# Local logics for traces

Igor Walukiewicz

**BRICS***

**Abstract**

A $\mu$-calculus over dependence graph representation of traces is considered. It is shown that the $\mu$-calculus cannot express all monadic second order (MSO) properties of dependence graphs. Several extensions of the $\mu$-calculus are presented and it is proved that these extensions are equivalent in expressive power to MSO logic. The satisfiability problem for these extensions is PSPACE complete.

## 1 Introduction

Infinite words, which are linear orders on *events*, are often used to model executions of systems. Infinite *traces*, which are partial orders on events, are often used to model concurrent systems when we do not want to put some arbitrary ordering on actions occurring concurrently. A *state* of a system in the linear model is just a prefix of an infinite word; it represents the actions that have already happened. A state of a system in the trace model is a *configuration*, i.e., a finite downwards closed set of events that already happened.

Temporal logics over traces come in two sorts: a *local* and a *global* one. The truth of a formula in a *local logic* is evaluated in an event, the truth of a formula in a *global logic* is evaluated in a configuration. Global logics (as for example the one in [11, 2]) have the advantage of talking directly about configurations hence potentially it is easier to write specifications in them. The disadvantage of global logics is the high complexity of

*Basic Research in Computer Science,
Centre of the Danish National Research Foundation.

the satisfiability problem [12]. Here we are interested in local temporal logics.

In this paper we present several local logics for traces and show that they have two desirable properties. First, the satisfiability problem for them is PSPACE complete. Next, these logics are able to express all the trace properties expressible in monadic second order logic (MSOL).

We start from the observation that branching time program logics, like the $\mu$-calculus, can be used to describe properties of traces. This is because these logics talk about properties of labelled graphs and a trace (represented by a dependence graph) is a labelled graph with some additional properties. It is well known that the $\mu$-calculus is equivalent to MSOL over binary trees but it is weaker than MSOL over all labelled graphs. It turns out that the $\mu$-calculus is weaker than MSOL also over dependence graphs.

To obtain a temporal logic equivalent to MSOL over traces we consider some extensions of the $\mu$-calculus. The one which is easiest to describe here is obtained by adding $co_a$ propositions. Such a proposition holds in a event $e$ of a trace if there is in the trace an event incomparable with $e$ which is labelled by $a$.

The first local temporal logic for traces was proposed by Ebinger [3]. This was an extension of LTL. He showed that over finite traces his logic is equivalent in expressive power to first order logic. The logic that is most closely related with the present work $\nu$TrPTL proposed by Niebert [8]. This is an extension of the $\mu$-calculus which captures the power of MSOL. Unfortunately the syntax of the logic is rather heavy. The proof that the logics captures the power of MSOL uses some kind of decomposition of traces and coding of asynchronous automata. The present work may be seen as an attempt to find another trace decomposition that makes the work easier, partly by allowing the use of standard facts about MSOL on trees. We do not use here any kind of automata characterisation of MSOL over traces or any other "difficult" result about traces.

### Outline of the paper

In the next section we define traces as labelled graphs representing partial orders on events. Such a representation is called *dependence graph* representation of traces. Next, we define MSO logic and the $\mu$-calculus over labelled graphs. We also recall results linking MSOL with the $\mu$-calculus and an automata characterisation of the later logic.

In Section 5 we describe a new representation of traces by trees

that we call lex-trees. These trees have the property that every trace is uniquely represented by such a tree. The other important property of lex-trees is that a lex-tree is MSOL definable in dependence graph representation of a trace and dependence graph is MSOL definable in lex-tree representation of a trace. Hence MSOL over dependence graphs is equivalent to MSOL over lex-trees. This allows us to use an equivalence of the $\mu$-calculus and MSOL over trees to obtain an extension of the $\mu$-calculus equivalent to MSOL over dependence graphs. This extension may not seem that natural as it is very much connected with particular representation.

In Section 6 we consider some other extensions of the $\mu$-calculus. One is $\mu(co)$, an extension with $co_a$ propositions. Such a proposition holds in a event $e$ if in the trace there is an event incomparable with $e$ which is labelled by $a$. The other is $\mu(Before)$ which is an extension with $Before_{ab}$ propositions. Such a proposition holds in a event, roughly, when among the events after it an $a$ event occurs before the first $b$ event.

In the same section we present the main result of the paper (Corollary 24) which says that the two logics can express all MSOL definable properties. The proof of this fact relies on existence of some automaton that can reconstruct a lex-tree inside a dependence graph. This construction is given in the next two sections.

In Section 7 we give a characterisation of lex-trees in terms of some local properties. Initially, we define lex-trees using some formulas with quantification over paths in dependence graph. Here, we show that lex-trees can be defined by existence of some marking of nodes satisfying some local consistency conditions.

In Section 8 we describe the construction of an automaton reconstructing lex-trees in dependence graphs. This construction uses the local definition of lex-trees from the preceding section.

In Section 9 we give translations of our logics to automata over infinite words. For a given formula we construct an exponential size automaton accepting linearizations of traces satisfying the formula. From this we deduce PSPACE-completeness of the satisfiability problem for our logics.

Section 10 contains a comparison with $\nu$TrPTL.

# 2 Traces and their representations

A *trace alphabet* is a pair $(\Sigma, D)$ where $\Sigma$ is a finite set of actions and $D \subseteq \Sigma \times \Sigma$ is a reflexive and symmetric *dependence relation.*

A $\Sigma$-labelled graph is $\langle V, R, \lambda\rangle$ where $V$ is the set of vertices, $R$ defines the edges and $\lambda : V \to \Sigma$ is a labelling function. A $\Sigma$-labelled partial order is a $\Sigma$-labelled graph where $R$ is a partial order relation.

**Definition 1** A *trace* or a *dependence graph* over a trace alphabet $(\Sigma, D)$ is a $\Sigma$-labelled partial order $\langle E, R, \lambda\rangle$ satisfying the following conditions:

(T1) $\forall e \in E. \quad \{e' : R(e', e)\}$ is a finite set.
(T2) $\forall e, e' \in E. \quad (\lambda(e), \lambda(e')) \in D \Rightarrow R(e, e') \vee R(e', e).$
(T2) $\forall e, e' \in E. \quad R(e, e') \Rightarrow (\lambda(e), \lambda(e')) \in D \vee$
$\exists e''.\ R(e, e'') \wedge R(e'', e') \wedge e \neq e'' \neq e'.$

The nodes of a dependence graph are called *events*. An *a-event* is an event $e \in E$ which is labelled by $a$, i.e., $\lambda(e) = a$. We say that $e$ is *before* $e'$ iff $R(e, e')$ holds. In this case we also say that $e'$ is *after* $e$.

The first condition of the definition of dependence graphs says that the past of each event (the set of the events before the event) is finite. The second one postulates that events labelled by dependent letters are ordered. The third, says that the order is induced by the order between dependent letters.

Below we describe two variations on the representation of dependence graphs. These variations will be important when defining logics for traces.

**Definition 2** A *Hasse diagram* of a trace $G = \langle E, R, \lambda\rangle$ is a labelled graph $\langle E, R_H, \lambda\rangle$ where $R_H$ is the smallest relation needed to determine $R$, i.e., $R_H^* = R$ and if $R_H(e, e')$ then there is no $e''$ different from $e$ and $e'$ such that $R_H(e, e'')$ and $R_H(e'', e')$ hold.

**Definition 3** A *process diagram* of a trace $G = \langle E, R, \lambda\rangle$ is a labelled graph $\langle E, R_P, \lambda\rangle$ where $R_P(e, e')$ holds if $e$ and $e'$ are labelled by dependent letters and $e'$ is the first $\lambda(e')$-event after $e$. More formally the condition is: $(\lambda(e), \lambda(e')) \in D$, $R(e, e')$ and $\forall e'' \neq e.\ R(e, e'') \wedge \lambda(e'') = \lambda(e') \Rightarrow R(e', e'')$.

**Proviso:** In the whole paper we fix a trace alphabet $(\Sigma, D)$ and a linear order $<_\Sigma$ on $\Sigma$. We also assume that we have a special letter $\perp \in \Sigma$ which is dependent on every other letter of the alphabet, i.e., $\{\perp\} \times \Sigma \subseteq D$. Finally, we assume that in every trace there is the least event (with respect to the partial order $R^*$) and it is labelled by $\perp$. We denote this least event also by $\perp$.

The assumption that every trace has the least event will turn out to be very useful for local temporal logics we consider in this paper. In particular the definition of a set of traces definable by a formula becomes unproblematic in this case.

# 3 MSOL over graphs and traces

In this section we give a definition of Monadic Second Order Logic (MSOL) over labelled graphs. Then we recall the known properties of this logic over the class of dependence graphs.

Let $\Gamma$ be a finite alphabet. We define MSOL suitable to talk about $\Gamma$-labelled graphs. The signature of the logic consists of one binary relation $R$ and a monadic relation $P_a$ for each $a \in \Gamma$. Let $Var = \{X, Y, \dots\}$ be the set of (second order) variables. The syntax of MSOL is given by the grammar:

$$X \subseteq Y \mid P_a(X) \mid R(X,Y) \mid \neg\alpha \mid \alpha \vee \beta \mid \exists X.\ \alpha$$

where $X$, $Y$ range over variables in $Var$, $a$ over letters in $\Gamma$, and $\alpha$, $\beta$ over formulas.

Given a $\Gamma$-labelled graph $M = \langle S, R \subseteq S \times S, \rho : S \to \Gamma\rangle$ and a valuation $V : Var \to \mathcal{P}(S)$ the semantics is defined inductively as follows:

- $M, V \vDash X \subseteq Y$ iff $V(X) \subseteq V(Y)$,
- $M, V \vDash P_a(X)$ iff there is $s \in S$ with $V(X) = \{s\}$ and $\rho(s) = a$,
- $M, V \vDash R(X,Y)$ iff there are $s, s' \in S$ with $V(X) = \{s\}$, $V(Y) = \{s'\}$ and $R(s, s')$,
- $M, V \vDash \exists X.\alpha$ iff there is $S' \subseteq S$ such that $M, V[S'/X] \vDash \alpha$,
- the meaning of boolean connectives is standard.

As usual, we write $M \vDash \varphi$ to mean that for every valuation $V$ we have $M, V \vDash \varphi$. A MSOL formula $\varphi$ defines a set of traces $\{G : G \vDash \varphi\}$. In the sequel we will sometimes use first order variables in MSOL formulas. To denote them we will use small letters $x, y, \dots$ The intention is that these variables range over nodes of a graph and not over sets of nodes as second order variables do. First order variables can be "simulated" with

second order variables because being a singleton set is expressible in our variant of MSOL.

Büchi theorem tells us that the class of finite or infinite words (dependence graphs for alphabets where all the letters are mutually dependent) the properties definable by MSOL are exactly the recognizable languages. This characterisation carries through to traces.

In case of traces there seem to be many possibilities to say what it means to be MSOL definable. That is, we can take MSOL over dependence graphs, or over Hasse diagrams of dependence graphs, or over process diagrams. Fortunately, MSOL has the same expressive power over each of these representations. Hence it makes sense to just say MSOL definable set of traces without mentioning a representation. The following theorem summarizing many results on traces can be found in [4].

**Theorem 4**
*Fix a given trace alphabet. For a set $L$ of traces the following are equivalent:*

- *$L$ is definable by a MSOL formula.*
- *$L$ is definable by a c-regular expression.*
- *$L$ is recognizable by an asynchronous automaton.*
- *The set of linearizations of traces in $L$ is a recognizable language of infinite words.*

We will not need asynchronous automata or c-regular expressions in this paper so we will not define them here. If linearizations are concerned, let us just say that a linearization of a trace is an infinite word which corresponds to some linear order of type $\omega$ extending the partial order of the trace.

The above theorem shows that the class of MSOL definable properties of traces is interesting because it admits many different characterisations. In case of infinite words MSOL has a role of some kind of expressibility yardstick. The theorem tells us that MSOL can play the same role also for traces.

To finish this section let us recall a tool for defining one labelled graph inside another one by means of MSOL formulas. This is a very simple instance of the method of interpretations of one structure inside the other.

Let $\xi(x,y)$ be a MSOL formula with two free first order variables $x, y$. In a given labelled graph $M = \langle S, R, \rho\rangle$ this formula defines a relation $R_M^\xi = \{(v,v') : M \vDash \xi(v,v')\}$. Let $M^\xi = \langle S, R_M^\xi, \rho\rangle$ be a labelled graph obtained from $M$ by using $R_M^\xi$ as an edge relation. We will use the following straightforward observation.

**Proposition 5** For every MSOL formula $\varphi$ there is an MSOL formula $\varphi^\xi$ such that $M^\xi \vDash \varphi$ iff $M \vDash \varphi^\xi$.

