Project Thesis

A Proof System for the Modal μ-calculus

Oliver Friedmann

Supervisor

Dr. Martin Lange
Prof. Dr. Martin Hofmann

Contents

1 Introduction

The modal μ-calculus is a modal logic that comprises constructs of propositional and first order logic and operators of temporal logic. It is mainly used to describe the temporal behaviour of abstract processes and for model-checking problems.

This thesis deals with the validity decision problem of the Modal μ-calculus, i.e. the question whether or not a given formula holds in all models.

The modal μ-calculus is an extension of the Linear Time μ-calculus (μTL) by replacing the next operator by modal operators interpreted over labelled transition systems. The validity decision problem for μTL is in PSPACE and all necessary proof rules are deterministic [Dax06]. For the modal μ-calculus, however, the decision problem is in EXPTIME and the branching operators introduce non-deterministic proof rules, hence the validity decision problem is more difficult.

We will develop a proof system that unfolds a given formula according to proof rules, quite similar to methods of natural deduction, and draft sufficient properties that a proof-tree has to fulfil for the formula to be valid.

We will create tree automata induced by our proof system to verify the sufficient properties of a proof-tree, and reduce the decision problem to an emptiness test. As this problem is equivalent to the question which player in a certain parity game has a winning strategy, we will utilise an appropriate method to solve such games.

To mention already existing approaches, there is the proof system of Kozen [Koz83] which is complete for the conjunctive fragment of the modal μ-calculus. As Walukiewicz proved [Wal95] that every formula of the modal μ-calculus is equivalent to a formula of the conjunctive fragment, the completeness of Kozen's proof system turned out to be true not only for the fragment but for the whole modal μ-calculus.

In contrast to our approach the proof trees in Kozen's system are of finite depth. Therefore it is not surprising that both approaches lead to very different solution concepts.

This thesis is inspired by Christian Dax's diploma thesis on games for the linear time μ-calculus [Dax06] and a follow-up paper by C. Dax, M. Hofmann and M. Lange [DHL06].

Rigorous proofs of soundness and completeness of our system will appear in the author's forthcoming diploma thesis.

2 Preliminaries

2.1 The Modal μ-calculus

Definition 2.1. *(Positive normal form)*
Let $\mathcal{P} = \{p, \neg p, q, \neg q, ...\}$ be a set of atomic propositions that is closed under complements, i.e. $\mathcal{P}$ fulfils the following properties:

1. $\forall p \in \mathcal{P} : \neg\neg p = p$
2. $\forall p : p \in \mathcal{P} \iff \neg p \in \mathcal{P}$

Let $\mathcal{V} = \{X, Y, ...\}$ be an infinite set of monadic second-order variables and let $\Sigma = \{a, b, ...\}$ be an infinite set of (transition) labels.
Formulas of the modal μ-calculus in *positive normal form* are given by the following grammar:

$$\varphi ::= q \mid \neg q \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \langle a \rangle \varphi \mid [a]\varphi \mid X \mid \mu X.\varphi \mid \nu X.\varphi$$

where $q \in \mathcal{P}$, $X \in \mathcal{V}$ and $a \in \Sigma$.
We will often refer to the μ- or ν-binder by σ and to positive or negative literals, i.e. q or $\neg q$, by $\pm q$.

Remark 2.2. *(True and false)*
The formulas true *and* false *are abbreviated by* $\texttt{tt} := q \vee \neg q$ *and* $\texttt{ff} := q \wedge \neg q$ *where* $q \in \mathcal{P}$.

Definition 2.3. *(Subformula)*
The set of subformulas $Sub(\varphi)$ of φ is the least set that fulfils the following properties:

1. $\varphi \in Sub(\varphi)$
2. $\psi_1 \vee \psi_2 \in Sub(\varphi) \Rightarrow \psi_1, \psi_2 \in Sub(\varphi)$
3. $\psi_1 \wedge \psi_2 \in Sub(\varphi) \Rightarrow \psi_1, \psi_2 \in Sub(\varphi)$
4. $\langle a \rangle \psi \in Sub(\varphi) \Rightarrow \psi \in Sub(\varphi)$
5. $[a]\psi \in Sub(\varphi) \Rightarrow \psi \in Sub(\varphi)$
6. $\sigma X.\psi \in Sub(\varphi) \Rightarrow \psi \in Sub(\varphi)$

Definition 2.4. *(Fischer-Ladner closure)*
The *Fischer-Ladner closure* $FL(\varphi)$ is the least set that fulfils the following properties:

1. $\varphi \in FL(\varphi)$
2. $\psi_1 \vee \psi_2 \in FL(\varphi) \Rightarrow \psi_1, \psi_2 \in FL(\varphi)$
3. $\psi_1 \wedge \psi_2 \in FL(\varphi) \Rightarrow \psi_1, \psi_2 \in FL(\varphi)$
4. $\langle a \rangle \psi \in FL(\varphi) \Rightarrow \psi \in FL(\varphi)$
5. $[a]\psi \in FL(\varphi) \Rightarrow \psi \in FL(\varphi)$
6. $\sigma X.\psi \in FL(\varphi) \Rightarrow \psi[\sigma X.\psi/X] \in FL(\varphi)$

Definition 2.5. *(Closed formulas, bound and free variables)*
An occurrence of a variable X in a $\mathcal{L}_\mu$-formula φ is *bound* iff it is in the scope of an $\sigma X.\psi$-subformula, otherwise it is *free*. The set $Free(\varphi)$ contains all free variables of φ.
A formula is *closed* iff it contains no free variables.

Definition 2.6. *(Substitution)*
Let $\varphi, \psi \in \mathcal{L}_\mu$. The formula $\varphi[\psi/X]$ is defined as the result of substituting all free occurrences of a variable X in φ by ψ.

Definition 2.7. *(Well-named)*
A formula φ is called *well-named* iff every bound variable is uniquely bound, i.e.

$$\forall\, \sigma X.\psi,\, \sigma' X.\psi' \in Sub(\varphi) : \sigma = \sigma' \wedge \psi = \psi'$$

We will assume from now on that a given formula is well-named.

Definition 2.8. *(Guarded form)*
A formula is in *guarded form* iff every occurrence of a bound variable X is in the scope of a $\langle * \rangle / [*]$-operator under its quantifier σ. We remark without proof that every formula can be transformed into guarded form. See for example [Wal00] or [Mat02].

Definition 2.9. *(Labelled transition system over $\mathcal{P}$)*
A *labelled transition system over* $\mathcal{P}$ is a quadruple $(\mathcal{S}, \theta, \mathcal{R}, \mathcal{L})$ where

1. $\mathcal{S}$ is a set of states
2. $\theta \in \mathcal{S}$ is an initial state
3. $\mathcal{R} \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ is a transition relation
4. $\mathcal{L} : \mathcal{S} \to 2^{\mathcal{P}}$ is a state description function that is compliant with $\mathcal{P}$, i.e. $\forall s \in \mathcal{S}, p \in \mathcal{P} : \; p \in \mathcal{L}(s) \iff \neg p \notin \mathcal{L}(s)$

We will often use the abbreviation $\mathcal{R}_a(s)$, where $s \in \mathcal{S}$ and $a \in \Sigma$, to denote the set of image states, i.e. $\mathcal{R}_a(s) := \{s' \in S \mid (s, a, s') \in \mathcal{R}\}$.
For a labelled transition system $\mathcal{K} = (\mathcal{S}, \theta, \mathcal{R}, \mathcal{L})$ the system $\mathcal{K}[\theta \mapsto s]$, where $s \in \mathcal{S}$, is defined as $(\mathcal{S}, s, \mathcal{R}, \mathcal{L})$.

