\mid_2) \end{cases}$$

where $\_$ stands for any function symbol from $\Gamma^{(n)}$. For instance, if t is the right subtree of the tree depicted in Figure 2 then $t\downarrow_1 = aabb$ and $t\downarrow_3 = fa$.

We define the signature of our automaton as $\{f, a, b\}^{(4)}$. The automaton L_P is constructed in three steps:

The first step is to construct an automaton that accepts a tree if it encodes two pairs of words such that the second pair is obtained from the first pair by application of one step of the PCP instance P. For reasons that will be apparent in the last step this automaton comes in a positive and in a negative version that differ in the choice of the components that code the two pairs. More exactly:

For any word $v \in \Gamma^*$, let $A_v^{i,j}$ for $i \neq j$ be the automaton that accepts t iff $t\downarrow_i = fw$ and $t\downarrow_j = wv$ for some $w \in \Gamma^*$. If $v = v_1 \cdots v_n$ then the rules of the automaton are (here for $i = 1$ and $j = 2$), where $\_$ stands for any symbol from $\{a, b, f, \perp\}$ and α, β for any symbol from $\{a, b\}$:

$$\begin{array}{rcl} (\perp\perp\perp\perp) & \to & q_* \\ (\_\_\_\_)(q_*, q_*) & \to & q_* \\ (\perp\perp\perp\perp) & \to & q_n \end{array}$$

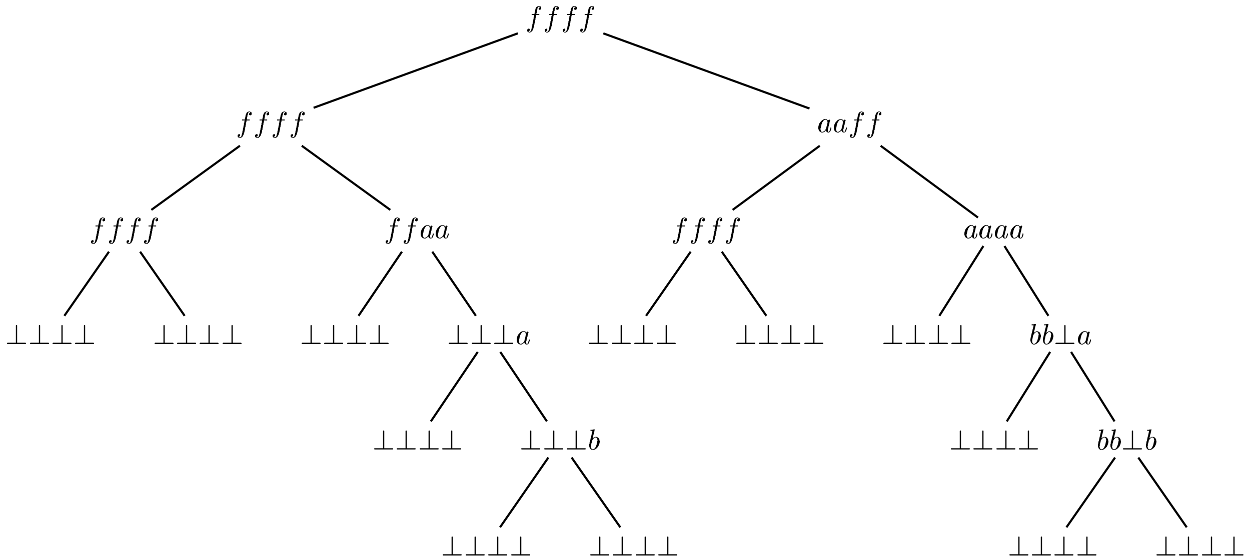


Figure 2: Example of the representation of the solution $((\epsilon, \epsilon), (a, aab), (aabb, aabb))$ of the PCP instance $((a, aab), (abb, b))$.

$$
\begin{array}{rcll}
(\perp\perp\_\_)(q_*, q_n) & \to & q_n & \\
(\perp v_i\_\_)(q_*, q_i) & \to & q_{i-1} & i = 2, \ldots, n \\
(f v_1\_\_)(q_*, q_1) & \to & q_0 & \\
(\alpha v_1\_\_)(q_*, q_1) & \to & q_\alpha & \\
(\alpha\beta\_\_)(q_*, q_\beta) & \to & q_\alpha & \\
(f\beta\_\_)(q_*, q_\beta) & \to & q_0 &
\end{array}
$$

The only accepting state is q_0.

From the boolean closure of (classical) tree automata we get an automaton A_P^+ that accepts t iff for some i we have that $t\downarrow_1 = fw_1$, $t\downarrow_2 = fw_2$, $t\downarrow_3 = w_1 p_i$ and $t\downarrow_4 = w_2 q_i$. Analogously, the automaton A_P^- accepts t iff for some i we have that $t\downarrow_1 = w_1 p_i$, $t\downarrow_2 = w_2 q_i$, $t\downarrow_3 = fw_1$ and $t\downarrow_4 = fw_2$. Let Q^+ resp. Q^- be the accepting states of these two automata. In case of the tree t of Figure 2, for example, A_P^+ accepts $t|_{12}$ and A_P^- accepts $t|_2$.

