BRICS RS-94-17 A. Cheng: Local Model Checking and Traces

MATEMATISK INSTITUT
AARHUS UNIVERSITET
TRYKKERIET

BRICS

Basic Research in Computer Science

Local Model Checking and Traces

Allan Cheng

BRICS Report Series RS-94-17

ISSN 0909-0878 June 1994

See back inner page for a list of recent publications in the BRICS Report Series. 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@daimi.aau.dk

Local Model Checking and Traces

Allan Cheng*
BRICS**
Computer Science Department, Aarhus University
Ny Munkegade, DK–8000 Aarhus C, Denmark
e-mail:acheng@daimi.aau.dk
Phone: +45 8942 3188 Fax: +45 8942 3255

June 13, 1994

Abstract

We present a *CTL*-like logic which is interpreted over labeled asynchronous transition systems. The interpretation reflects the desire to reason about these only with respect their progress fair behaviours. For finite systems we provide a set of tableau rules and prove soundness and completeness with respect to the given interpretation of our logic. We also provide a model checking algorithm which solves the model checking problem in deterministic polynomial time in the size of the formula and the labelled asynchronous transition system. We then consider different extensions of the logic with modalities expressing concurrent behaviour and investigate how this affects the decidability of the satisfiability problem.

1 Introduction

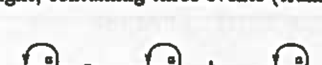
Recently attention has focused on behavioural views of concurrent systems in which concurrency or parallelism is represented explicitly [20, 11, 24, 22, 25]. This is is done by imposing more structure on models for concurrent systems, in our case an independence relation on the events.

Our main objective is to explore the use of the extra structure of independence in the context of *specification logics*. This paper introduces and studies a *CTL*-like logic, *P-CTL*, interpreted over *labelled asynchronous transition systems (l-ATS 's)* [21, 1, 25, 16].

l-ATS 's are generalizations of the standard labelled transition systems (*lts*'s) with extra structure specifying concurrent activity explicitly. Relations similar to those between labelled transition systems, nets, and process algebras exists between *l-ATS* 's, nets, and process algebras, see e.g. [14, 25, 15], and they provide a natural way of understanding other formalisms. Hence the choice of specification formalism is not too important, e.g. we could have used labelled 1-safe nets. The axioms of *l-ATS* 's are trivially satisfied for the case graph of labelled 1-safe nets, where the independence relation on events/transitions is interpreted as the disjointness of pre- and post-conditions.

*This work has been supported by The Danish Research Councils.
**Basic Research in Computer Science, Centre of the Danish National Research Foundation.

As an example let us consider the process agent $fix\,(X = a.X)|(\tau.b.0)$. Its transition graph is given below to the left. The agent can also be represented by the 1-safe net to the right, containing three events (transitions) labelled a, τ, and b respectively [14, 25].



The net gives us a more concrete model of the process agent. The transitions explicitly show that the event labelled a is independent of those labelled τ and b. We can therefore add *more structure* to the above transition system by providing a relation which explicitly states this *independence*. The new transition system is an example of a labelled asynchronous transition system.

P-CTL contains one important feature which is the model-theoretic incorporation of progress. What corresponds to quantified "until" path formulas in *CTL* is in our setting interpreted over paths of *l-ATS* 's respecting certain *progress assumptions*. This is formalized using maximal traces in the framework of Mazurkiewicz trace-theory. As an example the formula $Ev <b> tt$, "eventually b is enabled" (tt means "True"), is true of the process agent example under the assumption of progress (our interpretation) , but not without (standard *CTL* interpretation). Our interpretation is conservative in the sense that if we interpret *P-CTL* over standard *lts* we get the standard *CTL* interpretation.

Work on expressing progress assumptions and fairness assumptions can be found in e.g. Manna and Pnueli's book on temporal logic [10]. Often it involves "coding" these assumptions in the logic using formulas of the form $\phi_{fair} \Rightarrow \psi$, which require a more detailed knowledge of the particular system. We are able to avoid this obstacle and treat progress assumptions *uniformally*.

In this paper we present three contributions.

We give sound and complete tableau rules for *P-CTL* for finite *l-ATS* 's. The rules are a generalization of those in [8, 23] in the sense that if we restrict model checking to *lts*'s (a subclass of *l-ATS* 's) our tableau rules work in essentially the same way. Using the distinction between "local" and "global" model checking as advocated by Stirling and Walker in [23] our method must be classified as "local" model checking. Local model checking has the advantage that it isn't necessary to have an explicit representation of the states of the system being investigated. Labelled 1-safe P/T nets are examples of models which can be "locally" model checked without necessarily generating the entire reachability graph/state space.

In the standard setting of *CTL*-like logics interpreted over *lts*'s, model checking has been described in [3] using a state based algorithm, and in [8, 23] using tableau rules. Model checking in the framework of partial order semantics has been described in [19, 17]. We provide the first, to our knowledge, set of sound and complete tableau rules in the style of [8, 23] for a *CTL*-like logic interpreted in the trace theoretic framework.

We also provide a model checking algorithm. It resembles the algorithm in [3]. By chosing a state based approach, contrary to a tableau based as in [8, 23], we are able to obtain a polynomial time algorithm.

We present different extensions of *P-CTL* in the form of modal operators expressing concurrent or conflicting behaviour. It has been noted that when generalizing traditional specification logics with operators specifying concurrent behaviour, one gets undecidability results [9]. We prove that this is also the case for some of the extended versions of *P-CTL* by proving undecidability of satisfiability when imposing an injectivity constraint on the *l-ATS* 's as in [9]. The general satisfiability problem for different extensions of *P-CTL* is still open.

The rest of the paper is organized as follows. In section 2 we provide some necessary definitions. Partial order semantics of a class of concurrent systems is presented. In section 3 we present the logic and its interpretation. Section 4 contains a motivating example followed by the tableau rules and the definition of proof trees. In section 5 we state the main result, soundness and completeness of the proposed tableau rules. The model checking algorithm is presented in section 6. Section 7 presents different extensions of *P-CTL* and negative results on satisfiability of formulas in some of the extensions, and finally section 8 contains the conclusion and some open problems and suggestions for future work.

2 Basic Definitions

In this section we recall some basic definitions. Furthermore we state some facts and lemmas. First we define *concurrent alphabets*, the fundamental structure in Mazurkiewicz trace theory [11].

Definition 1 Concurrent alphabet and traces

- A concurrent alphabet (A, I) *is a set A and a relation $I \subseteq A \times A$, the independence relation, which is symmetric and irreflexive.*

In the following assume a fixed concurrent alphabet (A, I).

- *Given a set A we define $A^\infty = A^* \cup A^\omega$, i.e. A^∞ is the set of all finite and infinite sequences of elements from A.*
- *Define concatenation $\circ$ of elements in A^∞ as:*

$$u \circ v = \begin{cases} u & if\ |u| = \infty \\ uv & else \end{cases}$$

For notational convenience we will write uv instead of $u \circ v$.

- *Let $\leq_{prf}$ be the usual prefix ordering on sequences and $\pi_{(a,b)}$ the projection on $\{a, b\}^\infty$. We define a preorder $\preceq$ on A^∞ which demands that the relative order of arbitrary elements a and b, which are in conflict, i.e. $(a, b) \notin I$, must be the same when ignoring other elements of the sequences. Formally:*

$$u \preceq v \ \text{ iff } \ (\forall (a, b) \notin I.\ \pi_{(a,b)}(u) \leq_{prf} \pi_{(a,b)}(v))$$

- *Define an equivalence relation $\equiv$ on A^∞ by:*

$$u \equiv v \ \text{ iff } \ u \preceq v \text{ and } v \preceq u$$

Let $[u]$ denote the equivalence class containing u.

- *Fact: $\equiv$ is a congruence with respect to $\circ$.*
- *The elements of $A^\infty/\equiv$ will be called traces.*
- *Define a relation $\preceq$ on $A^\infty/\equiv$ as follows. For $[u],[v] \in A^\infty/\equiv$ we define:*

$$[u] \preceq [v] \text{ iff } u \preceq v$$

It can be shown that this relation is a partial order. We will write $[u] \prec [v]$ if and only if $u \preceq v$ and $u \not\equiv v$.

- *Fact: for $u, v \in A^*$:*
  - $u \equiv v$ *iff* $u \equiv_M v$
  - $[u] \preceq [v]$ *iff* $(\exists u' \in A^*. [uu'] = [v])$

  where $\equiv_M$ is the well known equivalence used by Mazurkiewicz for defining finite traces, see [11].

- *Fact: A set $C \subseteq A^*/\equiv$ is called a* chain *if $(\forall x, y \in C.\ x \preceq y \ \vee\ y \preceq x)$. A set $C \subseteq A^*/\equiv$ is a line if it is a chain that is maximal with respect to subset inclusion. For $C \subseteq A^*/\equiv$ we define $\downarrow C = \{[u'] \in A^*/\equiv \mid (\exists [u''] \in C.\ [u'] \preceq [u''])\}$. Then the following holds: There is a bijective correspondence between $\{[u] \mid u \in A^\omega\}$ and $\{\downarrow C \mid C$ is a line in $A^*/\equiv\}$.*

We have chosen to present traces using projections $\pi_{(a,b)}$. In this way finite as well as infinite traces are handled in a uniform way. Similar definitions can be found in e.g. [7].

The last *Fact* states the relationship to the work of Mazurkiewicz, Ochmański and Penczek [12]. For notational convenience we prefer to consider "infinite" traces as equivalence classes of infinite sequences rather than sets of the form $\downarrow C$.

We now define *labelled asynchronous transition systems* (*l-ATS*). Asynchronous transition systems have been presented by Shields [21] and Bednarczyk [1]. The labelled versions have been used by Mukund and Nielsen in [14] and are categorically related to labelled 1-safe nets [25], which are used by Olderog in [16].