Definition 2.10. *(Semantics of a $\mathcal{L}_\mu$-formula)*
The semantics of a $\mathcal{L}_\mu$-formula, relative to a labelled transition system $\mathcal{K}$ and an environment $\rho : \mathcal{V} \to 2^{\mathcal{S}}$ is an inductively defined subset of $\mathcal{S}$.

1. $[\![\pm q]\!]^{\mathcal{K}}_\rho := \{s \in \mathcal{S} \mid \pm q \in \mathcal{L}(s)\}$
2. $[\![\psi_1 \wedge \psi_2]\!]^{\mathcal{K}}_\rho := [\![\psi_1]\!]^{\mathcal{K}}_\rho \cap [\![\psi_2]\!]^{\mathcal{K}}_\rho$
3. $[\![\psi_1 \vee \psi_2]\!]^{\mathcal{K}}_\rho := [\![\psi_1]\!]^{\mathcal{K}}_\rho \cup [\![\psi_2]\!]^{\mathcal{K}}_\rho$
4. $[\![\langle a \rangle \psi]\!]^{\mathcal{K}}_\rho := \{s \in \mathcal{S} \mid \mathcal{R}_a(s) \cap [\![\psi]\!]^{\mathcal{K}}_\rho \neq \emptyset\}$
5. $[\![[a]\psi]\!]^{\mathcal{K}}_\rho := \{s \in \mathcal{S} \mid \mathcal{R}_a(s) \subseteq [\![\psi]\!]^{\mathcal{K}}_\rho\}$
6. $[\![X]\!]^{\mathcal{K}}_\rho := \rho(X)$
7. $[\![\mu X.\psi]\!]^{\mathcal{K}}_\rho := \bigcap\{T \subseteq \mathcal{S} \mid [\![\psi]\!]^{\mathcal{K}}_{\rho[X \mapsto T]} \subseteq T\}$
8. $[\![\nu X.\psi]\!]^{\mathcal{K}}_\rho := \bigcup\{T \subseteq \mathcal{S} \mid T \subseteq [\![\psi]\!]^{\mathcal{K}}_{\rho[X \mapsto T]}\}$

Definition 2.11. *(Model, Equivalence, Validity, Satisfiability)*
A labelled transition system $\mathcal{K}$, relative to an environment ρ, is a *model* of a $\mathcal{L}_\mu$-formula φ iff $\theta \in [\![\varphi]\!]^{\mathcal{K}}_\rho$, i.e.

$$\mathcal{K} \models_\rho \varphi \; :\iff \; \theta \in [\![\varphi]\!]^{\mathcal{K}}_\rho$$

Let $\mathcal{K}$ be a labelled transition system, $s \in \mathcal{S}$, ρ an environment and $\varphi \in \mathcal{L}_\mu$. Then we define

$$\mathcal{K}, s \models_\rho \varphi \; :\iff \; s \in [\![\varphi]\!]^{\mathcal{K}}_\rho$$

Two formulas φ, ψ are equivalent, $\varphi \equiv \psi$, iff

$$\forall \rho \forall \mathcal{K} : \mathcal{K} \models_\rho \varphi \iff \mathcal{K} \models_\rho \psi$$

A formula φ is valid, $\models \varphi$, iff every labelled transition system is a model of φ for every environment ρ, i.e.

$$\models \varphi :\iff \forall \rho \forall \mathcal{K} : \mathcal{K} \models_\rho \varphi$$

A formula φ is satisfiable iff there is a model of φ, i.e. $\exists \rho \exists \mathcal{K} : \mathcal{K} \models_\rho \varphi$.

Definition 2.12. *(Sequent)*
Let φ_0 be a $\mathcal{L}_\mu$-formula. A subset Γ of $FL(\varphi_0)$ is called a *sequent*.
Semantically, a sequent stands for the disjunction of its members; thus we will identify the set contextually with the logical disjunction of its members, i.e.

$$\gamma \equiv \emptyset :\iff \gamma \equiv \mathtt{ff}$$

$$\gamma \equiv \Gamma :\iff \gamma \equiv \bigvee_{\psi \in \Gamma} \psi$$

A sequent Δ is called *trivial* iff $\exists p \in \mathcal{P} : \{p, \neg p\} \subseteq \Delta$, otherwise it is called *non-trivial*.

2.2 Automata

We assume the reader to be familiar with parity and Büchi automata as we will use them to accept proof-trees that are induced by formulas of the modal μ-calculus in order to check their validity.
In this chapter we will introduce the concept of finiteness condition which allows us to accept trees with both finite and infinite branches as this idea will be quite useful to define our theoretic proof automaton.
Finally we will provide some required automata transformations.

Definition 2.13. *(Finiteness condition)*
A *nondeterministic parity automaton with finiteness condition (NPAF)* is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \delta, \Omega, F')$ where

1. Q, Σ, q_0, δ and Ω as usual

2. F' is the set of final states

A run on an infinite word is called *accepting* as usual, and a run on a finite word is called *accepting* iff the last state in the finite sequence is a final state.
The finiteness condition for the other automata - such as büchi and tree automata - is defined the same way.
Note that every NPA is also a NPAF with an empty set of final states.

Lemma 2.14. *(NPA to NBA)*
Let $\mathcal{A}$ be a NPAF with n states and index m. Then there is an equivalent NBAF $\mathcal{B}$ with at most $n \cdot \lceil \frac{m}{2} \rceil$ states.

Proof. Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, \Omega, F')$ be a NPAF with $|Q| = n$, $\max \Omega[Q] = m$, $k := \lceil \frac{m}{2} \rceil$ and $k' := \lfloor \frac{m}{2} \rfloor$.
Then we construct a NBAF $\mathcal{B} = (Q^{\mathcal{B}}, \Sigma, q_0^{\mathcal{B}}, \delta^{\mathcal{B}}, F^{\mathcal{B}}, F'^{\mathcal{B}})$ with at most $n \cdot k$ states as follows:

- $Q^{\mathcal{B}} := \bigcup_{i=0}^{k} (\{i\} \times Q_i)$ where $Q_k := Q$ and $Q_{i-1} := Q_i \setminus \Omega^{-1}(2i-1)$
- $q_0^{\mathcal{B}} := (k, q_0)$, $F^{\mathcal{B}} := \bigcup_{i=0}^{k'} (\{i\} \times \Omega^{-1}(2i))$ and $F'^{\mathcal{B}} := \{k\} \times F'$
- $\delta^{\mathcal{B}}((i,q),x) := \delta_i^{\mathcal{B}}(q,x) \cup \delta_{i-1}^{\mathcal{B}}(q,x)$ where $\delta_{-1}^{\mathcal{B}}(q,x) := \emptyset$ and for $i \geq 0$ $\delta_i^{\mathcal{B}}(q,x) := (\{i\} \times \delta(q,x)) \cap Q_i$

We have to show that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.
"$\Leftarrow$": Let $w \in \mathcal{L}(\mathcal{B})$. Then there is an accepting run $r : \mathbb{N} \to Q^{\mathcal{B}}$ of $\mathcal{B}$ on w. It is easy to see that by definition of the $\mathcal{B}$-automaton $p_Q \circ r$ is an accepting run of $\mathcal{A}$ on w, where $p_Q(i,q) := q$.
"$\Rightarrow$": Let $w \in \mathcal{L}(\mathcal{A})$. Then there is an accepting run $r : \mathbb{N} \to Q^{\mathcal{A}}$ of $\mathcal{A}$ on w. If w is of finite length, then $i \mapsto (k, r(i))$ is obviously an accepting run of $\mathcal{B}$ on w. If w is of infinite length, there is a greatest priority $2 \cdot p$ that occurs infinitely often, and moreover there is a position l s.t. $\forall l' > l : \Omega(r(l')) \leq p$. Thus we define the following transformation function $g : \mathbb{N} \to \mathbb{N}$:

$$g(i) := \begin{cases} k & i \leq l \\ k + l - i & l < i < k + l - p \\ p & i \geq k + l - p \end{cases}$$

It is not hard to see that $i \mapsto (g(i), r(i))$ is an accepting run of $\mathcal{B}$ on w. $\square$

Lemma 2.15. *(NBA to DPA)*
Let $\mathcal{A}$ be a NBA with n states. Then there is an equivalent DPA $\mathcal{B}$ with at most n^{2n+2} states and at most index $2n-1$.