The second step is to construct automata that in addition test whether the newly constructed pair is of the form (w, w) for some non-empty word w. We can easily derive from A_P^+ this automaton AE_P^+ (with set of accepting states $Q_=^+$) that emulates A_P^+ and verifies in addition that $t\downarrow_3 = t\downarrow_4 \neq \epsilon$, and analogously AE_P^- (with set of accepting states $Q_=^-$) that emulates A_P^- and verifies in addition that $t\downarrow_1 = t\downarrow_2 \neq \epsilon$. In case of the tree t of Figure 2, AE_P^- accepts $t|_2$.

Note that the first two steps used only classical tree automata. The third step is now to build the final automaton which uses these four automata as sub-automata (we hence assume their states-spaces to be disjoint) to accept a solution sequence of P. This is where equality tests come in.

$$
\begin{array}{rcll}
(\perp\perp\perp\perp) & \to & p_\perp & \\
(ffff)(p_\perp, p_\perp) & \to & p^+ & \\
(ffff)(p^+, q^+) & \to^{1.1=2.1\wedge 1.2=2.2} & p^- & q^+ \in Q^+ \\
(ffff)(p^-, q^-) & \to^{1.3=2.3\wedge 1.4=2.4} & p^+ & q^- \in Q^- \\
(ffff)(p^+, q^+) & \to^{1.1=2.1\wedge 1.2=2.2} & p_f & q^+ \in Q_=^+ \\
(ffff)(p^-, q^-) & \to^{1.3=2.3\wedge 1.4=2.4} & p_f & q^- \in Q_=^-
\end{array}
$$

where p_f is the only accepting state. An example of the representation of the solution of an instance of PCP as accepted by this automaton is given in Figure 2. In this example, the accepting run assigns the occurrence 11 the state p^+, the ocurrence 1 the state p^- and the occurrence ϵ the accepting state p_f.

From the construction of the automaton it is obvious that if it accepts some tree then the instance of PCP has a solution. For the other direction suppose that $(v_1, w_1), \ldots, (v_n, w_n)$ is a solution of P. We construct a sequence of trees $t_0, t_1, \ldots, t_n$ such that

1. t_0 is accepted in state $p_\perp$
2. For $1 \leq i \leq n$, i odd, t_i is accepted in some state $p^+ \in Q^+$
3. For $1 \leq i \leq n$, i even, t_i is accepted in some state $p^- \in Q^-$

4. t_n is accepted in state p_f

For any i let $n_i = max(|v_i|, |w_i|)$. For a letter c we write $c^{\leq n}$ for the set $\{w \in c^* \mid |w| \leq n\}$. We extend the definition of the i-th symbol of a word to positions exceeding the length of the word w by setting $w(i) := \perp$ if $i > |w|$.

We define $t_0 = (\perp\perp\perp\perp)$. If $0 \leq i < n$ then t_{i+1} is defined as $(ffff)(t_i, s_{i+1})$ where

$$O(s_{i+1}) = O(t_n) \cup 2^{\leq n_{i+1}} \cup 2^{\leq n_{i+1}-1}1$$

and the labeling of tree s_{i+1} for $i+1$ even (the analogous definition for the case $i+1$ odd is obtained by swapping component 1 for 3 and 2 for 4) is given by

- $s_{i+1}(\epsilon) = (ffv_{i+1}(1)w_{i+1}(1))$
- if $1 \leq m \leq n_{i+1}$ then $s_{i+1}(2^m) = (v_i(m)w_i(m)v_{i+1}(m-1)w_{i+1}(m-1))$
- if $\pi \in O(t_n) - 2^*$ then $s_{i+1}(\pi) = t_i(\pi)$
- if $0 \leq m < n_{i+1}$ and $2^m 1 \notin O(t_n)$ then $s_{i+1}(2^m 1) = (\perp\perp\perp\perp)$

□

The undecidability result holds even in case of signatures with only two components. For a proof, it suffices to encode in the above proof the first and second, respectively third and forth component of the signature into one symbol.

5 Tree Automata Testing Recognizable Relations

In this section we address the question whether the emptyness problem for tree automata that can test brothers by a recognizable relation have a decidable emptyness problem.

Definition 7 *A binary relation $R \subseteq T(\Sigma) \times T(\Sigma)$ is* recognizable *iff there is a TA-recognizable language $L \subseteq T(\Sigma^{(2)})$ such that*

$$R = \{(t_1, t_2) \mid \exists t \in L : t^{\langle 1 \rangle} = t_1 \text{ and } t^{\langle 2 \rangle} = t_2\}$$

Example 2 *The equality relation on Σ-terms is recognizable. The defining automaton on $T(\Sigma^{(2)})$ has one state q which is accepting, and rules*

$$\begin{array}{rcll} (\perp\perp) & \to & q & \\ (ff)(q,q) & \to & q & \quad \text{for all } f \in \Sigma \end{array}$$

Note that the automaton simply blocks when it encounters a symbol (fg) with differnt components that spoils equality.

