Automata for the modal μ-calculus and related results

David Janin
LaBRI[1]
U.E.R. de Mathématiques et d'Informatique
Université de Bordeaux I
351, Cours de la Libération,
FR-33405 Talence Cedex, France
e-mail: janin@labri.u-bordeaux.fr

Igor Walukiewicz
BRICS[2,3]
Department of Computer Science
University of Aarhus
Ny Munkegade
DK-8000 Aarhus C, Denmark
e-mail: igw@daimi.aau.dk

Abstract. The propositional μ-calculus as introduced by Kozen in [4] is considered. The notion of disjunctive formula is defined and it is shown that every formula is semantically equivalent to a disjunctive formula. For these formulas many difficulties encountered in the general case may be avoided. For instance, satisfiability checking is linear for disjunctive formulas. This kind of formula gives rise to a new notion of finite automaton which characterizes the expressive power of the μ-calculus over all transition systems.

1 Introduction

We consider the propositional μ-calculus as introduced by Kozen [4]. Subsequent research showed that the μ-calculus is an interesting logic when specification and verification is concerned. It is an expressive logic; on binary trees it is as expressive as the monadic second order logic of two successors [8, 3]. On the other hand, if computational complexity is concerned, the propositional μ-calculus is not much more difficult than classical propositional logic as its decidability problem is EXPTIME complete. Because of these and other features the logic is considered to be one of the most interesting logics of programs.

The two main results we present here are:

- Definition of the class of *disjunctive formulas* and the proof that every formula is equivalent to a disjunctive formula.
- Characterization of the μ-calculus by means of a new kind of automata on transition systems.

The methods developed here allow us also to obtain other know results as corollaries. In particular we show that our results subsume the results of Niwinski and Emerson and Jutla [8, 3]. We obtain yet another proof of Rabin's complementation lemma.

It was already discovered by Kozen in [4] that the interplay of all the connectives of the μ-calculus raises some challenging difficulties. Here we try to analyze these difficulties. Our first step is to give alternative "operational" semantics of the μ-calculus formulas. We look at a formula as an automaton-like device checking a property of the unwinding of a model from a given state.

[1] **La**boratoire **B**ordelais de **R**echerche en **I**nformatique
[2] **B**asic **R**esearch **i**n **C**omputer **S**cience,
Centre of the Danish National Research Foundation.
[3] On leave from: Institute of Informatics, Warsaw University, Banacha 2,
02-097 Warsaw, POLAND

If we are to check that $\langle a\rangle\alpha$ holds, we choose an edge from this state labeled by a leading to a state where α holds. If we are to check that $\alpha\vee\beta$ holds, we choose (nondeterministically) one of the disjuncts. If we are to check $\sigma X.\alpha(X)$, when σ is μ or ν, we try the equivalent formula $\alpha(\sigma X.\alpha(X))$. The distinction between least (μ) or greatest (ν) fixed points is achieved using suitable infinitary acceptance conditions. When we check $\alpha \wedge \beta$ we must check that this state satisfies both α and β.

While disjunctions act like nondeterministic choices, conjunctions act rather like universal branching of alternating automata. Such an alternating behavior of conjunctions is the source of many difficulties.

From automata theory we know that alternating automata are equivalent to nondeterministic ones [7]. This suggests that every formula should be equivalent to a formula which does not have universal branching behaviors represented by conjunctions. Of course we cannot discard conjunctions completely from positive formulas as shown by the formula $(\langle a\rangle p) \wedge [a](p \vee q)$. Note that the conjunction in this formula does not act as a universal branching. It is rather like an implicit conjunction from (usual, not alternating) automata on trees where transition relation forces the right son to be labeled by one state and the left son by another one. Such a kind of implicit conjunction is the only form of conjunction that appears in the fixpoint notation for sets of trees defined by Niwiński [8]. It was proved that this fixpoint language has the same expressive power as the monadic second order logic of n successors. Hence adding explicit conjunction to this language will not increase its expressive power.

These considerations lead to the notion of *disjunctive formulas* which are formulas where the role of conjunction is restricted so that it never acts as an universal branching. We show that every formula is equivalent to a disjunctive formula. It turns out that the satisfiability problem is linear for disjunctive formulas. There is also a straightforward method of model construction for such formulas. In comparison, the satisfiability problem for arbitrary formulas is EXPTIME-complete and the only known method of model construction involves nontrivial reduction to Rabin automata on infinite trees.

