Basic Research in Computer Science

BRICS RS-00-2 I. Walukiewicz: Local Logics for Traces

Local Logics for Traces

Igor Walukiewicz

BRICS Report Series **RS-00-2**

ISSN 0909-0878 **January 2000**

Copyright © 2000, Igor Walukiewicz.
BRICS, Department of Computer Science
University of Aarhus. All rights reserved.

Reproduction of all or part of this work
is permitted for educational or research use
on condition that this copyright notice is
included in any copy.

See back inner page for a list of recent BRICS Report Series publications.
Copies may be obtained by contacting:

BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark
Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `RS/00/2/`

Local logics for traces

Igor Walukiewicz

BRICS*

Abstract

A μ-calculus over dependence graph representation of traces is considered. It is shown that the μ-calculus cannot express all monadic second order (MSO) properties of dependence graphs. Several extensions of the μ-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 the formula in a *global logic* is evaluated in a configuration. Global logics (as for example the one in [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 [11]. 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 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 mu-calculus is equivalent to MSOL over binary trees but it is weaker than MSOL over all labelled graphs. It turns out that the μ-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 μ-calculus. The one which is easiest to describe here is obtained by adding co_a propositions. Such a proposition holds in a event e if there is in a trace an even incomparable with e which is labelled by a.

A 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 is the one proposed by Niebert [8]. This is an extension of the μ-calculus which captures the power of MSOL. Unfortunately the syntax of the logic is rather heavy. The proof that 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 μ-calculus over labelled graphs. We also recall results linking MSOL with the μ-calculus and an automata characterisation of the later logic.

In Section 3 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 μ-calculus and MSOL over trees to obtain an extension of the μ-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 4 we consider some other extensions of the μ-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 19) 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 5 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 6 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 7 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.

2 Preliminaries

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

We shall view (Mazurkiewicz) trace as a special Σ-labelled acyclic graph. Let $\langle E, R, \lambda\rangle$ be a Σ-labelled graph. In other words, (E, R) is a graph and $\lambda : E \to \Sigma$ is a labelling function. A graph is acyclic if the reflexive and transitive closure R^* of R is a partial order.

Definition 1 A *trace* or *dependence graph* over alphabet (Σ, D) is a Σ-labelled acyclic graph $G = \langle E, R, \lambda\rangle$ satisfying:

(T1) $\forall e \in E.$ $\{e' : R^*(e', e)\}$ is a finite set.
(T2) $\forall e, e' \in E.$ $R(e, e') \Rightarrow (\lambda(e), \lambda(e')) \in D.$
(T3) $\forall e, e' \in E.$ $(\lambda(e), \lambda(e')) \in D \Rightarrow R^*(e, e') \vee R^*(e', e).$
(T4) $\forall e, e' \in e.$ $R(e, e') \Rightarrow \neg\exists e''.\ R(e, e'') \wedge R(e'', e').$

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. We say that e' is a *successor* of e if $R(e, e')$ holds.

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 R relates only events labelled by dependent letters. The third, says that every two events labelled by dependent letters are related by R^*. The last condition requires that R has no redundancies. In the literature sometimes a trace is defined in terms of R^* relation and R relation is obtained by taking a Hasse diagram of R^* (cf. [5]).

Proviso: In the whole paper we fix a trace alphabet (Σ, D) and a linear order $<_\Sigma$ on Σ. We also assume that we have a special letter $\perp \in \Sigma$ such that $\{\perp\} \times \Sigma \subseteq D$. 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.

In the sequel we will also need other representations of traces than dependence graphs. Because of this the following definitions of logics are formulated more generally for Γ-labelled graphs where Γ is an arbitrary alphabet.

First, we define *monadic second logic* suitable to talk about Γ-labelled graphs. The signature of the logic contains 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 Γ, and α, β over formulas.

Given a Γ-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 φ 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 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.

Next we define the *μ-calculus* over an alphabet Γ. 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 Γ, and α, β over formulas. In the construction $\mu X.\alpha$ we require that X appears only positively in α (i.e., under even number of negations).

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

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

The omitted clauses for boolean constructors are standard. We write $M, V, s \vDash \alpha$ if $s \in [\![\alpha]\!]^M_V$. If G is a trace which has the least event $\perp$ then we write $G \vDash \alpha$ to mean that $G, V, \perp \vDash \alpha$ for all V. A μ-calculus formula defines a set of traces $\{G : G \vDash \alpha\}$.

