A Tableau Proof System for a Mazurkiewicz Trace Logic with Fixpoints

Peter Niebert[1] and Barbara Sprick[2]

[1]Institut für Informatik,
Universität Hildesheim, Germany
niebert@informatik.uni-hildesheim.de

[2]Fachbereich Informatik, Lehrstuhl 6
Universität Dortmund, Germany
sprick@ls6.informatik.uni-dortmund.de

Abstract. We present a tableau based proof system for νTrTL, a trace based temporal logic with fixpoints. The proof system generalises similar systems for standard interleaving temporal logics with fixpoints. In our case special attention has to be given to the modal rule: First we give a system with an interleaving style modal rule, later we use a technique similar to the sleep set method (known from finite state model checking) to obtain a more efficient proof rule. We briefly highlight the relation of the improved rule with recent advances in tableau systems for classical propositional logic, the *tamed cut* of the system *KE*.
The treatment of fixpoints leads to possibly infinite tableaux, which however can be finitely represented, yielding treelike structures with back loops: we show this using an automata construction. Indirectly we obtain a (known) decidability result.

1 Introduction

Temporal logics are a widely accepted specification and verification formalism in several areas of computer science, in particular in the field of *reactive systems*. The most widely known logic in this area is *linear time temporal logic*, *LTL*: this logic assumes a discrete and totally ordered time structure (natural numbers as time instances) and is based on two modalities, "Next" and "Until". This framework is appropriate for the specification of systems based on global states.

On the other hand a lot of effort went into the development of semantic frameworks for *distributed systems*, typically consisting of several subsystems (*locations*), which work independently and collaborate using a communication mechanism. In such a setting it is more appropriate to use *partially ordered* or *distributed runs* (which coincide with infinite Mazurkiewicz traces [Maz95]) as paradigm.

Recently Thiagarajan has defined *TrPTL* [Thi94] as a seemingly natural *trace based* generalisation of *LTL* to partially ordered runs. The key idea is to interpret temporal operators such as "Until" and in particular "Next" only relatively to the "view" of one location (local time), and it allows to *change the point of view* in order to express properties of several locations. In [Nie95, Huh96] νTrTL, a fixpoint extension of *TrPTL* has been developed. This logic combines the *local Next* modality with the possibility of fixpoint definitions (recursion in formulae), so that e.g. Thiagarajan's *local Until* can be defined as a derived operator. Similar fixpoint extensions are known for standard temporal logics (e.g. νTL [Var88]). The structure of νTrTL is advantageous over *TrPTL* in that it allows a semantically cleaner *changing of the*

point of view: νTrTL formulae only look into the future behaviour of a system, while the changing of the point of view in [Thi94] results in an uncontrolled jump into the past. This property of our logic is also essential for the proof system.

There are more reasons to be interested in νTrTL: it is well known, that νTL (the fixpoint extension of LTL) is expressively equivalent to Büchi-automata and to the *monadic second order theory of the temporal frames*. It is conjectured (work in progress), that an analogous result also holds for νTrTL, i.e. that this logic is expressively equivalent to the monadic second order theory of distributed runs.

In this work we develop a tableau based proof system for νTrTL. The structural properties of νTrTL allow an elegant formulation of proof rules. Since the treatment of fixpoints in proof systems requires lengthy constructions (see e.g. [Wal95] for the treatment of the propositional μ-calculus), we put the emphasis on the particular aspects of νTrTL. We treat the fixpoints in a system with infinite (depth, not width) tableaux. Using automata constructions it can be shown, that the infinite proofs can be finitely represented (as trees with back loops) and automatically constructed. Thus indirectly we obtain a decision procedure for the satisfiability problem (already directly addressed in [Nie95]).

The most important rule of our proof system is the *modal rule*, i.e. the rule for the treatment of the *Next* modalities. This rule uses a case analyses about the possible next steps occuring at some point in a distributed run and leads to branching in the tableaux similar to the rule for disjunction.

An observation concerning the modal rule interesting in two ways is that the tableau system can be made more efficient by making the cases of the previously mentioned case analysis *mutually exclusive*. On the one hand, we relate this modified modal rule to the *tamed cut* of D'Agostino's and Mondadori's system *KE* [DM94]. On the other hand, the construction we use to make the modal rule more efficient *operationally* behaves similarly to a reduction method used to make finite state model checking more efficient: the sleep set method [GW91].

Sleep sets are one way to exploit the independence of actions in the system in the search for particular states (e.g. a deadlock). Technically, during the depth first search, a set of actions (called sleep set), the exploration of which would be redundant (because they already occurred in another order), is carried around.

The analogy in the tableau system is the exploration of sub tableaux at applications of the modal rule: instead of actions we now have to deal with modalities. However instead of externally keeping track of modalities not to be explored anymore, the modified modal rule introduces formulae like $[a]\mathbf{false}$ (a cannot occur) into the sequents. The interplay with the other (standard) rules then *operationally* stops the expansion of redundant tableau parts.

The rest of the paper is structured as follows: In section 2 we give a simplistic example of a distributed system and formally introduce the notion of *distributed runs*, which constitute the semantic models over which νTrTL is interpreted. In section 3 we give the syntax and semantics of νTrTL and illustrate its use in a small example. In section 4 the tableau system is given, the proof of the soundness and completeness is given in section 5. In section 6 we discuss two aspects of automatic proof search: the finite representation of infinite rules and the improved modal rule.

The paper is based on the diploma thesis of the second author [Spr96], which contains a detailed presentation of the present work.

2 Runs of distributed programs

Distributed programs. The location based approach to Mazurkiewicz-Traces considers parallel systems of sequential processes, which communicate via joint actions [Thi94]. Let us first have a look at the Alternating Bit Protocol (ABP) [Mil89] as a small example of a concurrent program: The idea is that two components, a sender S and a receiver R communicate over two (unreliable) channels. The Message Channel transfers messages from S (send) to R (receive) and the Ack(nowledgement) Channel delivers acknowledgements from R (sack) to S (rack), both channels can lose (losem, losea) packets. Each data message sent by S contains a protocol bit, either 1 or 0 (send1/0). Let us assume the sender sends a 0-message (send0). Before receiving the corresponding acknowledgment (rack0) it can send the 0-message again after a timeout and ignore 1-acknowledgments. After receiving the corresponding ack (rack0) S stops transmitting the current message (send0) and flips the protocol bit to 1 for the next message. The receiver basically works in the same manner: After receiving a message (rec0), R returns an ack to S (sack0). Afterwards R can either resend the ack (sack0) after a timeout or ignore more messages with protocol bit 0 (rec0) before it finally receives a new message with the alternated protocol bit (rec1).

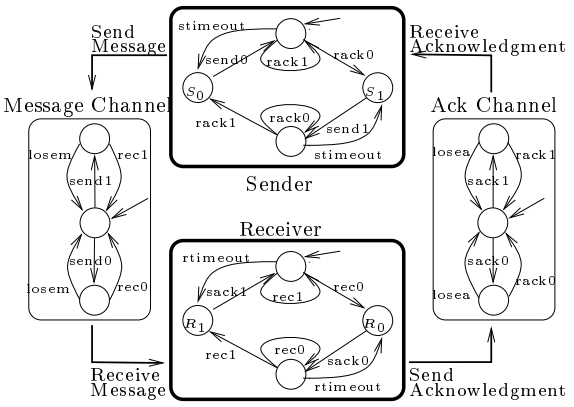

Fig. 1. Alternating Bit Protocol

Distributed runs. We will now introduce distributed runs to represent the behaviour of distributed programs. A run is one possible execution of a distributed program, e.g. figure 2 shows one possible run of the ABP example given above.