Definition 2 Labelled asynchronous transition system (*l-ATS*)

A *labelled asynchronous transition system* is a tuple *(S,i,E,I,Trans,l:E → Act)* such that

- S is a set of *states*, with initial state i.
- E is a set of *events*.
- *Trans* $\subseteq S \times E \times S$ is the *transition relation*

  (i) Determinism – every event has a unique effect:
  $(s,e,s') \in$ *Trans and* $(s,e,s'') \in$ *Trans* $\Rightarrow s' = s''$.

- $I \subseteq E \times E$ is an irreflexive, symmetric, *independence relation* satisfying:

  (ii) Coherence – simultaneously enabled independent events can occur interleaved:
  $e_1 I e_2$ *and* $(s,e_1,s_1),(s,e_2,s_2) \in$ *Trans* $\Rightarrow$
  $(\exists u.\ (s_1,e_2,u),(s_2,e_1,u) \in$ *Trans*$)$.

  (iii) Commutativity – adjacent independent events can be permuted:
  $e_1 I e_2$ *and* $(s,e_1,s_1),(s_1,e_2,u) \in$ *Trans* $\Rightarrow$
  $(\exists s_2.\ (s,e_2,s_2),(s_2,e_1,u) \in$ *Trans*$)$.

- $l : E \rightarrow Act$ is a labelling function associating to each event an action from a set *Act*.

Our definition corresponds exactly to the one presented in [25] except that we have dropped the following axiom since we do not need it.

(i) Every event occurs somewhere in the *l-ATS* :
$e \in E \Rightarrow (\exists\, s, s' \in S.(s,e,s') \in$ *Trans*$)$.

l-ATS 's arise naturally from labelled 1-safe nets. Intuitively we obtain a *l-ATS* from a labelled 1-safe net N by choosing S to be the reachable markings of the net, i to be the initial marking, E to the events of the nets, I to be the disjoint pre-/post/condition relation on events, *Trans* to be the occurrence relation of events, and l to be inherited from N. For details see [25]. In section 4 below we provide an example.

Note that a *l-ATS* is deterministic at the level of events but not necessarily at the level of the labels, since two different events might be labelled by the same label.

We use the notation $p \xrightarrow{e} q$ whenever $(p,e,q) \in$ *Trans*, $p \not\xrightarrow{e}$ whenever there is no $q \in S$ such that $p \xrightarrow{e} q$, and $p \not\rightarrow$ if $p \not\xrightarrow{e}$ for all events $e \in E$. We also assume that the size of E is at most countably infinite.

In the following we assume a fixed finite *l-ATS* *(S,i,E,I,Trans,l:E → Act)*. Notice that (E,I) is a concurrent alphabet. Whenever nothing else is mentioned it is implicitly assumed that (E,I) is used to generate the congruence $\equiv$.

Definition 3 Path

- *Define a* path *from $p \in S$ as a sequence, finite or infinite, of events $e_1, e_2, \ldots$, for which there exists states $p_1, p_2, \ldots$ such that $p \xrightarrow{e_1} p_1 \xrightarrow{e_2} p_2 \cdots$. Notice that the axioms of l-ATS ensures the uniqueness of the p_i's if they exist. We therefore also refer to $p \xrightarrow{e_1} p_1 \xrightarrow{e_2} p_2 \cdots$ as a path from p and use the notation $p \xrightarrow{\sigma}$ where $\sigma = e_1 e_2 \cdots$. Note that an event may be repeated along a path. Hence, in our definition the term event does not correspond to e.g. events in event structures [25]. The correspondence is obtained by considering event occurrences along the paths [25]. Define $path(p) \subseteq E^\infty$ to be all paths from p and use the notation $p \xrightarrow{\sigma}$ to indicate that $\sigma \in path(p)$.*
- *A path from p is maximal if it is either infinite or ends in a deadlocked state (or just a deadlock) p_n, i.e. a state p_n such that $p_n \not\rightarrow$.*
- *Due to the axioms of l-ATS we have that $\equiv$ respect the path property, formally: $(\forall \sigma \in path(p).\ (\forall \sigma' \in [\sigma].\ p \xrightarrow{\sigma'}))$. Hence $path(p)$ can be partitioned into elements of $E^\infty/\equiv$. Moreover if σ is finite then $p \xrightarrow{\sigma} q \Rightarrow (\forall \sigma' \in [\sigma].\ p \xrightarrow{\sigma'} q)$.*

- *Given $\sigma \in path(p), |\sigma| = \infty, \sigma = e_1 e_2 \cdots$. An event e is said to be* continuously enabled along $p \xrightarrow{\sigma} = p \xrightarrow{e_1} p_1 \xrightarrow{e_2} p_2 \cdots$ *if and only if e is enabled from a certain point and independent of the rest of the events taken along $p \xrightarrow{\sigma}$, formally: $(\exists n \in \mathbb{N}.\ (\forall j \geq n.\ p_j \xrightarrow{e} \wedge\ e I e_{j+1}))$. Notice that the irreflexivity of I implies that from a certain point e is never taken along the path $p \xrightarrow{\sigma}$. Whenever p is understood we simply say that e is continuously enabled along σ. In the process agent example from the introduction, τ is continuously enabled along $i \xrightarrow{a^\infty}$.*
- *Define $comp(p)$ as the maximal elements with respect to $\preceq$ of $path(p)/\equiv$. For $\sigma \in [\sigma'] \in comp(p)$ we refer to $p \xrightarrow{\sigma}$ as a computation from p. In the process agent example, $\tau b a^\infty$ is a computation from i while a^∞ is not. Because E is at most countably infinite $comp(p)$ is non-empty.*

Lemma 4 *If e is continuously enabled along $\sigma \in path(p)$ then for any $\sigma' \in [\sigma]$ e is continuously enabled along σ', i.e. $\equiv$ respects* continuously enabled. *Hence for $\sigma \in path(p)$ we say that $e \in E$ is* continuously enabled along $[\sigma]$ *if e is* continuously enabled along σ.

Lemma 5 *Given $\sigma \in path(p), |\sigma| = \infty$. Then $(\exists e \in E.\ e$ is continuously enabled along $\sigma)$ iff $(\exists \sigma' \in path(p).\ [\sigma] \prec [\sigma'])$.*

Proof. Proofs of both lemmas can be found in appendix A. □

Above we have identified the maximal traces as maximal elements in a partial order. Lemma 5 explains why we concentrate on these traces. They represent executions (of a concurrent system) which are fair with respect to progress of independent processes. In [12] the term "concurrency fairness" is used for such behaviours. Compared to other notions of "fairness" in the context of concurrent systems, "progress fairness" is a very weak assumption, see [10] for a comparison.

3 The Logic *P-CTL* and its Interpretation

In this section we assume a fixed *l-ATS* $K = (S,i,E,I,Trans,l{:}E \to Act)$. Our logic has the following syntax, where $\alpha \in Act$.

$$A\ ::=\ tt \mid \neg A \mid A_1 \wedge A_2 \mid \Diamond_\alpha A \mid A_1\, U_\exists\, A_2 \mid A_1\, U_\forall\, A_2$$

In Hennessy-Milner [13] logic $<a> A$ expresses the fact that an a can be performed reaching a state such that A holds. Here the $\Diamond_\alpha$ operator expresses the fact that an event labelled α can be performed reaching a state where A holds. The operators $U_\exists$ and $U_\forall$ are introduced as generalizations of their counterparts in e.g. [3], here interpreted over maximal traces, following Mazurkiewicz [11].

The logic is interpreted over K as follows, where $p \in S$, $\alpha \in Act$, and we have written $\models$ instead of $\models_K$ since K was fixed.

- $p \models tt$ always holds.
- $p \models \neg A$ iff not $p \models A$

- $p \models A_1 \wedge A_2$ iff $p \models A_1$ and $p \models A_2$
- $p \models \Diamond_\alpha A$ iff $(\exists e \in E, q \in S.\ l(e) = \alpha\ \wedge\ p \xrightarrow{e} q\ \wedge\ q \models A)$
- $p \models A_1\, U_\exists\, A_2$ iff $(\exists [\sigma] \in comp(p),\ p \xrightarrow{\sigma} = p_0 \xrightarrow{e_1} p_1 \xrightarrow{e_2} p_2 \cdots . (\exists 0 \leq n \leq |\sigma|.$ $(p_n \models A_2)\ \wedge\ (\forall 0 \leq i < n.\ p_i \models A_1)))$
- $p \models A_1\, U_\forall\, A_2$ iff $(\forall [\sigma'] \in comp(p).\ (\forall \sigma \in [\sigma'],\ p \xrightarrow{\sigma} = p \xrightarrow{e_1} p_1 \xrightarrow{e_2} p_2 \cdots .$ $(\exists 0 \leq n \leq |\sigma|.\ p_n \models A_2\ \wedge\ \forall 0 \leq i < n.\ p_i \models A_1)))$

Furthermore we define

- $ff \equiv \neg tt$
- $F\, A \equiv tt\, U_\exists A$
- $Ev\, A \equiv tt\, U_\forall A$
- $<\alpha> A \equiv \Diamond_\alpha A$
- $G\, A \equiv \neg F \neg A$
- $Al\, A \equiv \neg Ev \neg A$

The meaning of EvA is that eventually A will hold along any path that satisfies the progress assumptions (fair progress) while AlA means that along some progress fair path A always holds. In the process agent example from the introduction $i \models Ev\ \ <b>\ tt$ holds.

If we restrict attention to the subclass of *l-ATS* 's whose independence relation is empty we obtain the usual interpretation of the logical operators.