**Proof**
Formula $\varphi^\xi$ can be obtained by replacing every occurrence of $R$ in $\varphi$ by $\xi(x,y)$. □

# 4 The $\mu$-calculus over graphs and traces

Next we define the *$\mu$-calculus* over an alphabet $\Gamma$. For some fixed set *Var* of variables the syntax is defined by the grammar:

$$X \mid P_a \mid \neg\alpha \mid \alpha \vee \beta \mid \langle\cdot\rangle\alpha \mid \mu X.\alpha$$

where $X$ ranges over variables in *Var*, $a$ over letters in $\Gamma$, and $\alpha$, $\beta$ over formulas. In the construction $\mu X.\alpha$ we require that $X$ appears only positively in $\alpha$ (i.e., under even number of negations).

The meaning of a formula $\alpha$ in a $\Gamma$-labelled graph $M = \langle S, R, \rho\rangle$ with a valuation $V : Var \to \mathcal{P}(S)$ is a set of nodes $[\![\alpha]\!]_V^M \subseteq S$ defined by:

$$\begin{aligned}
[\![P_a]\!]_V^M &= \{s \in V : \rho(s) = a\}\\
[\![X]\!]_V^M &= V(X)\\
[\![\langle\cdot\rangle\alpha]\!]_V^M &= \{s \in S : \exists s'.\ R(s,s') \wedge s' \in [\![\alpha]\!]_V^M\}\\
[\![\mu X.\alpha(X)]\!]_V^M &= \bigcap\{A \subseteq S : [\![\alpha(A)]\!]_{V[A/S]}^M \subseteq A\}
\end{aligned}$$

The omitted clauses for boolean constructors are standard. We write $M, V, s \vDash \alpha$ if $s \in [\![\alpha]\!]_V^M$.

If $G$ is a trace that has the least event $\perp$ then we write $G \vDash \alpha$ to mean that $G, V, \perp \vDash \alpha$ for all $V$. A $\mu$-calculus formula $\alpha$ defines the set of traces $\{G : G \vDash \alpha\}$.

A $\Gamma$-labelled graph $\langle S, R, \rho\rangle$ is called *deterministic tree* if $\langle S, R\rangle$ is a tree and for every $v \in S$ and every $a \in \Gamma$ there is at most one $v' \in S$ with

$R(v, v')$ and $\rho(v') = a$. The following equivalence was shown by Niwinski (cf. [9]).

**Theorem 6**
*Over deterministic trees $\mu$-calculus is equivalent to MSOL. In other words, for every MSOL sentence $\varphi$ there is a $\mu$-calculus sentence $\alpha_\varphi$ such that for every deterministic tree $M$: $M \models \varphi$ iff $M \models \alpha_\varphi$. Also conversely, for every $\mu$-calculus sentence $\alpha$ there is an MSOL sentence $\varphi_\alpha$ such that: $M \models \alpha$ iff $M \models \varphi_\alpha$ for every deterministic tree $M$.*

The $\mu$-calculus cannot be equivalent to MSOL over all labelled graphs or even trees because in MSOL we can say that there is some fixed number of successors of the node and this is impossible in the $\mu$-calculus. Dependence graphs are deterministic acyclic graphs but usually they are not trees. Later we will see that the $\mu$-calculus is weaker than MSOL over dependence graphs.

We recall (from [6]) a characterisation of the $\mu$-calculus in terms of (alternating) automata. This will allow us to use automata instead of the $\mu$-calculus which is easier for some constructions. The only difference between these automata and alternating automata on binary trees is in the transition function which now needs to cope with nodes of arbitrary degree.

**Definition 7** A *$\mu$-automaton* over an alphabet $\Gamma$ is a tuple

$$\mathcal{A} = \langle Q, \Gamma, q_0, \delta, F, \Omega \rangle$$

where: $Q$ is a finite set of states, $\Gamma$ is a finite alphabet, $q_0 \in Q$ is an initial state, $\delta : Q \times \Gamma \to \mathcal{P}(\mathcal{P}(Q))$ is a transition function, $F \subseteq Q$ is a set of final states and $\Omega : Q \to \mathbb{N}$ defines a winning condition.

Let $M = \langle E, R, \rho \rangle$ be a $\Gamma$-labelled graph and $v_0$ a vertex of $M$. A *run* of $\mathcal{A}$ on $M$ starting from a vertex $v_0$ is a labelled tree $r : S \to E \times Q$; where $S$ is a tree and $r$ is a labelling function. We require that the root of $S$ is labeled with $(v_0, q_0)$ and for every node $v$ with $r(v) = (e, q)$ we have that either $q \in F$ or there is $W \in \delta(q, \rho(e))$ satisfying:

- for every successor $e'$ of $e$ there is a son of $v$ labelled by $(e', q')$ for some $q' \in W$;
- for every $q' \in W$ there is a son of $v$ labelled by $(e', q')$ for some successor $e'$ of $e$.

An infinite sequence $(e_0, q_0), (e_1, q_1), \ldots$ satisfies the *parity condition given by* $\Omega$ if the smallest number among those appearing infinitely often in the sequence $\Omega(q_0), \Omega(q_1), \ldots$ is even. We call a run *accepting* if every leaf of the run is labelled by a state from $F$ and every infinite path satisfies the parity condition. We say that $\mathcal{A}$ *accepts* $M$ from $v_0$ iff $\mathcal{A}$ has an accepting run on $M$ from $v_0$. Automaton $\mathcal{A}$ defines a set of traces $L(\mathcal{A}) = \{G : \mathcal{A} \text{ accepts } G \text{ from } \bot\}$.

The following theorem from [6] shows equivalence of $\mu$-automata and the $\mu$-calculus over $\Gamma$-labelled transition systems.

**Theorem 8**
*For every automaton $\mathcal{A}$ there is a $\mu$-calculus sentence $\alpha_{\mathcal{A}}$ such that for every $\Gamma$-labelled transition system $G$ we have: $G \vDash \alpha_{\mathcal{A}}$ iff $G \in L(\mathcal{A})$. Conversely; for every $\mu$-calculus sentence $\alpha$ there is an automaton $\mathcal{A}_\alpha$ such that $\{G : G \vDash \alpha\} = L(\mathcal{A}_\alpha)$.*

We finish this section with the proposition showing that over traces the $\mu$-calculus does not have sufficient expressive power. This example motivates the search for extensions of the $\mu$-calculus that can capture the power of MSOL over traces.

**Proposition 9** No $\mu$-calculus sentence can distinguish between the following two Hasse diagrams of traces presented in Figure 1. In the left

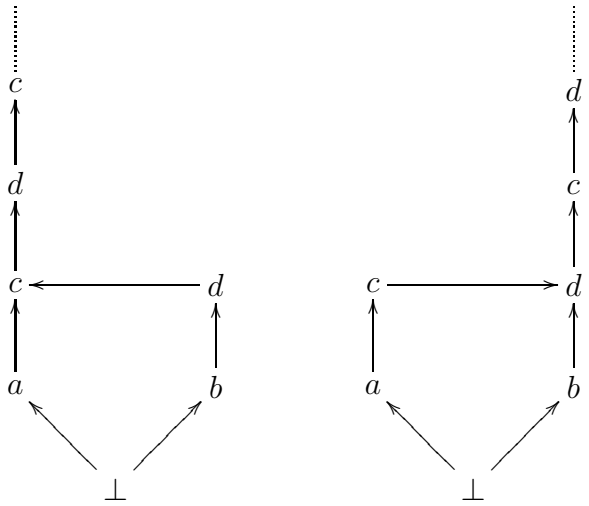


Figure 1: Indistinguishable traces

graph the dots stand for the sequence $(dc)^\omega$ and in the right graph for $(cd)^\omega$. In this example the trace alphabet $(\{\bot, a, b, c, d\}, D)$ where $D$ is the smallest symmetric and reflexive relation containing the pairs $\{(a, c), (b, d), (c, d)\} \cup \{\bot\} \times \{a, b, c, d\}$

**Proof**
The two dependence graphs are bisimilar. Proposition follows from the fact that no $\mu$-calculus formula can distinguish between two bisimilar graphs. □

Observe that this proposition holds independently of the representation of the traces. The figure above shows Hasse diagrams of traces. But also process diagrams of the two traces are bisimilar. This is also true for dependence graphs of the traces.

# 5 Lex-trees

In this section we describe a representation of traces by some kind of trees which we call lex-trees. This will allow us to use the equivalence of MSOL and the $\mu$-calculus over trees.

**Definition 10 (Lex-Tree)** Let $G = \langle E, R, \lambda \rangle$ be a trace. A path of events $e_1 e_2 \dots e_n$ in $G$ determines a sequence of labels $\lambda(e_1)\lambda(e_2)\dots\lambda(e_n)$. So, we can compare two such paths using the lexicographic ordering on $\Sigma^*$ obtained from our fixed ordering $<_\Sigma$ on $\Sigma$. We will denote this ordering also by $<_\Sigma$. For $e \in E$ let $lexp(e)$ be the smallest in lexicographical ordering path from the least element of $G$ to $e$.

*Lex-tree* of $G$, denoted $Lex(G)$, is a $\Sigma$-labelled graph $T = \langle E, Son, \lambda \rangle$ where $Son(e, e')$ holds iff $lexp(e') = lexp(e)e'$ (in words: if the lexicographic path to $e'$ goes through $e$ and $e'$ is a successor of $e$). In this case we will call $e'$ a *lex-son* of $e$ in $G$.

Definition of lex-trees gives a natural ordering on sons of a node that then can be extended to an ordering between any two nodes of lex-tree which are not on the same path.

**Definition 11 ("To the left" ordering)** We define "to the lex-left" ordering on events of $G$: $e \preccurlyeq e'$ iff $lexp(e) <_\Sigma lexp(e')$ but $lexp(e)$ is not a prefix of $lexp(e')$. We say that $e'$ is *to the right* of $e$ if $e$ is to the left of $e'$.

**Lemma 12** For every dependence graph $G$, $Lex(G)$, is a tree.

**Proof**
Every path in the lex-tree is a lex-path. There cannot be two lex-paths to the same event. □

**Lemma 13** There is a MSOL formula $\xi$ defining $Lex(G)$ in $G$ (i.e., $G^{\xi}$ is isomorphic to $Lex(G)$).

**Proof**
It is not difficult to write a formula $lpath(X, e)$ which says that $X$ is the lexicographic path to $e$. Using this we write a formula $\xi(x, y)$ such that $G \vDash \xi(e, e')$ iff $lexp(e') = lexp(e)e'$. □

**Lemma 14** There is a MSOL formula $\xi^{-1}$ defining $G$ from $Lex(G)$.

**Proof**
Let $G = \langle E, R, \lambda \rangle$ be a dependence graph. First observation is that for a pair of dependent letters $(a, b) \in D$ an $a$-event $e_a$ is before a $b$-event $e_b$ in $G$ iff $e_a$ is an ancestor or to the right of $e_b$ in the tree $Lex(G)$. Indeed if $e_a$ is before $e_b$ in $G$ then either $lexp(e_a)$ is a prefix of $lexp(e_b)$ or $lexp(e_b)$ is lexicographically smaller than $lexp(e_a)$. For the other direction, if $e_a$ is to the right of $e_b$ in $Lex(G)$ then $e_a$ is before $e_b$ in the trace ordering because otherwise we could have a path to $e_a$ going through $e_b$ (as $a$ and $b$ are dependent).

Let us define the relation $H(e, e')$ which holds if the two events are labelled by dependent letters and $e$ is an ancestor or to the right of $e'$. Clearly $H$ is definable in MSOL. The observation from the above paragraph can be reformulated as: $H(e, e')$ iff $R^{(}e, e')$ and $(\lambda(e), \lambda(e')) \in D$. In particular $H^* \subseteq R$ as $R$ is transitive. An easy induction using conditions (T1) and (T3) of the the definition of the trace shows $R \subseteq H^*$. We are done as the transitive closure of a relation is definable in MSOL. □

Using Proposition 5 we immediately obtain.

**Corollary 15** MSOL over dependence graphs and over lex-trees has the same expressive power. More precisely, for every MSOL formula $\varphi$ there is a MSOL formula $\varphi^T$such that for every dependence graph $G$ we have: $G \vDash \varphi$ iff $Lex(G) \vDash \varphi^T$. Vice versa, for every MSOL formula $\psi$ there is a MSOL formula $\psi^G$ such that for every graph $G$ we have: $G \vDash \psi^G$ iff $Lex(G) \vDash \psi$.

**Corollary 16** MSOL over dependence graphs is equivalent to the $\mu$-calculus over lex-trees. More precisely, for every MSOL formula $\varphi$ there is a $\mu$-calculus formula $\alpha$ such that for every dependence graph $G$ we have: $G \vDash \varphi$ iff $Lex(G) \vDash \alpha$. Moreover for every $\mu$-calculus formula $\alpha$ there is a MSOL formula $\varphi_\alpha$ such that for every dependence graph $G$ we have: $G \vDash \varphi_\alpha$ iff $Lex(G) \vDash \alpha$.