We will first define the alphabet for distributed runs as the set of actions, which can take place in such a run. Some of these actions are local to only one component (e.g. stimeout, rtimeout in figure 1), while others may belong to more than one component (e.g. send0, rack1). The latter we call "synchronisations" between the components involved in these actions. Actions which take place at different locations such as "send0" and "sack1" are called independent of each other: there is no natural way to observe a causal order between these actions.

Definition 2.1 (alphabet) *Let $K \in \mathbb{N}$ be fixed. Then $Loc = \{1, \ldots, K\}$ denotes a* **set of locations** *and $\widetilde{\Sigma} = (\Sigma_1, \ldots, \Sigma_K)$ a* **distributed alphabet**, *where each Σ_i is a finite, nonempty set of actions of location i. The sets Σ_i may overlap. We define $\Sigma := \bigcup_{1 \leq i \leq K} \Sigma_i$ as the global alphabet of the system. An action a with $a \in \Sigma_i \cap \Sigma_j$ is called a* **synchronisation** *between the locations i and j and $Loc(a) := \{i \in Loc | a \in \Sigma_i\}$ denotes the set of locations which are synchronised by action a.*
Two actions a and b are **independent**, *$(a \mathcal{I} b)$, iff $Loc(a) \cap Loc(b) = \emptyset$.*

It is easy to see, that the independence relation given in 2.1 is irreflexive and symmetric. Thus $(\Sigma, \mathcal{I})$ is a concurrent alphabet in the sense of Mazurkiewicz [Maz95] and we give here a location based approach to Mazurkiewicz traces.

Let us now define distributed runs as a tuple of a frame and interpretations of these frames. A distributed run stands for an execution of a distributed program:

Definition 2.2 (frame) *Let $\widetilde{\Sigma}$ be a distributed alphabet. A* **frame** *over $\widetilde{\Sigma}$ is a labelled poset $F = (E, \leq, l)$, where E is a countable set of events, $\leq$ a partial order on E and $l : E \to \Sigma$ a labelling-function. Let $E_i = \{e \in E \mid l(e) \in \Sigma_i\}$ be the set of i-events. For each i the restriction $\leq \cap (E_i \times E_i)$ is total, (i.e. the events of one location are causally ordered), and the global order $\leq$ is the least partial order containing the local (total) orders.*

Note that runs can either be finite or infinite. To talk about dynamic behaviour of these static frames we use the notion of configurations. A configurations of a frame gives informations about a certain state of a distributed program. The configuration c_1 in our example run in figure 2 represents the state of the program, in which only the sender and the message channel have performed the action send0, but the other components are still in their initial state. So a configuration contains all the actions of a distributed run, that have occurred so far in the computation.

Definition 2.3 *A* **configuration** *c of a frame F is a finite, with respect to $\leq$ downward closed set of events. $\mathcal{C}_F$ denotes the set of all configurations of F. Two configurations c, c' are i-***equivalent** *($c \equiv_i c'$), iff $c \cap E_i = c' \cap E_i$. They are A-***equivalent** *($\equiv_A$) for $A \subseteq Loc$ iff they are i-equivalent for each $i \in A$. We define a* **successor relation** *so that $c \xrightarrow{a} d$ iff $d = c \uplus \{e\}$ with $l(e) = a$.*

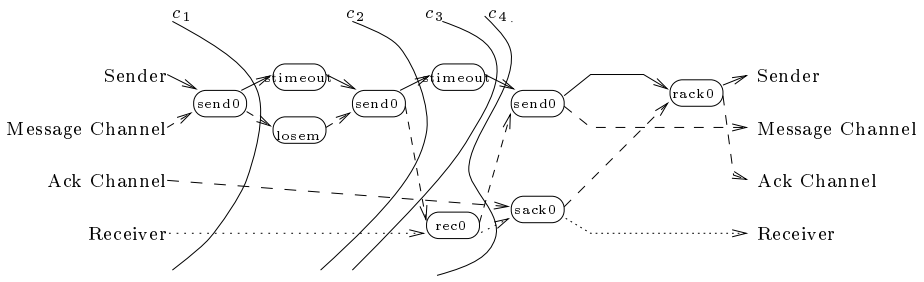

Fig. 2. One distributed run of the alternating bit protocol

Let us have a look at the example again: The configurations c_2 and c_3 in 2 are equivalent with respect to the receiver ($c_2 \equiv_{Receiver} c_3$) but not with respect to the sender ($c_2 \not\equiv_{Sender} c_3$): c_2 evolves into c_3 by performing an action which is local to the sender. From the receivers "point of view" the configuration did not change, the receiver does not "know" whether any of the other components has performed an action or not. Each configuration matches a certain state of the distributed program. At different states of the program, different properties might hold. We call these properties atomic propositions. Each location has its own atomic propositions. E.g.

we could take S_0 as an atomic proposition for "the sender is in S_0" or R_0 as "the receiver is in R_0". Looking at our example, these propositions are satisfied at the configuration c_3 but not at the configuration c_2 (R_0 is still true, but S_0 is not). Note that atomic propositions belonging to one location cannot be changed by actions which take place at other locations: The action which leads from c_2 to c_3 is local to the sender and thus does not have any effect to the proposition R_0, which is local to the receiver. We will now define an interpretation of a frame as a mapping from atomic propositions to the set of configurations:

Definition 2.4 *Let $\widetilde{\mathcal{P}} = (\mathcal{P}_1, \ldots, \mathcal{P}_K)$ be a distributed set of local propositions. Here, $\mathcal{P}_i$ and $\mathcal{P}_j$ are disjoint for $i \neq j$ and $\mathcal{P}_i$ denotes the set of propositions affiliated with location i.*
An **interpretation of a frame** *is a mapping $I : \widetilde{\mathcal{P}} \longrightarrow 2^{C_F}$ such that $c \equiv_i c'$ implies that $c \in I(P)$ iff $c' \in I(P)$ for all $P \in \mathcal{P}_i$, i.e. the interpretation of propositions of location i depends only on i-events.*

Now we are finally able to define a distributed run as one execution of a distributed program together with an interpretation:

Definition 2.5 *A frame F together with an interpretation I is called a* **distributed run** *$M = (F, I)$.*

3 A logic for distributed programs

We will now define a logic for the specification of distributed program executions as given in the previous section. The logic, called νTrTL, is the revised version of a logic first given in [Nie95] and later in [Huh96].

Syntax and Semantics. Let *Loc*, $\widetilde{\Sigma} = (\Sigma_1, \ldots, \Sigma_K)$ and $\widetilde{\mathcal{P}} = (P_1, \ldots, P_K)$ be defined as in section 2.

The propositions from the sets P_i will form a part of the atomic formulae of the logic. Similarly to these *propositional constants*, the meaning of which is given by the interpretation I of a distributed run (F, I), we also need *propositional variables*, written as $X, Y, Z, \ldots \in \mathcal{V}$. Just as I gives a meaning to the propositions we furthermore need a *valuation function* $v : \mathcal{V} \longrightarrow 2^{C_F}$, i.e. each variable stands for a set of configurations, which is given by v.