Definition 6 *Given a l-ATS $K = (S,i,E,I,Trans,l{:}E \to Act)$ and a formula A. The* model checking problem of K and A *is the problem of deciding whether or not $i \models A$.*

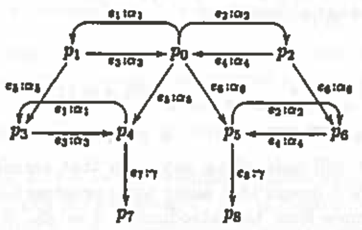
4 A Tableau Method for Model Checking

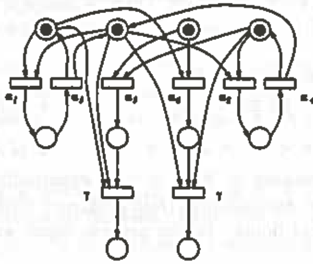
Local model checking as tableau systems has been presented in [23]. As opposed to a global model checker [3] (see also section 6, where a global model checker for *P-CTL* is presented), which checks if all states of the system satify a formula, a local model checker only checks if a specific state satisfies a given formula. For local model checkers based on tableau systems this is done by only visiting states if the tableau rules require it. Hence the local model checker may well be able to show that a state satisfies a formula without visiting all states of the system. For systems with a compact representation, such as 1-safe P/T nets (where a state of the system/net is considered to be a marking), a local model checker only has to generate new parts of the reachability graph when the tableau rules require it. Since the size of the reachability graph can be exponentially larger than the size of the net, a local model checker sometimes have an advantage over a global model checker.

In this section we present a local model checker based on a tableau system for model checking formulas from our logic with respect to *l-ATS* 's. We begin by considering an example to provide some intuition about the problems we are faced with when looking for a tableau system. Since our interpretation of the logical operators in *P-CTL* coincides with the usual interpretation we would like the tableau system to be a conservative extension of those presented in [8, 23], in the sense that when one only considers *l-ATS* 's with empty independence relation our tableau system should work in the same manner. The main difficulty is how to generalize the unfolding of minimal fixed-point assertions.

4.1 Unfolding Minimal Fixed-Point Assertions

Below we consider a very simple *l-ATS* , ats_1, which is generated by the 1-safe P/T net to the right. A good reference concerning the relationship between 1-safe nets and asynchronous transition systems is [25]. *l-ATS* 's also arise from certain process algebras, see [14].





The e_i's are the events, the Greek letters the labels, and p_0 the initial state. The independence relation is the smallest such containing $(e_1, e_5), (e_3, e_5), (e_2, e_6), (e_4, e_6)$. Now $p_0 \models_{ats_1} \neg Ev <\gamma> tt$ because of e.g. $[(e_1e_3e_2e_4)^\infty] \in comp(p_0)$ and no state along the computation $(e_1e_3e_2e_4)^\infty$ satisfies $<\gamma> tt$. However if we restrict yourselves to the states p_0, p_1, p_3, p_4, p_7 (referred to as ats_2) we do indeed have $p_0 \models_{ats_2} Ev <\gamma> tt$, since every computation from p_0 must eventually reach p_4 — e_5 cannot be continuously ignored.

Let us consider how a proof tree for $p_0 \models_{ats_2} Ev <\gamma> tt$ might look like:

$$\dfrac{p_0 \vdash Ev <\gamma> tt}{\dfrac{p_1 \vdash Ev <\gamma> tt}{p_0 \vdash Ev <\gamma> tt \quad \dfrac{p_3 \vdash Ev <\gamma> tt}{\dfrac{p_4 \vdash Ev <\gamma> tt}{\dfrac{p_4 \vdash <\gamma> tt}{p_7 \vdash tt}}}} \quad \dfrac{p_4 \vdash Ev <\gamma> tt}{\dfrac{p_4 \vdash <\gamma> tt}{p_7 \vdash tt}}}$$

The above tree is constructed according to some intuitive proof rules. Although informal, the example provides the first important observation. The leftmost branch begins and ends with the sequent $p_0 \vdash Ev <\gamma> tt$. In the μ-calculus $Ev <\gamma> tt$ is expressed by the formula $\mu X. <\gamma> tt \vee [Act]X$. Hence based on the tableau methods from [8, 23], one might expect that the above tree should be discarded as a proof tree since in the unfolding of the minimal fixed-point assertion reaches itself. However in the current framework, we interpret the logic over maximal traces, and the detected loop, $(p_0 \xrightarrow{e_1} p_1 \xrightarrow{e_3} p_0)^\infty$, is not a computation from p_0 since the event e_5 is continuously enabled. This example suggests that we might allow the unfolding of a minimal fixed-point assertion to reach itself. The cases in which this will be allowed should include the existence of an event that is continuously enabled along the loop represented by such a branch. Our solution to this problem is to annotate the logic used in the tableau rules. The idea is to keep track of the events which are continuously enabled and update this information as one unfolds the *l-ATS* via the tableau rules. So in our case e_5 would be "remembered" along the $p_0 \rightarrow p_1 \rightarrow p_0$ branch.

Let us consider a second example. This time we use ats_1. Again we construct in an intuitive and informal manner a tree rooted in the sequent $p_0 \vdash Ev <\gamma> tt$:

$$\dfrac{p_0 \vdash Ev <\gamma> tt}{\dfrac{p_1 \vdash Ev <\gamma> tt}{p_0 \vdash Ev <\gamma> tt \quad \dfrac{p_3 \vdash Ev <\gamma> tt}{\dfrac{p_4 \vdash Ev <\gamma> tt}{\dfrac{p_4 \vdash <\gamma> tt}{p_7 \vdash tt}}}} \quad \dfrac{p_4 \vdash Ev <\gamma> tt}{\dfrac{p_4 \vdash <\gamma> tt}{p_7 \vdash tt}} \quad \dfrac{p_8 \vdash Ev <\gamma> tt}{\dfrac{p_8 \vdash <\gamma> tt}{p_8 \vdash tt}} \quad \dfrac{p_2 \vdash Ev <\gamma> tt}{\dfrac{p_6 \vdash Ev <\gamma> tt}{\dfrac{p_5 \vdash Ev <\gamma> tt}{\dfrac{p_8 \vdash <\gamma> tt}{p_8 \vdash tt}}} \quad p_0 \vdash Ev <\gamma> tt}}$$

Again the interesting parts are the branches that unfold a minimal fixed-point assertion into itself. There are two such branches, the leftmost and the rightmost. However along both of these there are events which are continuously enabled, e_5 for the left branch and e_6 for the right branch. So according to the previous remarks these branches shouldn't discard the tree from being a proof tree. But we do wish to discard the tree as a proof tree since $p_0 \models_{ats_1} \neg Ev <\gamma> tt$. The problem is that by composing the two loops $(p_0 \xrightarrow{e_1} p_1 \xrightarrow{e_3} p_0)$ and $(p_0 \xrightarrow{e_2} p_2 \xrightarrow{e_4} p_0)$ we can obtain an infinite path $(p_0 \xrightarrow{e_1} p_1 \xrightarrow{e_3} p_0 \xrightarrow{e_2} p_2 \xrightarrow{e_4} p_0)^\infty$. Along this path there is no event which is continuously enabled, i.e. it is a computation from p_0. Moreover no state along the loop satisfies $<\gamma> tt$. This fact should discard the tree from being a proof tree.

One solution to the problem of detecting such "combined" loops is to continue to unfold the minimal fixed-point assertions $p_0 \vdash Ev <\gamma> tt$. If we unfold the fixed point assertion once more in the above example, still updating and propagating the information kept in the annotation, we will obtain a leaf with the information that we have found a looping path along which no event is continuously enabled. This could discard the tree from being a proof tree.

What remains is to find some bound on the number of times we allow the unfolding of a minimal fixed-point assertion. In the next subsection we provide the necessary definitions that take care of the general case. The bound we use is $|E|$, the number of events in the *l-ATS* .

4.2 Tableau Rules

In this section we consider a fixed *finite l-ATS* .

4.2.1 Annotated Logic

Before giving the tableau rules, we define the syntax of an annotated logic which is used in the tableau rules:

$$B \ ::= \ tt \mid \neg B \mid B_1 \wedge B_2 \mid \Diamond_a A \mid B_1 \ U^C_\exists \ B_2 \mid B_1 \ U^C_\forall \ B_2$$

where $1 \leq n < \infty$, $C \subseteq S$. The model checking will be performed by unfolding parts of the *l-ATS* into a tree structure. The unfolding will take place under certain constraints which restrict the size of the tree structure. The intuition is that C keeps track of which states have been visited and prevents unnecessary unfolding.

For the $U_\forall$ operator we need a more elaborate annotation. In the tableau rules we use the following annotations: $B_1 \ U_\forall^{(p,n,E')} \ B_2$, $B_1 \ U_\forall^{(p,n,E',S',-)} \ B_2$, and $B_1 \ U_\forall^{(p,n,E',S',-)} \ B_2$, where B_1, B_2 are formulas from the annotated logic and $S' \subseteq S$, $p \in S$, $E' \subseteq E$, and $n \in \mathbb{N}$. S' plays the same role as C and E' keeps track of which events have been concurrently enabled but ignored along the path corresponding to the current branch in the tree. p is a state from which we try to detect certain "critical loops". In the example from section 4.1 p would correspond to p_0.

4.2.2 Rules

In this section we present the tableau rules. There is a trade off between the rules and the defintion of proof trees. One can obtain simple rules at the cost of a complicated definition of proof trees. Moreover one needs a global side condition on the trees. We have chosen another approach. At the cost of presenting less simple tableau rules we keep the defintion of proof trees simple (definition 7 and 9) and avoid any global sidecondition.

The rules will consist of tableau rules for sequents of the form $p \vdash B$. The rules can be read from top to bottom as: "the top sequent holds if the bottom sequents and side conditions hold".

Tableau Rules

$\wedge$, 1) $\dfrac{p \vdash B_1 \wedge B_2}{p \vdash B_1 \quad p \vdash B_2}$

$\bigcirc_\alpha$, 3) $\dfrac{p \vdash \bigcirc_\alpha B}{q \vdash A}$ — $e \in E, q \in S, p \xrightarrow{e} q$ — $l(e) = \alpha$

$U_\exists^C$, 3) $\dfrac{p \vdash B_1\, U_\exists^C\, B_2}{p \vdash B_2}$ — $p \notin C$.

4) $\dfrac{p \vdash B_1\, U_\exists^C\, B_2}{p \vdash B_1 \quad q \vdash B_1\, U_\exists^{C \cup \{p\}}\, B_2}$ — $p \notin C, e \in E, q \in S, p \xrightarrow{e} q$.