Disjunctive formulas also hint the possibility of giving automata-like characterization of the μ-calculus. In [8, 3] it was shown that over binary trees the μ-calculus is as expressive as the monadic second order logic (MS-logic for short). Nevertheless it is not true that over arbitrary transition systems the μ-calculus is as expressive as MS-logic. It is not even true when we restrict the class of models to trees with nodes of finite but unbounded degrees. In both cases μ-calculus is strictly weaker than MS-logic.

Notice that these general trees can be encoded into binary trees. For a given μ-calculus formula we can construct, say, a Rabin automaton which recognizes codings of the models of the formula. But this is only a one way mapping. It is not true that for every Rabin automaton there is a μ-calculus formula having as models exactly the transition systems of which codings are accepted by the automaton.

We propose a notion of automaton of which expressive power is exactly the same as the μ-calculus. Restricted to binary trees these automata are just alternating automata with so-called parity conditions [6, 3]. They are more general because they admit runs over arbitrary transition systems. We show that there are direct translations between disjunctive formulas and this kind of automata. This proves that the set of recognizable languages induced by our notion of automata is closed under all boolean operators hence also complementation. If we consider μ-calculus restricted to binary trees then our constructions give us ordinary (non alternating) parity automata on trees. This way we obtain a proof of Rabin's complementation lemma and the results from [8, 3].

The paper is organized as follows. We start by giving basic definitions including a new formula constructor and the notion of binding functions. In the second section we describe operational semantics of formulas. In the third we present the notion of disjunctive formulas and prove properties of such formulas. Next section is devoted to the new kind of automata which we call μ-automata.

2 Preliminary definitions

Let $\mathit{Prop} = \{p, q, \ldots\} \cup \{\bot, \top\}$ be a set of propositional letters, $\mathit{Var} = \{X, Y, \ldots\}$ a set of variables and $\mathit{Act} = \{a, b, \ldots\}$ a set of actions. Formulas of the μ-calculus over these sets can be defined by the following grammar:

$$F := \mathit{Prop} \mid \neg \mathit{Prop} \mid \mathit{Var} \mid F \vee F \mid F \wedge F \mid \langle \mathit{Act} \rangle F \mid [\mathit{Act}] F \mid \mu \mathit{Var}.F \mid \nu \mathit{Var}.F$$

Note that we allow negations only before propositional constants. As we will be interested mostly in closed formulas this is not a restriction. All the results presented here extend to the general case when negation before variables is also allowed, restricting as usual to positive occurrences of bound variables.

In the following, $\alpha, \beta, \gamma, \ldots$ will denote formulas, and $A, B, C \ldots$ will denote finite sets of formulas. We shall use σ to denote either μ or ν. Variables, propositional constants and their negations will be called *literals*.

Formulas are interpreted in *transition systems* of the form $\mathcal{M} = \langle S, R, \rho \rangle$, where:

- S is a nonempty set of states,
- $R : \mathit{Act} \to \mathcal{P}(S \times S)$ is a function assigning a binary relation on S to each action in Act.
- $\rho : \mathit{Prop} \to \mathcal{P}(S)$ is a function assigning a set of states to every propositional constant.

For a given model $\mathcal{M}$ and an assignment $\mathit{Val} : \mathit{Var} \to \mathcal{P}(S)$, the set of states in which a formula α is true, denoted $\|\alpha\|^{\mathcal{M}}_{\mathit{Val}}$, is defined inductively as follows (we will omit superscript $\mathcal{M}$ when it causes no ambiguity):

$$
\begin{aligned}
\|p\|_{Val} = \rho(p) \quad \|\bot\|_{Val} &= \emptyset \quad \|\top\|_{Val} = S \\
\|\neg p\|_{Val} &= S - \rho(p) \\
\|X\|_{Val} &= Val(X) \\
\|\alpha \vee \beta\|_{Val} &= \|\alpha\|_{Val} \cup \|\beta\|_{Val} \\
\|\alpha \wedge \beta\|_{Val} &= \|\alpha\|_{Val} \cap \|\beta\|_{Val} \\
\|\langle a\rangle\alpha\|_{Val} &= \{s : \exists s'.(s,s') \in R(a) \wedge s' \in \|\alpha\|_{Val}\} \\
\|[a]\alpha\|_{Val} &= \{s : \forall s'.(s,s') \in R(a) \Rightarrow s' \in \|\alpha\|_{Val}\} \\
\|\mu X.\alpha(X)\|_{Val} &= \bigcap \{S' \subseteq S : \|\alpha\|_{Val[S'/X]} \subseteq S'\} \\
\|\nu X.\alpha(X)\|_{Val} &= \bigcup \{S' \subseteq S : S' \subseteq \|\alpha\|_{Val[S'/X]}\}
\end{aligned}
$$