Definition 8 *A* Tree automaton with recognizable tests between brothers (short: TART) *consists of a finite signature Σ, a finite set Q os states, a subset $Q_f \subseteq Q$ of accepting states, and a set of transition rules of one of the following forms*

- $a \to q$, $a \in \Sigma_0$, $q \in Q$

- $f(q_1,\ldots,q_n) \rightarrow^{R(i,j)} q$, $f \in \Sigma_n$, $q_1,\ldots,q_n,q \in Q$, $1 \leq i,j \leq n$ *and* R *a recognizable relation.*

The rewriting relation of a TART $A = (\Sigma, Q, Q_f, \Delta)$ *is the smallestbinary relation* $\rightarrow_A \subseteq T(\Sigma) \times Q$ *with*

- $a \rightarrow_A q$ *if* $a \rightarrow q \in \Delta$
- $f(t_1,\ldots,t_n) \rightarrow_A q$ *if* $f(q_1,\ldots,q_n) \rightarrow^{R(k,l)} q \in \Delta$, $t_i \rightarrow_A q_i$ *for* $1 \leq i \leq n$ *and* $(t_k, t_l \in R)$*.*

The TART A accepts *a tree* t *if* $t \rightarrow_A q$ *for some accepting state* $q \in Q_f$*. The set* L_A recognized *by* A *is the set of all trees accepted by* A*.*

Proposition 2 *Every TAT-recognizable, and hence every TA-recognizable set, is TART-recognizable.*

Proof: This follows immediately from the fact that the universal binary relation on trees, $T(\Sigma) \times T(\Sigma)$, and the equality relation (see Example 2) are recognizable. □

Theorem 2 *The emptyness problm for TART is undecidable.*

Proof: This follows immediately from Theorem 1 and the observation that for any i the relation

$$\{(t_1, t_2) \in T(\Gamma^{(n)}) \times T(\Gamma^{(n)} \mid t_1^i = t_2^i\}$$

is recognizable. □

Note that Theorem 2 stills holds when we restrict the class of recognizable relations to the the class of equivalence relations. If we restrict furtermore to equivalence relations with finite rang (that is, finite number of equivalence classes) then we fall back into the class of TA-recognizable languages, and hence decidability in this restricted claass is decidable.

Proposition 3 (*Result obtained in collaboration with Sophie Tison*) *Every language recognizable by a TART where all all relations have finite rang is TA-recognizable.*

Proof: [Sketch] We show how to translate TARTs into tree automata with regular constraints (to be defined below), and how to translate tree automata with regular constraints into classical tree autoamata.

A *Tree automaton with regular constraints* is defined as a classical tree automaton where in addition the transitions are annotated with membership tests in regular tree languages, like

$$f(q_1,\ldots,q_n) \rightarrow^{t_1 \in L_1 \wedge \ldots t_n \in L_n} q$$

where the L_i are (previously defined) regular tree languages. The notion of an accepting run is defined accordingly.

Let R be a recognizable relation. Then, for every term t, the set of trees $\{s \mid R(s,t)\}$ is recognizable. Hence, every equivalence classe of a recognizable relation is recognizable.

Let A be a TART where all relations have finite rang. We define an equivalent tree automaton with regular constraints by transforming every TART-transition

$$f(q_1, \ldots, q_n) \to^{R(i,j)} q$$

where $L_1, \ldots, L_k$ is the family of equivalence classes of R, into the following transitions with regular constraints:

$$\{f(q_1, \ldots, q_n) \to^{t_i in L_l, t_j \in L_l} q \mid 1 \leq l \leq k\}$$

Finally, to transform a tree automaton B with regular constraints into a classical tree automaton it suffices to execute the (deterministic) automata of all the regular languages used in B in parallel to the transitions of B. Whenever B tests membership in a regular language the transformed automaton just checks that the automaton corresponding to this regular set is in an accepting state. □

6 Closure Properties of TACT

In the case of the equivalence between *WS2S* and classical tree automata the translation of an arbitrary formula is obtained once the closure of the class of recognizable sets by the boolean operation, cylindrification and projection is established. In case of *WS2Sy* and TACT we need, however, a more specific property of TACT-recognizable sets than just closure by certain operations: The closure must not affect the set of components on which the automaton performs tests. This invariant of the closure operations is needed because in the closure by projection of the i-th component we need as a hypothesis that the automaton does not test on the i-th component. An even stronger invariant will be needed to prove that uniform *WS2Sy* translates to TAT automata.

6.1 Determinization

Proposition 4 *If L is recognizable by a TACT A that tests only on the set of components I then there exists a complete and deterministic TACT that tests only on I and that recognizes L.*

Proof: Completion is achieved by adding a sink state. The determinisation is the same as in [4]. Since all rules in the determined automaton have tests that are boolean combinations of tests in the original automaton the set of tested components is not touched by this operation. □

It should be remarked that the determinisation of a tree automaton with tests requires the set of constraints to be closed under negation (see [4]).

6.2 Boolean Operations

Proposition 5 *If L is recognized by a TACT A that tests only on the set of components I then there exists a TACT that tests only on I and that recognizes $T(\Gamma^{(n)}) - L$.*

If L_1, L_2 are recognized by TACTs A_1, A_2 that test only on the set of components I then there exists a TACT that tests only on I and that recognizes $L_1 \cup L_2$, resp. $L_1 \cap L_2$.