A Γ-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 2
Over deterministic trees μ-calculus is equivalent to MSOL. In other words, for every MSOL sentence φ there is a μ-calculus sentence α_φ such that: $M \vDash \varphi$ iff $M \vDash \alpha_\varphi$. Also conversely, for every μ-calculus sentence α there is an MSOL sentence φ_α such that: $M \vDash \alpha$ iff $M \vDash \varphi_\alpha$.

Clearly μ-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 μ-calculus. Dependence graphs are deterministic acyclic graphs but usually they are not trees. Later we will see that μ-calculus is weaker than MSOL over dependence graphs.

Next we recall (from [6]) a characterisation of the μ-calculus in terms of (alternating) automata. This will allow us to use automata instead of μ-calculus which is easier for some constructions.

Definition 3 A *μ-automaton* over an alphabet Γ is a tuple

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

where: Q is a finite set of states, Γ 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 Γ-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 (q', e') for some $q' \in W$;
- for every $q' \in W$ there is a son of v labelled by (q', e') for some successor e' of e.

An infinite sequence $(e_0, q_0), (e_1, q_1), \ldots$ satisfies a *parity condition given by* Ω 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 $L(\mathcal{A}) = \{G : \mathcal{A} \text{ accepts } G \text{ from } \bot\}$.

Theorem 4
μ-automata over Γ are equivalent to the μ-calculus over Γ. In other words, for every automaton $\mathcal{A}$ there is a μ-calculus sentence $\alpha_{\mathcal{A}}$ such that $\{G : G \vDash \alpha_{\mathcal{A}}\} = L(\mathcal{A})$; and conversely; for every μ-calculus sentence α there is an automaton $\mathcal{A}_\alpha$ such that $\{G : G \vDash \alpha\} = L(\mathcal{A}_\alpha)$.

Finally we will need a tool for defining one labelled graph inside another one by means of MSOL formulas. 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^ξ as an edge relation. We will use the following straightforward observation.

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

Proof
Just replace every occurrence of R in φ by $\xi(x, y)$. □

3 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 μ-calculus over trees.

Definition 6 (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 Σ^* obtained from our fixed ordering $<_\Sigma$ on Σ. 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 Σ-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-tree gives a natural ordering on sons of a node which then can be extended to an ordering between any two nodes of lex tree which are not on the same path.

Definition 7 ("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 8 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 9 There is a MSOL formula ξ defining $Lex(G)$ in G (i.e., G^ξ is isomorphic to $Lex(G)$).

Proof
We write a formula $\xi(x, y)$ such that $G \vDash \xi(e, e')$ iff $lexp(e') = lexp(e)e'$.
□

Lemma 10 There is a MSOL formula ξ^{-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 $H \subseteq R^*$. On the other hand we have $R \subseteq H$ but there may be no equality as H may contain some redundancies. Anyway, we have that the reflexive and transitive closure H^* of H is exactly R^*.

Consider $H' \equiv H^*(x, y) \wedge \neg\exists z.H^*(x, z) \wedge H^*(z, y)$. We claim that $H' = R$. By definition of a trace we have that $R \subseteq H'$. To show that $H' \subseteq R$ assume conversely and take $(e, e') \in H' \setminus R$. Then $R^*(e, e')$ holds. If not $R(e, e')$ then there is an event e'' labelled by a letter dependent on $\lambda(e)$ and satisfying $R^*(e, e'') \wedge R^*(e'', e)$. Then $H^*(e, e'')$ and $H^*(e'', e')$ hold. A contradiction. □

Using Proposition 5 we immediately obtain.

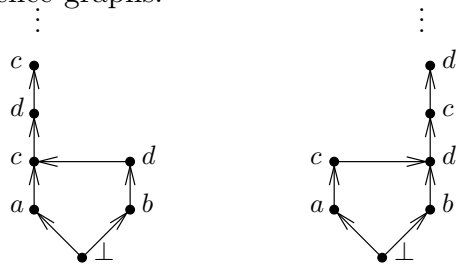
Corollary 11 MSOL over dependence graphs and lex-trees has the same expressive power. More precisely, for every MSOL formula φ there is a MSOL formula φ^Tsuch that for every dependence graph G we have: $G \vDash \varphi$ iff $Lex(G) \vDash \varphi^T$. Vice versa, for every MSOL formula ψ there is a MSOL formula ψ^G such that for every graph G we have: $G \vDash \psi^G$ iff $Lex(G) \vDash \psi$.