$U_\forall^C$, 5) $\dfrac{p \vdash B_1\, U_\forall^C\, B_2}{p \vdash B_2}$ — $p \notin C$.

6) $\dfrac{p \vdash B_1\, U_\forall^C\, B_2}{p \vdash B_1 \quad q_1 \vdash B_1\, U_\forall^{C \cup \{p\}}\, B_2 \cdots q_n \vdash B_1\, U_\forall^{C \cup \{p\}}\, B_2}$ — $p \notin C, next(p) = \{e_1, \ldots, e_n\}$, $0 < n \in \mathbb{N}$, — $(\forall 1 \le i \le n.\ p \xrightarrow{e_i} q_i)$

7) $\dfrac{p \vdash B_1\, U_\forall^C\, B_2}{p \vdash B_1\, U_\forall^{(p, |E|, next(p))}\, B_2}$ — $p \in C$

8) $\dfrac{p \vdash B_1\, U_\forall^{(p,n,E')}\, B_2}{p \vdash B_1\, U_\forall^{(p,n-1,E',\emptyset,-)}\, B_2}$ — $0 < n \in \mathbb{N}, E' \neq \emptyset$

9) $\dfrac{q \vdash B_1\, U_\forall^{(p,n,E',S',-)}\, B_2}{q \vdash B_1 \quad q_i \vdash B_1\, U_\forall^{(p,n,e_i I E', S' \cup \{q\}, -)}\, B_2}$ — $q \notin S', next(q) = \{e_1, \ldots, e_n\}$, $0 < n \in \mathbb{N}$, — $(\forall 1 \le i \le n.\ q \xrightarrow{e_i} q_i)$

and for $D \subseteq E$, $e \in E$ we define $eID = DIe = \{e' \in D \mid e'Ie\}$.

10) $\dfrac{q \vdash B_1\, U_\forall^{(p,n,E',S',-)}\, B_2}{q \vdash B_2}$ — $q \notin S'$

11) $\dfrac{q \vdash B_1\, U_\forall^{(p,n,E',S' \cup \{q\},-)}\, B_2}{q \vdash B_1\, U_\forall^{(p,n,E',\emptyset,-)}\, B_2}$

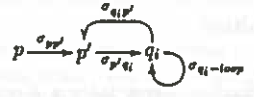
12) $\dfrac{q \vdash B_1\, U_\forall^{(p,n,E',S',-)}\, B_2}{q \vdash B_1 \quad q_i \vdash B_1\, U_\forall^{(p,n,e_i I E', S' \cup \{q\},-)}\, B_2}$ — $q \notin S', next(q) = \{e_1, \ldots, e_n\}$, $0 < n \in \mathbb{N}, q \neq p$ — $(\forall 1 \le i \le n.\ q \xrightarrow{e_i} q_i)$

13) $\dfrac{q \vdash B_1\, U_\forall^{(p,n,E',S',-)}\, B_2}{q \vdash B_2}$ — $q \notin S'$

14) $\dfrac{p \vdash B_1\, U_\forall^{(p,n,E',S',-)}\, B_2}{p \vdash B_1\, U_\forall^{(p,n,E')}\, B_2}$

Some explanation of the rules seem appropriate. The rules 1 to 4 should be reasonably clear. The annotation of the $U_\exists$ operator prevents unnecessary repetition.

The remaining rules are all concerned with the $U_\forall$ operator. If $p \not\models A_1\, U_\forall\, A_2$ then by definition of $\models$ we know that either there exists a finite path along which $A_1 \wedge \neg A_2$ holds until either $\neg A_1 \wedge \neg A_2$ holds or one reaches a deadlock, or else there exists an infinite computation along which $\neg A_1 \wedge A_2$ holds. The latter situation reduces to the existence of an infinite computation from p which consists of at most $|E|$ loops $\sigma_{p'q_i}\sigma_{q_i-loop}\sigma_{q_ip'}$ from a state p' reachable from p. This is illustrated as follows:



The rules 6 and 7 take care of the part denoted $\sigma_{pp'}$. Rules 8, 9, and 11 take care of $\sigma_{p'q_i}\sigma_{q_i-loop}$, and rules 12 and 14 take care of $\sigma_{q_ip'}$.

The next step is to define derivation trees which are build up according to the tableau rules.

4.2.3 Derivation Trees and Proof Trees

In this section we define proof trees. This is done by first defining a larger class of trees, derivation trees, which are generated according to the tableau rules. The next step is to restrict the class of derivation trees, using the annotation, to a subclass of derivation trees which will be defined to be the proof trees.

Derivation trees are defined inductively in the usual manner, except perhaps for negation. That is, if $T_1, \ldots, T_n$ are derivation trees with roots matching the sequents under the bar of a rule and the side condition is fulfilled then one obtains a new derivation tree by "pasting the derivation trees together" according to the rule. The root of the new derivation tree is labelled by the sequent above the bar. To be more specific:

Definition 7 Derivation Trees

First we define the basis, all trees having a single node, the root, labelled with the shown sequent.

- $p \vdash tt$ *is a derivation tree.*
- $p \vdash \neg B$ *is a derivation tree.*
- $p \vdash B_1\ U_\exists^C\ B_2$*, where* $p \in C$ *is a derivation tree.*
- $p \vdash B_1\ U_\forall^{(p,n,E')}\ B_2$*, where* $n = 0$ *or* $E' = \emptyset$.
- $q \vdash B_1\ U_\forall^{(p,n,E',S',-)} B_2$*, where* $q \in S'$.

By applying the rules we can obtain new derivation trees, e.g.:

- *If* T_1 *is a derivation tree with root* $p \vdash B_1$ *and* T_2 *is a derivation tree with root* $q \vdash B_1\ U_\exists^{C\cup\{p\}}\ B_2$*, where* $p \notin C$ *and* $\exists e \in E.p \xrightarrow{e} q$ *then* $\dfrac{p \vdash B_1\ U_\exists^C\ B_2}{T_1 \quad T_2}$ *is a derivation tree with root* $p \vdash B_1\ U_\exists^C\ B_2$.
- *If* T *is a derivation tree with root* $p \vdash B_2$ *and* $p \notin C$ *then* $\dfrac{p \vdash B_1\ U_\forall^C\ B_2}{T}$ *is a derivation tree with root* $p \vdash B_1\ U_\forall^C\ B_2$.

Nothing else is a derivation tree.

Before defining the proof trees we need an auxiliary definition.

Definition 8 Auxillary definition

- *Let Ann be the obvious function which takes a formula A from the first grammar and annotates all its* $U_\exists$ *and* $U_\forall$ *operators with* $C = \emptyset$*, giving a formula* $Ann(A)$ *from the second grammar. A formula B from the second grammar will be said to be* clean *if there exists a formula A from the first grammar such that B equals* $Ann(A)$.
- *A sequent of the form* $q \vdash B_1\ U_\forall^{(q,n,\emptyset)}\ B_2$*, where* $n \in \mathbb{N}, q \in S$*, will be called a* terminal sequent of type (B_1, B_2)*. When* B_1 *and* B_2 *are understood we will simply say that the sequent is terminal.*

We now define proof trees and will hereby restrict our attention to meaningful derivation trees. We get rid of meaningless derivation trees as for example $p \vdash \neg tt$.

Definition 9 Proof trees

A proof tree is a derivation tree T with root $p \vdash Ann(A)$ *such that either*

- $A = tt$ *or*
- $A = \neg A'$ *and there exists no proof tree with root* $p \vdash Ann(A')$ *or*
- *A is not of the above form and*
  - *every proper subtree* T' *of T whose root is labelled with a clean formula is itself a proof tree and*
  - *T has no leaves labelled with terminal sequents.*

5 Soundness and Completeness

Having given the necessary definitions we are now ready to state the main result, which holds for finite *l-ATS* 's:

Theorem 10

Soundness:

If T is a proof tree with root $p \vdash Ann(A)$ *then* $p \models A$.

Completeness:

If $p \models A$ *then there exists a proof tree with root* $p \vdash Ann(A)$.

Proof. The proof can be found in appendix B. It proceeds by structural induction, showing soundness and completeness simultaneously. □

Notice that if we restrict ourselves to *l-ATS* 's where the independence relation is empty and translate $A_1\ U_\exists\ A_2$ as $\mu X. A_2 \vee (A_1 \wedge \langle Act\rangle X)$ and $A_1\ U_\forall\ A_2$ as $\mu X. A_2 \vee (A_1 \wedge [Act]X)$ (actually applying this translation recursively on the subformulas A_1 and A_2) our tableau rules will work in essentially the same manner as those presented in [8, 23].

Also, the reader should have no trouble in adapting the tableau rules to the case where the considered models are 1-safe nets. Notice that no prior knowledge of the complete state space is required, but only of the set of labelled transitions (labelled events).

6 Model Checking *P-CTL* by Labelling States

In this section we sketch a state based algorithm, that solves the model checking problem. The algorithm essentially works as the one presented for *CTL* in [3] except for the $U_\forall$ operator.

Theorem 11 *There exists an algorithm which given* $K = (S,i,E,I,Trans,l{:}E \to Act)$ *and a formula F solves the model checking problem for K and F. Moreover its time complexity is* $O(|F|(|S| + |Trans||E|))$.

Proof. SKETCH:
Given a formula F and a *l-ATS* $K = (S,i,E,I,Trans,l{:}E \to Act)$ the algorithm proceeds in stages as follows: In the first stage all subformulas of length one are processed. In general, at stage i all subformulas of length i are processed, and at the end of stage i a state is labelled with a subformula F' of F (or its negation $\neg F'$) if and only if it is satisfied in that state.

Hence after the $|F|$'th stage, all states of K will have been labelled with either F or $\neg F$.

The data structures needed to perform the labelling are essentially as those described for the *CTL* model checker in [3]. The only exception is the case of the $U_\forall$ operator.

($U_\exists$ can be handled as the *EU* operator in *CTL*, since any finite prefix of a path can be extended to a computation.)