Proof: This is now straightforward (again, see [4]). We just remark that the closure of the language of constraints by conjunction is used in case of the intersection. □

6.3 Projection

Definition 9 *Let L be a set of $\Gamma^{(n)}$-trees. For $1 \leq i \leq n$ the i-th* projection *of L is*

$$\exists i : L \quad = \quad \{t^{\langle i \rangle} \mid t \in L\}$$

Proposition 6 *The class of TACT-recognizable languages is not closed by projection.*

Proof: Consider the following language over the signature $\{f, g\}^{(1)}$:

$$L := \{f(s,t) \mid O(s) = O(t)\}$$

Suppose that TACT A with n states recognizes L. Hence, A accepts the tree $f(s,t)$ where $s = f(\bot, f(\bot, \ldots, f(\bot, \bot) \ldots))$ is an f-comb of heigth $n+3$, and $t = g(\bot, g(\bot, \ldots, g(\bot, \bot) \ldots))$ is an g-comb of heigth $n+3$. No rule application in the accepting run of $f(s,t)$, except perhaps the ones acting on $f(\bot, \bot)$ and $g(\bot, \bot)$, uses an equality test. Hence, using a standard pumping technique, we obtain another tree $f(s', t)$ where s' is a larger f-comb than s, that is accepted by A. This yields a contradiction to the definition of L.

However, L can be defined with a TACT-automaton over two components and projection. The idea is to rewrite the definition of L as $\{f(s,t) \mid \rho(s) = \rho(t)\}$ where ρ is the renaming $f \mapsto f, g \mapsto f$. We use the second component to represent the image of the first component under ρ. The automaton verifies everywhere in the tree the renaming ρ by only acepting pairs (ff) and (gf). At the root of the tree we test equality of the second component before going into the accepting q_f state:

$$\begin{array}{rcl}
(\bot\bot) & \to & q \\
(ff)(q,q) & \to & q \\
(gf)(q,q) & \to & q \\
(ff)(q,q) & \to^{1.2=2.2} & q_f
\end{array}$$

□

However, closedness by projection is regained when we allow projection only on components that are not tested by the automaton:

Proposition 7 *Let $L \subseteq T(\Gamma^{(n)})$ be recognized by a TACT A that tests only on the set of components I and $i \notin I$. There exists a TACT that recognizes $\exists i : L$ and tests only on $\{j \mid j < i\} \cup \{j - 1 \mid j > i\}$.*

Proof: Let L be recognized by the TACT $A = (\Gamma, n+1, Q, Q_f, \Delta)$. To ease notation we show the construction for the case $i = n+1$.

We can assume w.l.o.g that A is reduced, that is that there is for every $q \in Q$ a term $t \in T(\Gamma^{(n)})$ such that $t \to_A^* q$ (this can simply be obtained by dropping all states that recognize the empty set). We construct an automaton for $\exists n : L$ as $A = (\Gamma, n, Q, Q_f, \Delta')$ where

$$
\begin{array}{rcl}
\Delta' & = & \{(\perp \cdots \perp) \to q \mid (\perp \cdots \perp\perp) \to q \in \Delta\} \\
& \cup & \{(\perp \cdots \perp) \to q \mid (\perp \cdots \perp f)(q_1, q_2) \to q \in \Delta, f \neq \perp\} \\
& \cup & \{\bar{f}(q_1, q_2) \to^c q \mid \bar{f} f(q_1, q_2) \to^c q, \bar{f} \neq \bar{\perp}\}
\end{array}
$$

The proof that A' accepts $\exists n+1 : L$ is exactly as in the case of classical tree automata (see, e.g., [4]). The restriction that the automaton A does not use a test on the $n+1$-th component ensures that the constraints pose no problem here. □

6.4 Cylindrification

Definition 10 *Let L be a set of $\Gamma^{(n)}$-trees. For $1 \leq i \leq n+1$, the $i+1$-*th *cylindrification of L is*

$$
\updownarrow i : L \quad = \quad \{t \in T(\Gamma^{(n+1)}) \mid t^{\langle i \rangle} \in L\}
$$

Proposition 8 *If L is recognizable by a TACT A that tests only on the set of components I then there exists a TACT that recognizes $\updownarrow i : L$ and tests only on $\{j \mid j < i\} \cup \{j + 1 \mid j > i\}$.*

Proof: Let L be recognized by the TACT $(\Gamma, n, Q, Q_f, \Delta)$. To ease notation we show the construction for the case $i = n+1$. It is straightforward to show that $\updownarrow n+1 : L$ is recognized by the TACT $(\Gamma, n+1, Q, Q_f, \Delta')$ where

$$
\begin{array}{rcl}
\Delta' & = & \{(\perp \cdots \perp\perp) \to q \mid (\perp \cdots \perp) \to q \in \Delta\} \\
& \cup & \{(\bar{f} f)(q_1, q_2) \to^c q \mid \bar{f}(q_1, q_2) \to^c q \in \Delta, f \neq \perp,\ \bar{f} \neq \bar{\perp}\} \\
& \cup & \{(\bar{\perp} f)(q_1, q_2) \to q \mid (\perp \cdots \perp) \to q \in \Delta, f \neq \perp\}
\end{array}
$$

□

7 The Logic *WS2Sy*

The logic *WS2Sy* is a two-sorted logic with the sorts `w` (for *word*) and `s` (for *set*). We will write variables of sort `w` in lower case and variables of sort `s` in upper case. Function symbols are a constant ϵ of sort `w` and two monadic function symbols 1 and 2 of profile `w` $\to$ `w`. We write applications of these function symbols in postfix notation, that is $x1121$ for $1(2(1(1(x))))$. Relation symbols are $\in$ of profile $\mathtt{w} \times \mathtt{s}$, and for every $n \geq 1$ a $n+1$-ary predicate Sy of profile $\mathtt{w} \times \mathtt{s} \times \ldots \times \mathtt{s}$.