# 6 Extended $\mu$-calculi for traces

Proposition 9 shows that it is not possible to reconstruct the shape of a trace in the $\mu$-calculus over dependence graph representation of a trace. The same is true for Hasse diagram or process diagram representations. This means that we need to consider richer representations of traces. Here we propose to extend the labelling with events. Before this we define some auxiliary relations.

**Definition 17** Let $G = \langle E, R, \lambda \rangle$ be a dependence graph. Relation *co* is the concurrency relation between events in the trace defined by $co(e, e')$ iff neither $R(e, e')$ nor $R(e', e)$ hold. We define relation $Before_{a,b}(e)$ for every event $e$ and every pair of dependent letters $(a, b) \in D$. The relation holds if $\lambda(e)$ depends on both $a$ and $b$ and moreover among events after $e$ there are $a$ and $b$-events and some $a$-event appears before all $b$-events. More formally $Before_{a,b}(e)$ holds if $\lambda(e)$ depends on $a$ and $b$ and there are events $e_a, e_b \neq e$ such that $R(e, e_a)$, $R(e, e_b)$ and $\lambda(e_a) = a$, $\lambda(e_b) = b$. Moreover for every $e'_b \neq e$ with $R(e, e'_b)$ and $\lambda(e'_b) = b$ we must have $R(e_a, e'_b)$.

**Definition 18** Let $G = \langle E, R, \lambda \rangle$ be a dependence graph or Hasse diagram of a dependence graph or a process diagram of it. We define its two annotations $M_{co}(G)$ and $M_B(G)$. Let $M_{co}(G) = \langle E, R, \lambda_{co} \rangle$ be labelled graph over an alphabet $\Gamma_{co} = \Sigma \times \mathcal{P}(\Sigma)$ where $\lambda_{co}(e) = (\lambda(e), \{\lambda(e') : co(e, e')\})$. Let $M_B(G) = \langle E, R, \lambda_{co} \rangle$ be a labelled graph over an alphabet $\Gamma_B = \Sigma \times \mathcal{P}(\Sigma \times \Sigma)$ where $\lambda_B(e) = (\lambda(e), \{(a, b) : Before_{ab}(e)\})$.

**Definition 19** For a trace $G$ we define six graph representations: $M_{co}(G)$, $M_B(G)$, $M^H_{co}(G)$, $M^H_B(G)$, $M^P_{co}(G)$, $M^P_B(G)$. Superscripts $H$ and $P$ stand for Hasse and process diagram representations respectively. The subscripts determine the kind of annotation.

Our goal is to show that the $\mu$-calculus over three out of these six representations is expressively complete. The next proposition says that the remaining three representations are to weak.

**Proposition 20** The $\mu$-calculus over $M_{co}(G)$, $M_B(G)$ and $M^P_{co}(G)$ representations cannot express all MSOL properties.

**Proof**
Let us first consider two representations with *co* annotation. Take an alphabet with three letters $a$, $b$, $c$ that are mutually dependent on each other, i.e., $\Sigma = (\{\bot, a, b\}, \{\bot, a, b\}^2)$. Recall that we assume that there is always special $\bot$ letter in the alphabet. Consider the two process diagrams of traces presented in Figure 6. The two diagrams are bisimilar so

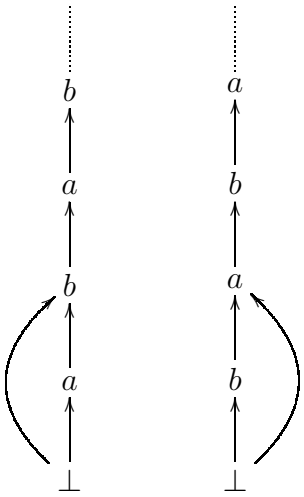


Figure 2: Indistinguishable process diagrams

no $\mu$-calculus formula can distinguish between them. The *co* annotation is void in this case as all the actions are mutually dependent. This settles the case of $M^P_{co}(G)$ representation as there is a FOL formula distinguishing between these two graphs. The same example also works for $M_{co}(G)$ representation. The difference is that in that representation we also have an arrow between every two places connected by a path. Still the two diagrams remain bisimilar in this case.

The case of $M_B(G)$ representation is more difficult. Consider the same trace alphabet as before. We claim that the property:

($*$) $a$ appears even number of times in $G$

is not expressible by a $\mu$-calculus formula over $M_B(G)$ representations. Even though all the letters in our alphabet are mutually dependent $M_B(G)$ representation of a trace is not an infinite word. For example in the graph $M_B(\bot ababb^\omega)$ there is an edge from each position to each later position. Predicate $Before_{ab}$ holds in a position if the next position is labelled by $a$; otherwise $Before_{ba}$ holds.

Suppose that the property $(*)$ is expressible in the $\mu$-calculus over $M_B(G)$ representations. Let $\mathcal{A} = \langle Q, \Sigma, q_0, \delta, \Omega, F\rangle$ be an automaton reconginizing this property. Take a trace of the form $(ab)^{2n}b^\omega$ for some $n$ and consider an accepting run of $\mathcal{A}$ on this trace. The first move of the automaton is to choose a set $W \in \delta(q_0, \bot)$ and assign a set of states $r(i) \neq \emptyset$ to each position $i > 1$. The assignment is such that $\bigcup_{i>1} r(i) = W$. Moreover for every $i$ and $q \in r(i)$ automaton $\mathcal{A}$ has an accepting run from $q$ starting at position $i$.

By finiteness of the automaton we must have $n$ and $k$ such that the first move of an accepting run of $\mathcal{A}$ on the words:

$$\bot(ab)^{2n}b^\omega \qquad \text{and} \qquad \bot(ab)^{2k}(ab)^{2n}b^\omega$$

uses the same set $W \in \delta(q_0, \bot)$. Let $r_1, r_2 : \{2, \dots\} \to \mathcal{P}(Q)$ be the assignments chosen by the automaton in the first move on the two words. We can assume that on the second word the assignment to the $(ab)^{2n}b^\omega$ suffix is exactly the same as the assignment in the first word, i.e., $r_1(i) = r_2(i + 2k)$ for $i > 1$. Consider the word

$$\bot ab(ab)^{2k-1}(ab)^{2n}b^\omega$$

It is not difficult to check that an assignment $r_3(i) = r_2(i + 2)$, for all $i > 1$, is a correct move of $\mathcal{A}$ and can be extended to an accepting run of $\mathcal{A}$. □

So we are left with three representations $M^H_{co}(G)$, $M^H_B(G)$ and $M^P_B(G)$. The first observation is that $M^H_{co}(G)$ representation gives at least as much information about $G$ as $M^H_B(G)$.

**Proposition 21** For every pair $(a, b) \in D$ there is a $\mu$-calculus formula $\beta_{ab}$ defining $Before_{ab}$ in $M^H_{co}(G)$; more formally for every $G$ and $e$ in $G$ we have: $M^H_{co}(G), e \vDash \beta_{ab}$ iff $Before_{ab}(e)$ holds in $G$.

**Proof**
Suppose $Before_{ab}$ holds in $G$ then:

- $e$ is labelled by a letter dependent on both $a$ and $b$;

- there is a $b$-event after $e$;
- there is a path from $e$ to an event $e_a$ labelled by $a$ such that no event on this path is a $b$-event or is concurrent with a $b$-event;

One can check that these three conditions are also sufficient for $Before_{ab}(e)$ to hold. The above conditions are expressed by the formula:

$$\Big[\bigvee_{c\in\Sigma_{ab}} P_c\Big] \wedge \big\langle\cdot\big\rangle[\mu X.P_b \vee \langle\cdot\rangle X] \wedge \big\langle\cdot\big\rangle[\mu Y.P_a \vee \langle\cdot\rangle(\neg co_b \wedge \neg P_b \wedge Y)]$$

where $\Sigma_{ab}$ is the set of all letters dependent on both $b$ and $c$, i.e., $\Sigma_{ab} = \{c : (a,c) \in D \wedge (a,b) \in D\}$. In the above we use $P_a$ to stand for a set of events with $a$ as the first component of the label and we use $co_b$ for the set of events with $b$ in the second component of the label. The proof that the formula expresses the required conditions is routine. □

We are now ready to formulate the main technical result of the paper.

**Theorem 22**
*For every MSOL sentence $\varphi$ there is a $\mu$-calculus sentence $\beta_\varphi$ such that for every dependence graph $G$: $G \vDash \varphi$ iff $M_B^H(G), \bot \vDash \beta_\varphi$ iff $M_B^P(G), \bot \vDash \beta_\varphi$*

**Proof**
The line of the proof is as follows. By Corollary 16 from $\varphi$ we construct a $\mu$-calculus formula $\alpha_\varphi$ such that $G \vDash \varphi$ iff $Lex(G), \bot \vDash \alpha_\varphi$. By Theorem 8 we get an equivalent automaton $\mathcal{A}_\varphi$ working on lex-trees. Next, we construct an automaton $\mathcal{C}$ which "reconstructs" a lex-tree in $M_B^H(G)$ as well as in $M_B^P(G)$. Then, we construct an automaton $\mathcal{B}$ which is a kind of product of $\mathcal{A}_\varphi$ and $\mathcal{C}$. This is an automaton running on $M_B^H(G)$ or $M_B^P(G)$ and accepting iff $\mathcal{A}_\varphi$ accepts $Lex(G)$. Using once again Theorem 8, automaton $\mathcal{B}$ can be translated back to a $\mu$-calculus formula $\beta_\varphi$.

The main difficulty in the proof is the construction of the automaton $\mathcal{C}$. Here we just state the lemma saying that it is possible. The proof of this lemma will be given in the two following sections. Recall that $M_B^H(G)$ and $M_B^P(G)$ are $\Gamma$-labelled graphs where $\Gamma = \Sigma \times \mathcal{P}(\Sigma \times \Sigma)$. So our automaton $\mathcal{C}$ will also use this alphabet. We will write $\downarrow_1$ and $\downarrow_2$ for projections on the first and second component respectively.

**Lemma 23** There is an automaton $\mathcal{C} = \langle Q_c, \Gamma, q_c^0, \delta_c, F_c, \Omega_c\rangle$ which reconstructs lexicographic trees, i.e., for every dependence graph $G = \langle E, R, \lambda\rangle$:

- $\mathcal{C}$ has unique accepting run $r : S \to E \times Q_c$ on $M_B^H(G)$ as well as on $M_B^P(G)$.
- there is a special state $tt \in F_{\mathcal{C}}$ such that when we restrict $r$ to $S' = \{v : r(v)\downarrow_2 \neq tt\}$ then $S'$ is a tree and $r\downarrow_1 : S' \to E$ is a tree isomorphism between $S'$ and $Lex(G)$.

Suppose that we have such an automaton $\mathcal{C}$ as in the lemma and let us proceed with the proof. Let $\mathcal{A}_\varphi = \langle Q_a, \Sigma, q_a^0, \delta_a, F_a, \Omega_a \rangle$ be an automaton over alphabet $\Sigma$. We construct an automaton $\mathcal{B}$:

$$\mathcal{B} = \langle Q_b, \Gamma, q_b^0, \delta_b, F_b, \Omega_b \rangle$$

where:

- $Q_b = (Q_a \times (Q_c \setminus \{tt\})) \cup Q_c$
- $q_b^0 = (q_a^0, q_c^0)$
- for $q_a \notin F_a$ we have $\delta_b((q_a, q_c), l) = \bigcup\{Choice(W_a, W_c) : W_a \in \delta(q_a, l\downarrow_1),\ W_c \in \delta(q_c, l)\}$ with $Choice(W_a, W_c)$ consisting of all the sets $W$ such that:

$$\{q_a' : \exists q_c'.\ (q_a', q_c') \in W\} = W_a$$
$$\{q_c' : \exists q_a'.\ (q_a', q_c') \in W\} \cup \{tt : tt \in W\} = W_c$$

- for $q_a \in F_a$ we have $\delta_b((q_a, q_c), l) = \delta_c(q_c, l)$
- $F_b = (F_a \times (F_c \setminus \{tt\})) \cup \{tt\}$
- $\Omega_b((q_a, q_c)) = \Omega_a(q_a)$ and $\Omega_b(q_c) = \Omega_c(q_c)$

We show that for every dependence graph $G$:

$$M_B^H(G) \in L(\mathcal{B}) \quad \text{iff} \quad Lex(G) \in L(\mathcal{A}_\varphi)$$

The proof for $M_H^P(G)$ representation is essentially the same.

First, let us show that if there is an accepting run $r_a : S_a \to E \times Q_a$ of $\mathcal{A}_\varphi$ on $Lex(G)$ then there is accepting run $r_b : S_b \to E \times Q_b$ of $\mathcal{B}$ on $M_B^H(G)$. Let $r_c : S_c \to E \times Q_c$ be the unique run of $\mathcal{C}$ on $G$.