The $U_\forall$ operator is handled as follows:

Assume we want to label the states with the subformula $F = A_1 U_\forall A_2$. All states must already have been labelled with A_1 or $\neg A_1$, and A_2 or $\neg A_2$. Then, states labelled with A_2 are labelled with F, and states labelled with $\neg A_1$ and $\neg A_2$ are labelled with $\neg F$. The remaining states must all be labelled with A_1 and $\neg A_2$. The next step is to compute the maximal strongly connected components of K restricted to these remaining states.

Let us denote the graph whose nodes are these maximal strongly connected components by G'. G' is a directed acyclic graph (*DAG*) whose nodes are sets of states of K. As long as there is a terminal node (call it n) in G', repeatedly do the following:

- if there is a state s in n, an event e and a state s' in K such that $s \xrightarrow{e} s'$ and s' is labelled with $\neg F$, then label all states in n with $\neg F$. Furthermore, for all node m in G', if n can be reached from m then label all states in m with $\neg F$. Remove all processed nodes from G' and let G' denote the new *DAG*.
- else, if there is a state s in n but no event e and state s' not in n such that $s \xrightarrow{e} s'$ then label all states in n (and m's above n, as described just above) with $\neg F$. Update G' as above.
- else if n only contains one state s then label this state by F (all successor states of s, and there exists at least one, are labelled F) and remove n from G'. Let G' denote the new *DAG*.
- else n must consist of more than one state, there must exists states, not in n, reachable from states in n, and all these states are labelled F. We assume $E = \{e_1, \ldots, e_m\}$.
  - Initialise a boolean array B of length m such that all its entries are set to *False*. Then for each edge $\xrightarrow{e_i:l(e_i)}$ between any to states in n, set all entries $B[j]$ such that $\neg(e_i I e_j)$ to *True*.
  - If there is an entry $B[k]$ which is *False* and e_k is enabled at any state in n, then label all states in n with F. Remove n from G' and let G' denote the new *DAG*.
  - Else label all states in n (and m's, as described in the first case) with $\neg F$ and update G' as above.

It should be obvious that cases one and three labels the states in n correctly. Case two is also correct because we can exhibit a computation in n whose states are labelled with A_1 and $\neg A_2$. Case four is correct because of the following observation: There exists a computation inside n if and only if there is no event e_k that is independent of all events labelling edges between states in n and that is enabled in (necessarily all) a state in n.

An analysis of the algorithm yields the complexity $O(|F|(|S| + |Trans||E|))$. Hence our algorithm is comparable to the one presented in [3]. □

7 Extending *P-CTL*

In this section we consider different modal operators expressing concurrent behaviour or conflicting behaviour. For the some of the logics we also prove that the satisfiability problem is undecidable for infinite as well as finite *l-ATS* 's, if we impose an "injectivity" constraint.

7.1 The New Modal Operators

First we give the syntax of the modal operators. This is done by extending the grammar from section 3 by the following rules, where $\alpha_i \in Act$ and $n > 0$:

$$A \ ::= \ \Diamond((\alpha_1, A_1), \ldots, (\alpha_n, A_n)) \mid \sharp((\alpha_1, A_1), \ldots, (\alpha_n, A_n)) \mid$$

$$(\alpha_1 \cdots \alpha_n)A \mid)\alpha_1 \cdots \alpha_n(A$$

The interpretations are:

- $p \models \Diamond((\alpha_1, A_1), \ldots, (\alpha_n, A_n))$ iff $(\exists e_1, \ldots, e_n \in E, q_1, \ldots, q_n \in S.$ $(\forall 1 \le i \le n.\ p \xrightarrow{e_i} q_i \wedge l(e_i) = \alpha_i \wedge q_i \models A_i) \wedge (\forall 1 \le i < j \le n.\ e_i I e_j))$
- $p \models \sharp((\alpha_1, A_1), \ldots, (\alpha_n, A_n))$ iff $(\exists e_1, \ldots, e_n \in E, q_1, \ldots, q_n \in S.$ $(\forall 1 \le i \le n.\ p \xrightarrow{e_i} q_i \wedge l(e_i) = \alpha_i \wedge q_i \models A_i) \wedge (\forall 1 \le i < j \le n.\ (e_i, e_j) \notin I))$
- $p \models (\alpha_1 \cdots \alpha_n)A$ iff $(\exists e_1, \ldots, e_n \in E, q \in S.\ (\forall 1 \le i \le n.\ l(e_i) = \alpha_i) \wedge$ $(\forall 1 \le i < j \le n.\ e_i I e_j) \wedge p \xrightarrow{e_1 \cdots e_n} q \wedge q \models A)$

  (This operator resembles the one proposed in [9])
- $p \models)\alpha_1 \cdots \alpha_n(A$ iff $(\exists e_1, \ldots, e_n \in E, q \in S, p \xrightarrow{e_1 \cdots e_n} q.\ (\forall 1 \le i \le n.\ l(e_i) = \alpha_i) \wedge$ $(\forall 1 \le i < n.\ (e_i, e_{i+1}) \notin I) \wedge q \models A)$

The $\Diamond$ and () operators specifies concurrent behaviour. The difference between them is that () requires a property A to hold *after* the execution of a set of mutually independent labelled events while $\Diamond$ require properties A_i to hold after the execution of each labelled event from a set of mutually independent labelled events. The $\sharp$ and *Cseq* operators specifies conflicting behaviour and are obtained from $\Diamond$ and)(by requirering the events to be mutually in conflict instead of independent. Each of the four modal operators captures some behaviour that cannot be specified by the others. In appendix D some *l-ATS* 's are given, which show that they are incomparable.

We will concentrate on the extensions of *P-CTL* which contains one of the operators $\Diamond$ or (). In these cases we can obtain undecidability results for the injective satisfiability problem (defined below).

Definition 12 *A l-ATS* $(S, i, E, I, Trans, l{:}E \to Act)$ *is said to be injective if for all* $s \in S, e, e' \in E$ *it is the case that* $s \xrightarrow{e}, s \xrightarrow{e'}$, *and* $l(e) = l(e')$ *implies* $e = e'$.

Definition 13 *The* (finite) injective satisfiability problem *is the problem of deciding, given a formula A, whether there exists a (finite) injective l-ATS* $K = (S, i, E, I, Trans, l : E \to Act)$ *such that* $i \models A$. *If this is the case, A will be said to be (finitely) injectively satisfiable.*

For the logics containing the $\Leftrightarrow$ operator we have the following result:

Lemma 14 *The set of formulas that are injectively satisfiable is nonrecursive, i.e. the injective satisfiability problem is undecidable. The same is true of the set of formulas that are finitely injectively satisfiable.*

Proof. The proofs can be found in appendix C. The $U_\forall$ operator is essential in the proof of the last lemma. □

For the logics containing the $\langle\rangle$ operator similar results hold, see [9]. Also, for the $\sharp$ and $\rangle\langle$ operators we are able to encode grids. For example $\sharp((up, tt)) \wedge \sharp((right, tt)) \wedge \neg\sharp((up, tt), (right, tt))$ specifies the existence of an "independence square". Hence we can prove similar undecidability results

8 Conclusion and Future Work

Partial order semantics for concurrent systems have gained interest because interleaving models of concurrency have failed to provide an acceptable interpretation of what it means for events of a concurrent system to be independent. Much work has been devoted to translate obtained results and notions from the interleaving models to the "true concurrency" models [6, 5, 25, 9]. Trying to contribute to the "translation of results" we have provided tableau rules for a *CTL*-like logic interpreted over maximal traces. The work which we have tried to "translate" can be found in [8, 23].

The choice of the modal operators in our extensions of the logic has partly been made because we wanted to draw attention to the fact that there is no obvious choice. Also one might argue both for and against that the presented *l-ATS* 's in appendix D should, given any reasonable operationally based notion of equivalence, be distinguished. Notice that when using the $\Leftrightarrow$ operator, the the proof of the undecidability of the restricted satisfiability problems uses the coherence axiom (ii) in the definition of *l-ATS* 's. Without it we are not able to encode a grid. When chosing the operator $\langle\rangle$, as is done in [9], axiom (iii) would enable us to encode grids.

Future research should look into the problem of finding some suitable notion of bisimulation characterizing the equivalences induces by the logics obtained by combining one or more of the above (or perhaps other) operators or finding other more fundamental situations, that the logical operators should be able to distinguish. Our approach is "ad hoc" in its nature and one might try to work in the setting presented in [6].

Another research area might be to handle a more expressive logic (perhaps one containing a recursion operator) in a similar way, that is, define the interpretation of the formulas over maximal traces and proving soundness and completeness of some tableau rules.

Finally the general satisfiability problem for some of the extended logics is still an open problem.

Acknowledgements: I thank Mogens Nielsen and Nils Klarlund for inspiring discussions and comments.

References

[1] M. A. Bednarczyk. *Categories of asynchronous systems.* PhD thesis, University of Sussex, 1988. PhD in computer science, report no.1/88.

[2] R. Berger. The undecidability of the domino problem. *Memoirs American Math. Sco.*, 66, 1966.

[3] Edmund M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite state concurrent system using temporal logic. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[4] David Harel. Recurring dominoes: making the highly undecidable highly understandable. *Ann. Disc. Math.*, 24:51–72, 1985.

[5] Lalita Jategaonkar and Albert Meyer. Deciding true concurrency equivalences on finite safe nets. In *Proc. ICALP'93*, pages 519–531, 1993.

[6] André Joyal, Mogens Nielsen, and Glynn Winskel. Bisimulation and open maps. In *Proc. LICS'93, Eighth Annual Symposium on Logic in Computer Science*, pages 418–427, 1993.

[7] Marta Z. Kwiatkowska. Event fairness and non-interleaving concurrency. *Formal Aspects of Computing*, 1:213–228, 1989.

[8] Kim G. Larsen. Proof systems for Hennessy-Milner logic with recursion. In *Proceedings of CAAP, Nancy France*, pages 215–230. Springer-Verlag (*LNCS* 299), March 1988.

[9] Kamal Lodaya, Rohit Parikh, R. Ramanujam, and P. S. Thiagarajan. A logical study of distributed transition systems. Technical report, School of Mathematics, SPIC Science Foundation, Madras, 1993. To appear in Information and Computation, a preliminary version appears as Report TCS–93–8.