The standard interpretation of *WS2Sy* assigns the sort $\mathtt{w}$ as universe the set $\{1,2\}^*$ and the sort $\mathtt{s}$ as universe the set of all finite subsets of $\{1,2\}^*$. The constant ϵ is interpreted as the empty word, and the function 1 (resp. 2) as the function that appends 1 (resp. 2) to a word. The predicate symbol $\in$ denotes set membership. The synchronisation predicate $Sy(w, M_1, \ldots, M_n)$ holds if for all $1 \leq i \leq n$ and all $x \in \{1,2\}^*$ we have that $w1x \in M_i$ iff $w2x \in M_i$.

The notion of a *free occurrence* of a variable in a formula is defined as usual. A *WS2Sy*-formula ϕ is called *restricted* if whenever ϕ contains a subformula of the form $Sy(x, X_1, \ldots, X_n)$ then $X_1, \ldots, X_n$ are free variable occurrences (here, there is no restriction on x being free or bound). A *WS2Sy*-formula ϕ is called *uniform* if whenever ϕ contains a subformula of the form $Sy(x, X_1, \ldots, X_n)$ then $X_1, \ldots, X_n$ are exactly the free variable occurrences of ϕ (hence, x is bound). As a consequence, every uniform formula of *WS2Sy* is restricted, and the classical logic *WS2S* is obtained from *WS2Sy* by omiting the synchronisation predicates.

The logic *WS2Sy* allows to define the usual set-theoretic notions as union, intersection, singleton sets, etc. that we will use freely in the sequel. Furthermore, the prefix relation on words (denoted $x \leq y$) is expressible in *WS2S* as has been demonstrated in Section 1, hence in *WS2Sy*.

Following the classical translation of the logic *WS2S* into tree automata, we define the one-sorted version $WS2Sy_0$ of *WS2Sy* where only set-variables and predicates on sets are used. The predicates of $WS2Sy_0$ are $X_1 = X_2 1$, $X_1 = X_2 2$, $X_1 = \epsilon$, $X_1 \subseteq X_2$, and $Sy(X_0, X_1, \ldots, X_n)$. In the standard interpretation of $WS2Sy_0$ we have that $M_1 = M_2 1$ is true iff M_2 is some singleton set $\{m_2\}$ and $M_1 = \{m_2 1\}$ and analogous for $M_1 = M_2 2$; $M_1 = \epsilon$ is true iff $M_1 = \{\epsilon\}$; $\subseteq$ is set inclusion; and $Sy(M_0, M_1, \ldots, M_n)$ is true in $WS2Sy_0$ iff M_0 is some singleton set $\{m_0\}$ and $Sy(m_0, M_1, \ldots, M_n)$ is true in *WS2Sy*. We can now translate in a straightforward way any formula $\Phi(x_1, \ldots, x_n, Y_1, \ldots, Y_m)$ of *WS2Sy* into a formula $\Phi_0(X_1, \ldots, X_n, Y_1, \ldots, Y_m)$ of $WS2Sy_0$ such that

$$w_1, \ldots, w_n, M_1, \ldots, M_m \models \Phi \quad \text{iff} \quad \{w_1\}, \ldots, \{w_n\}, M_1, \ldots, M_m \models \Phi_0$$

and any formula $\Phi(X_1, \ldots, X_n, Y_1, \ldots, Y_m)$ of $WS2Sy_0$ (for an appropriate enumeration of the variables) into a formula $\Phi_1(x_1, \ldots, x_n, Y_1, \ldots, Y_m)$ of *WS2Sy* such that

$$\begin{aligned} M_1, \ldots, M_n, N_1, \ldots, N_m \models \Phi \quad \text{iff} \quad & \text{exists } m_i \text{ such that } M_i = \{m_i\} \text{ for all } i \text{ and} \\ & m_1, \ldots, m_n, N_1, \ldots, N_m \models \Phi_1 \end{aligned}$$

We say that a set S of n-tuples of sets of words over $\{1,2\}$ is definable in *WS2Sy* (resp. restricted *WS2Sy* or uniform *WS2Sy*) if there is *WS2Sy*-formula (resp. restricted *WS2Sy*-formula or uniform *WS2Sy*-formula) $\Phi(X_1, \ldots, X_n)$ such that

$$(M_1, \ldots, M_n) \in S \quad \text{iff} \quad M_1, \ldots, M_n \models \Phi$$

According to the translation between *WS2Sy* and $WS2Sy_0$ that we have mentioned above it does not matter whether we take *WS2Sy* or $WS2Sy_0$ to define the notion of definability.

Draft 13/11/2000

8 Recognizability and Definability

Before we can formulate the correspondence between definability in (subclasses of) *WS2Sy* and recognizability by (subclasses of) TACT we need translations of tuples of finite sets of words into trees, and vice versa.