Corollary 12 MSOL over dependence graphs is equivalent to μ-calculus over lex-trees. More precisely, for every MSOL formula φ there is a μ-calculus formula α such that for every dependence graph G we have: $G \vDash \varphi$ iff $Lex(G) \vDash \alpha$. Moreover for every μ-calculus formula α there is a MSOL formula φ_α such that for every dependence graph G we have: $G \vDash \varphi_\alpha$ iff $Lex(G) \vDash \alpha$.

4 Extended mu-calculi for traces

We are now going to introduce two new representations of dependence graphs as labelled graphs. Before doing this we show that μ-calculus is weaker than MSOL on traces when traces are represented as dependence graphs.

Proposition 13 No μ-calculus sentence can distinguish between the following two dependence graphs:



In the left graph the dots stand for the sequence $(dc)^\omega$ and in the right graph for $(cd)^\omega$. These two dependence graphs are over trace alphabet $(\{\bot, a, b, c, d\}, D)$ where D is the smallest symmetric and reflexive relation containing $\{(a, c), (b, d), (c, d)\} \cup \{\bot\} \times \{a, b, c, d\}$

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

The above proposition shows that we need to have more information in order to recover the structure of a dependence graph. We will do this by introducing more information into labels of events. First we define some auxiliary relations.

Definition 14 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 15 Let $G = \langle E, R, \lambda \rangle$ be a dependence graph. We define its two representations $M_{co}(G)$ 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 $\lambda_B(e) = (\lambda(e), \{(a, b) : \mathit{Before}_{ab}(e)\})$.

Our goal is to show that the μ-calculus over $M_{co}(G)$ as well as over $M_B(G)$ is expressively complete. The first observation is that $M_{co}(G)$ representation gives at least as much information about G as $M_B(G)$.

Proposition 16 For every pair $(a, b) \in D$ there is a μ-calculus formula β_{ab} defining Before_{ab} in $M_{co}(G)$; more formally for every G and e in G we have: $M_{co}(G), e \vDash \beta_{ab}$ iff $\mathit{Before}_{ab}(e)$ 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 $\mathit{Before}_{ab}(e)$ to hold. The above conditions are expressed by the formula:

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

where Σ_{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 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. □

Our goal can be formulated as follows.

Theorem 17
For every MSOL sentence φ there is a μ-calculus sentence β_φ such that for every dependence graph G: $G \vDash \varphi$ iff $M_B(G), \bot \vDash \beta_\varphi$.

Proof
The line of the proof is as follows. By Corollary 12 from φ we construct a μ-calculus formula α_φ such that $G \vDash \varphi$ iff $Lex(G), \bot \vDash \alpha_\varphi$. By Theorem 4 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 $M_B(G)$. 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 $M_B(G)$ and accepting iff $\mathcal{A}_\varphi$ accepts $Lex(G)$. Using once again Theorem 4, automaton $\mathcal{B}$ can be translated back to a μ-calculus formula β_φ.

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. Please observe that $M_B(G)$ is a Γ-labelled graph 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 18 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 $M_B(G)$,
- 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 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 Σ. 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 claim that for every dependence graph G:

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

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 $M_B(G)$. 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 $(\perp_a, \perp_c)$ where $\perp_a$, $\perp_c$ are the roots of S_a and S_c respectively. We put $r_b(\perp_a, \perp_c) = (\perp, (q_a^0, q_c^0))$, where $\perp$ 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 $v^i = (v^i_a, v^i_c)$ for all i. Recall that $r_b(v^i)\downarrow_2 = (r_a(v^i_a)\downarrow_2, r_a(v^i_c)\downarrow_2)$. By definition of Ω_b we have that $\Omega_b(r_a(v^i_a)\downarrow_2, r_a(v^i_c)\downarrow_2) = \Omega_a(r_a(v^i_a)\downarrow_2)$. Hence $v^0, v^1, \ldots$ satisfies the parity condition Ω_b because by the assumption $v^0_a, v^1_a, \ldots$ satisfies the parity condition Ω_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 Ω_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 $\mu(Before)$ stand for the extension of the μ-calculus over the alphabet Σ with propositions $Before_{ab}$ for every $(a, b) \in D$. The meaning of such a proposition is: $G, e \vDash Before_{ab}$ iff $Before_{ab}(e)$ holds in G. It is straightforward to see that $\mu(Before)$ over dependence graph representation of traces is equivalent to the plain μ-calculus over the alphabet $\Gamma_B = \Sigma \times \mathcal{P}(\Sigma \times \Sigma)$ and $M_B(G)$ representation of traces.