here $Val[S'/X]$ is the valuation such that, $Val[S'/X](X) = S'$ and $Val[S'/X](Y) = Val(Y)$ for $Y \neq X$. We shall write $\mathcal{M}, s, Val \models \alpha$ when $s \in \|\alpha\|^{\mathcal{M}}_{Val}$.

2.1. Definition (Binding). We call a formula *well named* iff every variable is bound at most once in the formula and free variables are distinct from bound variables. For a variable X bound in a well named formula α there exists a unique subterm of α of the form $\sigma X.\beta(X)$, from now on called the *binding definition of X in α* and denoted $\mathcal{D}_\alpha(X)$. We will omit subscript α when it causes no ambiguity. We call X a *μ-variable* when $\sigma = \mu$, otherwise we call X a *ν-variable*.

The function $\mathcal{D}_\alpha$ assigning to every bound variable its *binding definition* in α will be called the *binding function* associated with α.

2.2. Definition (Dependency order). Given a formula α we define the *dependency order* over the bound variables of α, denoted $\leq_\alpha$, as the least partial order relation such that if X occurs free in $\mathcal{D}_\alpha(Y)$ then $X \leq_\alpha Y$. We will say that a bound variable Y depends on a bound variable X in α when $X \leq_\alpha Y$.

2.3. Definition. Variable X in $\mu X.\alpha(X)$ is *guarded* iff every occurrence of X in $\alpha(X)$ is in the scope of some modality operator $\langle\ \rangle$ or $[\ ]$. We say that a formula is guarded iff every bound variable in the formula is guarded.

2.4. Proposition (Kozen). *Every formula is equivalent to some guarded formula.*

This proposition allows us to restrict ourselves to guarded, well-named formulas. From now on, we shall only consider formulas of this kind. This restriction is not essential to what follows but simplifies definitions substantially.

In construction of our tableaux we shall distinguish some occurrences of conjunction which should not be reduced by ordinary (*and*) rule.

2.5. Definition. We extend the syntax of the μ-calculus by allowing new construction of the form $a \to A$, where a is an action and A is a finite set of formulas. Such a formula will be semantically equivalent to $\bigwedge\{\langle a\rangle\alpha | \alpha \in A\} \wedge [a] \bigvee A$. Namely state q satisfies formula $a \to A$ when any formula of A is satisfied by at least one a-successor of state q, and any a-successor of state q satisfies at least one formula of A. We adopt the convention that the conjunction of the empty set of formulas is the formula $\top$ and disjunction of the empty set is $\bot$.

2.6. Remark. A formula $\langle a\rangle\alpha$ is equivalent to $a \to \{\alpha, \top\}$ and a formula $[a]\alpha$ is equivalent to $a \to \{\alpha\} \vee a \to \emptyset$. It follows that any formula can be written with this new construction in place of modalities. All the notions defined in this section like bound variable definitions, guardedness, etc. extend to formulas with this new construction.

3 "Operational semantics"

Here we will describe alternative "operational" semantics for the formulas of the μ-calculus. We will show that a formula is satisfied in a state s of a structure $\mathcal{M}$ with a valuation *Val* iff there is a consistent marking of a tableau for the formula. This characterization gives us a tool for proving equivalence of formulas.

Let γ be a well-named, guarded formula where construction $a \to A$ is used instead of $\langle a\rangle\alpha$ and $[a]\alpha$ constructions.