We construct a run $r_b : S_b \to E \times Q$ by induction on the distance of a node from the root. The nodes of $S_b$ will come from the set $(S_a \times S_c) \cup S_c$. If a node of $S_b$ will be of the form $(v_a, v_c) \in S_a \times S_c$ then we will have:

$$\begin{array}{l} r_b(v_a, v_c) = (e, (q_a, q_c)) \\ \text{with } e = r_a(v_a)\downarrow_1 = r_c(v_c)\downarrow_1,\ q_a = r_a(v_a)\downarrow_2 \text{ and } q_c = r_c(v_c)\downarrow_2 \end{array} \tag{1}$$

If a node of $S_b$ will be of the form $v_c \in S_c$ then we will have:

$$r_b(v_c) = r_c(v_c) \tag{2}$$

The root of $S_b$ is $(\bot_a, \bot_c)$ where $\bot_a$, $\bot_c$ are the roots of $S_a$ and $S_c$ respectively. We put $r_b(\bot_a, \bot_c) = (\bot, (q_a^0, q_c^0))$, where $\bot$ is the least event of $G$.

Suppose we have a node $(v_a, v_c)$ of $S_b$ and (1) holds. We have several cases depending on whether $v_a$ or $v_c$ have sons.

If $v_c$ has no sons in $S_c$ then $e$ is a leaf in $Lex(G)$. So $q_a \in F_a$ as $r_a$ is an accepting run of $\mathcal{A}_\varphi$ on $Lex(G)$. Hence $(q_a, q_c) \in F_b$ as $q_c \neq tt$.

If $v_a$ has no sons and $v_c$ has sons $w_c^1, \ldots, w_c^n$ then we know that $q_a \in F_a$. For each $i = 1, \ldots, n$ we make $w_c^i$ a son of $(v_a, v_c)$ and put $r_b(w_c^i) = r_c(w_c^i)$.

The last case is when both $v_a$ and $v_c$ have sons. Let $w_a^1, \ldots, w_a^m$ and $w_c^1, \ldots, w_c^n$ be sons of $v_a$ and $v_c$ respectively. For each $i, j$ such that $r_a(w_a^i)\downarrow_1 = r_c(w_c^j)\downarrow_1$ we create a son $(w_a^i, w_c^j)$ of $(v_a, v_c)$ labelled by $(r_a(w_a^i)\downarrow_1, (r_a(w_a^i)\downarrow_2, r_c(w_c^j)\downarrow_2))$. This way we have taken care of all the events that are sons of $e$ in $Lex(G)$. For every event $e'$ which is a successor of $e$ but not a son of $e$ in $Lex(G)$ there is $j$ with $r_c(w_c^j)\downarrow_1 = (e', tt)$. We make $w_c^j$ a son of $(v_a, v_c)$ and label it with $r_c(w_c^j)$.

Finally, we define $r_b$ for nodes of $S_b$ of the form $v_c \in S_c$. In this case we know by (2) that $r_b(v_c) = r_c(v_c)$ and we just copy the run of $\mathcal{C}$. More precisely for each son $w_c$ of $v_c$ in $S_c$ we make $w_c$ also a son of $v_c$ in $S_b$ and put $r_b(w_c) = r_c(w_c)$.

It is not difficult to check that $r_b$ is a locally consistent run. Clearly, every leaf is labelled by a state from $F_b$. So it remains to show that every infinite path satisfies the parity condition of $\mathcal{B}$. Suppose $v^0, v^1, \ldots$ is an infinite path in $S_b$ and $v^i \in S_a \times S_c$ for all $i$. Let $v^i = (v_a^i, v_c^i)$ for all $i$. Recall that $r_b(v^i)\downarrow_2 = (r_a(v_a^i)\downarrow_2, r_a(v_c^i)\downarrow_2)$. By definition of $\Omega_b$ we have that $\Omega_b(r_a(v_a^i)\downarrow_2, r_a(v_c^i)\downarrow_2) = \Omega_a(r_a(v_a^i)\downarrow_2)$. Hence $v^0, v^1, \ldots$ satisfies the parity condition $\Omega_b$ because by the assumption $v_a^0, v_a^1, \ldots$ satisfies the parity condition $\Omega_a$. The other case is when for an infinite path $v^0, v^1, \ldots$ we have $v^i \in S_c$ for some $i$. Then $v^j \in S_c$ and $r_b(v_j) = r_c(v_j)$ for all $j \geq i$. As $\Omega_b(v_j) = \Omega_c(v_j)$, we get that this path satisfies the parity condition.

Now we want to show that whenever $\mathcal{B}$ accepts $G$ then $\mathcal{A}_\varphi$ accepts $Lex(G)$. Let $r_b : S_b \to E \times Q_b$ be an accepting run of $\mathcal{B}$ on $G$. Let $r_c : S_c \to E \times Q_c$ be the unique accepting run of $\mathcal{C}$ on $G$. Let us define $f : G \to Q_c$ by $f(e) = q$ iff there is $v \in S_c$ with $r_c(v) = (e, q)$ and $q \neq tt$. This function is well defined by our assumption on $\mathcal{C}$.

We claim that for every $v \in S_b$ if $r_b(v) = (e, (q_a, q_c))$ then $q_c = f(e)$. This follows by an easy induction on the distance of $v$ from the root.

Let $S_a = \{v \in S_b : r_b{\downarrow_2}\ (v) \in Q_a \times Q_c\}$. Clearly $S_a$ is a tree by the definition of automaton $\mathcal{B}$. We define $r_a : S_a \to E \times Q_a$ by $r_a(v) = (e, q_a)$ whenever $r_b(v) = (e, (q_a, q_c))$.

We want to show that $r_a$ is an accepting run of $\mathcal{A}_\varphi$ on $Lex(G)$. It is easy to see that every infinite path in $S_a$ satisfies the parity condition given by $\Omega_a$. So it remains to check if $r_a$ is locally consistent. Let $v \in S_a$ with $r_b(v) = (e, (q_a, q_c))$. As $q_c = f(e)$ we know that the sons of $v$ which are assigned state other than $tt$ are labelled with lex-sons of $e$ and every lex-son of $e$ is in a label of one of the sons of $v$. Then by the definition of $\mathcal{B}$ we get that $r_a$ is locally consistent in $v$. □

We sum up the results of this section in the corollary below. This is the main result of the paper.

Let $\mu(\mathit{Before})$ stand for the extension of the $\mu$-calculus over the alphabet $\Sigma$ with added proposition $\mathit{Before}_{ab}$ for every $(a, b) \in D$. The meaning of such a proposition is: $G, e \vDash \mathit{Before}_{ab}$ iff $\mathit{Before}_{ab}(e)$ holds in $G$. It is straightforward to see that $\mu(\mathit{Before})$ over dependence graph representation of traces is equivalent to the plain $\mu$-calculus over the alphabet $\Gamma_B = \Sigma \times \mathcal{P}(\Sigma \times \Sigma)$ and $M_B^H(G)$ representation of traces.

Similarly let $\mu(co)$ stand for the extension of the $\mu$-calculus over the alphabet $\Sigma$ with propositions $co_a$ for every $a \in \Sigma$. The meaning of such a proposition is: $G, e \vDash co_a$ iff there is an event $e'$ in $G$ labelled with $a$ and such that $co(e, e')$ holds. Once again $\mu(co)$ corresponds to the plain $\mu$-calculus over $M_{co}^H(G)$ representations of traces.

**Corollary 24 (Expressive completeness)** For every MSOL formula $\varphi$ there are equivalent formulas $\alpha_\varphi$, $\beta_\varphi$ and $\gamma_\varphi$ of the $\mu$-calculus such that: $G \vDash \varphi$ iff $M_B^P(G) \vDash \alpha_\varphi$ iff $M_B^H(G) \vDash \beta_\varphi$ iff $M_{co}^H(G) \vDash \gamma_\varphi$. Also the converse holds: for a $\mu$-calculus formula over one of the three representations of traces there is an equivalent MSOL formula.

# 7 Local characterisation of lex-trees

What remains to be done is the construction of an automaton $\mathcal{C}$ reconstructing lex-trees in dependence graphs. We have defined lex-trees using some global properties of events. In this section we would like to show that there is a labelling of events which is defined by some local conditions and such that a label of an event identifies which among the

successors of the event are lex-sons (i.e., sons in the lex-tree). We will use this labelling in the next section to construct an automaton reconstructing the lex-tree in a given dependence graph. For this section let us fix a dependence graph $G = \langle E, R, \lambda \rangle$.

**Definition 25** In a lex-tree a *left split from* $e$ is an event $e'$ which is a son of an ancestor of $e$ and which is to the left of $e$ (i.e., $lexp(e') <_\Sigma lexp(e)$).

**Lemma 26** For every $e$ there are no more than $|\Sigma|$ left splits from $e$.

**Proof**
Let $e$ be an event and let $e_a$, $e_b$ be its two sons labelled $a$ and $b$ respectively. Assume that $a$ is smaller than $b$ in our fixed ordering on $\Sigma$. The lemma follows from the observation that there cannot be an $a$ labelled descendant of $e_b$ in the lex-tree. Suppose conversely that there is an $a$-event $e'_a$ which is a lex-descendant of $e_b$. Then $lexp(e'_a)$ goes through $e$ and $e_b$ but not through $e_a$. So $e_a$ is after $e'_a$ in the trace ordering. Hence $e_a$ cannot be a direct successor of $e$ in the trace as we have a path to $e_a$ going through $e_b$ and $e'_a$. □

**Definition 27** A *lex-slice from an event* $e$, denoted $G(e)$, is the restriction of $G$ to the events:

$$\{e' : R(e, e') \text{ or } R(e'', e') \text{ for } e'' \text{ a left split from } e\}$$

In words, this is the set of events which are after $e$ or after some left split from $e$.

Next, we define a concept of a view. Intuitively, a view from an event $e$ describes the dependencies one can see in the lex-slice of $e$. As we want views to be finite, we just note the dependencies between first occurrences of actions. A view is something that will be guessed, so we define it without a reference to a particular event or trace.

**Definition 28** A *view* is a binary relation $V$ on a set $X \subseteq \Sigma$ such that $V$ relates two letters $a, b \in X$ iff $(a, b) \in D$ and such that a reflexive and transitive closure $V^*$ of $V$ is a partial order. Let *Views* be the set of all the views.

**Definition 29** For a view $V$, let $Alph(V) \subseteq \Sigma$ be the set of letters the view relates. Let $Min(V)$ be the set of minimal elements of $V$ (i.e., minimal in the partial order $V^*$). For a letter $a \in Min(V)$ let $Left(V, a) =$

$\{b : \exists_{c \neq a} V(c,b)$ and $c$ minimal in $V\}$ be the set of those letters from $Alph(V)$ which are bigger than some minimal element of $V$ other than $a$ (the name comes from the fact that usually $a$ will be the "rightmost" minimal element).

**Definition 30** Let $e$ be an event and let $V$ be a view. By $V\downarrow_e$ we denote the view obtained from $V$ by possibly changing the relation of $\lambda(e)$ to letters $a$ such that $(a, \lambda(e)) \in D$. We put $(a, \lambda(e))$ in $V(e)\downarrow_e$ if $Before_{a\lambda(e)}(e)$ holds and we put $(\lambda(e), a)$ in $V(e)\downarrow_e$ if $Before_{\lambda(e)a}(e)$ holds. If none of these holds then $\lambda(e)$ does not appear at all in $G(e) \setminus \{e\}$. In this last case $V\downarrow_e$ does not relate $\lambda(e)$ at all and the domain of $V\downarrow_e$ becomes $Alph(V) \setminus \{\lambda(e)\}$. If $\lambda(e) \notin Alph(V)$ then $V\downarrow_e = V$.

We define a projection from an event to be the correct view from the event.

**Definition 31 (Projection from an event)** Let $G(e)$ be the lex-slice for $e$. We define $P(e)$, the *projection from* $e$. For every two letters $(a,b) \in D$ such that both of them appear in $G(e)$ we put $(a,b) \in P(e)$ if in $G(e)$ the first $a$-event is before the first $b$-event; we put $(b,a) \in P(e)$ otherwise.

The lemmas below show what kind of information we can deduce from a projection of an event.

**Lemma 32** The minimal elements of $P(e)$ are exactly the labels of the minimal events in $G(e)$, and these are $e$ and all left splits of $e$.

**Proof**
If $a$ is minimal in $P(e)$ then by the definition the first $a$ event is minimal in $G(e)$. Hence this $a$-event must be $e$ or the left split from $e$.

We want to show that if $e'$ is a left split from $e$ then $e'$ is minimal in $G(e)$. This will also show that $\lambda(e')$ is minimal in $P(e)$.