Proof. See for example [Pit06]. □

Theorem 2.16. *(Solving parity games)*
Given a parity game, there is an algorithm that computes winning sets for both players.
It works in space $O(dn)$, and its running time is

$$O\left(d \cdot m \cdot \left(\frac{n}{\lfloor d/2 \rfloor}\right)^{\lfloor d/2 \rfloor}\right)$$

where n is the number of vertices, m is the number of edges, and d is the maximum priority in the parity game.

Proof. See for example [Jur00]. □

3 A Proof System

This chapter comprises the core of this thesis. We will create a proof system in order to decide whether a formula is valid or not, although we will not proof it sound and complete.

3.1 Proof rules

We will introduce proof rules motivated by natural deduction for the modal μ-calculus in this chapter. We will get a (possibly infinite) proof tree by applying the proof rules to a formula.
As we will see later on, a formula is valid iff there is a proof tree s.t. a certain condition is fulfilled on every branch, hence a parity tree automaton is predestined to accept such trees.

Definition 3.1. *(Proof rule)*
Let $\varphi \in \mathcal{L}_\mu$ and $\Gamma, \Gamma_1, ..., \Gamma_n \subseteq FL(\varphi)$ sequents.
A *proof rule* (R) is defined as follows:

$$(R): \frac{\vdash \Gamma_1 \qquad \vdash \Gamma_2 \qquad ... \qquad \vdash \Gamma_n}{\vdash \Gamma}$$

Every member Γ_i of the top line of (R) is called $(i\text{-}th)$ *premiss* of (R) and the bottom line Γ is called *conclusion* of (R).
A proof rule (R) is *sound* iff $\models \Gamma_1 \wedge \Gamma_2 \wedge ... \wedge \Gamma_n$ implies $\models \Gamma$.
Accordingly a proof rule (R) is *complete* iff $\models \Gamma$ implies $\models \Gamma_1 \wedge \Gamma_2 \wedge ... \wedge \Gamma_n$.
Note that we are using the implicit disjunction over sequents as introduced in Definition 2.12.

Definition 3.2. *(Proof rules of the modal μ-calculus)*
Let $\varphi \in \mathcal{L}_\mu$, $\psi_1, \psi_2, \psi \in FL(\varphi)$, let $\Gamma \subseteq FL(\varphi)$ be a sequent and let $\Pi \subseteq FL(\varphi) \cap \mathcal{P}$ be a non-trivial sequent.
The proof rules $(\vee)$, $(\wedge)$, (σ) and $(\Box)$ of the modal μ-calculus are defined as follows:

$$(\vee): \frac{\vdash \psi_1, \psi_2, \Gamma}{\vdash \psi_1 \vee \psi_2, \Gamma} \qquad (\wedge): \frac{\vdash \psi_1, \Gamma \qquad \vdash \psi_2, \Gamma}{\vdash \psi_1 \wedge \psi_2, \Gamma} \qquad (\sigma): \frac{\vdash \psi[\sigma X.\psi / X], \Gamma}{\vdash \sigma X.\psi, \Gamma}$$

$$(\Box): \frac{\vdash \psi_i, \{\gamma_j \mid b_j = a_i\}}{\vdash [a_1]\psi_1, [a_2]\psi_2, ..., [a_n]\psi_n, \langle b_1\rangle\gamma_1, \langle b_2\rangle\gamma_2, ..., \langle b_m\rangle\gamma_m, \Pi}$$

where $a_1, ..., a_n, b_1, ..., b_m \in \Sigma$.
A *principal formula* in a rule application is a formula that gets transformed by this rule.

Definition 3.3. *(Pre-Proof)*
Let $\varphi \in \mathcal{L}_\mu$. A *pre-proof* for φ is a possibly infinite tree whose nodes are labelled with sequents, whose root is labelled with $\vdash \varphi$ and which is built according to the proof rules $(\vee)$, $(\wedge)$, (σ) and $(\Box)$ of the modal μ-calculus .

Definition 3.4. *(Connection relation)*
Let $\varphi \in \mathcal{L}_\mu$. For all sequents Γ, Δ and all rules (R) occurring in a pre-proof for φ, s.t. Γ is the conclusion of (R) and Δ is a premiss of (R) the *connection relation* $Con_R(\Gamma, \Delta) \subseteq \Gamma \times \Delta$ is defined as follows:

$$(\psi_1, \psi_2) \in Con_R(\Gamma, \Delta) : \Longleftrightarrow \begin{array}{l} \psi_1 \text{ is no principal formula in } (R) \\ \text{and } \psi_1 = \psi_2 \text{ or } \psi_2 \text{ results} \\ \text{from } \psi_1 \text{ in the application of } (R) \end{array}$$

We drop the index R if the actual rule (R) is irrelevant.

Definition 3.5. *(σ-threads)*
Let $\varphi_0 \in \mathcal{L}_\mu$ and $\pi = \Gamma_0, \Gamma_1, ...$ be an infinite branch in a pre-proof for φ_0 resulting from the rule application $R_0, R_1, ...$
A *thread* in π is a sequence of formulas that is compliant with π and Con, i.e. a sequence $\varphi_0, \varphi_1, ...$ s.t. $\forall i \in \mathbb{N} : (\varphi_i, \varphi_{i+1}) \in Con_{R_i}(\Gamma_i, \Gamma_{i+1})$ holds.
A thread is called a *ν-thread* iff $\exists \nu X.\psi \in FL(\varphi_0)$ s.t.

1. $\forall i \, \exists j \geq i : \varphi_j = \nu X.\psi$
2. $\forall \mu Y.\gamma \in FL(\varphi_0) : (\nu X.\psi \in Sub(\mu Y.\gamma) \Rightarrow \exists i \, \forall j \geq i : \varphi_j \neq \mu Y.\gamma)$

A *μ-thread* is defined the same way.
We remark without proof that every thread is either a ν-thread or a μ-thread as it is easy to see.

Definition 3.6. *(Proof)*
Let $\varphi \in \mathcal{L}_\mu$. A *proof* for φ is a pre-proof s.t. the following holds:

1. Every finite branch ends in a trivial sequent.
2. Every infinite branch contains a ν-thread.

We also write $\vdash\varphi$ to indicate that there is a proof for φ.
Note that we will sometimes call a pre-proof that fulfils at least the first condition as *pre-proof with valid leaves only*.

3.2 Proof automata

We will now define the core of our proof system that is used to decide validity:
A thread-automaton that traces non-deterministically a ν-thread in a given branch of a pre-proof and a tree automaton that simply accepts well-formed pre-proofs.
By gluing the determinization of the thread automaton and the tree automaton together, the resulting automaton exactly accepts proof-trees. Hence we just have to check the latter automaton for emptiness in order to decide validity.