3.1. Definition. We define the system of tableau rules $\mathcal{S}^\gamma$ parameterized by a formula γ, or rather its binding function:

$$(and)\ \frac{\{\alpha, \beta, C\}}{\{\alpha \wedge \beta, C\}} \quad (or)\ \frac{\{\alpha, C\} \quad \{\beta, C\}}{\{\alpha \vee \beta, C\}}$$

$$(\mu)\ \frac{\{\alpha(X), C\}}{\{\mu X.\alpha(X), C\}} \quad (\nu)\ \frac{\{\alpha(X), C\}}{\{\nu X.\alpha(X), C\}}$$

$$(reg)\qquad \frac{\{\alpha(X), C\}}{\{X, C\}} \qquad \begin{array}{l}\text{whenever } X \text{ is a bound variable of } \gamma\\ \text{and } \mathcal{D}_\gamma(X) = \sigma X.\alpha(X)\end{array}$$

$$(mod)\quad \frac{\{\alpha\} \cup \{\bigvee B : a \to B \in \{C\}, B \neq A\} \ \text{ for every } a \to A \in \{C\}, \alpha \in A}{\{C\}}$$

with $\bigvee \emptyset$ interpreted as $\perp$.

3.2. Remark. The rule (mod) has as many premises as there are formulas in the sets A such that $a \to A \in C$. For instance

$$\frac{\{\alpha_1, \alpha_3\} \quad \{\alpha_2, \alpha_3\} \quad \{\alpha_1 \vee \alpha_2, \alpha_3\} \quad \{\beta_1\} \quad \{\beta_2\}}{\{a \to \{\alpha_1, \alpha_2\}, a \to \{\alpha_3\}, b \to \{\beta_1, \beta_2\}\}}$$

is an instance of the rule.

3.3. Remark. We see applications of the rules as a process of reduction. Given a finite set of formulas C we want to derive, we look for the rule the conclusion of which matches our set. Then we apply the rule and obtain the assumptions of the instance of the rule whose conclusion is C.

3.4. Definition. A *tableau* for γ is a pair $\langle T, L\rangle$, where T is a tree and L is a labeling function such that:

1. the root of T is labeled by $\{\gamma\}$,

2. the sons of any node n are created and labeled according to the rules of system $\mathcal{S}^\gamma$, with *rule (mod) applied only when no other rule is applicable.*

Leaves and nodes where (*mod*) rule was applied will be called *modal nodes*. The root of $\mathcal{T}$ and sons of modal nodes will be called *choice nodes*. We will say that m is *near* n iff there is a path from n to m in the tableau without an application of modal rule.

3.5. Remark. Returning to our example of an instance of the rule (*mod*) from Remark 3.2. If a node n is labeled by the conclusion of this instance then it has five sons labeled by corresponding assumptions. We will call a son obtained by *reducing* an action a an *a-son*. In our example n has three a-sons and two b-sons. Node n is a modal node, its sons are choice nodes.

3.6. Definition (Marking). For a tableau $\mathcal{T} = \langle T, L \rangle$ we define its *marking* with respect to a structure $\mathcal{M} = \langle S, R, \rho \rangle$ and state s_0 to be a relation $M \subseteq S \times T$ satisfying the following conditions:

1. $(s_0, r) \in M$, where r is a root of T.
2. If some pair (s, m) belongs to M and a rule other than (*mod*) was applied in m, then for some son n of m, $(s, n) \in M$.
3. If $(s, m) \in M$ and rule (*mod*) was applied in a node m then for every action a for which exists a formula of the form $a \to A$ in $L(n)$:
   (a) for every a-son n of m there exists a state t such that $(s, t) \in R(a)$ and $(t, n) \in M$.
   (b) for every state t such that $(s, t) \in R(a)$ there exists an a-son n of m such that $(t, n) \in M$.

3.7. Definition (Trace). Given a path $\mathcal{P}$ of a tableau $\mathcal{T} = \langle T, L \rangle$, a *trace* on $\mathcal{P}$ will be a function F assigning a formula to every node in some initial segment of $\mathcal{P}$ (possibly to the whole $\mathcal{P}$), satisfying the following conditions:

- If $F(n)$ is defined then $F(n) \in L(n)$.
- Let m be a node with $F(m)$ defined and let $n \in \mathcal{P}$ be a son of m. If a rule applied in m does not reduce the formula $F(m)$ then $F(n) = F(m)$. If $F(m)$ is reduced in m then $F(n)$ is one of the results of the reduction. This should be clear for all the rules except (*mod*). In case m is a modal node and n is labeled by $\{\delta\} \cup \{\bigvee B : a \to B \in C, B \neq A\}$ for some $a \to A \in L(m)$ and $\delta \in A$, then $F(n) = \delta$ if $F(m) = a \to A$ and $F(n) = \bigvee B$ if $F(m) = a \to B$ for some $a \to B \in C$, $B \neq A$. Traces from other formulas end in node m.