Definition 11 *The* tree representation *of a finite set* $W \subset \{1,2\}^*$ *is the tree* $\overline{W} \in T(\{0,1\}^{(1)})$ *defined by*

$$\overline{W} = \begin{cases} \perp & \text{if } W = \emptyset \\ 0(\overline{1^{-1}W}, \overline{2^{-1}W}) & \text{if } W \neq \emptyset \text{ and } \epsilon \notin W \\ 1(\overline{1^{-1}W}, \overline{2^{-1}W}) & \text{if } W \neq \emptyset \text{ and } \epsilon \in W \end{cases}$$

where $1^{-1}W = \{w \mid 1w \in W\}$ *and* $2^{-1}W = \{w \mid 2w \in W\}$.

The tree representation $\overline{(W_1, \ldots, W_n)}$ *of an* n*-tuple* $(W_1, \ldots, W_n)$ *of finite subsets of* $\{1,2\}^*$ *is the tree* $t \in T(\{0,1\}^{(n)})$ *with* $t^{\langle i \rangle} = \overline{W_i}$ *for all* i.

The tree representation $\overline{S}$ *of a set of tuples of finite subsets of* $\{1,2\}^*$ *is* $\{\overline{(W_1, \ldots, W_n)} \mid (W_1, \ldots, W_n) \in S\}$.

Note that the tree representation of a tuple of sets is uniquely defined since $(\perp \ldots \perp)$ is a constant.

Example 3 *The tree representation of the set* $\{\epsilon, 2\}$ *is the right tree of Figure 1.*

The tree representation of the triple of sets $(\{\epsilon, 1\}, \{1\}, \{\epsilon, 2\})$ *is the left tree of Figure 1.*

Theorem 3 *Let* S *be a set of* n*-tuples of words.*

1. *If* S *is definable in restricted WS2Sy then* $\overline{S}$ *is TACT-recognizable.*
2. *If* S *is definable in uniform WS2Sy then* $\overline{S}$ *is TAT-recognizable.*
3. *If* S *is definable in WS2S then* $\overline{S}$ *is TA-recognizable ([11]).*

The proof of the first two items of this theorem is subject of Section 9, the last item is due to [11].

Definition 12 *The* tuple representation $\langle\!\langle t \rangle\!\rangle$ *of a tree* $t \in T(\Gamma^{(n)})$ *for finite* Γ, *say* $\Gamma = \{c_1, \ldots, c_m\}$, *is the* $n(m+1)$*-tuple of finite subsets of* $\{1,2\}^*$

$$(M_0^1, M_1^1, \ldots, M_m^1, \ldots, M_0^n, M_1^n, \ldots, M_m^n)$$

defined by

$$\begin{aligned} M_0^i &= \{w \mid \text{the } i\text{-th component of the root of } t\,|_w \text{ is } \perp\} \\ M_j^i &= \{w \mid \text{the } i\text{-th component of the root of } t\,|_w \text{ is } c_j\} \end{aligned}$$

The tuple representation $\langle\!\langle S \rangle\!\rangle$ *of a set of trees in* $T(\Gamma^{(n)})$ *is* $\{\langle\!\langle t \rangle\!\rangle \mid t \in S\}$.

Example 4 *The tuple representation of the middle tree of Figure 1 is*

$$(\{11,12,2\},\{\},\{\epsilon,1\}\ ,\ \{11,12,2\},\{\epsilon\},\{1\})$$

Theorem 4 *Let $S \subseteq T(\Gamma^{(n)})$.*

1. *If S is TACT-recognizable then $\langle\langle S\rangle\rangle$ is definable in restricted WS2Sy.*
2. *If S is TAT-recognizable then $\langle\langle S\rangle\rangle$ is definable in uniform WS2Sy.*
3. *If S is TA-recognizable then $\langle\langle S\rangle\rangle$ is definable in WS2S ([11]).*

The proof of the first two items of this theorem is subject of Section 10, the last item is due to [11].

9 Translating Formulae to Automata

In this section we prove Theorem 3 by translating a restricted $WS2Sy_0$-formula $\Phi(X_1,\ldots,X_n)$ into a TACT-automaton A_Φ over $\{0,1\}^{(n)}$ such that A_Φ is a TAT when Φ is uniform, and that A_Φ is a classical tree automaton when Φ is a *WS2S* formula.

The first step is to construct automata for the atomic formulae of $WS2Sy_0$. Automata for the atomic formulae $X_1 = X_2.0$, $X_1 = X_2.1$, $X_1 = \epsilon$ and $X_1 \subseteq X_2$ can be found in the literature (for instance [4]). It remains to give an automaton for the synchronization predicates:

The automaton A_n is defined as $(\{0,1\}, n, \{q_0,q_1\},\{q_1\},\Delta)$ where Δ contains the rules

$$\begin{array}{rcc}
(\bot\ldots\bot) & \rightarrow & q_0\\
(\bot_\ldots_)(q_0,q_0) & \rightarrow & q_0\\
(0_\ldots_)(q_0,q_0) & \rightarrow & q_0\\
(1_\ldots_)(q_0,q_0) & \rightarrow^{1.2=2.2\wedge\ldots\wedge 1.n=2.n} & q_1\\
(0_\ldots_)(q_0,q_1) & \rightarrow & q_1\\
(0_\ldots_)(q_1,q_0) & \rightarrow & q_1
\end{array}$$

Proposition 9 *Let S^n be the set of all n-tuples of sets of words such that $S \models Sy(X_1,X_2,\ldots,X_n)$. The language recognized by A_n is $\overline{S_n}$.*

Proof of Theorem 3: Let a restricted formula Φ be given. We can assume without loss of generality that every bound variable of Φ has exactly one binding quantifier and that no variable of Φ has both bound and free occurrences in Φ. We construct inductively an automaton for all subformulae of Φ.