Definition 3.7. *(Thread automaton)*
Let $\varphi_0 \in \mathcal{L}_\mu$. The *thread automaton* $\mathcal{A}_{\varphi_0}$ is a nondeterministic parity automaton that is defined as follows.
Define $\mathcal{A}_{\varphi_0} := (Q^{\mathcal{A}}, \Sigma^{\mathcal{A}}, q_0^{\mathcal{A}}, \delta^{\mathcal{A}}, \Omega^{\mathcal{A}})$ where

- $Q^{\mathcal{A}} := FL(\varphi_0)$ and $q_0^{\mathcal{A}} := \varphi_0$
- $\Sigma^{\mathcal{A}} := \quad \{L(\psi_1 \wedge \psi_2), R(\psi_1 \wedge \psi_2) \mid \psi_1 \wedge \psi_2 \in FL(\varphi_0)\}$

  $\cup \{P(\psi_1 \vee \psi_2) \mid \psi_1 \vee \psi_2 \in FL(\varphi_0)\}$

  $\cup \{P(\sigma X.\psi) \mid \sigma X.\psi \in FL(\varphi_0)\}$

  $\cup \{N([a]\psi) \mid [a]\psi \in FL(\varphi_0)\}$

- $\delta^{\mathcal{A}} := \quad \{(\psi_1 \wedge \psi_2, L(\psi_1 \wedge \psi_2), \{\psi_1\}) \mid \psi_1 \wedge \psi_2 \in FL(\varphi_0)\}$

  $\cup \{(\psi_1 \wedge \psi_2, R(\psi_1 \wedge \psi_2), \{\psi_2\}) \mid \psi_1 \wedge \psi_2 \in FL(\varphi_0)\}$

  $\cup \{(\psi_1 \vee \psi_2, P(\psi_1 \vee \psi_2), \{\psi_1, \psi_2\}) \mid \psi_1 \vee \psi_2 \in FL(\varphi_0)\}$

  $\cup \{(\sigma X.\psi, P(\sigma X.\psi), \{\psi[\sigma X.\psi/X]\}) \mid \sigma X.\psi \in FL(\varphi_0)\}$

  $\cup \{([a]\psi, N([a]\psi), \{\psi\}) \mid [a]\psi \in FL(\varphi_0)\}$

  $\cup \{(\langle a\rangle\psi_1, N([a]\psi_2), \{\psi_1\}) \mid \langle a\rangle\psi_1, [a]\psi_2 \in FL(\varphi_0)\}$

  $\cup \{(\psi, r, \{\psi\}) \mid \psi \in FL(\varphi_0),\ r \notin \{L(\psi), R(\psi), P(\psi), N(*)\}\}$

- $\Omega^{\mathcal{A}}(\mu X.\psi) := \Omega^{\mathcal{A}}(\psi) + ((\Omega^{\mathcal{A}}(\psi) + 1) \mod 2)$

  $\Omega^{\mathcal{A}}(\nu X.\psi) := \Omega^{\mathcal{A}}(\psi) + ((\Omega^{\mathcal{A}}(\psi)) \mod 2)$

  $\Omega^{\mathcal{A}}([a]\psi) := \Omega^{\mathcal{A}}(\langle a\rangle\psi) := \Omega^{\mathcal{A}}(\psi)$

  $\Omega^{\mathcal{A}}(\psi_1 \wedge \psi_2) := \Omega^{\mathcal{A}}(\psi_1 \vee \psi_2) := \max \Omega^{\mathcal{A}}[\{\psi_1, \psi_2\}]$

  $\Omega^{\mathcal{A}}(q) := 0$ otherwise

Note that an infinite branch $\pi = \Gamma_0, \Gamma_1, ...$ in a pre-proof for φ_0 induces a word $r_0, r_1, ...$ as follows:

$$r_i := \begin{cases} L(\psi) & \text{if } \psi \text{ is principal in } \Gamma_i \text{ and } \Gamma_{i+1} \text{ is left premiss of } \Gamma_i \\ R(\psi) & \text{if } \psi \text{ is principal in } \Gamma_i \text{ and } \Gamma_{i+1} \text{ is right premiss of } \Gamma_i \\ P(\psi) & \text{if } \psi \text{ is principal in } \Gamma_i \text{ and } \Gamma_{i+1} \text{ is the premiss of } \Gamma_i \text{ and } \psi \notin \{[*]*\} \\ N([a]\psi) & \text{if } [a]\psi \text{ is principal in } \Gamma_i \text{ and } \Gamma_{i+1} \text{ is the premiss of } \Gamma_i \end{cases}$$

We will not distinguish formally between a branch π and its induced word. We identify rule application here with an infinite word, therefore our automaton just resembles the connection relation and traces nondeterministically a ν-thread. We will elaborate on that fact in the following Lemma.

Lemma 3.8. *(Correctness of the thread automaton)*
Let $\varphi_0 \in \mathcal{L}_\mu$ and $\mathcal{A}_{\varphi_0}$ the corresponding thread automaton. Then for all branches π of a pre-proof for φ_0 the following holds:

$$\pi \in \mathcal{L}(\mathcal{A}_{\varphi_0}) \iff \pi \text{ contains a } \nu\text{-thread}$$

Proof. Let $\varphi_0 \in \mathcal{L}_\mu$.
"$\Rightarrow$": Let $\pi \in \mathcal{L}(\mathcal{A}_{\varphi_0})$. Thus there is an accepting run $q_0, q_1, ...$ on π which is obviously a thread as the transition function resembles the connection relation. Therefore we have to show that $q_0, q_1, ...$ is a ν-thread.
As $q_0, q_1, ...$ is an accepting run, there is a greatest priority $m \in \mathbb{N}$ that occurs infinitely often and is even. According to the definition of $\Omega^{\mathcal{A}}$ there is a formula $\nu X.\psi$ s.t. $\Omega^{\mathcal{A}}(\nu X.\psi) = m$, i.e. $q_i = \nu X.\psi$ infinitely often.
Thus it remains to prove that there is no $\mu Y.\psi'$ s.t. $\nu X.\psi \in Sub(\mu Y.\psi')$ that occurs infinitely often in $q_0, q_1, ...$
Assume that there is such a $\mu Y.\psi'$. This implies by definition that $\Omega^{\mathcal{A}}(\mu Y.\psi') > \Omega^{\mathcal{A}}(\nu Y.\psi)$ which is impossible as m is the highest priority that occurs infinitely often. Hence $q_0, q_1, ...$ is a ν-thread.
"$\Leftarrow$": Let π be an infinite branch that contains a ν-thread $q_0, q_1, ...$ As the transition function of $\mathcal{A}_{\varphi_0}$ resembles the connection relation, $q_0, q_1, ...$ is also a run on π. Therefore it remains to prove that $q_0, q_1, ...$ is accepting.
Since $q_0, q_1, ...$ is a ν-thread there is an outermost $\nu X.\psi$ s.t. $q_i = \nu X.\psi$ infinitely often, consequently there is an even priority $\Omega^{\mathcal{A}}(\nu Y.\psi) =: m \in \mathbb{N}$ that occurs infinitely often. As $\nu X.\psi$ is the outermost formula that occurs infinitely often (!), m is the greatest priority that occurs infinitely often, thus $\pi \in \mathcal{L}(\mathcal{A}_{\varphi_0})$. □

Definition 3.9. *(Pre-proof automaton)*
Let $\varphi_0 \in \mathcal{L}_\mu$. The *pre-proof automaton* $\mathcal{B}_{\varphi_0}$ is a deterministic büchi tree automaton with finiteness condition that is defined as follows.
Define $\mathcal{B}_{\varphi_0} := (Q^{\mathcal{B}}, \Sigma^{\mathcal{B}}, q_0^{\mathcal{B}}, \delta^{\mathcal{B}}, F^{\mathcal{B}}, F'^{\mathcal{B}})$ where