Suppose not, then there is another event $e''$ before $e'$ which is a left split of $e$ or $e$ itself. Let $e^{(3)}$, $e^{(4)}$ be events on the lex-path to $e$ of which $e'$ and $e''$ are respectively lex-sons. If $e^{(3)}$ is before $e^{(4)}$ then we get a path from $e^{(3)}$ to $e'$ (it goes from $e^{(3)}$ to $e^{(4)}$ to $e''$ and then to $e'$). This contradicts the fact that $e'$ is a successor of $e^{(3)}$ in the Hasse diagram of $G$. If $e^{(4)}$ is before $e^{(3)}$ then $e'$ is not a left split of $e$ as the lex path to $e'$ does not go through $e^{(3)}$ but through $e''$.

□

**Lemma 33** If $b \in \mathit{Left}(P(e), \lambda(e))$ then the first $b$ event in $G(e)$ is not a lex-descendant of $e$ in $\mathit{Lex}(G)$.

**Proof**
If $(c, b) \in P(e)$ for some minimal letter $c \neq \lambda(e)$ then there is a path from the first $c$ event to the first $b$ event in $G(e)$. By Lemma 32, the first $c$ event is a left split from $e$. □

**Lemma 34** For every event $e$ the set $\mathit{Min}(P(e)\downarrow_e) \setminus (\mathit{Min}(P(e)) \setminus \{\lambda(e)\})$ is the set of labels of lex-sons of $e$.

**Proof**
If $\lambda(e) \in \mathit{Min}(P(e)\downarrow_e) \setminus (\mathit{Min}(P(e)) \setminus \{\lambda(e)\})$ then $\mathit{Before}_{\lambda(e)a}(e)$ holds for every letter $a$ dependent on $\lambda(e)$. Hence there is the unique successor $e'$ of $e$ labelled by $\lambda(e') = \lambda(e)$. It can be checked that this successor is a lex-son because every path to $e'$ goes through $e$.

For the other case suppose $b \in \mathit{Min}(P(e)\downarrow_{\lambda(e)}) \setminus (\mathit{Min}(P(e)) \setminus \{\lambda(e)\})$ with $b \neq \lambda(e)$. In this case $\mathit{Before}_{b\lambda(e)}(e)$ holds.

Let $e_b$ be the first $b$-event in $G(e)$. We want to show that $e_b$ is a lex-son of $e$. Suppose first that $e_b$ is not a successor of $e$. Then, as $b$ depends on $\lambda(e)$, there is a path from $e$ to $e_b$ and say $e'$ is just before $e_b$ on it. We have that $\lambda(e')$ is dependent on $b$ and $\lambda(e') \neq \lambda(e)$ hence $(\lambda(e'), b) \in P(e)\downarrow_e$ a contradiction with the minimality of $b$ in $P(e)\downarrow_e$. To see that $e_b$ is a lex-son of $e$ observe that for a similar reason there cannot be a path from some left split of $e$ to $e_b$.

Finally observe that every lex-son of $e$ is labelled by some letter from $\mathit{Min}(P(e)\downarrow_e) \setminus (\mathit{Min}(P(e)) \setminus \{\lambda(e)\})$. This is because whenever $e''$ is a lex-son of $e$ then $\mathit{Before}_{\lambda(e'')a}(e)$ holds for all $a$ dependent on $\lambda(e'')$. So $\lambda(e'') \in \mathit{Min}(P(e)\downarrow_e)$ and $\lambda(e'') \notin (\mathit{Min}(P(e)) \setminus \lambda(e))$. □

Next, we define a pair of labelling functions which we call consistent view assignment. Each of this functions will assign a view to an event of a trace. The main future of these labellings is that some local consistency conditions are enough to determine them uniquely. Having local conditions is important because they can be easily checked by an automaton.

**Definition 35** *Consistent view assignment* for a trace $G$ is a pair of functions $(V_L, V)$ each assigning a view to every event of $G$. For every event $e$, these functions have to satisfy the following consistency conditions.

1. If $e$ is the root of $G$ then $V(e) = P(e)$ and $V_L(e) = \emptyset$.

2. If $Min(V(e)\downarrow_e) = Min(V(e)) \setminus \{\lambda(e)\}$ (intuitively $e$ has no lex-sons) then $V_L(e) = V(e)\downarrow_e$.

3. If $Min(V(e)\downarrow_e) = Min(V(e))$ (intuitively $e$ has a unique lex son labelled by $\lambda(e)$) then there is a successor $e'$ of $e$ labelled with $\lambda(e)$ and we must have $V(e') = V(e)$ and $V_L(e') = V_L(e)$.

4. If $Min(V(e)\downarrow_e) = (Min(V(e)) \setminus \{\lambda(e)\}) \cup \{b_1, \ldots, b_k\}$ with $b_1 <_\Sigma \cdots <_\Sigma b_k$ in our fixed ordering on $\Sigma$ then there must be successors $e_1, \ldots, e_k$ of $e$ labelled by $b_1, \ldots, b_k$ respectively and we must have:

   (a) $V(e_k) = V(e)\downarrow_e$,

   (b) $Alph(V_L(e_i)) \subseteq Alph(V(e_i))$ and $V_L(e_i)$ agrees with $V(e_i)$ on $Left(V(e_i), \lambda(e_i))$ for $i = 1, \ldots, k$,

   (c) $V_L(e) = V_L(e_1)$ and $V(e_{i-1}) = V_L(e_i)$ for $i = 2, \ldots, k$.

**Proposition 36** For every dependence graph $G$ there is a consistent view assignment.

**Proof**
Define a view assignment by letting $V(e) = P(e)$ and $V_L(e) = P(e_L)$ where $e_L$ is the biggest in "to the left" ordering split from $e$ (we will call it biggest left split for short). We put $V_L(e) = \emptyset$ if there is no such $e_L$. We have several cases to consider.

Clearly the root condition of the definition of consistent assignment is satisfied.

Suppose $Min(V(e)\downarrow_e) = Min(V(e)) \setminus \{\lambda(e)\}$ then by Lemma 34 there are no lex-sons of $e$. We have that $P(e_L) = P(e)\downarrow_e$.

Suppose $Min(V(e)\downarrow_e) = Min(V(e))$ then by Lemma 34 there is the unique lex-son $e'$ of $e$ labelled $\lambda(e)$. We have that $P(e') = P(e)$ and that $e_L$ is the biggest left split also for $e'$.

Finally suppose $Min(V(e)\downarrow_e) = (Min(V(e)) \setminus \{\lambda(e)\}) \cup \{b_1, \ldots, b_k\}$ with $b_1 <_\Sigma \cdots <_\Sigma b_k$ listed according to our ordering on $\Sigma$. By Lemma 34 there are lex-sons $e_1, \ldots, e_k$ of $e$ labelled with $b_1, \ldots, b_k$ respectively and these are the only lex-sons of $e$.

It is not difficult to check that $P(e_k) = P(e)\downarrow_e$. To check the next condition observe that $e_L$ is the biggest left split for $e_1$ and $e_{i-1}$ is the biggest left split for $e_i$ $(i = 2, \ldots, k)$. This means that $V(e_{i-1}) = V_L(e_i)$ and $V_L(e) = V_L(e_1)$. We have that $Alph(P(e_{i-1})) \subseteq Alph(P(e_i))$ because

the left slice $G(e_{i-1})$ is a suffix of $G(e_i)$. Finally let us check that $V_L(e_i)$ and $V(e_i)$ agree on the letters from $\mathit{Left}(V(e_i), \lambda(e_i))$. We have that $V_L(e_i) = P(e'_i)$ where $e'_i$ is the biggest left split of $e_i$ (i.e. $e'_i = e_{i-1}$ or $e'_i = e_L$ if $i = 1$). By Lemma 33 for every letter $a \in \mathit{Left}(P(e_i), \lambda(e_i))$ the first $a$-event in $G(e_i)$ is also the first event in $G(e'_i)$. □

We finish this section with a proposition showing that there is exactly one consistent view assignment.

**Proposition 37** If $(V_L, V)$ is a consistent view assignment then for every event $e$ we have $V(e) = P(e)$. (Consistency conditions imply that $V_L$ is also determined)

Before proving the proposition we need some lemmas. For the rest of this section let us fix a consistent view assignment $(V_L, V)$.

**Definition 38** We say that an event $e$ is *good* if $V(e) = P(e)$ (it does not matter what $V_L(e)$ is)

**Lemma 39** If $e$ is good and $e'$ is its rightmost lex-son then $e'$ is good.

**Proof**
This is because the only difference between $V(e)$ and $V(e')$ is for pairs containing letter $\lambda(e)$. The correct pairs for $V(e')$ are calculated with $\mathit{Before}_{ab}(e)$ predicates. □

**Lemma 40** Let $e$ be a good event with lex-sons $e_1, \ldots, e_k$ listed in "to the left" ordering (with $e_k$ rightmost). Suppose that $e_i$ and all descendants of $e_i$ in $\mathit{Lex}(G)$ are good then $e_{i-1}$ is good.

**Proof**
There are two cases to consider.

Suppose there is a leftmost node, $e'$, in the subtree of the lex-tree rooted in $e_i$. As $e'$ is good we get $V(e') = P(e')$. Then, by the consistency conditions (1) and (4$c$) we have $V_L(e') = P(e')\downarrow_{e'}$ and $V(e_{i-1}) = V_L(e')$. Hence $e_{i-1}$ is good as $P(e_{i-1}) = P(e')\downarrow_{e'}$.

The other case is when there is an infinite leftmost lex-path $P = e'_1 e'_2 \ldots$ from $e_i$. To shorten the notation let us write $\mathit{Left}(e')$ for the set of letters $\mathit{Left}(V(e'), \lambda(e'))$ and $\mathit{Right}(e')$ for $\mathit{Alph}(V(e') \setminus \mathit{Left}(e'))$. Observe that the sequence $\mathit{Right}(e'_1), \mathit{Right}(e'_2), \ldots$ is not increasing. Hence it stabilizes on some set $\mathit{Inf}$.

Let $e'$ be an event on $P$ with $Right(e') = Inf$. We claim that for every letter $a \in Inf$ the first $a$ event in $G(e')$ is a descendant of $e'$ in $Lex(G)$. Suppose conversely that $e_a$ is to the left. As $a \in Inf$ we also know that $e_a$ is after $e'$. Going down the path $P$ we show that $e_a$ is to the left and after every event in $P$. But then we would have infinitely many events before $e_a$. This is impossible by the definition of traces.

This argument actually shows that there are no $a$ events to the left of $e'$. So no event labelled by a letter from $Right(e')$ can appear in $G(e_{i-1})$. By definition of consistent views $V(e')$ is consistent with $V_L(e')$ on $Left(e')$. Moreover, as no more event is going to get to the left, $V(e_{i-1})$ is just $V(e')$ restricted to $Left(e')$. □

Now we are ready to prove Proposition 37

**Proof** (of Proposition 37)
By definition, the root event is good. Assume that in tree $Lex(G)$ there is an event which is not good. Let us go down the tree always choosing the rightmost son in which subtree there is a not good event. By Lemma 34 we know that the label of a good event determines the lex-sons of the event. Finally we must get to a not good event $e$ as we can make at most $|\Sigma|$ right turns. Let $e_1$ be the father of $e$. By assumption $e_1$ is good so $e$ cannot be the rightmost son of $e_1$ by Lemma 39. Let $e_2$ be the son of $e$ immediately to the right of $e$. By our choice of $e$ all events in the lex-subtree of $e_2$ are good so $e$ is good by Lemma 40. A contradiction. □

# 8 Automaton reconstructing lex-trees

Recall that *Views* is the set of views over the alphabet $\Sigma$ (cf. Definition 28). Before defining an automaton reconstructing lex-trees we will need one auxiliary operation. Suppose $V$ is a view, $a$ is a minimal element in $V$ and $B \subseteq (\Sigma \times \Sigma)$ is a partial order relation on $\Sigma$. We define updated view $V{\downarrow_{(B,a)}}$ to be the same as $V$ on letters other than $a$ and to have $(a, b)$ if $(a, b) \in B$ and $(b, a)$ if $(b, a) \in B$. If $a$ is related to no element in $B$ then $V{\downarrow_{(B,a)}}$ is $V$ without pairs containing $a$. The intention is that when we have a trace $G$ and an event $e$ with $V = P(e)$, $a = \lambda(e)$ and $B = \{(b, c) : Before_{b,c}(e)\}$ then $V{\downarrow_{(B,a)}}$ is $P(e){\downarrow_e}$.

We define automaton $\mathcal{C} = \langle Q, \Gamma, q^0, \delta, F, \Omega\rangle$ as follows:

- $Q = (\Sigma \times Views \times Views) \cup \{q^0, tt\}$,

- $\Gamma = \Sigma \times \mathcal{P}(\Sigma \times \Sigma)$,
- $\Omega(q) = 0$ for every state $q$.

It remains to define $F$ and the transition function. The set $F$ contains $tt$ and all the pairs $((a, V_L, V), (B, a))$ such that $Min(V{\downarrow_{(B,a)}}) = Min(V) \setminus \{a\}$ and $V_L = V{\downarrow_{(B,a)}}$. Intuitively in this case from $V$, $B$ and $a$ we can determine that the current event has no lex sons.