We get an automaton for atomic formulae as in the case of classic tree automata, resp. by Proposition 9. Then we cylindrify to have components according to all free and bound variables of Φ. This yields, by Proposition 8, an automaton which does not test on components corresponding to bound variables. In case of composite formulae we conclude by Propositions 5 and 7.

If the formula Φ is uniform then the automata corresponding to the atomic formulae have (after cylindrification) only tests `True`, $1.1 = 2.1 \wedge \ldots \wedge 1.n = 2.n$ and $1.1 \neq 2.1 \vee \ldots \vee 1.n \neq 2.n$, where we assume for simplicity that the components corresponding to the global variables are $1, \ldots, n$. Since all closure operations on automata construct tests that are boolean combinations of tests of the input automata, the final automaton too has only tests of this form. Since the final automaton works on the components $1, \ldots, n$ it is indeed a TAT. $\square$

10 Translating Automata to Formulae

In this section we prove Theorem 4 by translating a TACT-automaton $A = (\Gamma, n, Q, Q_f, \Delta)$, where Γ has cardinality m, into a formula Φ with $n(m+1)$ free variables.

Let $\Gamma = \{c_1, \ldots, c_m\}$, we will use in addition the convention $c_0 = \perp$. Let $Q = \{q_1, \ldots, q_q\}$. We can assume w.l.o.g. that all constraints of the productions of A are conjunctions of equations and inequations since a boolean combination of constraints can be brought into an equivalent disjunctive normal form and a rule $f(q_1, q_2) \rightarrow^{c_1 \vee c_2} q$ can be split into two rules $f(q_1, q_2) \rightarrow^{c_1} q$ and $f(q_1, q_2) \rightarrow^{c_2} q$.

First we define some auxiliary formulas: The formula $\mathit{partition}(Y, Y_1, \ldots, Y_k)$ states that $(Y_1, \ldots, Y_k)$ is a partition of Y:

$$\mathit{partition}(Y, Y_1, \ldots, Y_k) \quad = \quad Y = Y_1 \cup \ldots \cup Y_k \wedge \bigwedge_{i \neq j} Y_i \cap Y_j = \emptyset$$

The formula $\mathit{term}(X, X_1, \ldots, X_{n(m+1)})$ states that, for some tree $t \in T(\Gamma^{(n)})$, X is the set of occurrence of t and $(X_1, \ldots, X_{n(m+1)}) = \langle\langle t \rangle\rangle$:

$$\begin{aligned}\mathit{term}(X, X_1, \ldots, X_{n(m+1)}) \quad = \quad & \bigwedge_{i=0\ldots n-1} \mathit{partition}(X, X_{i(m+1)+1}, \ldots, X_{i(m+1)+m+1}) \\ & \wedge \forall x, y (x \leq y \wedge y \in X \rightarrow x \in X) \\ & \wedge \forall x (x \in X_0^1 \cap \ldots \cap X_0^n \rightarrow (x0 \notin X \wedge x1 \notin X)) \\ & \wedge \forall x (x \notin X_0^1 \cap \ldots \cap X_0^n \rightarrow (x0 \in X \wedge x1 \in X))\end{aligned}$$

The first clause of the formula term says that each component of every occurrence is marked with exactly one symbol and that X is the set of occurrences, the second clause says that the set of occurrences is prefix-closed, the third clause says that any occurrence marked $(\perp \ldots \perp)$ is a leaf and the forth clause that every occurrence not marked with $(\perp \ldots \perp)$ has two subtrees. Finally, the formula is

$$\begin{aligned}\phi_A(X_0^1, X_1^1, \ldots, X_m^1, \ldots, X_0^n, X_1^n, \ldots, X_m^n) \quad & = \exists X, Y_1, \ldots, Y_k \Big(\\ & \mathit{term}(X, X_0^1, \ldots, X_m^n) \\ \wedge \quad & \mathit{partition}(X, Y_1, \ldots, Y_q) \\ \wedge \quad & \bigvee_{q_i \in Q_f} \epsilon \in Y_i\end{aligned}$$

$$
\begin{aligned}
\wedge \quad & \forall x (x \in X_0^1 \cap \ldots \cap X_0^n \rightarrow \bigvee_{(\perp \cdots \perp \rightarrow q_i \in \Delta)} x \in Y_i) \\
\wedge \quad & \bigwedge_{(i_1 \ldots i_n) \neq (0 \ldots 0)} \forall x (x \in X_{i_1} \cap \ldots \cap X_{i_n} \rightarrow \bigvee_{(i_1 \ldots i_n)(q_j, q_k) \rightarrow^c q_l \in \Delta} (\\
& \quad x \in Y_l \wedge x1 \in Y_j \wedge x2 \in Y_k \\
& \quad \wedge \bigwedge_{1.i = 2.i \in c} Sy(x, X_0^i, \ldots, X_m^i) \wedge \bigwedge_{1.i \neq 2.i \in c} \neg Sy(x, X_0^i, \ldots, X_m^i)))\Big)
\end{aligned}
$$

In this formula, the first conjunct says that the free variables are the encoding of a $\Gamma^{(n)}$-tree with set of occurrences X. The second conjunct says that every occurrence of the tree is assigned a state. The third conjunct expresses that the root of the tree is assigned an accepting state, and the last two conjuncts express that the assignment is according to the transitions rules of the automaton: The forth conjunct covers the case of the constant $(\perp \ldots \perp)$ and the complex last conjunct the case of a binary function symbol.