- $Q^{\mathcal{B}} := 2^{FL(\varphi_0)} \cup \{v^{\mathcal{B}}\}$ and $q_0^{\mathcal{B}} := \{\varphi_0\}$
- $F^{\mathcal{B}} := Q^{\mathcal{B}}$ and $F'^{\mathcal{B}} := \{v^{\mathcal{B}}\}$
- $\Sigma^{\mathcal{B}} := \{LR(\psi_1 \wedge \psi_2) \mid \psi_1 \wedge \psi_2 \in FL(\varphi_0)\}$
  $\cup \{P(\psi_1 \vee \psi_2) \mid \psi_1 \vee \psi_2 \in FL(\varphi_0)\}$
  $\cup \{P(\sigma X.\psi) \mid \sigma X.\psi \in FL(\varphi_0)\}$
  $\cup \{N([a]\psi) \mid [a]\psi \in FL(\varphi_0)\}$

- $\delta^{\mathcal{B}}(\Gamma \dot\cup \{\psi_1 \wedge \psi_2\}, LR(\psi_1 \wedge \psi_2)) := (v'(\Gamma \cup \{\psi_1\}), v'(\Gamma \cup \{\psi_2\}))$

  $\delta^{\mathcal{B}}(\Gamma \dot\cup \{\psi_1 \vee \psi_2\}, P(\psi_1 \vee \psi_2)) := v'(\Gamma \cup \{\psi_1, \psi_2\})$

  $\delta^{\mathcal{B}}(\Gamma \dot\cup \{\sigma X.\psi\}, P(\sigma X.\psi)) := v'(\Gamma \cup \{\psi[\sigma X.\psi / X]\})$

  $\delta^{\mathcal{B}}(\Gamma' \dot\cup \{[a]\psi\}, N([a]\psi)) := v'(\{\psi\} \cup \{\psi' \mid \langle a \rangle \psi' \in \Gamma'\})$

  where $\Gamma' \subseteq \mathcal{P} \cup \{\langle a \rangle \psi \in FL(\varphi_0)\} \cup \{[a]\psi \in FL(\varphi_0)\}$

  $\delta^{\mathcal{B}}(v^{\mathcal{B}}, *) := v^{\mathcal{B}}$

  where $v'(\Gamma) := \begin{cases} v^{\mathcal{B}} & \text{if } \Gamma \text{ is trivial} \\ \Gamma & \text{otherwise} \end{cases}$

Note that the alphabet is identified with the rule application in a node of a pre-proof tree.

Corollary 3.10. *(Correctness of the pre-proof automaton)*
Let $\varphi_0 \in \mathcal{L}_\mu$ and $\mathcal{B}_{\varphi_0}$ the corresponding pre-proof automaton. Then the following holds:

$$P \in \mathcal{L}(\mathcal{B}_{\varphi_0}) \iff P \text{ is a pre-proof with valid leaves only for } \varphi_0$$

Proof. Let $\varphi_0 \in \mathcal{L}_\mu$.
"$\Rightarrow$": Let $P \in \mathcal{L}(\mathcal{B}_{\varphi_0})$. As $\mathcal{L}(\mathcal{B}_{\varphi_0})$ obviously follows the proof rules, P is a pre-proof. Moreover all finite branches are valid, because the finiteness condition implies that all finite branches end with the $v^{\mathcal{B}}$ state. As $v^{\mathcal{B}}$ is reached iff a branch ends with a valid sequent, P is a pre-proof with valid leaves only.
"$\Leftarrow$": This can be shown the same way. □

Definition 3.11. *(Proof automaton)*
Let $\varphi_0 \in \mathcal{L}_\mu$, $\mathcal{A}'_{\varphi_0}$ a deterministic parity thread automaton for φ_0 (which exists according to Lemma 2.14 and Lemma 2.15) and $\mathcal{B}_{\varphi_0}$ the pre-proof automaton for φ_0 as defined in Definition 3.9. The *proof automaton* $\mathcal{C}_{\varphi_0}$ is a deterministic parity tree automaton with finiteness condition that is defined as follows.
Define $\mathcal{C}_{\varphi_0} := (Q^{\mathcal{C}}, \Sigma^{\mathcal{C}}, q_0^{\mathcal{C}}, \delta^{\mathcal{C}}, \Omega^{\mathcal{C}}, F'^{\mathcal{C}})$ where

- $Q^{\mathcal{C}} := (2^{FL(\varphi_0)} \times Q^{\mathcal{A}'}) \cup \{v^{\mathcal{C}}\}$ and $q_0^{\mathcal{C}} := (\{\varphi_0\}, \varphi_0)$

- $\Sigma^{\mathcal{C}} := \Sigma^{\mathcal{B}}$ and $F'^{\mathcal{C}} := \{v^{\mathcal{C}}\}$
- $\Omega^{\mathcal{C}}(\Gamma, \psi) := \Omega^{\mathcal{C}}(\psi)$ and $\Omega^{\mathcal{C}}(v^{\mathcal{C}}) := 0$
- $\delta^{\mathcal{C}}((\Gamma, \psi), r) := t(\delta^{\mathcal{B}}(\Gamma, r), \delta^{\mathcal{A}'}(\psi, r))$ if $r \in \Sigma^{\mathcal{A}'} \cap \Sigma^{\mathcal{B}}$

  $\delta^{\mathcal{C}}((\Gamma, \psi), LR(\psi')) := (t(\delta^{\mathcal{B}}(\Gamma, LR(\psi'))_1, \delta^{\mathcal{A}'}(\psi, L(\psi'))),$

  $t(\delta^{\mathcal{B}}(\Gamma, LR(\psi'))_2, \delta^{\mathcal{A}'}(\psi, R(\psi'))))$

  $\delta^{\mathcal{C}}(v^{\mathcal{C}}, *) := v^{\mathcal{C}}$

  where $t(q_{\mathcal{B}}, q_{\mathcal{A}'}) := \begin{cases} v^{\mathcal{C}} & \text{if } q_{\mathcal{B}} = v^{\mathcal{B}} \\ (q_{\mathcal{B}}, q_{\mathcal{A}'}) & \text{otherwise} \end{cases}$

This automaton intuitively does the same as the pre-proof automaton, but in addition to that, it runs the thread-checking automaton $\mathcal{A}'_{\varphi_0}$ along each path of the tree.

Theorem 3.12. *(Correctness of the proof automaton)*
Let $\varphi_0 \in \mathcal{L}_\mu$ and $\mathcal{C}_{\varphi_0}$ the corresponding proof automaton. Then the following holds

$$\vdash \varphi_0 \iff \mathcal{L}(\mathcal{C}_{\varphi_0}) \neq \emptyset$$

Proof. Let $\varphi_0 \in \mathcal{L}_\mu$.
"$\Rightarrow$": Let $P \in \mathcal{L}(\mathcal{C}_{\varphi_0})$, thus there is an accepting run r on P.
We define the state projection functions $p_{\mathcal{B}} : Q^{\mathcal{C}} \to Q^{\mathcal{B}}$ and $p_{\mathcal{A}} : Q^{\mathcal{C}} \setminus \{v^{\mathcal{C}}\} \to Q^{\mathcal{A}}$ as follows:

$$p_{\mathcal{B}}(q) := \begin{cases} q_{\mathcal{B}} & \text{if } q = (q_{\mathcal{B}}, q_{\mathcal{A}}) \\ v^{\mathcal{B}} & \text{if } q = v^{\mathcal{C}} \end{cases} \qquad \text{and} \qquad p_{\mathcal{A}}(q_{\mathcal{B}}, q_{\mathcal{A}}) := q_{\mathcal{A}}$$