The key idea of the logic itself - and of the presentation given here - is that the formulae of the logic look at the configurations, i.e. the global state during a run, from a *local* point of view: Some formulae look at the state of the system from the point of view of a single location (e.g. in the case of local propositions), others may involve a joint look from several locations (e.g. at the very beginning of a run or after a joint action of these locations). This idea is reflected in the syntax by a *family* of sets of formulae Φ_A (looking from the point of view of $A \subseteq Loc$):

Definition 3.1 (Syntax) *The syntax of the logic νTrTL consists of sets Φ_A of formulae, where $A \subset Loc$ denotes the* **type** *of the formulae in Φ_A, i.e. a set of locations (to which the formulae directly refer). We also write $type(\phi) = A$ for*

$\phi \in \Phi_A$. *The set of all formulae is denoted by* $\Phi := \bigcup_{A \subseteq Loc} \Phi_A$.
The sets Φ_A *are defined to be the least sets, such that:*

$$\begin{aligned}
\{\mathbf{true}, \mathbf{false}\} \cup \mathcal{V} &\subseteq \Phi_\emptyset \\
P, \neg P &\in \Phi_{\{i;\ P \in \mathcal{P}_i\}} \\
\phi \in \Phi_A, \psi \in \Phi_B &\Rightarrow \phi \wedge \psi, \phi \vee \psi \in \Phi_{A \cup B} \\
i \in Loc(a), A \subseteq Loc(a), \phi \in \Phi_A &\Rightarrow \langle a \rangle_i \phi, [a]_i \phi \in \Phi_{\{i\}} \\
\phi[X := \phi] \in \Phi_A &\Rightarrow \mu X.\phi, \nu X.\phi \in \Phi_A
\end{aligned}$$

The operators μ *and* ν *bind the variables. By* $\phi[X := \psi]$ *we denote the formula obtained by substituting all free occurrences of* X *in* ϕ *by* ψ. *A formula that does not contain any free variables is* **closed**.

Note that for $A, B \subseteq Loc$ with $A \neq B$ we have $\Phi_A \cap \Phi_B = \emptyset$, i.e. every formula has a *unique type*. We will give some examples of νTrTL formulae later.

Further note that using sets Φ_A of formulae there is a subtle restriction on the the way formulae can be constructed: e.g. within a formula $[a]_i\phi$ the formula ϕ may only directly refer to locations from $Loc(a)$. Hence *changing the point of view* in a formula is only possible via a modality refering to a common action of the *old* and the *new* point of view. We will further comment this issue at the example section.

For convenience we only allow negation of atomic propositions. However the logic is closed under negation, because every operator has its *dual*, and negations can be drawn inside down to the atomic propositions. Let $not(\phi)$ denote the negation of ϕ.

The operator μ defines the least and ν the greatest fixpoint. Since both kinds of fixpoints are often treated equally, we use σ as wild card for both operators.

Definition 3.2 (Semantics of the logic νTrTL) *Let* (F, I) *be a distributed run and* v *a valuation function. The semantics of a formula* $\phi \in \Phi$ *with respect to* (F, I) *and* v *is denoted by* $[\![\phi]\!]_v^{(F,I)}$. *It is inductively defined by:*

- $[\![\mathbf{true}]\!]_v^{(F,I)} = C_F$, $[\![\mathbf{false}]\!]_v^{(F,I)} = \emptyset$ *(all configurations satisfy* **true**, *none* **false**)
- $[\![P]\!]_v^{(F,I)} = I(P)$, $[\![\neg P]\!]_v^{(F,I)} = \mathcal{C}_F \setminus [\![P]\!]_v^{(F,I)}$, $[\![X]\!]_v^{(F,I)} = v(X)$ *for* $P \in \mathcal{P}$, $X \in \mathcal{V}$
- $[\![\phi \wedge \psi]\!]_v^{(F,I)} = [\![\phi]\!]_v^{(F,I)} \cap [\![\psi]\!]_v^{(F,I)}$, $[\![\phi \vee \psi]\!]_v^{(F,I)} = [\![\phi]\!]_v^{(F,I)} \cup [\![\psi]\!]_v^{(F,I)}$
- $[\![\langle a \rangle_i \phi]\!]_v^{(F,I)} = \{c \mid \exists r, c' \ \ with\ (c \equiv_i c')\ and\ c' \xrightarrow{a} r\ and\ r \in [\![\phi]\!]_v^{(F,I)}\}$
- $[\![[a]_i \phi]\!]_v^{(F,I)} = \{c \mid \forall c', r \ \ with\ (c \equiv_i c')\ and\ c' \xrightarrow{a} r\ implies\ r \in [\![\phi]\!]_v^{(F,I)}\}$
- $[\![\nu X.\phi]\!]_v^{(F,I)} = \bigcup\{A \mid A \subseteq [\![\phi]\!]_{v[X:=A]}^{(F,I)}\}$, $[\![\mu X.\phi]\!]_v^{(F,I)} = \bigcap\{A \mid [\![\phi]\!]_{v[X:=A]}^{(F,I)} \subseteq A\}$
  where $v[X := A](X) = A$ *and for* $Y \neq X$ *we have* $v[X := A](Y) = v(Y)$.

A configuration $c \in \mathcal{C}_F$ *satisfies a formula* ϕ *iff* $c \in [\![\phi]\!]_v^{(F,I)}$. *A distributed run* (F, I) *satisfies a formula* ϕ *iff the initial configuration does, i.e. iff* $\emptyset \in [\![\phi]\!]_v^{(F,I)}$. *A distributed program satisfies a formula, iff all of its distributed runs do.*

The following observation [Koz83] eases the formulation of our proof-system:

Proposition 1 (guarded formulae). *A formula is* ψ **guarded** *iff in each subformula of the form* $\sigma X.\phi$ *of* ψ *all free occurrences of the fixpoint variable* X *lie in the scope of a modality* $\langle a \rangle_i \phi$ *or* $[a]_i \phi$. *Every formula is equivalent to a guarded formula.*

Sample Properties. In this subsection we want to illustrate the way the logic can express properties of distributed runs. Consider the example run in figure 2.

In section 2 we have already shown at which configurations propositions (as simpliest formulae) hold, so let us now look at a formula with a single modality: $\langle rec0\rangle_{Receiver} R_0$. Informally we should read this formula as "*From the point of view of the receiver the next action to occur is rec0 and afterwards it will be in state* R_0". Formally this formula holds for instance at the configuration c_1, because that configuration (where the receiver has not even started) is equivalent to c_3 from the receiver's point of view, and in c_3 the action rec_0 is enabled leading to configuration c_4, where (according to the interpretation) R_0 holds. Note that the formula does not say, that rec_0 is immediately enabled in c_1, but will rather be enabled by some preparatory events ($timeout, losem, send0$) of the other locations.

Next we will build up a more complex formula to illustrate the use of fixpoints in formulae. We want to formalise the following property (of all runs of the ABP):

AlwsAckMeansRcvd: "*Whenever the sender participates in a send*0*-event and eventually (after finitely many stimeout and repeated send*0 *actions) receives a rack*0*, then (after the first send*0*) the message channel will only finitely often lose a message (and receive another send*0*-request) before transmitting rec*0 *to the receiver. The receiver then will enter the* R_0 *state (indicating the reception of a* 0*-tagged message)*".