Similarly let $\mu(co)$ stand for the extension of the μ-calculus over the alphabet Σ 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 μ-calculus over $M_{co}(G)$ representations of traces.

Corollary 19 For every formula φ of MSOL there are equivalent formulas α_φ and β_φ of $\mu(co)$ and $\mu(Before)$ calculi, i.e., formulas such that for every dependence graph G: $G \vDash \varphi$ iff $G \vDash \alpha_\varphi$ iff $G \vDash \beta_\varphi$. For every formula of $\mu(co)$ or $\mu(Before)$ μ-calculus there is an equivalent formula of MSOL.

5 Local characterisation of lex-trees

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 20 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 21 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 Σ. 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 22 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 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 23 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 24 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 $Left(V, a) \subseteq \Sigma$ 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 25 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 $Before_{a\lambda(e_k)}(e)$ holds and we put $(\lambda(e), a)$ in $V(e)\downarrow_e$ if $Before_{\lambda(e_k)a}(e)$ 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. This will be our intended labelling of the events.

Definition 26 (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 fist 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 27 The minimal elements of $P(e)$ are exactly the labels of the minimal events in $G(e)$, which are e and all left splits of e.

Proof
First, we want to show that if e' is a left split of e then e' is minimal in $G(e)$. 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' contradicting 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 $e^{(3)}$

So if e' is a left split from e then $\lambda(e')$ is minimal in $P(e)$. If a is minimal in $P(e)$ then in $G(e)$ there is no event above the first event labelled a. Hence the first a-event must be e or the left split from e. □

Lemma 28 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
By assumption there is a minimal element in $c \neq \lambda(e)$ in $P(e)$ with a path from c to b in $P(e)$. By induction on the length of this path we show that there is a path in $G(e)$ from the first event labelled e_c to the first event labelled e_b. Then to get the statement of the lemma observe that e_c must be a left split of e by Lemma 27. □

Lemma 29 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 a dependent on $\lambda(e)$. Hence there is the unique successor e' of e labelled by $\lambda(e') = \lambda(e)$. This successor is a lex-son because every path to e' goes through e.

For the other case suppose $b \in \mathit{Min}(P(e)\downarrow_{\lambda(e)}) \setminus (\mathit{Min}(P(e)) \setminus \{e\})$ 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 $Min(P(e)\downarrow_e) \setminus (Min(P(e)) \setminus \{\lambda(e)\})$. This is because whenever e'' is a lex-son of e then $Before_{\lambda(e'')a}(e)$ holds for all a dependent on $\lambda(e'')$. So $\lambda(e'') \in Min(P(e)\downarrow_e)$ and $\lambda(e'') \notin Min(P(e))$. □

Definition 30 *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 Σ 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 31 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 29 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 29 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 Σ. By Lemma 29 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 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 $Left(V(e_i), \lambda(e_i))$. 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 28 for every letter $a \in Left(P(e_i), \lambda(e_i))$ 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 32 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 33 We say that an event e is *good* if $V(e) = P(e)$ (it does not matter what $V_L(e)$ is)

Lemma 34 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 $Before_{ab}(e)$ predicates. □

Lemma 35 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 $Lex(G)$ 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 (4c) 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 $P = e'_1 e'_2 \dots$ 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 $\mathit{Right}(e'_1), \mathit{Right}(e'_2), \dots$ is not increasing. Hence it stabilizes on some set Inf.

Let e' be an event on P with $\mathit{Right}(e') = \mathit{Inf}$. We claim that for every letter $a \in \mathit{Inf}$ the first a event in $G(e')$ is a descendant of e' in $\mathit{Lex}(G)$. Suppose conversely that e_a is to the left. As $a \in \mathit{Inf}$ 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 $\mathit{Right}(e')$ can appear in $G(e_{i-1})$. By definition of consistent views $V(e')$ is consistent with $V_L(e')$ on $\mathit{Left}(e')$. Moreover, as no more event is going to get to the left, $V(e_{i-1})$ is just $V(e')$ restricted to $\mathit{Left}(e')$. □

Now we are ready to prove Proposition 32

Proof (of Proposition 32)
By definition, the root event is good. Assume that in tree $\mathit{Lex}(G)$ 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 29 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 34. 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 35. A contradiction. □

6 Automaton reconstructing lex-trees

Recall that *Views* is the set of views over the alphabet Σ (cf. Definition 23). 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 Σ. 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 we have a trace G and an event e with $V = P(e)$, $a = \lambda(e)$ and $B = \{(b, c) : \mathit{Before}_{b,c}(e)\}$. In this case $V{\downarrow_{(B,a)}}$ is $P(e){\downarrow_e}$.