For the transition function $\delta$ we define $\delta((a, V_L, V), (b, B))$ by cases:

- if $Min(V{\downarrow_{(B,a)}}) = Min(V)$ then
  $\delta((a, V_L, V), (a, B)) = \{\{(a, V_L, V), tt\}\}$
- If $Min(V{\downarrow_{(B,a)}}) = (Min(V) \setminus \{a\}) \cup \{b_1, \ldots, b_k\}$ (where $b_1, \ldots, b_k$ are $<_\Sigma$-ordered by our order on $\Sigma$) then $\delta((a, V_L, V), (a, B))$ contains all the sets $\{(b_1, V_1', V_1), \ldots, (b_k V_k', V_k), tt\}$ such that:
  - $V_k = V{\downarrow_{(B,a)}}$,
  - $Alph(V_i') \subseteq Alph(V_i)$ and $V_i'$ agrees with $V_i$ on $Left(V_i, b_i)$
  - $V_L = V_1'$ and $V_{i-1} = V_i'$ for $i = 2, \ldots, k$.
- $\delta((a, V_L, V), (b, B)) = \emptyset$ otherwise.

Finally we let $\delta(q^0, (\bot, B)) = \delta((\bot, \emptyset, B), (\bot, B))$ as we can consider $B$ to be a view. There are no tranistions from state $tt$.

The definition of transition relation directly reflects the definition of consistent view assignment (cf. Definition 35). The idea of the construction is that a run of $\mathcal{C}$ on $G$ corresponds to a consistent view assignment. As there is exactly one consistent view assignmnet for every trace, automaton $\mathcal{C}$ will have exactly one accepting run on each trace.

**Theorem 41**
*For every dependence graph $G$ there is a unique accepting run of $\mathcal{C}$ on $M_B^H(G)$ or $M_B^P(G)$. The restriction of this run to nodes having states other than tt in their label is isomorphic to the lexicographic tree $Lex(G)$.*

**Proof**
For simplicity we will concentrate on $M_B^H(G)$ representation of traces. The proof for the other representation is very similar. We will mention the differences in the text.

First, let us show that there is a run of $\mathcal{C}$ on $M_B^H(G)$ for every dependence graph. Let $G$ be a dependence graph and let $Lex(G)$ be the lex-tree of $G$. Consider the function $r : G \to E \times Q$ defined by:

$$r(e) = (e, (\lambda(e), P_L(e), P(e)))$$

where $P_L(e)$ is the projection from the leftmost split of $e$ or $P_L(e) = \emptyset$ if there is no such split.

Function $r$ is almost a run of $\mathcal{C}$ on $M_B^H(G)$. We just need to do a couple of cosmetic changes. We change the value of $r(\perp)$ to $(\perp, q_0)$. Next, for every $e \in G$ and every successor $f$ of $e$ which is not a son of $e$ in $Lex(G)$ we create a new son $v_f^e$ of $e$. If there are successors of $e$ but all of them are sons of $e$ in $Lex(G)$ then we choose one such successor $f$ arbitrary and create a new son $v_f^e$ of $e$. We put $r(v_f^e) = (f, tt)$ for each of the new vertices. Using Proposition 36 one can check that $r$ is a run of $\mathcal{C}$. As $\Omega(q) = 0$ for all states, every run is accepting. If we had $M_B^P(G)$ representation of a trace then the only difference would be that there would be more $v_f^e$ nodes created.

Now assume that there is an accepting run of $\mathcal{C}$ on $G$. Let $r : S \to E \times Q$ be the part of this run restricted to nodes $v$ such that the state in $r(v)$ is different from $tt$. In other words $r$ is obtained from the run by cutting of the leaves labelled with $tt$. We have a function $r{\downarrow_1}$: $S \to E$. We will show that it is an isomorphism.

We say that a vertex $v$ of $S$ is *good* if $r(v) = (e, V_L, V)$ for some $e$, $V_L$, $V$ and $V = P(e)$.

**Lemma 42** Suppose that $v$ is good then then $r{\downarrow_1}$ is a bijection between sons of $v$ and sons of $e$ in $Lex(G)$.

**Proof**
From Lemma 36 we get that the sons of $e$ in lex-tree are determined by $P(e)$ and $Before_{ab}(e)$ relations. Similarly $P(e)$ and the label of $e$ determine the first components of states in the transition of the automaton. □

For an event $e$ define the set $AR(e)$ of events which are ancestors or to the right of $e$ in $Lex(G)$:

$$\begin{aligned} AR(e) =& \{e' : lexp(e') \text{ is a prefix of } lexp(e)\} \cup \\ & \{e' : lexp(e) <_\Sigma lexp(e') \text{ and } lexp(e) \text{ is not a prefix of } lexp(e')\} \end{aligned}$$

A path $v_0 v_1 \dots$ in $S$ defines a sequence of letters $a_0 a_1 \dots$ which are the letters appearing in $r(v_0), r(v_1), \dots$ By the definition of the automaton

such a sequence of letters determines uniquely the path in $S$. Hence we can compare vertices in $S$ by comparing lexicographically the labels of paths leading to them. We can also define the set $AR(v)$ of vertices above or to the right of $v$ in the same way as we did for the events.

We say that $r\downarrow_1$ is a *good bijection* between a subset of $S$ and the subset of $G$ iff it is a bijection and each node in the domain is good.

Suppose that we have a vertex $v$ of $S$ such that:

1. $r\downarrow_1$ is a good bijection between $AR(v)$ and $AR(e)$.
2. $r\downarrow_1$ is not a good bijection between subtree of $S$ rooted in $v$ and the subtree of $Lex(G)$ rooted in $r\downarrow_1(v)$.

We will show that if we have such $v$ then we can find its son $v'$ with the same properties and such that $r\downarrow_1(v')$ is a son of $e$ in $Lex(G)$.

By Lemma 42 $r\downarrow_1$ is a bijection between the sons of $v$ and the sons of $r\downarrow_1(e)$. Let $v_1, \ldots, v_k$ be the sons of $v$ listed in the order of letters in their labels. Vertex $v_k$ is the rightmost son of $v$ and $r\downarrow_1(v_k)$ is the rightmost lex-son of $e$. By Lemma 39 we have that $v_k$ is good. Hence clause 1 is satisfied for $v_k$. If $r\downarrow_1$ is not a good bijection between the subtrees rooted in $v_k$ and $r\downarrow_1(v_k)$ then $v_k$ is the son we are looking for. Otherwise, $v_{k-1}$ satisfies clause 1 because by Lemma 40 we know that $v_{k-1}$ is good. Continuing like this we must find a son $v_i$ of $v$ which satisfies both clauses. Otherwise, we would have that $r\downarrow_1$ is a good bijection between descendants of $v$ and descendants of $r\downarrow_1(v)$ which is impossible by clause 2 of our assumption.

Let us iterate this construction to infinity. Let $v'_1, v'_2, \ldots$ be the events chosen in successive iterations of the construction. We have that the events $r\downarrow_1(v'_1), r\downarrow_1(v'_2), \ldots$ form a path in $Lex(G)$. As every infinite path in $Lex(G)$ is eventually leftmost there is an event $r\downarrow_1(v'_i)$ starting from which the path goes only from a father to the leftmost son. But then $r\downarrow_1$ is an isomorphism between descendants of $v'_i$ and descendants of $r\downarrow_1(v'_i)$ which was assumed not to exist. This shows that $v$ with the above two properties cannot exist.

Take the root $v_0$ of $S$ and the least element $\bot$ of $G$. We have $r(\bot) = (\bot, q_0)$. By definition of the automaton, its move from $q_0$ on the letter $(\bot, P(\bot))$ is exactly the same as from the state $(\lambda(\bot), \emptyset, P(\bot))$ on this letter. So we can pretend $r(v_0) = (\bot, (\lambda(\bot), \emptyset, P(\bot)))$ and not $(\bot, q_0)$. But then $v_0$ satisfies clause 1 from the above. Hence we must have that 2 is not satisfied. So $r\downarrow_1$ is a good bijection between $S$ and $E$. This shows that a run of $\mathcal{C}$ is uniquely determined. $\square$

# 9 Complexity issues

In this section we will show that the satisfiability problem for the logics proposed in this paper is PSPACE-complete. For a given formula $\alpha$ we will construct an automaton $\mathcal{A}(\alpha)$ recognizing all linearisations of all the traces satisfying $\alpha$. This way, to check if a formula $\alpha$ is satisfiable it is enough to check if $\mathcal{A}(\alpha)$ accepts some word. As it is usually the case with linear time logics, the model checking problem has the same complexity as the satisfiability problem.

**Definition 43** A *linearization* of a trace $G = \langle E, R, \lambda \rangle$ is a word $w \in \Sigma^\omega$ which corresponds to some linear order containing partial order $R$, i.e., $w$ is the sequence of labels of events in the chosen linear order. In particular we consider only linear orders of type $\omega$. Let $Lin(G)$ denote the set of all linearizations of $G$.

A word $w \in Lin(G)$ determines the linear order extending $R$. We will use $w(i)$ for $i$-th letter of $w$ and $e^w(i)$ for the event it represents, namely, the event which is on $i$-th position in the linear ordering determined by $w$.

First, we will deal with the $\mu$-calculus over dependence graphs without additional information in the labels. Later, we will extend the constructions to other $\mu$-calculi.

A $\mu$-calculus formula is *positive* if all the negations appear only before propositional constants. To have equivalent positive formula for every formula of the $\mu$-calculus we have to extend the syntax, which is now given by the grammar:

$$X \mid P_a \mid \neg P_a \mid \alpha \vee \beta \mid \alpha \wedge \beta \mid \langle\cdot\rangle\alpha \mid [\cdot]\alpha \mid \mu X.\alpha(X) \mid \nu X.\alpha(X)$$

the meaning of the two new constructs is defined by:

$$\begin{aligned} [\![[\cdot]\alpha]\!]_V^G &= \{e \in E : \forall e'.\ R(e,e') \wedge e' \in [\![\alpha]\!]_V^G\} \\ [\![\nu X.\alpha(X)]\!]_V^G &= \bigcup \{S \subseteq E : S \subseteq [\![\alpha(X)]\!]_{V[S/X]}^G\} \end{aligned}$$

It is well known that every formula of the $\mu$-calculus is equivalent to a formula generated by the above grammar. We will use $\sigma$ to denote either $\mu$ or $\nu$. So $\sigma X.\alpha(X)$ can be either $\mu X.\alpha(X)$ or $\nu X.\alpha(X)$.

A formula is *well-named* if every variable is bound at most once in the formula. Obviously, every formula is equivalent to a well named one.

**Definition 44** If $X$ is bound in a well-named formula $\alpha$ then the *binding definition* of $X$ in $\alpha$ is the (unique) fixpoint formula of the form $\sigma X.\gamma(X)$. The *definition list* for $\alpha$ is the function $D_\alpha$ assigning to each fixpoint variable in $\alpha$ its binding definition. A variable $X$ is called *$\mu$-variable* if $D_\alpha(X)$ is a $\mu$-formula; similarly we define $\nu$-variables. Let $\lessdot_\alpha$ be a binary relation on variables bound in $\alpha$ defined by $X \lessdot_\alpha Y$ iff $X$ occurs free in $D_\alpha(Y)$.

It is easy to check that the transitive closure of $\lessdot_\alpha$ is a partial order. This allows us to formulate the following definition.

**Definition 45** A *dependency order* for a well named formula $\alpha$ is a linear order $\leq_\alpha$ that extends $\lessdot_\alpha$

**Definition 46** A *closure* of a formula $\alpha$, denoted $cl(\alpha)$, is the smallest set of formulas containing $\alpha$ and closed under taking subformulas.

With these definitions we can describe a construction of an alternating parity automaton over $\omega$-words for a given positive and well-named formula $\alpha$. This construction works for $M_B^H(G)$ representations of traces, i.e., when the modalities in $\alpha$ are interpreted as edges in Hasse diagram of a trace.

$$\mathcal{A}(\alpha) = \langle Q, \Sigma, q_0 \in Q, \delta : Q \times (\Sigma \cup \{\varepsilon\}) \to \mathcal{P}(\mathcal{P}(Q)), \Omega : Q \to \mathbb{N} \rangle$$

where we define the components below. We put $Q = cl(\alpha) \cup (cl(\alpha) \times \Sigma \times \mathcal{P}(\Sigma))$; with the second summand needed for checking formulas of the form $\langle\cdot\rangle\gamma$. The initial state $q_0$ is $\alpha$. For the acceptance condition $\Omega$ we put

$$\Omega(q) = \begin{cases} 2i & q = X \text{ is } i\text{-th in } \leq_\alpha \text{ ordering and } X \text{ is a } \nu\text{-variable} \\ 2i+1 & q = X \text{ is } i\text{-th in } \leq_\alpha \text{ ordering and } X \text{ is a } \mu\text{-variable} \\ 2m+3 & q \text{ is of the form } (\langle\cdot\rangle\gamma, a, S) \\ 2m+4 & q \text{ is of the form } ([\cdot]\gamma, a, S) \\ 2m+5 & \text{otherwise} \qquad (\text{where } m \text{ is the length of } \leq_\alpha) \end{cases}$$

Finally, we need to define the transition function. Notice that we allow $\varepsilon$-moves in the automaton. For readability we will represent an element $Z \in \mathcal{P}(\mathcal{P}(Q))$ by a DNF formula: $\bigvee_{Y \in Z} \left( \bigwedge_{q \in Y} q \right)$. So if $Z$ is for example $\{\{q_1, q_2\}, \{q_3\}\}$ we get $(q_1 \wedge q_2) \vee q_3$. To be consistent with this convention we also write *true* for $\{\emptyset\}$ and *false* for $\emptyset$.