[10] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems.* Springer Verlag, 1992.

[11] Antoni Mazurkiewicz. Trace theory. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, pages 279–324. Springer-Verlag (*LNCS* 255), 1986.

[12] Antoni Mazurkiewicz, Edward Ochmański, and Wojciech Penczek. Concurrent systems and inevitability. *Theoretical Computer Science*, 64():281–304, 1989.

[13] Robin Milner. *Communication and Concurrency.* Prentice Hall International Series In Computer Science, C. A. R. Hoare series editor, 1989.

[14] Madhavan Mukund and Mogens Nielsen. CCS, Locations and Asynchronous Transition Systems. *Proc. Foundations of Software Technology and Theoretical Computer Science 12*, pages 328–341, 1992.

[15] Mogens Nielsen, Gordon Plotkin, and Glynn Winskel. Petri nets, event structures and domains, Part I. *Theoretical Computer Science*, 13:85–108, 1981.

[16] Ernst R. Olderog. *Nets, Terms and Formulas.* Cambridge University Press, 1991. *Number 23 Tracts in Theoretical Computer Science.*

[17] Doron Peled and Amir Pnueli. Proving partial order liveness properties. In *Proc. ICALP'90*, pages 553–571. Springer-Verlag (*LNCS* 443), 1990.

[18] Wojciech Penczek. On undecidability of propositional temporal logics on trace systems. *Information Processing Letters*, 43:147–153, 1992.

[19] Wojciech Penzcek. Temporal logics for trace systems: On automated verification. *International Journal of Foundations of Computer Science*, 4 (1):31–67, 1993.

[20] Wolfgang Reisig. *Petri Nets – An Introduction.* EATCS Monographs in Computer Science Vol.4, 1985.

[21] M. W. Shields. Concurrent machines. *Computer Journal*, 28:449–465, 1985.

[22] Eugene W. Stark. Concurrent transition systems. *Theoretical Computer Science*, 64():221–269, 1989.

[23] Colin P. Stirling and David Walker. Local model checking in the modal mu-calculus. Technical Report ECS-LFCS-89-78, Laboratory for Foundations of Computer Science, Department of Computer Science – University of Edinburgh, May 1989.

[24] Glynn Winskel. Event structures. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, pages 325–390. Springer-Verlag (*LNCS* 255), 1986.

[25] Glynn Winskel and Mogens Nielsen. Models for concurrency. Technical Report DAIMI PB-429, Computer Science Department, Aarhus University, November 1992. To appear as a chapter in the Handbook of Logic and the Foundations of Computer Science, Oxford University Press.

A Proofs of Lemmas

Claim 15 *Given a concurrent alphabet (A, I). For $\sigma, \sigma' \in A^\infty$ we have that*

$$\sigma \preceq \sigma' \Leftrightarrow (\forall \sigma_1 \in pref_{fin}(\sigma). (\exists \sigma_1' \in pref_{fin}(\sigma'). \sigma_1 \prec \sigma_1'))$$

Proof.

"$\Leftarrow$" Easy proof by contradiction.

"$\Rightarrow$" First choose a finite prefix σ_1' of σ' s.t. its Parikh vector (for each event e this vector provides the number of occurrences of e in σ_1') is greater than or equal to that of σ_1. Assuming $\sigma_1 = a_1 \cdots a_n$ and $\sigma_1' = b_1 \cdots b_m$ find the first occurrence of a_1, say b_{j_1} in σ_1'. Also $(\forall 1 \leq j < j_1. b_j I b_{j_1})$, since we have $\sigma \preceq \sigma'$. Hence $b_1 \cdots b_m \equiv b_{j_1} b_1 \cdots b_{j_1 - 1} b_{j_1+1} \cdots b_m$. Continuing this procedure for $a_2, \ldots, a_n$ we eventually get that $\sigma_1' \equiv \sigma_1 \gamma$ for some $\gamma \in A^*$. But then clearly $\sigma_1 \preceq \sigma_1'$.

□

Claim 16 *Given a l-ATS $= (S,i,E,I,Trans,l{:}E \to Act)$, $\sigma \in path(p)$ s.t. $|\sigma| = \infty$, and an $e \in E$ that is continuously enabled along σ. Then $(\forall \sigma' \in [\sigma]. e$ is continuously enabled along $\sigma')$.*

Proof. Clearly by definition there exists a finite $\sigma_1 \in path(p)$, a $p' \in S$, and a $\sigma_2 \in path(p')$ s.t. $p \xrightarrow{\sigma}= p \xrightarrow{\sigma_1} p' \xrightarrow{\sigma_2}$ and e is enabled at p' and independent of all events in σ_2. Since $[\sigma] \preceq [\sigma']$ we have from the previous lemma that there exists a finite prefix of σ', say σ_1', s.t. $[\sigma_1] \preceq [\sigma_1']$. But then by the facts from section 2 we know that there exists a $\gamma \in E^\infty$ s.t. $\sigma_1 \gamma \equiv \sigma_1'$ and all events in γ are independent of e. We conclude that e must be enabled at p'' where $p \xrightarrow{\sigma_1'} p''$, since $p \xrightarrow{\sigma_1 \gamma} p''$. Choosing σ_2' s.t. $\sigma' = \sigma_1' \sigma_2'$ we also conclude that all events in σ_2' are independent of e since all events in σ_2' are events of σ_2. Hence e is continuously enabled along σ'. □

Claim 17 *Given a l-ATS $= (S,i,E,I,Trans,l{:}E \to Act)$. For $[\sigma] \in comp(p), p \in S, |\sigma| = \infty$ we have*

$$(\exists [\sigma'] \in comp(p). [\sigma] \prec [\sigma']) \textit{ iff } (\exists e \in E.\ e \textit{ is continuously enabled along } \sigma).$$

Proof.

"$\Leftarrow$" This direction is easy.

"$\Rightarrow$" First we observe the following: Since $[\sigma] \prec [\sigma']$ there must exist an $e \in E$ s.t. $\pi_{(e,e)}(\sigma) < \pi_{(e,e)}(\sigma')$. Clearly $|\pi_{(e,e)}(\sigma)| = n < \infty$ for some $n \in \mathbb{N}$.

Let $\sigma = \sigma_1 \sigma_2$ where $\#_e(\sigma_1) = n, \#_e(\sigma_2) = 0$, and $|\sigma_1| < \infty$.

By the above lemma we know that there exists a finite prefix of σ', σ_3, s.t. $[\sigma_1] \preceq [\sigma_3]$. Furthermore there must exists a suffix of σ s.t. all events of it are independent of e. To see this assume that there were infinitely many indexes $i_j \in \mathbb{N}$ for $0 \leq j$ s.t. $(e_{i_j}, e) \notin I$, where $\sigma = e_1 e_2 \cdots$. Since $(\forall j \in \mathbb{N}. \pi_{(e,e_{i_j})}(\sigma) < \pi_{(e,e_{i_j})}(\sigma'))$ all e_{i_j}'s must occur before the $n+1^{th}$ e in $\pi_{(e,e_{i_j})}(\sigma')$. But this clearly means that there must be infinitely many events between the n^{th} and $n+1^{th}$ e in σ' which is impossible.

We now show that there must exist an event e' which is continuously enabled along σ.

First choose the first occurrence of an $e' \in E$ along σ' such that $\#_{e'}(\sigma) < \#_{e'}(\sigma')$. Next split σ into σ_1 and σ_2 s.t. $\sigma = \sigma_1\sigma_2$ and all events in σ_2 are independent of e'. Then choose σ_1' as the shortest prefix of σ' s.t. $\#_{e'}(\sigma_1') = \#_{e'}(\sigma_1)+1$ and the Parikh vector of σ_1' is greater than that of σ_1. By an argument similar to that above one can rearrange σ_1' by continuously interchanging adjacent independent events and obtain $\sigma_1\gamma \in path(p)$. Now $\#_{e'}(\gamma) > 0$. Let $\gamma = a_1 \cdots a_r e' b_1 \cdots b_s$ where $r, s \geq 0$ and all a_i's are different from e'. Now assume that $(\exists 1 \leq i \leq r. (a_i, e') \notin I)$. Choose the first such i. Then $\pi_{(a_i,e')}(\sigma_1') > \pi_{(a_i,e')}(\sigma_1)$ and since the relative occurrence of e' and a_i's in σ_1' and $\sigma_1\gamma$ are the same an a_i must occur before the $\#_{e'}(\sigma_1)+1^{th}$ e' in σ'. But $\#_{a_i}(\sigma) = \#_{a_i}(\sigma')$ by choice of e'. But then there must exist an a_i in σ_2 and this contradicts the assumption that all events in σ_2 were independent of e'. So by using the axioms of *l-ATS* 's we conclude that e' must be enabled at p' where $p \xrightarrow{\sigma_2} p'$. Hence e' is continuously enabled along σ.

□

B Proof of Soundness and Completeness

Claim 18 *Given a l-ATS $K = (S,i,E,I,Trans,l{:}E \rightarrow Act)$. Then for any $p \in S$ and any formula A we have:*

$$p \models A \text{ iff } (\exists \text{ a proof tree with root (labelled) } Ann(A))$$

Proof. Assume a fixed finite *l-ATS* $K = (S,i,E,I,Trans,l{:}E \rightarrow Act)$. We will show that for any $p \in S$

$$p \models A \text{ iff } \exists \text{ a proof tree with root } p \vdash Ann(A)$$

by structural induction in A.

$\underline{tt}$ Clearly we always have $p \models tt$ and the proof tree $p \vdash tt$.

$\underline{\neg}$ We have $p \models \neg A$ iff not $p \models A$ iff (induction hypothesis) there exists no proof tree with root $p \vdash Ann(A)$ iff $p \vdash \neg Ann(A)$ is a proof tree.