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

- $Q = (\Sigma \times \mathit{Views} \times \mathit{Views}) \cup \{q^0, tt\}$,
- $\Gamma = \Sigma \times \mathcal{P}(\Sigma \times \Sigma)$,
- $\Omega(q) = 0$ 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 $\mathit{Min}(V{\downarrow_{(B,a)}}) = \mathit{Min}(V) \setminus \{a\}$ 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 δ we define $\delta((a, V_L, V), (b, B))$ by cases:

- if $\mathit{Min}(V{\downarrow_{(B,a)}}) = \mathit{Min}(V)$ then
  $\delta((a, V_L, V), (a, B)) = \{\{(a, V_L, V), tt\}\}$
- If $\mathit{Min}(V{\downarrow_{(B,a)}}) = (\mathit{Min}(V)\setminus\{a\})\cup\{b_1, \ldots, b_k\}$ (where $b_1, \ldots, b_k$ are $<_\Sigma$-ordered by our order on Σ) 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:
  - $V_k = V{\downarrow_{(B,a)}}$,
  - $\mathit{Alph}(V_i') \subseteq \mathit{Alph}(V_i)$ and V_i' agrees with V_i on $\mathit{Left}(V_i, b_i)$
  - $V_L = V_1'$ 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 30). 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 36
For every dependence graph G there is a unique accepting run of $\mathcal{C}$ on $M_B(G)$. The restriction of this run to nodes having states other than tt in their label is isomorphic to the lexicographic tree $Lex(G)$.

Proof
First, let us show that there is a run of $\mathcal{C}$ on a dependence graph G. Let S be a tree containing $Lex(G)$ and moreover for every $e \in Lex(G)$ and every successor f of e which is not a son of e in $Lex(G)$ let S contain a new node v_f^e which is a son 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 put a new node v_f^e which is a son of e into S. Define $r : S \to E \times Q$ by:

$$r(v) = \begin{cases} (\bot, q_0) & \text{if } v = \bot \\ (e, (\lambda(a), P_L(e), P(e)) & \text{for } v \in Lex(G) \setminus \{\bot\} \\ (f, tt) & \text{if } v = v_f^e \end{cases}$$

Here $P_L(e)$ is the projection from the leftmost split from e or it is $\emptyset$ if there is no such split. By Proposition 31 function r is a locally consistent run of $\mathcal{C}$ on G. As $\Omega(q) = 0$ for all states, every locally consistent run is accepting.

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 such that the state in the label 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 show that it is an isomorphism.

Lemma 37 Suppose v satisfies:

$$r(v) = (e, (\lambda(e), V_L, P(e))) \qquad \text{for some } V_L \tag{3}$$

then $r{\downarrow_1}$ is a bijection between sons of v and sons of e in $Lex(G)$.

Proof
From Lemma 31 we get that the sons of e in lex-tree are determined by $P(e)$. Similarly $P(e)$ and the label of e determine 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 $Lex(G)$:

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

Suppose we have an event e such that:

1. $(r\downarrow_1)^{-1}(e')$ is a singleton for every $e' \in AR(e)$. So we have a function $rr : AR(e) \to S$ which is the reverse of $r\downarrow_1$.
2. For every $AR(e')$ we have $rr(e')$ satisfies (3).
3. $r\downarrow_1$ is not an isomorphism between the subtree of $Lex(G)$ rooted in e and the subtree of S rooted in $rr(e)$

We will show that if we have such e then we can find a son of e in $Lex(G)$ with the same properties.