We formulate the property using several abbreviations for subformulae (for better readability we use $\phi \to \psi$ as an abbreviation for $not(\phi) \vee \psi$, S for $Sender$ and MC for $MessageChannel$):

$$AlwsAckMeansRcvd = \nu X.\left(AckMeansRcvd \wedge \left(\bigwedge_{a\in\Sigma_S} [a]_S X\right)\right)$$

$$AckMeansRcvd = (\langle send0\rangle_S \mathbf{true} \wedge EvtllyRcvsAck) \to ChanWillTransmt$$

$$EvtllyRcvsAck = \mu Y.\langle rack0\rangle_S \mathbf{true} \vee \langle stimeout\rangle_S \langle send0\rangle_S Y \vee \langle rack1\rangle_S Y$$

$$ChanWillTransmt = \mu Z.\langle rec0\rangle_{MC} R_0 \vee \langle losem\rangle_{MC} \langle send0\rangle_{MC} Z.$$

Note how recursion is used in the above formulae: to formulate "always" we use a greatest fixpoint, which corresponds to an infinite unfolding of the formula. In the other cases we deal with *finite recursion*, i.e. we use a least fixpoint to allow an arbitrary but finite expansion of the formula: e.g. we may only cycle finitely often through the $losem-send0$-cycle in ChanWillTransmt until finally we have to commit to the action $rec0$.

In the formulation of the property we walk along a *causal chain* (i.e. a maximal totally ordered set) of events, which is typical for this logic. Also note that the logic does not directly allow us to say things involving changes of the point of view except via common actions (in the example via $send0$ from Sender to Message Channel, and via $rec0$ from Message Channel to Receiver). For instance we cannot directly say "when the sender receives a $rack0$ the receiver is in state R_0".

This does not mean a general restriction of the expressiveness of the logic for the specification of properties of *complete runs*, only it may not be possible to write down "global" invariants (for example) in a compositional way.

4 A tableaux system

Let us now define a tableau proof system for the validity of formulae of νTrTL. Let Γ be a finitary set of formulae. We will call $\Gamma \vdash$ a *sequent*. Given a distributed run (F, I) and a configuration $c \in \mathcal{C}_F$, the configuration c satisfies a sequent $\Gamma \vdash$ iff a formula $\gamma \in \Gamma$ exists such that $c \notin [\![\gamma]\!]_v^{(F,I)}$.
For these sequents we want to define a tableau proof system, which accepts a sequent iff the sequent is valid and rejects it otherwise. Here validity is defined as follows:

Definition 4.1 *A sequent $\Gamma \vdash$ is* valid *iff* $\forall (F, I), v : \bigcap_{\gamma \in \Gamma} [\![\gamma]\!]_v^{(F,I)} = \emptyset$

Definition 4.2 (rules) *In the following let $\Gamma_{\langle\,\rangle}$ ($\Gamma_{[\,]}$) $\subseteq \Gamma$ denote the set of formulae in Γ of the form $\langle a\rangle_i\phi$ ($[a]_i\phi$) $\in \Gamma$, and let Γ_P be the set of propositions in Γ. Let $\mathcal{T}$ be the following set of tableau-rules, which are divided into three groups: axioms, logical rules and a modal rule.*

Axioms and logical rules

$$\frac{\Gamma, P, \neg P \vdash}{} \quad (1) \qquad \frac{\Gamma, \mathbf{false} \vdash}{} \quad (2) \qquad \frac{\Gamma, \mathbf{true} \vdash}{\Gamma \vdash} \quad (3)$$

$$\frac{\Gamma, \phi \wedge \psi \vdash}{\Gamma, \phi, \psi \vdash} \quad (4) \qquad \frac{\Gamma, \phi \vee \psi \vdash}{\Gamma, \phi \vdash \;\; \Gamma, \psi \vdash} \quad (5) \qquad \frac{\sigma X.\alpha, \Gamma \vdash}{\alpha[X := \sigma X.\alpha], \Gamma \vdash} \quad (6)$$

Modalities

$$\frac{\Gamma_P, \Gamma_{\langle\,\rangle}, \Gamma_{[\,]} \vdash}{\Gamma_{a_1} \vdash,\; \Gamma_{a_2} \vdash, \ldots \Gamma_{a_n} \vdash} \quad (7)$$

In the modal rule $\Gamma_{\langle\,\rangle}$ must be non-empty and Σ' and Γ_a are defined as follows:

- $a_i \in \Sigma', \Sigma' := \{a \in \Sigma | \exists \phi \in \Gamma_{[\,]}, \Gamma_{\langle\,\rangle}$ *s.t. type*$(\phi) \in Loc(a)\}$
- $[a]_i\phi \in \Gamma_{[\,]} \Rightarrow \phi \in \Gamma_a, \langle a\rangle_i\phi \in \Gamma_{\langle\,\rangle} \Rightarrow \phi \in \Gamma_a$
- $[b]_i\phi \in \Gamma_{[\,]}, i \in Loc(a) \Rightarrow \mathbf{true} \in \Gamma_a,\ \langle b\rangle_i\phi \in \Gamma_{\langle\,\rangle}, i \in Loc(a) \Rightarrow \mathbf{false} \in \Gamma_a$
- $[b]_i\phi \in \Gamma_{[\,]}, i \notin Loc(a) \Rightarrow [b]_i\phi \in \Gamma_a, \langle b\rangle_i\phi \in \Gamma_{\langle\,\rangle}, i \notin Loc(a) \Rightarrow \langle b\rangle_i\phi \in \Gamma_a$
- $P \in \Gamma_P, P \in \mathcal{P}_i, i \notin Loc(a) \Rightarrow P \in \Gamma_a$

The axioms and logical rules are standard. The goal of such a rule is valid, iff all the subgoals are valid. In rule 6 we unwind a fixpoint; only the use of this rule can lead to infinite tableau-paths.

Let us now have a more detailed look at the rule 7. While the rules 1-6 are local to one configuration (a configuration c satisfies the goal of a rule, if it satisfies at least one of the subgoals), the modal rule refers to a step from one configuration c to the next configuration c' by adding one event. This explains, why the modal rules may only be applied after none of the other (local) rules is applicable any more. In the case of $\Gamma_{\langle\,\rangle} = \emptyset$ every configuration in which no more actions can occur can satisfy Γ (depending on the interpretation) Thus we have to make sure, that $\Gamma_{\langle\,\rangle} \neq \emptyset$. Note that the fixpoints are guarded. Thus, always after applying a finite number of rules 1–5, a modal rule has to be applied. Just as the or-rule 5 investigates the two possibilities to satisfy a disjunction, the modal rule does a case-analysis depending on which actions could be performed next in an arbitrary configuration of an arbitrary

run. According to this choice and the semantics of the logic each formula in the goal is linked to a corresponding formula in each subgoal.

In a modal step a with $i \in Loc(a)$, propositions $P \in \mathcal{P}_i$ will be deleted because their interpretation in c' can become different from the one in c. Propositions $P \notin \mathcal{P}_i$ are not concerned by such a step and will thus not be deleted.