$\underline{\Delta}$ We have $p \models A_1 \wedge A_2$ iff $p \models A_1$ and $p \models A_2$ iff (induction hypothesis) there exists proof trees $T1$ with root $p \vdash Ann(A_1)$ and T_2 with root $p \vdash Ann(A_2)$ iff there exists a proof tree with root $p \vdash Ann(A_1 \wedge A_2)$ because $Ann(A_1 \wedge A_2) = Ann(A_1) \wedge Ann(A_2)$.

$\underline{\Leftrightarrow}$ Let $List = ((\alpha_1, A_1), \ldots, (\alpha_n, A_n))$, where $n \geq 1$. Then we have $p \models \Leftrightarrow List$ iff $(\exists e_1, \ldots, e_n \in E,$
$q_1, \ldots, q_n \in S.\ (\forall 1 \leq i \leq n.\ p \xrightarrow{e_i} q_i \ \wedge\ q_i \models A_i \ \wedge\ l(e_i) = \alpha_i) \ \wedge\ (\forall 1 \leq i < j \leq n.\ e_i I e_j))$ iff there exists proof trees $T_1, \ldots, T_n$ and $(\exists e_1, \ldots, e_n \in E, q_1, \ldots, q_n \in S.\ (\forall 1 \leq i \leq n.\ p \xrightarrow{e_i} q_i \ \wedge\ l(e_i) = \alpha_i \ \wedge\ T_i$ has root $q_i \vdash Ann(A_i)) \ \wedge\ (\forall 1 \leq i < j \leq n.\ e_i I e_j))$ iff there exists a proof tree T with root $p \vdash Ann(\Leftrightarrow List)$, since $Ann(\Leftrightarrow List) = \Leftrightarrow ((\alpha_1, Ann(A_1)), \ldots, (\alpha_n, Ann(A_n)))$.

$\underline{U_\exists}$ We have that $p \models A_1\ U_\exists\ A_2$ iff $(\exists\, p_1, p_2, \ldots, p_n \in S, e_1, e_2, \ldots, e_n \in E, n \geq 0.$
$p = p_0 \xrightarrow{e_1} p_1 \xrightarrow{e_2} p_2 \cdots \xrightarrow{e_n} p_n \ \wedge\ p_n \models A_2 \ \wedge\ (\forall 0 \leq i < n.\ p_i \models A_1) \ \wedge\ (\forall 0 \leq i < j \leq n.\ p_i \neq p_j))$ iff there exists proof trees $T_0, \ldots, T_{n-1}$ with roots $p_0 \vdash Ann(A_1), \ldots, p_{n-1} \vdash Ann(A_1)$ and T_n with root $p_n \vdash Ann(A_2)$ and events $e_1, \ldots, e_n$ such that $p = p_0 \xrightarrow{e_1} p_1 \xrightarrow{e_2} p_2 \cdots \xrightarrow{e_n} p_n$ is loop free iff there exists a proof tree with root $p \vdash Ann(A_1)\ U_\exists^{\emptyset}\ Ann(A_2)$ because $Ann(A_1\ U_\exists\ A_2) = Ann(A_1)\ U_\exists^{\emptyset}\ Ann(A_2)$.

$\underline{U_\forall}$ We show the bi-implication by showing the left and right implications separately.

"$\Rightarrow$" We show how to obtain a derivation tree with root labelled $p \vdash Ann(A_1\ U_\forall\ A_2)$. This will be done by providing an algorithm which will be shown to terminate and produce the desired tree. We then argue that it is a proof tree.

The tree will be constructed from the root and expanded downwards. Only so called "active" leaves of the current tree will be expanded. We try to keep the tree as small a possible by first trying to prove that B_2 holds at a state. Only if this isn't possible do we expand the tree.

During the presentation fo the algorithm several claims will be stated. All of them will be shown to be valid in the succeeding paragraph. For convenience we will use B_1 for $Ann(A_1)$ and B_2 for $Ann(A_2)$. So $Ann(A_1\ U_\forall\ A_2) \equiv B_1\ U_\forall^{\emptyset} B_2$.

The algorithm consists of the following steps:

1 The root is labelled by $p \vdash B_1\ U_\forall^{\emptyset} B_2$. Mark this node as *active*.

2 If possible choose an active node N, labelled by a sequence of one of the following forms:

i) $q \vdash B_1\ U_\forall^{C}\ B_2$

ii) $q \vdash B_1\ U_\forall^{(q,n,E')}\ B_2$

iii) $q \vdash B_1\ U_\forall^{(p,n,E',S',\leftrightarrow)}\ B_2$

where $\leftrightarrow$ stands for either $\leftarrow$ or $\rightarrow$. Else terminate.

3 If $q \models A_2$ then by induction we have the existence of a proof tree T' with root $q \vdash B_2$. Deactivate N and paste T' below N using rule 5, 10, or 13. None of the added nodes are active.

4 Else if $q \models \neg A_2$ then necessarily (claim 1) $q \models A_1$. By induction there exists a proof tree T' with root $q \vdash B_1$.

- If N is of the form i), and $q \notin C$ then (claim 2) $next(q) \neq \emptyset$ and apply rule 6, using T'. Deactivate N and activate the new leaves labelled $q_i \vdash B_1\ U_\forall^{C \cup \{q\}}\ B_2$ that were added by application of rule 6.
- If N is of the form i) and $q \in C$ then deactivate N and, using rule 7, add a node below N labelled $q \vdash B_1\ U_\forall^{(q,|E|,next(q))}\ B_2$. Using rule 8, because (claim 3) $next(q) \neq \emptyset$, add yet another node below labelled $q \vdash B_1\ U_\forall^{(q,|E|-1,next(q),\emptyset,\rightarrow)}\ B_2$ which is activated.

- If N is of the form *ii*) then (claim 4) $E' \neq \emptyset$. If $n = 0$ then deactivate N. Else if $n > 0$ then deactivate N and apply rule 8, adding a node below N labelled $q \vdash B_1\ U_\forall^{(q,n-1,E',\emptyset,\leftarrow)}$. Activate this node.
- If N is of the form *iii*) ($\rightarrow$) and $q \notin S'$ then (claim 5) $next(q) \neq \emptyset$ and we deactivate N. By induction we have the existence of the proof tree T' with root labelled $q \vdash B_1$. Using rule 9 add this tree below N and add nodes labelled $q_i \vdash B_1\ U_\forall^{(p,n,e_i IE',S' \cup \{q\},\rightarrow)}\ B_2$. Only the last nodes are activated.
- If N is of the form *iii*) ($\rightarrow$) and $q \in S'$ then deactivate N and, using rule 11 add a node below N labelled $q \vdash B_1\ U_\forall^{(p,n,E',\emptyset,\leftarrow)}\ B_2$. Activate this node.
- If N is of the form *iii*) ($\leftarrow$) and $q \notin S'$ and $q \neq p$ then deactivate N. Because (claim 6) $next(q) \neq \emptyset$, we can use rule 12 and the induction hypothesis to add proof tree T'. Also add nodes labelled $q_i \vdash B_1\ U_\forall^{(p,n,e_i IE',S' \cup \{q\},\leftarrow)}\ B_2$. Only these last nodes will be activated.
- If N is of the form *iii*) ($\leftarrow$) and $q \in S'$ and $q \neq p$ then deactivate N.
- If N is of the form *iii*) ($\leftarrow$) and $q = p$ the apply rule 14. Deactivate N and activate the added node labelled $q \vdash B_1\ U_\forall^{(q,n,E')}\ B_2$

5 Goto 2.

We now observe the following:

* The above "algorithm" terminates: One only expands *active* nodes and since the underlying *l-ATS* is finite expansion cannot continue indefinitely because of the annotation of the formulas.

* All claims stated in the algorithm are valid: Since the strategy used to compute the tree is first to try to prove that A_2 holds at a state, and if not then expand the tree, we conclude that:

  - Claim 1 is valid: If $q \models \neg A_2$ and $q \models \neg A_1$ then because of the way we expand the tree we could exhibit a finite path along which $A_1 \wedge \neg A_2$ holds until $\neg A_1 \wedge \neg A_2$ holds. But since any finite path can be extended to a computation (K is assumed to be finite) we obtain a contradiction with the assumption $p \models A_1\ U_\forall\ A_2$.
  - Claim 2 is valid: If $next(q) = \emptyset$ then we would have found a finite path starting at p and ending in q, a deadlock. This would be a computation from p to q along which no state satisfied A_2. Again this would contradict $p \models A_1\ U_\forall\ A_2$.
  - Claim 3 is valid: As for claim 2.
  - Claim 4 is valid: If $E' = \emptyset$ then because E' keeps track of which events have been concurrently enabled along the loop starting and ending at q (along the branch from the root of the tree to the current node), we would have detected one or more loops (see figure)

along which A_2 never holds and by repeating these loops we could exhibit an infinite computation along which A_2 never holds. This contradicts $p \models A_1\ U_\forall\ A_2$.

  - Claim 5 and 6 are valid: As for claim 2.

Assume the produced tree isn't a proof tree. Then, using the indcution hypothesis, we conclude that the only reason why the tree isn't a proof tree is that it has leaves labelled by terminal sequents. But then an argument similar to that used to show the validity of claim 4 gives us a contradiction with the assumption $p \models A_1\ U_\forall\ A_2$.

"$\Leftarrow$" We show that if there exists a proof tree T with root $p \vdash Ann(A_1\ U_\forall\ A_2)$ then $p \models A_1\ U_\forall\ A_2$. So assume that $p \models \neg(A_1\ U_\forall\ A_2)$ i.e. $p \models \neg A_2$ and there exists a $\sigma \in [\sigma'] \in comp(p)$ such that either:

- $|\sigma| < \infty, p = p_0 \xrightarrow{e_1} p_1 \xrightarrow{e_2} \cdots \xrightarrow{e_m} p_m \not\rightarrow, \sigma = e_1 \cdots e_m$, and

$$(\forall n \leq |\sigma|. (p_n \models \neg A_2) \vee (\exists 0 \leq i < n. p_i \models \neg A_1))$$

There are two cases.