- $\delta(P_a, a) = \mathit{true}$ and $\delta(P_a, b) = \mathit{false}$ for $b \neq a$;
- $\delta(X, \varepsilon) = D_\alpha(X)$;
- $\delta(\alpha \wedge \beta, \varepsilon) = \alpha \wedge \beta$ and $\delta(\alpha \vee \beta, \varepsilon) = \alpha \vee \beta$;
- $\delta(\sigma X.\gamma, \varepsilon) = \gamma$;
- $\delta(\langle\cdot\rangle\gamma, a) = (\langle\cdot\rangle\gamma, a, \emptyset)$ and $\delta([\cdot]\gamma, a) = ([\cdot]\gamma, a, \emptyset)$
- $\delta((\langle\cdot\rangle\gamma, a, S), b) = \begin{cases} (\langle\cdot\rangle\gamma, a, S \cup \{b\}) & \text{if } bDS \\ (\langle\cdot\rangle\gamma, a, S) & \text{if } \neg[bD(S \cup \{a\})] \\ \gamma \vee (\langle\cdot\rangle\gamma, a, S \cup \{b\}) & \text{if } (a,b) \in D \text{ and } \neg[bDS] \end{cases}$
- $\delta(([\cdot]\gamma, a, S), b) = \begin{cases} ([\cdot]\gamma, a, S \cup \{b\}) & \text{if } bDS \\ ([\cdot]\gamma, a, S) & \text{if } \neg[bD(S \cup \{a\})] \\ \gamma \wedge ([\cdot]\gamma, a, S \cup \{b\}) & \text{if } (a,b) \in D \text{ and } \neg[bDS] \end{cases}$

In the above, for a set $S \subseteq \Sigma$ we write $bDS$ to mean that $(b, c) \in D$ for some $c \in S$.

The definition of a run of such an automaton is standard (cf. [7]). In particular a run is a tree labelled with pairs consisting of a position in $w$ and a state of $\mathcal{A}$. A run of $\mathcal{A}$ is *accepting* if on every path $P$ of it the number $\min\{\Omega(q) : q \text{ appears infinitely often on } P\}$ is even.

Most of the cases of the definition of transition function are standard. The interesting part happens for formulas of the from $\langle\cdot\rangle\gamma$ or $[\cdot]\gamma$. Suppose we want to check $\langle\cdot\rangle\gamma$ from a position $i$ of the word. In state $\langle\cdot\rangle\gamma$ on letter $a = w(i)$ there is only one transition and it leads to the state $(\langle\cdot\rangle\gamma, a, \emptyset)$. If in a position $j > i$ the automaton is still in a state of the form $(\langle\cdot\rangle\gamma, a, S)$ for some $S$ then $S = \{\lambda(e^w(k)) : R(e^w(i), e^w(k))\ k = i, \ldots, j\}$; in words $S$ contains labels of those events represented by positions $i, \ldots, j$ of $w$ which are after (in the trace ordering) the event represented by position $i$. When reading letter $w(j+1)$ we know that $e^w(j+1)$ is a successor of $e^w(i)$ iff $w(j+1)$ depends on $a$ and is independent on all the letters in $S$. In this case $\mathcal{A}$ can start checking $\gamma$ or skip this successor. As the priority of states of the form $(\langle\cdot\rangle\gamma, a, S)$ is odd the automaton must finally decide to start checking $\gamma$ from some successor. The case for $[\cdot]\gamma$ is dual.

**Proposition 47** For every formula $\alpha$ of the $\mu$-calculus, every trace $G$ and every $w \in Lin(G)$: $M_B^H(G) \vDash \alpha$ iff $w \in L(\mathcal{A}(\alpha))$.

The proof of this proposition follows standard lines of other translations of the $\mu$-calculus to alternating automata [5, 10, 1].

The case of process diagram representation of traces is very similar. It is only necessary to change the definition of the transition function for formulas starting with modalities. In this case we have:

- $\delta(\langle\cdot\rangle\gamma, a) = (\langle\cdot\rangle\gamma, a, \emptyset)$ and $\delta([\cdot]\gamma, a) = ([\cdot]\gamma, a, \emptyset)$
- $\delta((\langle\cdot\rangle\gamma, a, S), b) = \begin{cases} \gamma \vee (\langle\cdot\rangle\gamma, a, S \cup \{b\}) & \text{if } (a,b) \in D \text{ and } b \notin S \\ (\langle\cdot\rangle\gamma, a, S) & \text{otherwise} \end{cases}$
- $\delta(([\cdot]\gamma, a, S), b) = \begin{cases} \gamma \wedge ([\cdot]\gamma, a, S \cup \{b\}) & \text{if } (a,b) \in D \text{ and } b \notin S \\ ([\cdot]\gamma, a, S) & \text{otherwise} \end{cases}$

The next step is to extend this construction to $\mu(Before)$ calculus. For a formula $\alpha \in \mu(Before)$ we want to construct an automaton $\mathcal{A}^b(\alpha)$ which accepts all words $w \in \Sigma^*$ such that $w \in Lin(G)$ and $G \vDash \alpha$. For this we extend the construction of $\mathcal{A}(\alpha)$ from above by adding new states:

$$\{Before^i_{ab}, NBefore^i_{ab} : i \in 0, 1, 2 \quad a, b \in \Sigma\}$$

We also add transitions which make the automaton accept from a state $Before^0_{ab}$ at position $i$ if $w(i)$ depends both on $a$ and $b$ and in the suffix of the word $w(i)w(i+1)\ldots$ the first $a$ appears before the first $b$. This is why we need states $Before^1_{ab}$ and $Before^2_{ab}$. State $Before^1_{ab}$ waits for the first $a$ and makes sure it comes before any $b$. State $Before^2_{ab}$ makes sure there is a $b$ in the sequence. From the state $NBefore^0_{ab}$ we accept the complement of the language accepted from $Before^0_{ab}$. It should be clear how to define transitions from these states. Finally we add transitions:

$$\delta(Before_{ab}, \varepsilon) = \delta(Before^0_{ab}) \qquad \delta(\neg Before_{ab}, \varepsilon) = \delta(NBefore^0_{ab})$$

Let us denote the obtained automaton by $\mathcal{A}^b(\alpha)$.

**Proposition 48** For every formula $\alpha$ of the $\mu(Before)$-calculus, every trace $G$ and every $w \in Lin(G)$: $M^H_B(G) \vDash \alpha$ iff $w \in L(\mathcal{A}^b(\alpha))$.

The final step is to consider $\mu(co)$ calculus, i.e., the $\mu$-calculus over $M^H_{co}(G)$ representations of traces. The construction of an automaton for this extension is not that straightforward. To check that $\langle co_a\rangle tt$ holds in some position we need to keep some information about what was already

read. This information comes in the form of past view. We assume that while reading a word $w$ in each position $j$ we calculate the binary relation:

$$C_j = \{(a,b) \in \Sigma^2 : (a,b) \in D \text{ and in } w[1,\ldots,j] \text{ the last } a \text{ is before the last } b\} \quad (3)$$

In other words $C_j$ is the set of pairs $(a,b) \in D$ such that the last $a$ appears before the last $b$ in the prefix of $G$ determined by the events $e^w(1),\ldots,e^w(j)$.

Given a $C \subseteq \Sigma^2$ and $b \in \Sigma$ we define $\mathit{Update}(C,b)$ to be the relation identical to $C$ on all the pairs not containing $b$ and containing:

$$\{a : (a,b) \in D \text{ and } a \text{ appears in } C\} \times \{b\} \cup \{(b,b)\}$$

Clearly there is a deterministic automaton $\mathcal{D}$ which states are subsets of $\Sigma \times \Sigma$ and such that after reading $j$-th letter from a word $w$ it reaches the state $C_j$. This automaton starts with the empty set as the initial state and uses *Update* operation on each letter it reads.

To extend our construction of alternating automata to handle $co_a$ propositions, we make the product of the previous automaton $\mathcal{A}$ with $\mathcal{D}$. Then we add to the set of states the set $\mathcal{P}(\Sigma) \times \Sigma$. Finally we add the transitions

- $\delta((C, co_a), b) = \mathit{true}$ if $a$ is incomparable with $b$ in $\mathit{Update}(C,b)$;
- $\delta((C, co_a), b) = (\{b\}, a)$ if $a$ is smaller than $b$ in $\mathit{Update}(C,a)$ or $a$ does not appear in $C$;
- $\delta((S,a),a) = \mathit{true}$ if $a$ depends on no letter from $S$;
- $\delta((S,a),a) = \mathit{false}$ if $a$ depends on some letter from $S$;
- $\delta((S,a),b) = (S',a)$ for $b \neq a$; where $S'$ is $S$ if $b$ does not depend on any of the letters from $S$ and $S'$ is $S \cup \{b\}$ otherwise.

Let us denote the obtained automaton by $\mathcal{A}^c(\alpha)$. The behaviour of $\mathcal{A}^c(\alpha)$ is such that after reading $j$-th letter from $w$ its first component is in a sate $C_j$ which is the relation as defined in (3). Being in a state $(C_j, co_a)$ and reading a letter $b$ at position $j+1$ the automaton can decide that there is $a$-event concurrent with $e^w(j+1)$ if $a$ is incomparable with $b$ in $\mathit{Update}(C_j,b)$. If it is not the case then the automaton enters a state $(\{b\},a)$. From this state it accumulates, in the first component, labels

of all the events after $e^w(j+1)$ in the trace. If, when reaching the first $a$, we have that $a$ is independent from all the letters accumulated in the first component then we know that it represents an event incomparable with $e^w(j+1)$.

**Theorem 49**
*For every formula $\alpha$ of the $\mu(co)$-calculus, every trace $G$ and every $w \in Lin(G)$: $M_{co}^H(G) \vDash \alpha$ iff $w \in L(\mathcal{A}^c(\alpha))$. The size of $\mathcal{A}^c(\alpha)$ is $\mathcal{O}(|\alpha| \times 2^{|\Sigma|^2})$. There is a nondeterministic automaton equivalent to $\mathcal{A}^c(\alpha)$ of size $2^{\mathcal{O}(|\Sigma|^2|\alpha|\log(|\alpha|))}$.*

The bound on the size of nondeterministic automaton is obtained by observing that one can glue parts of the states of alternating automaton that correspond to $\mathcal{D}$ automaton. One can also use $\mathcal{D}$ automaton to take care of $\langle\cdot\rangle\gamma$ and $[\cdot]\gamma$ formulas.

**Corollary 50** The satisfiability problem for the $\mu$-calculus over $M_B^P(G)$, $M_B^H(G)$ and $M_{co}^H(G)$ representations of traces is PSPACE-complete.

# 10 Comparison with $\nu$TrPTL

In this section we want to compare the logics form this paper with the logic $\nu$TrPTL introduced by Peter Niebert [8]. This was the first local and expressively complete logic for traces. Although it is also an extension of the $\mu$-calculus, it is quite different from the logics discussed above.

For the start, Niebert works with *distributed alphabets* instead of trace alphabets. Such an alphabet consist of a finite set of locations $Loc$ and a family of finite alphabets $(\Sigma_l)_{l\in Loc}$. Distributed alphabet determines a trace alphabet $(\Sigma, D)$ where $\Sigma = \bigcup_{l\in Loc} \Sigma_l$ and $(a,b) \in D$ iff $\{a,b\} \subseteq \Sigma_l$ for some $l \in Loc$. The first problem is that there is no canonical translation from a trace alphabet to a distributed alphabet. Given a trace alphabet $(\Sigma, D)$ one can take as a set of locations $Loc$ any set of cliques in $D$ which covers all pairs in $D$, i.e., for every $(a,b) \in D$ there should be clique $l \in Loc$ with $(a,b) \in l$. Then, for every $l$ one defines $\Sigma_l$ to be the set of letters appearing in $l$. It is easy to verify that the trace alphabet obtained from such $(\Sigma_l)_{l\in Loc}$ is exactly $(\Sigma, D)$ we have started from. As there can be many different coverings of $D$ with cliques, there can be many different distributed alphabets corresponding to the same trace alphabet. Unfortunately, the semantics of modality in $\nu$TrPTL is sensitive to the choice of distributed alphabet.