It is not hard to see that $p_{\mathcal{B}} \circ r$ is an accepting run of $\mathcal{B}_{\varphi_0}$ on P, hence P is a pre-proof with valid leaves only (Corollary 3.10). As moreover every infinite branch π of P is annotated with the deterministic thread automaton $\mathcal{A}'_{\varphi_0}$ that accepts a branch iff it contains a ν-thread (Lemma 3.8), every infinite branch has to contain such a thread since $p_{\mathcal{A}} \circ \pi$ is an accepting run of $\mathcal{A}'_{\varphi_0}$ on π. Therefore P is a proof.
"$\Leftarrow$": Let P be a proof for φ_0. According to Corollary 3.10 there is an accepting run r of $\mathcal{B}_{\varphi_0}$ on P and because of Lemma 3.8 there is, for every

infinite branch π in P, an accepting run r_π of $\mathcal{A}'_{\varphi_0}$ on π as π contains a ν-thread by assumption.
Because $\mathcal{A}'_{\varphi_0}$ is deterministic we can construct a run r' of $\mathcal{C}_{\varphi_0}$ on P by annotating every infinite branch π in r with the accepting run r_π. It is not hard to see that this run is accepting by definition of $\mathcal{C}_{\varphi_0}$. □

3.3 A Reduction from validity to parity games

Theorem 3.13. *(The proof system is sound and complete)*
Let $\varphi_0 \in \mathcal{L}_\mu$. Then the following holds:

$$\vdash \varphi_0 \iff \models \varphi_0$$

Definition 3.14. *(Induced parity game)*
Let $\mathcal{A} = (Q, \Sigma, q_0, \delta, \Omega, F')$ be a parity automaton.
The induced parity game $G_\mathcal{A} := (Pos_\exists, Pos_\forall, \delta', \Omega')$ is defined as follows:

1. $Pos_\exists := Q$
2. $Pos_\forall := Q^+ \setminus Q$
3. $\delta'(q) := \delta(q, \Sigma)$
4. $\delta'(q_1, ..., q_n) := \{q_1, ..., q_n\}$ where $n \geq 2$
5. $\Omega'(q_1) := \Omega(q_1)$ and $\Omega'(q_1, ..., q_n) := 0$ where $n \geq 2$

Theorem 3.15. *(Solution of the proof system)*
As a formula is valid iff the corresponding proof automaton is not empty, all we have to do is to solve the induced parity game. Thus a formula is valid iff the corresponding parity game starting in the initial state of the proof automaton has a positional winning strategy for the automaton-player.

Lemma 3.16. *(Game size)*
Let $\varphi_0 \in \mathcal{L}_\mu$, let A_{φ_0} be the thread parity automaton of Definition 3.7, let P_{φ_0} be the Piterman determinization (Lemma 2.14 in combination with Lemma 2.15) of A_{φ_0} and let C_{φ_0} be the proof automaton.
Let G_{φ_0} be the induced parity game.
Then the following estimations hold:

1. $|Q_A| \leq k \in O(k)$

2. $Index_A \leq k \in O(k)$

3. $|Q_P| \leq 2^{(2 \cdot k \cdot \lceil \frac{k}{2} \rceil + 2) \cdot ld(k \cdot \lceil \frac{k}{2} \rceil)} \in 2^{O(k^2 \cdot ld(k))}$

4. $Index_P = Index_C = Index_G = 2 \cdot k \cdot \lceil \frac{k}{2} \rceil - 1 \in O(k^2)$

5. $|Q_C| \leq 2^{k + (2 \cdot k \cdot \lceil \frac{k}{2} \rceil + 2) \cdot ld(k \cdot \lceil \frac{k}{2} \rceil)} \in 2^{O(k^2 \cdot ld(k))}$

6. $|\delta_C| \leq 2^{k + (2 \cdot k \cdot \lceil \frac{k}{2} \rceil + 2) \cdot ld(k \cdot \lceil \frac{k}{2} \rceil) + ld(k)} \in 2^{O(k^2 \cdot ld(k))}$

7. $|G| \leq 2^{2 \cdot (k + (2 \cdot k \cdot \lceil \frac{k}{2} \rceil + 2) \cdot ld(k \cdot \lceil \frac{k}{2} \rceil))} \in 2^{O(k^2 \cdot ld(k))}$

8. $|\delta_G| \leq 2^{2 \cdot (k + (2 \cdot k \cdot \lceil \frac{k}{2} \rceil + 2) \cdot ld(k \cdot \lceil \frac{k}{2} \rceil)) + ld(k)} \in 2^{O(k^2 \cdot ld(k))}$

where $k := |\varphi_0|$

Proof. Directly.

- Proposition 1 and 2 follow by definition, 3 and 4 by applying Lemma 2.14 and 2.15. Then 5 is obvious.
- Proposition 6 holds, since the highest possible transition degree occurs if a ($\Box$)-rule is played with a sequent full of boxes. Therefore the transition degree can be limited by k.
- Proposition 7 holds, since the state blow-up by the $\forall$-player is only generated by the ($\wedge$)-rule, the state size can be limited quadratically.
- Same way as in 6.

□

Corollary 3.17. *(Complexity)*
Deciding whether $\varphi_0 \in \mathcal{L}_\mu$ *is valid or not is in*

$$2^{O(k^5 \cdot ld(k))}$$

where $k := |\varphi_0|$

Proof. Lemma 3.16 in combination with Lemma 2.16 □

4 A simple implementation

The implementation of the three automata and the realisation of the solving algorithm described in the preceding chapter have been done in the object-oriented functional language Ocaml.
The Piterman construction [Pit06] to determinize the thread automaton has been implemented by Christian Dax [Dax06].
In this chapter, we will describe the usage of the validity checking application, its core implementation and some experimental results.

4.1 Usage

The main application is implemented in *checker/checker.ml* and compiles to a binary with the same name in the same directory. The command line options are:

```
1   -info          Show information about the given formula
2   -show_game     Show parity game
3   -show_size     Compute and show size of the game
4   -hash          Enable hashing of transition functions
5   -nosimplify    Disable pre-simplification of formula
6   -valid_stir    Check validity using the Stevens/Stirling algorithm
7   -sat           Check satisfiability instead of validity
8   -about         Show about screen
9   -parse         Mu-calculus formula
10  -help          Display this list of options
```

The *parse*-parameter expects the formula the algorithm has to solve. The syntax is quite like in Definition 2.1 extended by a rudimentary define-syntax s.t. reoccurring formulas do not have to be rewritten:

$$\varphi ::= q \mid -q \mid \varphi \& \varphi \mid \varphi | \varphi \mid < a > \varphi \mid [a]\varphi \mid X \mid mu\ X.\varphi \mid nu\ X.\varphi \mid \#def$$

$$term ::= \varphi \mid \#def := \varphi, term$$

Note that a formula has to be closed and every variable has to be under an even number of negations under its quantifier. It is neither necessary to bind all variables uniquely nor to give a formula in guarded form, all these transformations are done automatically.