3.8. Definition (μ-trace). We say that there is a *regeneration* of a variable X on a trace F on some path iff for some node m and its son n on the path $F(m) = X$ and $F(n) = \alpha(X)$ with $\mathcal{D}_\gamma(X) = \sigma X.a(X)$, i.e. rule (*reg*) was applied to variable X.

We call a trace *μ-trace* iff it is an infinite trace (defined for the whole path) on which the smallest variable, with respect to $\leq_\alpha$ ordering, regenerated i.o. is a μ-variable. Similarly a trace will be called a *ν-trace* iff it is an infinite trace where the the smallest variable, with respect to $\leq_\alpha$ ordering, regenerated i.o. is a ν-variable.

3.9. Remark. Every infinite trace is either a μ-trace or a ν-trace because all the rules except regenerations decrease the size of formulas and formulas are guarded hence every formula is eventually reduced. Observe that even though $\leq_\alpha$ is a partial ordering there is always the least variable required in the above definition.

3.10. Definition (Consistent marking). Using notation from the Definition 3.6, a marking M of $\mathcal{T}$ with respect to $\mathcal{M}$ and s is *consistent* with respect to $\mathcal{M}$, s, Val iff it satisfies the following conditions:

local consistency for every modal node m and state t, if $(t, m) \in M$ then $\mathcal{M}, t, Val \models A'$, where A' is the set of all the literals occurring in $L(m)$,

global consistency for every path $\mathcal{P} = n_0, n_1, \ldots$ of T such that for every $i = 0, 1, \ldots$ there exist s_i with $(s_i, n_i) \in M$ there is no μ-trace on $\mathcal{P}$.

The following theorem gives a characterization of satisfiability by means of consistent markings.

3.11. Theorem. *A positive guarded formula γ is satisfied in a structure $\mathcal{M}$, state s and valuation Val iff there exists a marking M of a tableau for γ consistent with $\mathcal{M}$, s, Val.*

Proof. The proof uses transfinite approximations of fixed point expressions and signatures in the style of those defined in [9].

4 A disjunctive normal form theorem

In this section we define a notion of *disjunctive formula* and show that every formula is equivalent to a disjunctive formula.

4.1. Definition. The set of *disjunctive formulas*, $\mathcal{F}_d$ is the smallest set defined by the following clauses:

1. every variable is a disjunctive formula,
2. if $\alpha, \beta \in \mathcal{F}_d$ then $\alpha \vee \beta \in \mathcal{F}_d$; if moreover X occurs only positively in α and not in the context $X \wedge \gamma$ for some γ, then $\mu X.\alpha, \nu X.\alpha \in \mathcal{F}_d$,
3. formula $\alpha_1 \wedge \ldots \wedge \alpha_n$ is a disjunctive formula provided that every α_i is either a literal or a formula of a form $a \to A$ with $A \subseteq \mathcal{F}_d$. Moreover we require that for any action a there can be at most one conjunct of the form $a \to A$ among $\alpha_1, \ldots, \alpha_n$.

4.2. Remark. Many properties can be "naturally" expressed by disjunctive formulas. For example the properties q holds almost always and q holds infinitely often can be written as the following disjunctive formulas:

$$\mu X.((a \to \{X\}) \vee \nu Y.(q \wedge a \to \{Y\})) \qquad \nu X.\mu Y.((q \wedge a \to \{X\}) \vee a \to \{Y\})$$

4.3. Remark. Modulo the order of application of (*and*) rules, disjunctive formulas have unique tableaux. Moreover on any infinite path there is one and only one infinite trace.

4.4. Theorem. *For every formula there exists an equivalent disjunctive formula.*

Proof. Let $\mathcal{T}$ be a regular tableau for a formula γ. A graph obtained from a tree by adding edges from some leaves to their ancestors will be called a *tree with back edges*. Added edges will be called *back edges*. First one needs to prove:

4.5. Lemma. *It is possible to construct a finite tree with back edges $\mathcal{T}_l = \langle T_l, L_l \rangle$, satisfying the following conditions:*

1. *$\mathcal{T}_l$ unwinds to $\mathcal{T}$.*
2. *Every node to which a back edge points can be assigned color magenta or navy in such a way that for any infinite path from the unwinding of $\mathcal{T}_l$ we have: there is a μ-trace on the path iff the highest node of $\mathcal{T}_l$ through which the path goes i.o. is colored magenta.*

To prove the lemma one takes a deterministic parity automaton $\mathcal{A}$ [6, 3] which recognizes paths of $\mathcal{T}$ having μ-trace on them. Then one can run $\mathcal{A}$ on every path of $\mathcal{T}$. This gives an assignment of states of $\mathcal{A}$ to nodes of $\mathcal{T}$. Obtained tree is still regular and we can use parity condition to present it in the desired form.