The other small difficulty is that in our logics we have many propositions and one modality and in $\nu$TrPTL there are many modalities but just one proposition $tt$. This is rather cosmetic difference. Let us consider $M_B^P(G)$ representations of traces and the fixpoint logic $\mu_{\langle a\rangle}$ with the following syntax:

$$X \mid tt \mid \mathit{Before}_{ab} \mid \neg\alpha \mid \alpha \vee \beta \mid \langle a\rangle\alpha \mid \mu X.\alpha(X)$$

where in the last construction $X$ is required to appear only positively in $\alpha(X)$.

The meaning of a formula $\alpha$ in a $M_B^P(G) = \langle E, R^P, \lambda\rangle$ representation of a trace is a set of events defined by the standard set of clauses plus:

- $e \vDash \mathit{Before}_{ab}$ iff $\mathit{Before}_{ab}(e)$ holds
- $e \vDash \langle a\rangle\alpha$ iff there is an event $e'$ labelled with $a$, s.t., $R^P(e, e')$, $\lambda(e') = a$ and $e' \vDash \alpha$.

The important fact here is that the modality is interpreted using arrows in a process diagram representation of a trace. So the meaning of $e \vDash \langle a\rangle\alpha$ is that in the first $a$-event after $e$ the formula $\alpha$ holds. The following fact says that this logic is expressively complete.

**Fact 51** There is a translation of the $\mu$-calculus over $M_B^P(G)$ representations of traces into $\mu_{\langle a\rangle}$.

**Proof**
Recall that in the $\mu$-calculus over $M_B^P(G)$ representations we have just one modality $\langle\cdot\rangle\alpha$ and a proposition $P_{a,S}$ for every $a \in \Sigma$ and $S \subseteq \Sigma \times \Sigma$. Such a proposition holds in a event $e$ iff $\lambda(e) = a$ and $S = \{(a, b) : \mathit{Before}_{ab}(e) \text{ holds}\}$.

Proposition $\mathit{Before}_{ab}$ from $\mu_{\langle a\rangle}$ can be defined by a disjunction of $P_{a,S}$ propositions. Similarly for propositions $P_a$ saying that the label of an event is $a$. Clearly, having $\mathit{Before}_{ab}$ and $P_a$ propositions is equivalent to having $P_{a,S}$ propositions.

To see that we can use $\langle a\rangle$ modalities instead of $P_a$ propositions observe that $\langle\cdot\rangle\alpha$ is equivalent to $\langle\cdot\rangle \bigvee_{a\in\Sigma}(P_a \wedge \alpha)$ which is equivalent to $\bigvee_{a\in\Sigma}\langle a\rangle\alpha$. Then all propositions in $\alpha$ that are not guarded by some modality can be replaced by conjunctions of $\mathit{Before}_{ab}$ propositions, or by either $tt$ or $\neg tt$. □

The expressive completeness proof for the $\mu$-calculus over $M_B^P(G)$ representations allows us to restrict the logic $\mu_{\langle a\rangle}$ even further. Instead of

predicates $Before_{ab}$ we can consider predicates $Before'_{ab}$ with the semantics:

$e \vDash Before'_{ab}$ iff $Before_{ab}(e)$ holds and $\lambda(e) \in \{a, b\}$

This predicate allows to check the ordering between the next even labelled by the current letter and the next event labelled by some other dependent letter. Additionally to $Before'_{ab}$ propositions we need $Start_{ab}$ predicates with the semantics:

$e \vDash Start_{ab}$ iff $e$ is the lest event in the trace and $Before_{ab}(e)$ holds

So $Start_{ab}$ can hold only in the least event of the trace and in this event it is equivalent to $Before_{ab}$. Essentially we have restricted the use of $Before_{ab}$ predicate for all but the least events. Let us call the obtained logic $\mu^{-}_{\langle a \rangle}$.

**Fact 52** The logic $\mu^{-}_{\langle a \rangle}$ over $M^P_B(G)$ representations of traces is expressively complete.

This fact follows from the inspection of the proof of the expressive completeness of the $\mu$-calculus over $M^P_B(G)$ representations. The only place were we use $Before_{ab}$ information is in the translation from MSOL formulas to the $\mu$-calculus is in the construction of $\mathcal{C}$ automaton. In that construction the use of $Before_{ab}$ is limited exactly to the cases we have in $\mu^{-}_{\langle a \rangle}$ logic.

After this preparation we can go back to $\nu$TrPTL. The semantics of this logic is defined over configurations. Then a type system is introduced and it is shown that all typeable formulas can be given local semantics, i.e., can be evaluated in events. Here we will try to shortcut this path, hopefully getting definitions equivalent to original ones.

Given a distributed alphabet $(\Sigma_l)_{l \in Loc}$ and a trace $G = \langle E, R, \lambda \rangle$ consider the relation $\xrightarrow{a}_l$ is defined by:

$e \xrightarrow{a}_l e'$ if $a, \lambda(e) \in \Sigma_l$, $\lambda(e') = a$ and $e'$ is the first $\Sigma_l$-labelled event after $e$, i.e., $\forall e''.\ (\lambda(e'') \in \Sigma_l \wedge R(e, e'')) \Rightarrow R(e', e'')$.

Consider the set of $\mu$-calculus formulas given by the grammar:

$$X \mid tt \mid \neg\alpha \mid \alpha \vee \beta \mid \langle a \rangle_l \alpha \mid \mu X.\alpha(X)$$

The formulas of $\nu\text{TrPTL}^{con}$ are the formulas generated by the above grammar which are typeable. The full logic $\nu$TrPTL has modalities $\langle a \rangle_L$

for $L$ a set of locations. We will not need them here. It seems that the use of type system can be avoided if the logic is interpreted on events and not on configurations. So instead of dealing with the original definition of $\nu$TrPTL$^{con}$ we just take the logic given by the above grammar and the usual semantics of the $\mu$-calculus were modalities are interpreted as:

$$e \vDash \langle a \rangle_l \alpha \text{ iff there is } e' \text{ with } e \xrightarrow{a}_l e' \text{ and } e' \vDash \alpha.$$

This is not an expressively complete logic as it cannot distinguish between the two traces shown in Figure 3. In this example the distributed alphabet consists of alphabets: $\{a, b\}$, $\{b, c\}$, $\{\bot, a\}$, $\{\bot, b\}$, $\{\bot, c\}$. The problem is that for this alphabet we have $\bot \not\vDash \langle b \rangle_{b,c} tt$ and $\bot \not\vDash \langle c \rangle_{b,c} tt$. Hence we cannot tell whether $b$ or $c$ comes first.

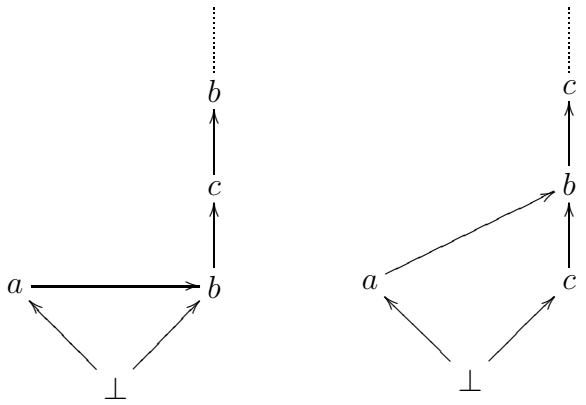


Figure 3: Indistinguishable traces

Fortunately, the original definition of the logic is such that, apart from the formulas we have defined, it also gives the ability to check the order between the first occurrences of actions. (This is one of the advantages of defining the logic over configurations). So we can add to the logic $Start_{a,b}$ propositions for every pair of dependent letters $(a, b) \in D$. Such a proposition is true only in the least event of a trace and only when the first $a$ event in the trace is before the first $b$ event. Hence these are the same propositions as in $\mu^{-}_{\langle a \rangle}$ logic.

Let us call the obtained logic $\mu$TrPTL$^{con}$. This is the $\mu$-calculus with $\langle a \rangle_l$ modalities and $Start_{a,b}$ propositions with the semantics described above. It seems that this logic is directly translatable into $\nu$TrPTL$^{con}$. What we can show is that $\mu$TrPTL$^{con}$ is expressively complete for every

distributed alphabet. The expressive completeness of $\nu$TrPTL$^{con}$ was left open in [8].

**Fact 53** For every distributed alphabet $(\Sigma_l)_{l\in Loc}$ the logic $\mu$TrPTL$^{con}$ is expressively complete.

**Proof**
We show a translation of $\mu^-_{\langle a\rangle}$ over $M_B^P(G)$ representations into $\mu$TrPTL$^{con}$. The first observation is that if $\Sigma_l = \{a, b\}$ then for every event $e$ we have $e \vDash \mathit{Before}'_{ab}$ iff $e \vDash \langle a\rangle_l tt$. The translation is more complicated if we just have $\{a, b\} \subseteq \Sigma_l$. In this case $e \vDash \mathit{Before}'_{ab}$ iff $e \vDash \mu X.\langle a\rangle_l tt \vee \bigvee_{c\in\Sigma_l, c\neq b}\langle c\rangle_l X$ (intuitively going through events of $\Sigma_l$ we meet $a$ before $b$). This solves the case of $\mathit{Before}'_{ab}$ propositions of $\mu^-_{\langle a\rangle}$.

It remains to deal with modalities of $\mu^-_{\langle a\rangle}$. Let $\alpha$ be a $\mu^-_{\langle a\rangle}$ formula and let $\alpha^{con}$ be an equivalent $\mu$TrPTL$^{con}$ formula. Take an event $e$ and a location $l$ such that $a \in \Sigma_l$. For every event $e$ with $\lambda(e) \in \Sigma_l$ we have: $e \vDash \langle a\rangle\alpha$ iff $e \vDash \mu X.\langle a\rangle_l\alpha^{con} \vee \bigvee_{c\in\Sigma_l, c\neq a}\langle c\rangle_l X$. The idea is that the first $a$ action after $e$ can be found by going through the actions in $\Sigma_l$. Let us denote this formula by $\gamma_l$. We cannot just use $\gamma_l$ in our translation because in $\mu$TrPTL$^{con}$ we have no means to check what is the label of the current event. Fortunately, if $\lambda(e) \notin \Sigma_l$ then $e \nvDash \gamma_l$. Hence the formula $\bigvee\{\gamma_l : a \in \Sigma_l\}$ is equivalent to $\langle a\rangle\alpha$ for all events. □

# References


[1] O. Bernholtz, M. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In D. L. Dill, editor, *Computer Aided Verification, Proc. 6th Int. Conference*, volume 818 of *Lecture Notes in Computer Science*, pages 142–155, Stanford, June 1994. Springer-Verlag, Berlin.

[2] V. Diekert and P. Gastin. An expressively complete temporal logic without past tense operators for mazurkiewicz traces. In *CSL*, volume 1683 of *LNCS*, pages 188–203, 1999.

[3] W. Ebinger. *Charakterisierung von Sprachklassen unendlicher Spuren durch Logiken*. PhD thesis, Institut f"ur Informatik, Universit"at Stuttgart, 1994.

[4] W. Ebinger. Logical definability of trace languages. In V. Diekert and G. Rozenberg, editors, *The Book of Traces*, pages 382–390. World Scientific, 1995.

[5] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *Proc. FOCS 91*, pages 368–377, 1991.

[6] D. Janin and I. Walukiewicz. Automata for the $\mu$-calculus and related results. In *MFCS '95*, volume 969 of *LNCS*, pages 552–562, 1995.

[7] D. Muller and P. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54:267–276, 1987.

[8] P. Niebert. *A Temporal Logic for the Specification and Verification of Distributed Behaviour*. PhD thesis, Universität Hildesheim, March 1998. Also available as *Informatik-Bericht Nr. 99-02,Institut für Software, Abteilung Programmierung, Technische Universität Braunschweig, Gaußstraße 11, D-38092 Braunschweig/Germany*.

[9] D. Niwiński. Fixed point characterization of infinite behaviour of finite state systems. *Theoretical Computer Science*, 189:1–69, 1997.

[10] C. S. Stirling. Modal and temporal logics. In S.Abramsky, D.Gabbay, and T.Maibaum, editors, *Handbook of Logic in Comuter Science*, pages 477–563. Oxford University Press, 1991.

[11] P. S. Thiagarajan and I. Walukiewicz. An expressively complete linear time temporal logic for Mazurkiewicz traces. In *LICS'97*, pages 183–194. IEEE, 1997.

[12] I. Walukiewicz. Difficult configurations – on the complexity of LTrL. In *ICALP '98*, volume 1443 of *LNCS*, pages 140–151, 1998.