Example 4.1. *(Explanatory usage example)*
Let $\varphi := \mu X.\langle a \rangle(\nu Y.[b](X \wedge (Y \vee p)))$ and $\psi := \varphi \longleftrightarrow \varphi$. Obviously ψ is true, although we want to be given that sophisticated answer by our implementation. Therefore we could call the application as follows...

```
checker -parse "#phi := mu X.<a>(nu Y.[b](X & (Y | p))), #psi := #phi <==> #phi,
    #psi" -valid_stir -nosimplify
```

...and the output looks like...

```
Validity Checker for the Modal mu-calculus 0.1 (c) O. Friedmann, 2006

Using raw formula:
  mu X.<a>nu Y.[b](X & (Y | p)) <==> mu X.<a>nu Y.[b](X & (Y | p))
Using uniquely bound formula:
  (mu X.<a>nu Y.[b](X & (Y | p)) & mu X0.<a>nu Y0.[b](X0 & (Y0 | p))) | (-mu
      X1.<a>nu Y1.[b](X1 & (Y1 | p)) & -mu X2.<a>nu Y2.[b](X2 & (Y2 | p)))
Simplifying to:
  (mu X.<a>nu Y.[b](X & (Y | p)) & mu X0.<a>nu Y0.[b](X0 & (Y0 | p))) | (-mu
      X1.<a>nu Y1.[b](X1 & (Y1 | p)) & -mu X2.<a>nu Y2.[b](X2 & (Y2 | p)))
Transforming to:
  (mu X.<a>nu Y.[b](X & (Y | p)) & mu X0.<a>nu Y0.[b](X0 & (Y0 | p))) | (nu
      X1.[a]mu Y1.<b>(X1 | (Y1 & -p)) & nu X2.[a]mu Y2.<b>(X2 | (Y2 & -p)))

Building formula access tables...
Building thread automaton...
Building piterman automaton...
Building proof game...

Check using Stevens/Stirling

Formula is valid.
CPU time: 0:0:0
```

4.2 Concrete implementation

We will only present the core components of the source code implementation in this chapter. The complete source code is available at [Fri06].
Generally the implementation splits into separate modules and folders:

- *app* - main application source, comprises the command line behaviour and the interaction of the core components
- *automata* - implementation of the thread and the proof automata
- *data* - formula definition, guarded transformation and formula utility routines
- *parser* - simple parser for formulas
- *utils* - project-independent utilities and algorithms

The syntax of a formula of the modal μ-calculus is defined in *data/formula.ml* and looks as follows.

Listing 1: Formula

```
type formula = FProp of string
```

```
2               | FVariable of string
3               | FNeg of formula
4               | FAnd of formula * formula
5               | FOr of formula * formula
6               | FDiamond of string * formula
7               | FBox of string * formula
8               | FMu of string * formula
9               | FNu of string * formula;;
```

The parity game solving algorithm of Stevens / Stirling is implemented according to [SS98] in *utils/stirlingTest.ml*

Listing 2: Stirling test

```
1  module StirlingTest = struct
2
3  let eloise = true;;
4
5  let abelard = false;;
6
7  type timestamp = int;;
8
9  let nextTimestamp ts = ts + 1;;
10
11 type priority = int;;
12
13 type 'q decisions = ('q * timestamp) list;;
14
15 type 'q assumption = 'q * priority * timestamp;;
16
17 type 'q assumptions = 'q assumption list;;
18
19 type 'q playListEntry = 'q * priority * 'q list * timestamp;;
20
21 type 'q playListStack = 'q playListEntry list;;
22
23 class ['q] stirlingTest (parityGame: ('q) Automata.parityGame) (cb: unit ->
       unit) =
24   let timer = ref 0 in
25   let eloiseAssumptions = ref [] in
26   let abelardAssumptions = ref [] in
27   let eloiseDecisions = ref [] in
28   let abelardDecisions = ref [] in
29   object(self)
30
31     method private getPlayerDecisions = function
32       true -> eloiseDecisions
33     | false -> abelardDecisions
34
35     method private  getPlayerAssumptions = function
36       true -> eloiseAssumptions
37     | false -> abelardAssumptions
38
39     method private getPlayer = parityGame#isExistsPos
40
41     method private getPlayerByPriority i = (i mod 2 = 0)
42
43     method private other p = not p
44
45     method private dropDecisions ts ds = List.filter (fun (_, ts') -> ts' < ts)
           ds
46
47     method private addDecision ts dr c =
48
49       let rec addDec = function
```

```
50          [] -> [(c, ts)]
51        | ((c', ts')::xs) -> if c' = c then (c', ts')::xs
52                             else (c', ts')::(addDec xs)
53        in
54          addDec dr
55
56      method private decisionApplies dr c = List.mem_assoc c dr
57
58      method private getAssumption assums c prio = match assums with
59        [] -> None
60      | ((c', prio', ts)::xs) -> if (c=c') && (prio = prio') then Some ts else
            self#getAssumption xs c prio
61
62      method private addAssumption assums c prio ts = match (self#getAssumption
            assums c prio) with
63        None -> (c, prio, ts)::assums
64      | Some _ -> assums
65
66      method private dropAssumptions ts assums = List.filter (fun (_, _, ts') ->
            ts' < ts) assums
67
68      method private getCycle c l =
69        let rec getCycle' c acc = function
70          [] -> []
71        | ((c', p, o, ts)::xs) -> if c = c' then (c', p, o, ts)::acc
72                                             else getCycle' c ((c', p, o, ts)::acc)
                                                      xs
73        in getCycle' c [] l
74
75      method private getCyclePriority cycle =
76        let rec getCycleP p = function
77          [] -> p
78        | ((_, p', _, _)::ps) -> getCycleP (max p p') ps
79        in getCycleP 0 cycle
80
81      method private explore
82                     (config: 'q)
83                     (playList: 'q playListStack) =
84        timer := nextTimestamp !timer;
85        cb();
86        let player = self#getPlayer config in
87        let dr = self#getPlayerDecisions player in
88            if self#decisionApplies !dr config then
89                self#backtrack player playList config
90            else match (self#getCycle config playList) with
91                [] -> let succs = parityGame#transition config in
92                      if List.length succs = 0 then
93                         self#backtrack (self#other player) playList config
94                      else
95                         let config2 = List.hd succs in
96                         let prio = parityGame#omega config in
97                         let playList2 = (config, prio, List.tl succs, !timer) ::
                                playList in
98                         self#explore config2 playList2
99              | cycle ->
100                   let pr = self#getCyclePriority cycle in
101                   let pl = self#getPlayerByPriority pr in
102                   let ass = self#getPlayerAssumptions pl in
103                  (ass := self#addAssumption !ass config pr !timer;
104                   self#backtrack pl playList config)
105
106     method private backtrack
107                    (pA: bool)
108                    (playList: 'q playListStack)
109                    (config: 'q) =
110       timer := nextTimestamp !timer;
111       match playList with
```

```
112           [] -> pA
113         | ((c, pr, o, _)::t) -> let pB = self#other pA in
114             if (self#getPlayer c = pB) && (List.length o > 0) then
115                 self#explore (List.hd o) ((c, pr, List.tl o, !timer)::t)
116             else let dA = self#getPlayerDecisions pA in
117                  let dB = self#getPlayerDecisions pB in
118                  let aB = self#getPlayerAssumptions pB in
119                  (dA := self#addDecision !timer !dA config;
120                   (match (self#getAssumption !aB c pr) with
121                      Some ts -> (aB := self#dropAssumptions ts !aB;
122                                  dB := self#dropDecisions ts !dB)
123                    | None -> ());
124                   self#backtrack pA t c)
125
126     method isValid = self#explore (parityGame#q0) []
127
128 end;;
129
130 let isValid pG = let t = new stirlingTest (pG) (fun () -> ()) in t#isValid;;
131
132 let isValid' pG cb = let t = new stirlingTest (pG) cb in t#isValid;;
133
134 end;;
```

The three automata have been implemented quite straight forwardly, therefore they are just looking as in the definitions of the preceding chapter.