Next one constructs from $\mathcal{T}_l$ a disjunctive formula $\widehat{\gamma}$ which has a tableau equivalent to $\mathcal{T}$. The construction starts in the leaves of the tree and proceeds to the root. All back edges leading to a node n are assigned the same variable X_n and the color of the node is used to decide which fixpoint operator should be used to close this variable when we reach n in our construction.

In [9] the general technique of model construction for the μ-calculus formulas was described. Till now it remains essentially the only known technique for model construction (see [5] for different approach). It turns out that in case of disjunctive formulas model construction is much easier. This is described in the following theorem.

4.6. Theorem. *A closed disjunctive formula α is satisfiable iff the formula β obtained from α by replacing all occurrences of μ-variables by $\bot$ and all ν-variables by $\top$ is satisfiable.*

Proof. Let $\mathcal{T}_\alpha$ and $\mathcal{T}_\beta$ be tableaux for α and β respectively. There exists a function h which for any node of $\mathcal{T}_\alpha$ gives us the corresponding node of $\mathcal{T}_\beta$, mapping variables to corresponding constants. This situation is schematically represented in Figure 1. It is quite easy to show that if α is satisfiable then β is satisfiable. This can be done by induction on the structure of α.

Conversely, assume β is satisfiable and let M be a minimal (w.r.t. inclusion) marking of $\mathcal{T}_\beta$ consistent w.r.t. some arbitrary model $\mathcal{M}$ for β. It is quite easy, using h^{-1} and the modal nodes occurring in M, to build a regular model of β together with a marking M' of $\mathcal{T}_\alpha$ consistent with that model such that no μ-variable is ever regenerated on any path of that marking.

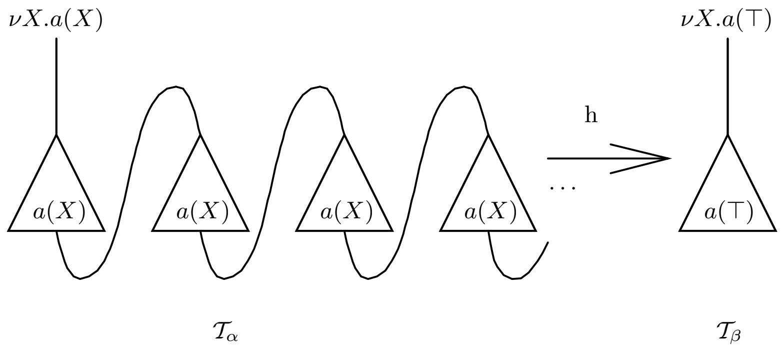

Fig. 1. Relation between $\mathcal{T}_\alpha$ and $\mathcal{T}_\beta$

Because β in the above theorem is a disjunctive formula without fixpoint operators we have:

4.7. Corollary. *Satisfiability checking for disjunctive formulas can be done in linear time.*

5 Automata for the μ-calculus

The question we ask here is what concept of automaton characterizes the expressive power of the μ-calculus over transition systems.

The idea of constructing automata able to deal with arbitrary branching was already considered in [1] but the construction proposed there would give us to big expressive power.

5.1. Definition. A *μ-automaton* is a tuple $\mathcal{A} = \langle Q, \Sigma_p, \Sigma_a, q_0, \delta, \Omega \rangle$ where: Q is a finite set of states, Σ_p, Σ_a are finite alphabets called *proposition* and *action* alphabets respectively, $q_0 \in Q$ is the initial state, $\delta : Q \to \mathcal{P}(\Sigma_p \times (\Sigma_a \dot{\to} \mathcal{P}(Q)))$ is a transition function and $\Omega : Q \to \mathcal{N}$ is a indexing function defining acceptance conditions. Here, $\Sigma_a \dot{\to} \mathcal{P}(Q)$ denotes the set of partial functions from Σ_a to $\mathcal{P}(Q)$.