Note that the formula ϕ_A is restricted. If the automaton is in fact a TAT automaton then the formula ϕ_A is equivalent to a uniform *WS2Sy*-formula since

$$Sy(x, X_0^i, \ldots, X_m^i) \wedge Sy(x, X_0^j, \ldots, X_m^j)$$

is equivalent to the formula

$$Sy(x, X_0^i, \ldots, X_m^i, X_0^j, \ldots, X_m^j)$$

Finally, if the automaton A is a classical tree automaton then ϕ_A contains no synchronization predicate and is hence a *WS2S*-formula.

11 Other Classes of Automata with Tests

So far we have only considered the class of automata with tests between brothers introduced in [1]. We can easily extend the results of this paper to classes of automata with "deep" tests that can perform tests like $11.i = 222.i$, where a tree t satisfies this test if $(t\mid_{11})^{\langle i \rangle} = (t\mid_{222})^{\langle i \rangle}$ [6]. This generalization is easily achieved by defining a generalization of *WS2Sy* with stronger synchronization predicates. These classes of automata are, however, of limited interest since already the class of automata with tests between cousin positions (that is $p.i = q.i$ where p and q have length 2) has an undecidable emptiness problem [10].

There is a very interesting class of tree automata with tests that has a decidable emptiness problem: the class of *reduction automata* [3] and *generalized reduction automata* [2]. These automata allow "deep" (in the above sense) tests but come with a syntactic restriction that ensures that on every branch of a tree only a bounded number of equality tests can be performed. Hence, the undecidability proof of this paper does not directly apply to a generalization of these automata to component-wise tests since we used the fact that we can perform an unbounded number of equality tests on the left spine of the tree (see Section 4). Hence, it is still possible that the results known for (generalized) reduction automata can be lifted to component-wise tests, and that a corresponding logic with a restriction on the set of admissible formulae can be found.

Note, however, that the proof of Proposition 6 applies to generalized reduction autoamata since only one test is performed. Hence, the class of langages recognizable by component-wise generalized reduction automata is not closed under arbitrary projections.

Accknowledgements

Discussions with Joachim Niehren and Sophie Tison are acknowledged. Thanks to Zhendong Su for reminding me of a bug in an earlier version of this paper.

Bibliography

[1] Bruno Bogaert and Sophie Tison. Equality and disequality constraints on brother terms in tree automata. In P. Enjalbert, A. Finkel, and K. W. Wagner, editors, *9th Annual Symposium on Theoretical Aspects of Computer Science*, volume 577 of *Lecture Notes in Computer Science*, pages 161–171, Paris, France, 1992. Springer-Verlag.

[2] Anne-Cécile Caron, Hubert Comon, Jean-Luc Coquidé, Max Dauchet, and Florent Jacquemard. Pumping, cleaning and symbolic constraints solving. In *Proc. Int. Conference on Algorithms, Languages and Programming*, pages 436–449, 1994.

[3] Anne-Cécile Caron, Jean-Luc Coquidé, and Max Dauchet. Encompassment properties and automata with constraints. In Claude Kirchner, editor, *5th International Conference on Rewriting Techniques and Applications*, volume 690 of *Lecture Notes in Computer Science*, pages 328–342, Montreal, Canada, June 1993. Springer-Verlag.

[4] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata. Techniques and Applications, April 1999. Available at `http://www.grappa.univ-lille3.fr/tata/`.

[5] Aart Middeldorp and Yoshihito Toyama. Completeness of combinations of constructor systems. Technical Report R9058, CWI, October 1990.

[6] J. Mongy. *Transformations de noyaux reconnaissables d'arbres. Forêts RATEG.* PhD thesis, Laboratoire d'Informatique Fondamentale de Lille, Université Lille 1, Villeneuve d'Asq, France, 1981. In French.

[7] Emil L. Post. A variant of a recursively unsolvable problem. *Bulletin of the AMS*, 52:264–268, 1946.

[8] Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141:1–35, 1969.

[9] Franck Seynhaeve. *Automates, Réécriture et Contraintes : Résultats de Décidabilité et d'Indécidabilité.* PhD thesis, Université de Lille 1, Villeneuve d'Ascq, France, May 1999. In French.

[10] Franck Seynhaeve, Marc Tommasi, and Ralf Treinen. Grid structures and undecidable constraint theories. In Michel Bidoit and Max Dauchet, editors, *Theory and Practice of Software Development*, volume 1214 of *Lecture Notes in Computer Science*, pages 357–368, Lille, France, April 1997. Springer-Verlag. Extended version to appear in *Theoretical Computer Science*.

[11] J.W. Thatcher and J.B. Wright. Generalized finite automata with an application to a decision problem of second-order logic. *Math. Systems Theory*, 2:57–82, 1968.

[12] Ralf Treinen. Predicate logic and tree automata with tests. In Jerzy Tiuryn, editor, *Foundations of Software Science and Computation Structures*, volume 1784 of *Lecture Notes in Computer Science*, pages 329–343, Berlin, Germany, March 2000. Springer-Verlag.