By the above lemma $r\downarrow_1$ is a bijection between the sons of e and the sons of $rr(e)$. Let us extend rr to these sons. Clearly clause 1 is satisfied for e_k. By Lemma 34 we have that e_k satisfies clause 2. If $r\downarrow_1$ is not a bijection between the subtrees rooted in e_k and $rr(e_k)$ then e_k is the son we were looking for. Otherwise e_{k-1} satisfies clause 1. From Lemma 35 we know that e_{k-1} satisfies clause 2. Continuing like this we must find a son e_i of e which satisfies all the clauses. Otherwise we would have that $r\downarrow_1$ is a bijection between descendants of e and descendants of $rr(e)$ which is impossible by clause 3 of our assumption.

Let us iterate this construction to infinity. Let $e'_1, e'_2, \ldots$ be events chosen in successive iterations of the construction. We have that these events form a path in $Lex(G)$. As every infinite path in $Lex(G)$ is eventually leftmost there is an event e'_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 e'_i and descendants of $rr(e'_i)$ which was assumed not to exist. This shows that e with the above properties cannot exist.

Take the least element $\perp$ of G which is also the root of S. We have $r(\perp) = (\perp, q_0)$. By definition of the automaton, its move from q_0 on the letter $(\perp, P(\perp))$ is exactly the same as from the state $(\lambda(\perp), \emptyset, P(\perp))$ on this letter. So we can pretend that the root of S is labelled with $(\perp, (\lambda(\perp), \emptyset, P(\perp)))$ and not $(\perp, q_0)$. But then $\perp$ satisfies the clauses 1–3 above. As this is impossible and clauses 1–2 hold we must have that 3 is not satisfied. So $r\downarrow_1$ is an isomorphism between S and E. $\square$

7 Complexity issues

In this section we will show that the model checking problem for the logics proposed in this paper is PSPACE-complete. For a given formula α we will construct an automaton $\mathcal{A}(\alpha)$ recognizing all linearisations of all the traces satisfying α.

Definition 38 A *linearization* of a trace $G = \langle E, R, \lambda \rangle$ is a word $w \in \Sigma^w$ 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. Let $Lin(G)$ denote the set of all linearizations of G.

If $w \in Lin(G)$ then it 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 μ-calculus over dependence graphs without additional information in the labels. Later we will extend the constructions to other μ-calculi.

A μ-calculus formula is *positive* if all the negations appear only before propositional constants. To have equivalent positive formula for every formula of the μ-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 μ-calculus is equivalent to a formula generated by the above grammar. We will use σ to denote either μ or ν. 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 39 If X is bound in a well-named formula α then the *binding definition* of X in α is the (unique) fixpoint formula of the form $\sigma X.\gamma(X)$. The *definition list* for α is the function D_α assigning to each fixpoint variable in α its binding definition. A variable X is called *μ-variable*

if $D_\alpha(X)$ is a μ-formula; similarly we define ν-variables. Let $\lessdot_\alpha$ be a binary relation on variables bound in α 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 40 A *dependency order* for a well named formula α is a linear order $\leq_\alpha$ that extends $\lessdot_\alpha$

Definition 41 A *closure* of a formula α, denoted $cl(\alpha)$, is the smallest set of formulas containing α and closed under taking subformulas.

With these definitions we can define an alternating parity automaton for a given positive and well-named formula α:

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

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

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

Finally we need to define the transition function. Please notice that we allow ε-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 which leads to a state $(\langle\cdot\rangle\gamma, a, \emptyset)$. For every position $j > i$ if 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 γ 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 γ from some successor. The case for $[\cdot]\gamma$ is dual.

Proposition 42 For every formula α of the μ-calculus, every trace G and every $w \in Lin(G)$: $G \vDash \alpha$ iff $w \in L(\mathcal{A}(\alpha))$.

The proof of this proposition follows standard lines of other translations of the μ-calculus to alternating automata [4, 10, 1].

The next step is to extend this construction to $\mu(Before)$ calculus. This is the μ-calculus over $M_B(G)$ representation of G. One can equivalently see the $\mu(Before)$ calculus as the extension of the above μ-calculus with propositions $Before_{ab}$ with the meaning: $G, e \vDash Before_{ab}$

iff $Before_{ab}(e)$ holds in G. 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 $M_B(G) \vDash \alpha$. 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. Let us denote the obtained automaton by $\mathcal{A}^b(\alpha)$.

Proposition 43 For every formula α of the $\mu(Before)$-calculus, every trace G and every $w \in Lin(G)$: $M_B(G) \vDash \alpha$ iff $w \in L(\mathcal{A}^b(\alpha))$.