Definition 4.3 (tableaux) *Given a set of formulae Γ. A tableau $\mathcal{T}$ for Γ is a labelled tree $< K, L >$, where K is a tree and L is the labelling function, such that*
- the root of K is labelled with $\Gamma \vdash$,
- if $L(n)$ is a tableau-axiom, then n is a leaf of K
- if $L(n)$ is not an axiom, then the children of n in K are created and labelled according to the tableau-rules. $L(n)$ is the goal and the labels of the children the subgoals of a tableau-rule.

An acceptance condition is needed such that a tableau is accepted iff the sequent is valid. In the case of finite tableaux, we can accept, if all leaves are labelled with axioms. Hence the proof-system is already sound and complete for logic νTrTL without fixpoints. Because of the induction-rule 6 which unwinds fixpoints, we can also create infinite paths. These paths require a different acceptance condition. For this we have to face two problems: One is the satisfaction of fixpoint-formulae, the second is a particular fairness problem.

To solve the first problem we need the following definition which allows us to observe a particular formula over a path:

Definition 4.4 (trace, μ-trace) *Let $\mathcal{T}$ be a tableau for$\Gamma \vdash$ and $\sigma = v_1, v_2, \ldots$ be an infinite path in the tableau, i.e. v_{i+1} is a child of v_i. A* trace *on the path σ is each sequence of formulae $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ such that $\alpha_j \in L(v_k)$ and α_{j+1} is either*

1. *α_j, if the formula α_j is not reduced by the rule applied in node v_k or*
2. *if α_j was reduced, then α_{j+1} is one of the formulae produced out of α_j.*

A least fixpoint formula $\mu X.\phi = \alpha_i$ is regenerated from α_i to $\alpha_j, i < j$ on a trace, if $\mu X.\phi = \alpha_i$ derives $\mu X.\phi = \alpha_j$ in such a way, that $\mu X.\phi$ is a subformula of each $\alpha_k, i < k \leq j$. We call a trace on which a least fixpoint formula is regenerated infinitely often, a μ-trace.

Considering the second problem, we define a particular notion of fairness for paths. Paths in which a formula $\langle a \rangle_i \psi$ is never evaluated are called *unfair*:

Definition 4.5 (fair tableau paths) *A tableau-path $\sigma = (v_0, v_1, v_2, ...)$ is called* fair, *iff for each i and for each $\langle a \rangle_i \phi \in v_i$ there exists a node $v_j \in \sigma$ with $j \geq i$, such that the modal rule is applied to v_j and there exists a b-child of v_j with $b \in \Sigma_i$. A path, which is not fair, is called* unfair.

Definition 4.6 (acceptance-condition) *Given a tableau $\mathcal{T}$ for a sequent $\Gamma \vdash$. The tableau will accept, iff every leaf is labelled with an axiom, and every infinite tableau-path is either unfair or contains an infinite μ-trace.*

The requirement for finite paths is easy to see, for infinite paths we will give a detailed explanation a bit further on in this paper. The given rules together with the acceptance condition form a sound and complete proof-system for the logic νTrTL. We will prove this in the next section.

Theorem 2 (soundness and completeness). *Let Γ be a set of formulae and let $\mathcal{T}$ be a tableau for the sequent $\Gamma \vdash$. The tableau $\mathcal{T}$ accepts $\Gamma \vdash$ iff $\Gamma \vdash$ is valid.*

5 Soundness and Completeness

We will now give the outline of the soundness and completeness proofs of the proof-system given above (for more details see [Spr96]). In the "standard" way, the soundness and completeness proofs are done by induction over tableau paths: a leaf is valid iff it is an axiom, and an inner goal is valid, iff all its subgoals are valid.
In the case of the proof system without the rules for fixpoints (i.e. a proof system for a reduced logic without the fixpoint operators) we could have applied the induction method for the soundness and completeness proof. But due to the fixpoint rules we have to deal with possibly infinite paths (which might be accepted as well) in the tableau. Thus this induction method cannot be used here.
Instead we will give a global proof based on the correspondence of tableau paths and distributed runs. As a formal framework we use an established tool for fixpoint logics, the Streett/Emerson theorem [ES89]: it gives a different characterisation for the logical satisfaction of fixpoints in the propositional μ-calculus, and it can be adapted to several fixpoint-frameworks. The idea is first to weaken the semantics of μ and ν to arbitrary (not least or greatest) fixpoints, represented in derivation graphs (vaguely corresponding to the interleaved representation of distributed runs) and to then regain the proper semantics by a separate criterion.

We will now use this characterisation to prove soundness and completeness.

Definition 5.1 (derivation graph) *A derivation graph $DG((F,I), d_0, \Gamma)$ for a distributed run (F,I) with the configuration d_0 and a set of formulae Γ is a tuple $(V, \rightsquigarrow)$, with $V \subseteq \Phi \times \Phi$, $\rightsquigarrow \subseteq V \times V$, where $(V, \rightsquigarrow)$ is minimal, such that the following holds (we implicitly assume $(c,\phi),(c',\phi') \in V$ when we write $(c,\phi) \rightsquigarrow (c',\phi')$):*

- $(d_0, \gamma) \in V \quad \forall \gamma \in \Gamma$
- *For $(c,P) \in V, P \in \mathcal{P}$ and $c \notin I(P)$ we get $(c,P) \rightsquigarrow (c, \mathbf{false})$*
- *For $(c,\neg P) \in V, P \in \mathcal{P}$ and $c \in I(P)$ we get $(c,\neg P) \rightsquigarrow (c, \mathbf{false})$*
- *For $(c, \phi \vee \psi) \in V$ we either get $(c, \phi \vee \psi) \rightsquigarrow (c,\phi)$ or $(c, \phi \vee \psi) \rightsquigarrow (c,\psi)$*
- *For $(c, \phi \wedge \psi) \in V$ we get $(c, \phi \wedge \psi) \rightsquigarrow (c,\phi)$ and $(c, \phi \wedge \psi) \rightsquigarrow (c,\psi)$*
- *For $(c, [a]_i\phi) \in V, c \xrightarrow{a} c'$ we get $(c, [a]_i\phi) \rightsquigarrow (c', \phi)$*
- *For $(c, [b]_i\phi) \in V, c \xrightarrow{a} c', a \in \Sigma_i$ we get $(c, [a]_i\phi) \rightsquigarrow (c', \mathbf{true})$*
- *For $(c, [b]_i\phi) \in V, c \xrightarrow{a} c', a \notin \Sigma_i$ we get $(c, [a]_i\phi) \rightsquigarrow (c', [a]_i\phi)$*
- *For $(c, \langle a \rangle_i\phi) \in V, c \xrightarrow{a} c'$ we get $(c, \langle a \rangle_i\phi) \rightsquigarrow (c', \phi)$*
- *For $(c, \langle b \rangle_i\phi) \in V, c \xrightarrow{a} c', a \in \Sigma_i$ we get $(c, \langle b \rangle_i\phi) \rightsquigarrow (c', \mathbf{false})$*
- *For $(c, \langle b \rangle_i\phi) \in V, c \xrightarrow{a} c', a \notin \Sigma_i$ we get $(c, \langle b \rangle_i\phi) \rightsquigarrow (c', \langle b \rangle_i\phi)$*
- *For $(c, \sigma X.\alpha(X)) \in V$ we get $(c, \sigma X.\alpha(X)) \rightsquigarrow (c, \alpha(\sigma X.\alpha(X)))$*

Definition 5.2 (correct) *A derivation graph is called* correct, *iff*

1. *it does not contain a node $(c, \mathbf{false}) \in DG((F,I), d_0, \Gamma)$,*
2. *for each node $(c, \langle a \rangle_i\phi) \in DG((F,I), d_0, \Gamma)$ we have $(c, \langle a \rangle_i\phi) \rightsquigarrow^* (d, \langle a \rangle_i\phi) \rightsquigarrow (d', \phi')$ and $(d', \phi') \in DG((F,I), d_0, \Gamma)$ with $\langle a \rangle_i\phi \neq \phi'$,*

3. it does not contain any path with an infinite regeneration of a least fixpoint.