5.2. Definition. A *labeled transition system restricted* to Σ_a and Σ_p is a tuple

$$\mathcal{T} = \langle S, s_0 \in S, R : \Sigma_a \to \mathcal{P}(S \times S), \rho : S \to \Sigma_p \rangle$$

where s_0 is the *initial state*, ρ defines labeling of states and R defines edges between states together with their labeling.

5.3. Remark. From given finite sets of actions Σ_a and literals L_p and given transition system with some valuation it is straightforward to construct a labeled transition system restricted to Σ_a and $\Sigma_p = \mathcal{P}(L_p)$. There is also an obvious translation in the opposite direction assuming, say, that all variables not in Σ_p are always false and that there are no other edges.

5.4. Definition. For a restricted transition system $\mathcal{T}$ and an automaton $\mathcal{A}$ as the above we define a *run* of $\mathcal{A}$ on $\mathcal{T}$ to be an infinite labeled tree T satisfying the following conditions:

- The root is labeled by (q_0, s_0).
- For any node of the tree labeled (q, s) there is a pair $(p, f) \in \delta(q)$ such that $p = \rho(s)$ and for every action a in the domain of f (assumed partial):
  - ⋆ for every $q_a \in f(a)$ there is a son labeled (q_a, t) for some t with $(s, t) \in R(a)$,
  - ⋆ for every t with $(s, t) \in R(a)$ there is $q_a \in f(a)$ and a son labeled (q_a, t)

The run is accepting iff for any path $\mathcal{P}$, $\min\{\Omega(q) : q \text{ appears i.o. on } \mathcal{P}\}$ is even.

We would like to prove that automata of this kind have exactly the same expressive power as the μ-calculus.

5.5. Theorem. *For any disjunctive formula γ there is an equivalent μ-automaton.*

Proof. Let $\mathcal{T}$ be a tree with back edges unwinding into a tableau for γ. We build an automaton recognizing exactly the models of γ. Its set of states is $Q = (CN \times BV)$ where CN is the set of choice nodes (see Definition 3.4). Transitions are build from a state (n, X) to a state (m, Y) whenever m is near n, with Y the smallest variables regenerated on the path from n to m or $*$ if there is no such.

5.6. Theorem. *For any μ-automaton $\mathcal{A}$ we can construct an equivalent disjunctive formula.*

Proof. Construct a tree with back edges from the automaton in the style of Lemma 4.5. Then construct a formula from this tree.

5.7. Corollary. *μ-automata are closed under all the connectives of the μ-calculus. In particular they are closed under negation.*

5.8. Remark. A transition system is called *labeled binary tree* iff every state has exactly two transitions: one labeled "l" and one labeled "r". Exactly the same argument as in Theorem 4.4 shows that over labeled binary trees every formula of the μ-calculus is equivalent to a disjunctive formula where each special conjunction is of the form $l \to \{\alpha_l\} \wedge r \to \{\alpha_r\} \wedge \Gamma$ with Γ containing only literals. It follows that μ-automata built from such formulas correspond exactly to usual (non alternating) automata with so called *parity condition*[6, 3]. This shows that μ-calculus is equivalent to Rabin automata and that μ-automata are closed under boolean connectives. In particular this gives a proof of Rabin's complementation lemma.

References

1. O. Bernholtz and O. Grumberg. Branching time temporal logic and amorphous tree automata. In *Proc. 4th Conference on Concurrency Theory*, volume 715 of *Lecture Notes in Computer Science*, pages 262–277. Springer-Verlag, 1993.
2. E.A. Emerson and C.S. Jutla. The complexity of tree automata and logics of programs. In *29th IEEE Symp. on Foundations of Computer Science*, 1988.
3. E.A. Emerson and C.S. Jutla. Tree automata, mu calculus and determinacy. In *Proc. FOCS 91*, 1991.
4. D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
5. D. Kozen. A finite model theorem for the propositional μ-calculus. *Studa Logica*, 47(3):234–241, 1988.
6. A.W. Mostowski. Regular expressions for infinite trees and a standard form of automta. In A. Skowron, editor, *Fith Symposium on Computation Theory*, volume 208 of *LNCS*, pages 157–168, 1984.
7. D.E. Muller and P.E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54:267–276, 1987.
8. D. Niwiński. Fixed points vs. infinite generation. In *Proc. 3rd. IEEE LICS*, pages 402–409, 1988.
9. R.S. Street and E.A. Emerson. An automata theoretic procedure for the propositional mu-calculus. *Information and Computation*, 81:249–264, 1989.