The final step is to consider $\mu(co)$ calculus, i.e., the μ-calculus over $M_{co}(G)$ representations of traces. One can think of $\mu(co)$ as the extension of the plain μ-calculus with propositions co_a with the meaning: $G, e \vDash co_a$ iff there is $e' \in G$ such that $\lambda(e') = a$ and $co(e, e')$ holds in G. The construction of an automaton for this extension is not that straightforward. To check that co_a 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:

$$C_j = \{(a, b) \in \Sigma^2 : (a, b) \in D \text{ and in } w[1, \ldots, j] \text{ the last } a \text{ is before the last } b\} \quad (4)$$

The other characterisation of C_j is that it 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 $Update(C, b)$ to be the relation identical to C on all the pairs not containing b and containing:

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

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 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 (4). 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 44
For every formula α of the $\mu(co)$-calculus, every trace G and every $w \in Lin(G)$: $M_{co}(G) \vDash \alpha$ 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.

By Proposition 16, $\mu(Before)$ calculus can be translated to $\mu(co)$ calculus with only linear increase in the size of the formula. Using the standard technique of calculating states of an automaton on demand we obtain.

Corollary 45 The satisfiability problem for $\mu(Before)$ and $\mu(co)$ logics is PSPACE complete. More precisely given a formula α of $\mu(Before)$ or $\mu(co)$ logics one can decide in PSPACE if there is a trace G such that $M_B(G) \models \alpha$ or $M_{co}(G) \models \alpha$ respectively.

References

[1] O. Bernholtz, M. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. In *CAV*, volume 818 of *LNCS*, pages 142–155, 1994.

[2] V. Diekert and P. Gastain. An expressively complete temporal logic without past tense operators for mazurkiewicz traces. In *CSL'99*, volume 1683 of *LNCS*, pages 188–203, 1999.

[3] W. Ebinger. *Charakterisierung von Sprachklassen unendlicher Spuren durch Logiken*. PhD thesis, Institut für Informatik, Universität Stuttgart, 1994.

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

[5] P. Gastin and A. Petit. *The Book of Traces*, chapter Infninite Traces. World Scientific, 1995.

[6] D. Janin and I. Walukiewicz. Automata for the μ-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] I. Walukiewicz. Difficult configurations – on the complexity of LTrL. In *ICALP '98*, volume 1443 of *LNCS*, pages 140–151, 1998.

Recent BRICS Report Series Publications

RS-00-2 Igor Walukiewicz. *Local Logics for Traces*. January 2000. 30 pp.

RS-00-1 Rune B. Lyngsø and Christian N. S. Pedersen. *Pseudoknots in RNA Secondary Structures*. January 2000. 15 pp. To appear in *Fourth Annual International Conference on Computational Molecular Biology*, RECOMB '00 Proceedings, 2000.

RS-99-57 Peter D. Mosses. *A Modular SOS for ML Concurrency Primitives*. December 1999. 22 pp.

RS-99-56 Peter D. Mosses. *A Modular SOS for Action Notation*. December 1999. 39 pp. Full version of paper appearing in Mosses and Watt, editors, *Second International Workshop on Action Semantics*, AS '99 Proceedings, BRICS Notes Series NS-99-3, 1999, pages 131–142.

RS-99-55 Peter D. Mosses. *Logical Specification of Operational Semantics*. December 1999. 18 pp. Invited paper. Appears in Flum, Rodríguez-Artalejo and Mario, editors, *European Association for Computer Science Logic: 13th International Workshop*, CSL '99 Proceedings, LNCS 1683, 1999, pages 32–49.

RS-99-54 Peter D. Mosses. *Foundations of Modular SOS*. December 1999. 17 pp. Full version of paper appearing in Kutyłowski, Pacholski and Wierzbicki, editors, *Mathematical Foundations of Computer Science: 24th International Symposium*, MFCS '99 Proceedings, LNCS 1672, 1999, pages 70–80.

RS-99-53 Torsten K. Iversen, Kåre J. Kristoffersen, Kim G. Larsen, Morten Laursen, Rune G. Madsen, Steffen K. Mortensen, Paul Pettersson, and Chris B. Thomasen. *Model-Checking Real-Time Control Programs — Verifying LEGO Mindstorms Systems Using UPPAAL*. December 1999. 9 pp.

RS-99-52 Jesper G. Henriksen, Madhavan Mukund, K. Narayan Kumar, and P. S. Thiagarajan. *Towards a Theory of Regular MSC Languages*. December 1999.