This correctness definition of derivation graphs needs some further explanations. Item 1 makes sure that all leaves of the derivation graph are satisfied. Since the conditions on derivation graphs are necessarily local (single step), but the semantics of $\langle\,\rangle_i$ refers to global (multi step) jumps, we need the additional condition 2 for resolving $\langle\,\rangle_i$ formulae. Condition 3 is the heart of the original Streett/Emerson theorem ensuring the proper semantics of least fixpoints.

Thus we recast the Streett/Emerson theorem to our framework as follows:

Theorem 3 (Streett/Emerson [ES89]). *A configuration d_0 of a distributed run (F, I) satisfies a formula ϕ iff there exists a correct derivation graph for $((F, I), d_0, \phi)$*

The proof of theorem 2 will be divided into two parts. We will show both directions separately. First we will show that if a tableau $\Gamma \vdash$ accepts a sequent, the sequent is valid. Then, in a second part, we will show that a sequent rejected by the tableau is not valid. Note that for every sequent there exists either a rejecting or an accepting tableau. For finite tableaux the soundness and completeness can easily be shown by induction over the length of the tree. Showing the soundness and completeness for infinite tableaux requires some more work.

Lemma 4. *For $\Gamma \vdash$ not valid there exist rejected paths in any tableau for $\Gamma \vdash$.*

Proof (sketch): Let Γ_0 be a set of formulae and let (F, I) be a distributed run. The configuration d_0 of (F, I) satisfies all formulae in Γ_0.
Now we choose a fair execution of the distributed run, e.g. using round robin. With theorem 3 we know, that there exists a correct derivation graph for $(F, I), d_0, \Gamma_0$.
We then have to show that the tableau $\mathcal{T}$ built for Γ_0 either contains a leaf which is not labelled with an axiom or it has at least one infinite *fair* path without a $\mu - trace$. With the help of the derivation graph we will inductively identify either this leaf or the infinite path $\pi = v_0, v_1, v_2, \ldots$ in the tableau. During this process we assume that each v_i corresponds to a configuration $c_i \in \mathcal{C}_F$ which is given as interleaving in the sequence of actions taken in modal rules along the path from the root to v_i. Further more we assume that for each $\gamma \in \Gamma(v_i)$, (c_i, γ) belongs to $DG((F, I), d_0, \Gamma_0)$. This invariant is easily seen to carry over for rule applications with only one subgoal v_{i+1} (the rules 1, 4, 6). For rules with several subgoals we choose the successor v_{i+1} according to the derivation graph. Let $\Gamma(v_i) = \Gamma_i, \phi \vee \psi$ and let the or-rule be applied next. Depending on $\Gamma(v_{i+1}) = \Gamma_i, \phi$ or $\Gamma(v_{i+1}) = \Gamma_i, \psi$ we either choose the left or the right successor of v_i.
Similarly, we proceed for the modal rule. If this process ends at a leaf v of $\mathcal{T}$, it cannot be an axiom (with $P, \neg P \in \Gamma(v)$) because our invariant would obviously lead to a violation of condition 1 of definition 5.2. On the other hand the path π would be fair by construction, and from the invariant and because of condition 3 of 5.2 the path would not contain an infinite μ-trace. Hence, the tableau will not accept $\Gamma \vdash$.

Lemma 5. *Let Γ_0 be a finite set of formulae and let $\mathcal{T}$ be a tableau starting from Γ_0. If $\mathcal{T}$ contains a rejected path then $\Gamma_0 \vdash$ is not valid and there exists a distributed run (F, I) such that the initial configuration d_0 of (F, I) satisfies $\Gamma_0 \vdash$.*

Proof (sketch): The proof of this lemma requires a bit more effort. Let Γ_0 be fixed and let $\pi = v_1, v_2, \ldots$ be a rejected path in a tableau $\mathcal{T}$ for Γ_0. The idea is to construct a distributed run (F, I) with a correct derivation graph $DG((F, I), d_0, \Gamma_0)$. First, we have to construct a distributed run (F, I) such that its initial configuration satisfies $\Gamma_0 \vdash$.

- From π we first construct the frame F by considering the sequence of actions taken in applications of the modal rule in π as one interleaving of the distributed run. This interleaving also corresponds to a sequence of configurations $\theta = c_1, c_2, c_3, \ldots$.
- To construct the interpretation of configurations occurring in θ it is sufficient to set all propositions occurring just before the next application of a modal rule to **true**. Unfortunately, not all configurations $c \in \mathcal{C}_F$ are met along the path. But for each $i \in Loc$ there exists at least one i-equivalent configuration $c' \in \theta$ from which the interpretation of propositions $P \notin \mathcal{P}_i$ can be adopted consistently.

The main part now is the construction of the derivation graph.

- For configurations $c \in \theta$ and formulae ϕ, such that (c, ϕ) already belongs to the derivation graph, the definition of successor nodes is basically the reverse of the identification of the successor goal in the soundness proof. For other configurations $c' \notin \theta$ a similar method as for the interpretation construction is used: If the type of ϕ is A, it is possible to show that there exists an A-equivalent configuration $c \in \theta$.
- A problem not mentioned so far occurs with the definition of successors of nodes $(c, \phi_1 \vee \phi_2)$: While there can only exist one successor $(c, \phi_l), l \in \{1, 2\}$ in the derivation graph, it can happen that the or-rule 5 is applied differently on several occurrences of $\phi_1 \vee \phi_2$ on the section of the tableau path relevant to (c, ϕ_l). However, it is easy to construct from any rejected path in the tableau another rejected path in the same tableau, where disjunctions $\phi_1 \vee \phi_2$ are always resolved in the same way between two applications of the modal rule. Hence, in the above construction we can assume to have such a path.
- Finally we observe that the constructed derivation graph is correct: condition 1 in definition 5.2 is satisfied as the derivation graph and the interpretation are constructed consistently. Condition 2 is satisfied because for a finite path we get a finite derivation graph without $\langle\ \rangle_i$ at the leaves, whereas for an infinite path the condition follows from the fairness of π. Condition 3 is inherited from the absence of μ-traces in the path.