4.3 Experimental results

We have tested our rudimentary and unoptimized implementation with several examples, in the following we present three of our tests.
The first formula $TransInv_n := Sys_n \Rightarrow Inv_n$ describes an axiomatic system and an invariant. The axiomatic system roughly says that if there is a p_i-edge on a path in the transition system, then there will be also a $p_{i'}$-edge afterwards, where $i' := (i \mod n) + 1$:

- $Single_n := \nu X.(\langle x\rangle X \wedge \neg \bigvee_j \langle p_j\rangle tt) \vee \bigvee_{i=1}^{n} (\langle p_i\rangle X \wedge \neg\langle x\rangle tt \wedge \neg \bigvee_{j\neq i} \langle p_j\rangle tt)$
- $Item_i := \nu X.(\langle x\rangle X \vee \bigvee_{j\neq i} \langle p_j\rangle X \vee \langle p_i\rangle(\mu Y.(\langle x\rangle Y \vee \langle p_i'\rangle X \bigvee_{j\neq i'} \langle p_j\rangle Y))))$

where $Sys_n := Single_n \wedge \bigwedge_{i=1}^{n} Item_i$.
Our invariant demands that if there is a p_1-edge on a path, then there will be infinitely many p_1-edges:

$$Inv_n := \nu X.(\langle x\rangle X \vee \bigvee_{j\neq 1} \langle p_j\rangle X \vee \langle p_1\rangle(\nu Y.\mu Z.(\langle x\rangle Z \vee \bigvee_{j\neq 1} \langle p_j\rangle Z \vee \langle p_1\rangle Y))))$$

The second formula $Petri_n := Model_n \Rightarrow Net_n$ is a practical test that describes a simple petri net and a valid model in an axiomatic way. There are n processes, one operator p and one access pebble that can be moved

$TransInv_n$				
n	States	Transitions	Index	Stevens / Stirling
1	1143	1814	75	00:00
2	21523	39927	137	00:00
3	-	-	-	00:25
4	-	-	-	23:53
5	-	-	-	-

Figure 1: Complexity measures for $TransInv_n$

from a process p_i to the operator using the edge s_i or vice versa using the edge t_i.

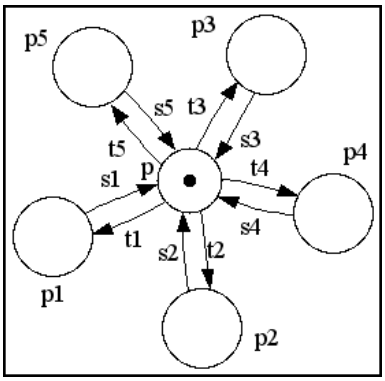

Figure 2: Petri net of size $n = 5$

Our net demands the following axioms:

- Well-definedness: $Well_n := \nu X.(p \wedge \bigvee_{i=1}^{n}(\langle t_i\rangle(p_i \wedge \langle s_i\rangle X)))$
- Fairness: $Fair_n := \bigwedge_{i=1}^{n} \nu X_i.\mu Y_i.(\langle t_i\rangle X_i \vee \langle s_i\rangle Y_i \vee \bigvee_{j\neq i}(\langle t_j\rangle Y_i \vee \langle s_j\rangle Y_i))$
- Exlusiveness: $Excl_n := \nu X. \bigvee_{i=1}^{n}(\langle t_i\rangle(X \wedge p_i \wedge \neg p \wedge \neg \bigvee_{j\neq i} p_j) \vee \langle s_i\rangle(X \wedge p \wedge \neg \bigvee_j p_j))$

where $Net_n := Well_n \wedge Fair_n \wedge Excl_n$.
And our simple model just circulates the pebble:

$$Model_n := \nu X.((p \wedge \neg \bigvee_j p_j) \wedge \langle t_1 \rangle ((p_1 \wedge \neg p \wedge \neg \bigvee_{j \neq 1} p_j) \wedge \langle s_1 \rangle ((p \wedge \neg \bigvee_j p_j) \wedge \langle t_2 \rangle (.... \wedge \langle s_n \rangle X)....)))$$

$Petri_n$				
n	States	Transitions	Index	Stevens / Stirling
1	365	538	77	00:00
2	1731	2583	173	00:00
3	8408	12737	305	00:00
4	31983	49340	473	00:00
5	102700	161870	677	00:02
6	-	-	-	00:05

Figure 3: Complexity measures for $Petri_n$

The third and last formula $Nester_n := \varphi \vee \neg\varphi$ is taken from [DHL06] and is chosen as an example with several alternating fixpoints:

$$\varphi := \mu X_1.\nu X_2.\mu X_3....\sigma X_n.q_1 \vee \langle a \rangle (X_1 \wedge (q_2 \vee \langle a \rangle (X_2 \wedge ...(q_n \vee \langle a \rangle X_n)...)))$$

$Nester_n$				
n	States	Transitions	Index	Stevens / Stirling
1	54	79	21	00:00
2	8668	14654	65	00:00
3	-	-	-	-

Figure 4: Complexity measures for $Nester_n$

Although the system is quite unoptimized - both theoretically and practically - the results are quite promising. Looking at the petri net example for instance, our implementation terminated successfully within seconds. Other formulas, however, with many alternating fix points, could only be solved upto very low values of n.

4.4 Further work

As already mentioned, our rudimentary implementation is hardly optimized and so is the induced parity game. There are many optimizations that could be done, for instance, by determinization of the proof-rule application.
In addition to that it would be very interesting to see how other promising parity game solving algorithms are performing in comparison to the Stevens/Stirling-method, such as the strategy improvement algorithm of Jens Vöge or the sat reduction of Martin Lange which is based on [Jur00].
Another very interesting question is whether the category-theoretical method for μTL is applicable to decide the validity since such construction performed quite better in the case of the Linear Time μ-calculus [DHL06].
Finally the main task that remains is obviously the correctness of our system. Although the system intuitively appears to be correct, the proof for that seems to be quite difficult and requires a deeper analysis of the fixpoint structure and the parity game properties.

References

[Dax06] Christian Dax. Games for the linear time μ-calculus. Master's thesis, Dep. of Computer Science, University of Munich, 2006. Available from http://www.tcs.ifi.lmu.de/lehre/da_fopra/Christian_Dax.pdf.

[DHL06] Christian Dax, Martin Hofmann, and Martin Lange. A proof system for the linear time μ-calculus. 2006.

[Fri06] Oliver Friedmann. A proof system for the modal μ-calculus, 2006. Dep. of Computer Science, University of Munich. Available from http://www.tcs.ifi.lmu.de/lehre/da_fopra/.

[Jur00] Marcin Jurdzinski. Small progress measures for solving parity games. *Proceedings of 17th Int. Symp. on Theoretical Aspects of Computer Science, STACS 2000*, February 2000. Available from http://www.dcs.warwick.ac.uk/~mju/Papers/Jur00-STACS.ps.

[Koz83] Dexter Kozen. Results on the propositional mu-calculus. 1983.

[Mat02] Radu Mateescu. Local model-checking of modal mu-calculus on acyclic labeled transition systems. *Proc. 8th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, TACAS02*, 2002. Available from ftp://ftp.inrialpes.fr/pub/vasy/publications/cadp/Mateescu-02.pdf.

[Pit06] Nir Piterman. From nondeterministic buchi and streett automata to deterministic parity automata. *Proc. 21st Symposium on Logic in Computer Science*, 2006. Available from http://mtc.epfl.ch/~piterman/publications/2006/Pit06.pdf.

[SS98] Perdita Stevens and Colin Stirling. Practical model-checking using games. *1/98 TACAS 98*, 1998. Available from http://homepages.inf.ed.ac.uk/perdita/tacas98tech.ps.

[Wal95] Igor Walukiewicz. Completeness of Kozen's axiomatization of the propositional μ-calculus. In *Proceedings 10th Annual IEEE Symp.*

on Logic in Computer Science, LICS'95, San Diego, CA, USA, 26–29 June 1995, pages 14–24. IEEE Computer Society Press, Los Alamitos, CA, 1995.

[Wal00] Igor Walukiewicz. Completeness of kozens axiomatisation of the propositional μ-calculus. *Inf. Comput., 157(1-2):142182*, 2000. Available from https://www.labri.fr/publications/l3a/2000/Wal00/igw_kozen_compl.pdf.