6 Automated proof search

The taming of the modal rule. Although the logic is defined on traces, the proof system works in an interleaving fashion (in particular the modal rule). Thus the question arises whether we can apply partial order reductions as known from model checking ([Pel93, GW91]) to the proof-search. Obviously each tableau path represents one interleaving of a distributed run. Thus equivalent interleavings should either both accept or both reject the path. Consider two actions $a, b \in \Sigma, a \mathcal{I} b$, and a configuration c. As a and b are independent, the order in which a and b are executed should not make any difference, since both ways lead to the same configuration c' in

a distributed run. Thus, to investigate, whether c' satisfies an arbitraty formula ϕ, it is not necessary to follow both paths leading to the same configuration. In model checking the so called sleep-set algorithm ([GW91]) is developed to keep track of this problem. There, a sleep-set belonging to a node v in the state-graph of a system specifies all directions, which do not have to be developed from that node. An action a in the sleep-set of a node v means, that the a-path outgoing from v need not be investigated. Instead of actually adding sleep-sets to tableau-nodes, we can naturally (i.e. logically) incorporate an equivalent of the sleep-set method into the modal rule. Consider the modal rule as given in rule 7. In this rule we do a case analysis between all actions in Σ: a_1 or a_2 or ... or a_n. By using the modal rule like this we produce some redundant paths in the tableau, as each interleaving of a trace is represented in the tableau. But now we can change this case analysis to an exclusive one by defining a total order on actions $a_1 < a_2 < \ldots < a_n$. The case analysis will now be: a_1 or not a_1 but a_2 or ... or not $\Sigma \setminus \{a_n\}$ but a_n. This order needs to be fixed for one application of the modal rule. Due to a particular fairness problem which we will discuss below we have to change the order in a fair way (e.g. Round Robin) to make sure, that we do not produce unfair paths.

We can logically code this in the modal rule by the following modification (where Γ_{a_i}, Σ' are defined as before): Let $A_i = \{a \in \Sigma' | (a, a_i) \in \mathcal{I}, a < a_i\}$. And let $[a]\mathbf{false}$ abbreviate $\bigvee_{i \in Loc(a)} [a]_i \mathbf{false}$ and $[A]\mathbf{false}$ abbreviate $\bigwedge_{a \in A} [a]\mathbf{false}$. Then the reduced modal rule 8 is defined as:

$$\frac{\Gamma_P, \Gamma_{\langle\ \rangle}, \Gamma_{[\ ]} \vdash}{\Gamma_{a_1} \vdash \quad [A_2]\mathbf{false}, \Gamma_{a_2} \vdash \quad \ldots \quad [A_n]\mathbf{false}, \Gamma_{a_n} \vdash} \tag{8}$$

This modification turns out to be the logical encoding of sleep-sets into the modal rule. Consider a modal step with action a_j. Any action $a_i < a_j, (a_i, a_j) \in \mathcal{I}$, which was not enabled before the execution of a_j is not enabled after the modal step either. Thus we add action a_i to the set A_j which can be viewed as the "sleep-set".

The method we use here is not only the logical encoding of the sleep-sets which are a partial order reduction method known from model-checking. The idea, to make the branches mutually exclusive, is similar to the one shown in [DM94]. There, a tableau-system (called **KE**) with an analytic *cut*-rule (**PB**) is given, where the *cut*-rule does not contradict the subformula principle. The idea of that paper is to separate the branching from the logical rules, such that the rule for disjunction is linear and the branching takes place only in the cut-rule(**PB**):

$$\frac{}{A \mid \neg A}(PB) \qquad \frac{A \vee B,\ \neg A}{B}(E \vee 1) \qquad \frac{A \vee B,\ \neg B}{A}(E \vee 2)$$

Our reduced modal rule could also be seen as the combination of a built-in analytic cut-rule **PB** (only there the case analysis is done on subformulae and not on actions, as we do it here) and a modified modal rule.
Let $\langle b \rangle \mathbf{true}$ be an abbreviation for $\bigwedge_{i \in Loc(b)} . \langle b \rangle_i \mathbf{true}$ and $[b]\mathbf{false}$ an abbreviation for $\bigvee_{i \in Loc(b)} . [b]_i \mathbf{false}$. Then in our system the analogue of the cut-rule **PB** would be for $b \in \Sigma'$: *"do b or (do) not do b"* (rule 9). We can then immediately apply the modified modal rule 10 to the left subgoal of rule 9 (modified in the sense, that we have separated the case-analysis from the modal rule such that we now only have got one subgoal):

$$\frac{\Gamma \vdash}{\langle b\rangle \mathbf{true}, \Gamma \vdash \qquad [b]\mathbf{false}, \Gamma \vdash} \quad (9) \qquad \frac{\langle b\rangle \mathbf{true}, \Gamma_P, \Gamma_{\langle\ \rangle}, \Gamma_{[\ ]} \vdash}{\Gamma_b \vdash} \quad (10)$$

Here Γ_b is constructed as defined in 7. To the right subgoal we would apply rule 9 again for the "next" action in Σ'. The following figure will illustrate how our *new* modal rule 8 can be seen as a combination of the rules 9 and 10.

$$\dfrac{\Gamma \vdash}{\dfrac{\langle a_1\rangle \mathbf{true}, \Gamma \vdash}{\Gamma_{a_1} \vdash} \qquad \dfrac{[a_1]\mathbf{false}, \Gamma \vdash}{\dfrac{[a_1]\mathbf{false}, \langle a_2\rangle \mathbf{true}, \Gamma \vdash}{[A_2]\mathbf{false}, \Gamma_{a_2} \vdash} \qquad \dfrac{[a_1]\mathbf{false}, [a_2]\mathbf{false}, \Gamma \vdash}{\vdots \qquad \dfrac{\vdots}{[A_n]\mathbf{false}, \Gamma_{a_n} \vdash}}}}$$

In rule 9 we only create a subgoal for an action $a \in \Sigma'$.

A particular fairness problem occurs on infinite paths: we must find a fair method for finding a representative for an infinite trace. Consider e.g. an infinite trace with infinitely many a's but only one b. Let a and b be independent. With $a < b$ the above rule would put a asleep forever on every b-path such that there is no representation of the trace $[a^\omega b]$ We can take care of this problem by redefining the total ordering of actions in each modal step with a fair method (e.g. Round Robin).

It seems that this reduction will often lead to significantly smaller proofs (with the usual worst case exceptions). However, we have not practically checked this.

Handling infinite tableaux. Our tableau system often requires infinite proofs, so one might ask for practical use of the system. With the help of tree automata (see [Tho90] for an introduction) we can give a decision method for the existence of accepting tableaux. More precisely we can construct tree automata running on tableaux by regarding them as trees with sequents as node labels[1] in the obvious way. The construction is sketched in the appendix, for further details see [Spr96].

Theorem 6. *For each sequent $\Gamma \vdash$ we can canonically construct a Rabin-tree-automaton $\mathcal{A}_T$ recognising exactly the accepting tableaux for $\Gamma \vdash$.*

These automata give us a finitary representation of proofs: instead of constructing tableaux with infinite paths we now represent the proof as a tree with backloops.

Definition 6.1 (regular tableaux) *An infinite tree $\mathcal{T}$ is said to be regular if there are only finitely distinct non-isomorphic subtrees $\mathcal{T}'$ in $\mathcal{T}$. An infinite tableau is called regular, iff it is a regular tree.*

Obviously regular trees and tableaux can be finitely represented. Since the emptiness problem for tree automata is decidable (see [Tho90]) and moreover in the nonempty case the decision procedure returns a regular tree, we can deduce the following:

Corollary 7. *The set of sequents $\Gamma \vdash$ with an accepting tableau is decidable. If there exists any accepting tableau for $\Gamma \vdash$, then there also exists a regular one.*

[1] Note that for an initial sequent $\Gamma_0 \vdash$ only a finite and easy to determine set of sequents $\Gamma \vdash$ can occur in any tableau, the Fischer-Ladner closure.

Acknowledgemets

We thank Michaela Huhn, R. Ramanujam and P.S. Thiagarajan for fruitful discussions. Rajeev Goré made very valuable comments to the presentation of the paper and has pointed us to the connection with tamed cut. Part of this work was financially supported by the Human Capital and Mobility Cooperation Network "EXPRESS" (Expressivity of Languages for Concurrency).

References

[Dam92] Mads Dam. Fixpoints of Büchi automata. In *International Conference on the Foundations of Software Technology and Theoretical Computer Science (FST&TCS)*, Lecture Notes in Computer Science, pages 39–50, 1992.

[DM94] Marcello D'Agostino and Marco Mondadori. The taming of the cut. Classical refutations with analytic cut. *Logic and Computation*, 4(3):285–319, 1994.

[ES89] E. Allen Emerson and Robert S. Streett. An automata theoretic decision procedure for the propositional mu-calculus. *Information and Computation*, 81:249–264, 1989.

[GW91] Patrice Godefroid and Pierre Wolper. A partial approach to model checking. In *IEEE Symposium on Logic in Computer Science*, volume 6, pages 406–415, 1991.

[Huh96] Michaela Huhn. Action refinement and property inheritance in systems of sequential agents. In U. Montanari and V. Sassone, editors, *International Conference on Concurrency Theory (CONCUR)*, volume 1119 of *Lecture Notes in Computer Science*, pages 639–654. Springer-Verlag, 1996.

[Koz83] Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[Maz95] Antoni Mazurkiewicz. Introduction to trace theory. In Volker Diekert and Grzegorz Rozenberg, editors, *The Book of Traces*, chapter 1, pages 1–42. World Scientific, 1995.

[Mil89] Robin Milner. *Communications and Concurrency*. Prentice-Hall, 1989.

[MP92] Manna and Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.

[Nie95] Peter Niebert. A ν-calculus with local views for systems of sequential agents. In *MFCS*, volume 969 of *Lecture Notes in Computer Science*, 1995.

[Pel93] Doron Peled. All from one, one for all: on model checking using representatives. In *International Conference on Computer Aided Verification (CAV)*, Lecture Notes in Computer Science, 1993.

[Spr96] B. Sprick. Ein Beweissystem für die modale Tracelogik ν-**TrTl** und seine Optimierung durch Halbordnungsreduktionen. Master's thesis, Universität Hildesheim, 1996.

[Thi94] P.S. Thiagarajan. A trace based extension of Linear Time Temporal Logic. In *IEEE Symposium on Logic in Computer Science (LICS)*, volume 9, 1994.

[Tho90] Wolfgang Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 4, pages 133–191. Elsevier Science Publishers B.V., 1990.

[Var88] Moshe Y. Vardi. A temporal fixpoint calculus. In *ACM Symposium on Principles of Programming Languages*, pages 250–259, 1988.

[Wal95] Igor Walukiewicz. A complete deductive system for the μ-calculus. BRICS Report Series RS-95-6, Danish Center for Basic Research in Computer Science (BRICS), 1995.

A Construction of automata recognising tableaux

Here we give a brief description of a Rabin tree automaton which for input $\Gamma \vdash$ recognises exactly the accepting tableaux for $\Gamma \vdash$. The required automaton is a product of a nondeterministic tree automaton $\mathcal{A}_{tree}$ recognising tableaux and a deterministic Rabin-word-automaton $\mathcal{A}_L$ recognising accepted tableau-paths.

Tree-automaton recognising tableaux: Its states are sequents $\Gamma \vdash \in Seq(\Gamma_0)$ and its transition relation is defined by the tableau-rules. $\mathcal{A}_{tree}$ nondeterministically "guesses" a tableau and checks that the input tableau matches the guess. Any tree meeting the transition relation is accepted by the automaton.

Word-automaton recognising accepting tableau paths: This automaton is constructed as the union of three sub-automata which all run on tableau-paths: $\mathcal{A}_{\mathit{fin}}$ accepts finite paths ending with an axiom, $\mathcal{A}_{\mathit{unfair}}$ recognises unfair paths, and $\mathcal{A}_\mu$ accepts exactly all paths with a $\mu - \mathit{trace}$. As the construction of $\mathcal{A}_{\mathit{fin}}$ is trivial, we only describe the construction of $\mathcal{A}_{\mathit{unfair}}$ and $\mathcal{A}_\mu$ and show how to construct their union and the product with the tree automaton.

Automaton recognising unfair paths: For $\mathcal{A}_{\mathit{unfair}}$ we need two states q_{in_i} and q_i for each location $i \in \mathit{Loc}$. In q_{in_i} we check, if there exists a formula $\langle\ \rangle_i$ in the next input sequent Γ. If it does the automaton goes over to q_i where it stays until the next action at location i is performed and then it goes over to $q_{in_{i+1}}$. Otherwise it directly goes over to $q_{in_{i+1}}$. This automaton accepts, if it meets q_i infinitely often while it meets q_{in_i} only finitely many times for any i as there is no more action at location i though $\langle\ \rangle_i$ is still in the input sequent.

Automaton recognising μ-traces: $\mathcal{A}_\mu$ checks for the existence of a μ-trace. Given here as a nondeterministic Büchi-automaton $\mathcal{A}'_\mu$ it can clearly be determinised with Safra's determinization construction. $\mathcal{A}'_\mu$ "guesses" one formula from each sequent along the input sequence so that the sequence of chosen formulae forms a trace and ensures the trace being a μ-trace.

Safras's construction can be used for determization or, as shown in [Wal95], a deterministic (Rabin) automaton can be constructed directly. Both ways lead to a deterministic Rabin-automaton for the acceptance of μ-traces.

Union of $\mathcal{A}_{\mathit{fin}}$, $\mathcal{A}_{\mathit{unfair}}$,and $\mathcal{A}_\mu$: The states of this union-automaton ($\mathcal{A}_L$) are the products of the sub-automata with the tuple of all three initial states as new initial state. A state (q_1, q_2, q_3) goes over to (q'_1, q'_2, q'_3) iff each state q_1 in the sub automaton goes over to q'_i with the same input. The Rabin-acceptance condition of $\mathcal{A}_L$ can be defined as a product of all three sub-acceptance conditions.

Product-automaton recognising accepting tableaux: The idea is to let $\mathcal{A}_L$ run simultaneously on all paths of the tree. Thus we first convert the deterministic word-automaton $\mathcal{A}_L$ into a tree-automaton $\mathcal{A}_{tree(L)}$. $\mathcal{A}_{tree(L)}$ is then running on trees with each path in the tree being a sequence of sequents and accepts, iff all paths of the tree are accepted by $\mathcal{A}_L$. Finally we compose the product of the tree automaton $\mathcal{A}_{tree(L)}$ and the tree automaton $\mathcal{A}_{tree}$, to obtain the intersection of both tree languages. The states of this (final) product automaton are again sequents $\Gamma \vdash \in Seq(\Gamma_0)$. The acceptance condition is the same as for $\mathcal{A}_{tree(L)}$ and the intersection of the tranistion relations of both automata gives the new transition relation.

This tree automaton finally gives us a finitary representation of a proof and thus a basis for the construction of a finitary Hilbert-style proof system as in [Wal95].