* Assume $(\exists 0 < i \leq m.\ p_i \models A_2)$. Let $i_0 > 0$ denote the least such index. We know that there must exist an index $0 \leq j < i_0$ such that $p_j \models \neg A_1$. Let j_0 denote the least such index. Clearly the path $e_1 \cdots e_{j_0}$ can be made loop free and traceable in T (using the induction hypothesis to obtain contradictions). But this gives a contradiction since T must then have a subtree which is a proof tree labelled with root $q_{j_0} \vdash Ann(A_1)$, i.e. $q_{j_0} \models A_1$.
* No states along σ satisfies A_2. If there is a state which satisfies $\neg A_1$ along the path the argument above can be applied. Else $(\forall 0 \leq i \leq m.\ p_i \models (A_1 \wedge \neg A_2))$. But then there must exist a loop free path from p to p_m such that $A_1 \wedge (\neg A_2)$ is satisfied along it. This path must be traceable in T. But this means there must exist a leaf labelled $p_m \vdash Ann(A_1)\ U_\forall^C\ Ann(A_2)$ such that $p_m \notin C$ and since $p_m \not\rightarrow$, T cannot be a derivation tree.

- $|\sigma| = \infty, p = p_0 \xrightarrow{e_1} p_1 \xrightarrow{e_2} \cdots, \sigma = e_1 e_2 \cdots$, and

$$(\forall n \in \mathbb{N}. (p_n \models \neg A_2) \vee (\exists 0 \leq i < n. p_i \models \neg A_1))$$

As before we extract two cases:

* $(\exists i \in \mathbb{N}.\ p_i \models A_2)$. Let $i_0 > 0$ be the least such index. As before we have a least index $0 \leq j_0 < i_0$ such that $p_{j_0} \models \neg A_1$. By repeating the argument above, we obtain a contradiction.

* $(\forall n \in \mathbb{N}.\ p_n \models \neg A_2)$. If there is a state which satisfies $\neg A_1$ along the path the argument above can be applied. Else we can obtain from σ a path $\sigma' \in [\sigma'] \in comp(p)$ such that $p \xrightarrow{\sigma'} = p'_0 \xrightarrow{e'_1} p'_1 \xrightarrow{e'_2} \cdots \xrightarrow{e'_{m-1}} p'_{m-1} \xrightarrow{e'_m} p''_{k_0} \xrightarrow{e'''_{k_0+1}} p''_{k_0+1} \cdots$ where $p'_0, \ldots, p'_{m-1}$ occurs only once along σ' while $p''_{k_0}, \ldots$ occurs infinitely often along σ'. (We simply remove a finite number of loops in σ since the underlying transition system is finite). The common suffix ensures that no event is continuously enabled along σ'.
Since no event is continuously enabled along σ' there must exist finitely many loops starting and ending at p''_{k_0}, $\sigma_{loop_1}, \ldots, \sigma_{loop_r}$, such that no event from $next(p''_{k_0})$ is continuously enabled along $(\sigma_{loop_1} \cdots \sigma_{loop_r})^\infty$ i.e. no enabled event at p_{k_0} is independent of all events taken the r loops. Notice that these loops might themselves contain loops. Moreover, since $|next(p''_{k_0})| \leq |E|$ we can assume that $r \leq |E|$. Let $next(p''_{k_0}) = \{e'''_{j_1}, \ldots, e'''_{j_r}\}$. We can also assume that σ_{loop_l} corresponds to a loop along which some event in conflict with e'''_{j_l} is taken.
From each loop σ_{loop_l} we can extract, by deleting inner loops, three loopfree subpaths σ'_{loop_l}, σ''_{loop_l}, and σ'''_{loop_l} such that
  · σ''_{loop_l} contains an event in conflict with e'''_{j_l}.
  · $p''_{k_0} \xrightarrow{\sigma'_{loop_l}} \xrightarrow{\sigma''_{loop_l}} \xrightarrow{\sigma'''_{loop_l}} p''_{k_0}$.
  · All states along this new loop satisfies $A_1 \wedge \neg A_2$.

But then, using the induction hypothesis, a prefix of the following path must be traceable in the proof tree T:

$$p \xrightarrow{e'_1} \cdots \xrightarrow{e'_m} p''_{k_0} \xrightarrow{\sigma'_{loop_1}} \xrightarrow{\sigma''_{loop_1}} \xrightarrow{\sigma'''_{loop_1}} \cdots \xrightarrow{\sigma'_{loop_r}} \xrightarrow{\sigma''_{loop_r}} \xrightarrow{\sigma'''_{loop_r}} p''_{k_0}$$

and must end in a leaf labelled $p''_{k_0} \vdash Ann(A_1)\ U_\forall^{(p''_{k_0}, n, \emptyset)}\ Ann(A_2)$, because the rules 9 and 12 keeps track (in the annotation) of which events have been concurrently enabled. In our case there are no such events. But then T cannot be a proof tree and we obtain the desired contradiction.

□

C Proof of undecidability of (finite) satisfiability problem

Some parts can be carried out essentially as in [18, 9] and these will therefore be omitted.
The following problem is known to be undecidable [2, 4].

Definition 19 *The colouring problem, CP.*
An instance of the problem is a quadruple $C = (C, R, U, c_0)$ where $C = \{c_0, \ldots, c_k\}$ is a finite nonempty set of colours and $R, U : C \to \mathcal{P}(C) - \{\emptyset\}$ are the "right" and "up" adjacency functions.
A solution to C is a function $Col : \mathbb{N} \times \mathbb{N} \to C$ such that:

* $Col(0,0) = c_0$
* $(\forall (i,j) \in \mathbb{N} \times \mathbb{N}.\ Col(i, j+1) \in U(Col(i,j)) \ \wedge\ Col(i+1, j) \in R(Col(i,j)))$

The following problem is known to be undecidable [9].

Definition 20 *The finite colouring problem, FCP.*
An instance of the problem is a quintuple $C_F = (C, R, U, c_0, c_f)$ where $C = \{c_0, \ldots, c_k\}$ is a finite nonempty set of colours, $c_f \in C$ and $R, U : C \to \mathcal{P}(C) - \{\emptyset\}$ are the "right" and "up" adjacency functions.
A solution to C_F is a triple (Col, M, N), where $M, N \in \mathbb{N}$, $Col : \{0, \ldots, M\} \times \{0, \ldots, N\} \to C$ is such that:

* $Col(0,0) = c_0$
* $(\forall 0 \leq i < M, 0 \leq j \leq N.\ Col(i+1, j) \in R(Col(i,j)))$
* $(\forall 0 \leq i \leq M, 0 \leq j < N.\ Col(i, j+1) \in U(Col(i,j)))$
* $Col(M, N) = c_f$

Claim 21 *The set of formulas that are injectively satisfiable is nonrecursive, i.e. the injective satisfiability problem is undecidable.*

Proof. We reduce the colouring problem to the injective satisfiability problem. Given $C = (C, R, U, c_0)$ we provide a formula A_C, a conjunct of five formulas given below, such that A_C is injectively satisfiable if and only if C has a solution.
We assume that all the labels $\alpha_0, \ldots, \alpha_k, up, right$ are distinct symbols. The five conjuncts are the following:

* $A_1 = <\alpha_0> tt$ — $Col(0,0) = c_0$
* $A_2 = G(\Leftrightarrow ((<right>, tt), (<up>, tt)))$ — coding of grid
* $A_3 = G(\bigwedge_{i=0}^{k}(<\alpha_i> tt \Leftrightarrow \bigwedge_{j \neq i}[\alpha_j]ff))$ — exactly one colour
* $A_4 = G(\bigwedge_{i=0}^{k}(<\alpha_i> tt \Rightarrow [right](\bigvee_{c_j \in R(c_i)} <\alpha_j> tt)))$ — right adjacency
* $A_5 = G(\bigwedge_{i=0}^{k}(<\alpha_i> tt \Rightarrow [up](\bigvee_{c_j \in U(c_i)} <\alpha_j> tt)))$ — up adjacency

We claim that $A_C = \bigwedge_{i+1}^{5} A_i$ is injectively satisfiable if and only if C has a solution. The "if" direction is easy and therefore omitted. The "only if" direction follows the lines in [9], and makes essential use of the axioms of *l-ATS* and injectivity. □

Claim 22 *The set of formulas that are finitely injectively satisfiable is nonrecursive, i.e. the finite injective satisfiability problem is undecidable.*

Proof.
We reduce the finite colouring problem to the finite injective satisfiability problem. Given $C_F = (C, R, U, c_0, c_f)$ we provide a formula A_{C_F}, a conjunct of seven formulas given below, s.t. A_{C_F} is finitely injectively satisfiable if and only if C_F has a solution.
Again we assume that all the labels $\alpha_0, \ldots, \alpha_k, UM, RM, up, right$ are distinct symbols. The seven conjuncts are the following:

- $A_1 = <\alpha_0> tt$ $\qquad Col(0,0) = c_0$

- $A_2 = G((\Leftrightarrow ((right, tt), (up, tt)) \wedge [RM]ff \wedge [UM]ff) \vee$
  $(<up> tt \wedge <RM> tt \wedge [right]ff \wedge [UM]ff) \vee$
  $(<right> tt \wedge <UM> tt \wedge [up]ff \wedge [RM]ff) \vee$
  $(<c_f> tt \wedge <RM> tt \wedge <UM> tt \wedge [right]ff \wedge [up]ff))$ $\qquad$ grid structure

- $A_3 = G(\bigwedge_{i=0}^{k}(<\alpha_i> tt \Leftrightarrow \bigwedge_{j\neq i}[\alpha_j]ff))$ $\qquad$ exactly one colour

- $A_4 = G(\bigwedge_{i=0}^{k}(<\alpha_i> tt \Rightarrow [right](\bigvee_{c_j \in R(c_i)} <\alpha_j> tt)))$ $\qquad$ right adjacency

- $A_5 = G(\bigwedge_{i=0}^{k}(<\alpha_i> tt \Rightarrow [up](\bigvee_{c_j \in U(c_i)} <\alpha_j> tt)))$ $\qquad$ up adjacency

- $A_6 = EV(<\alpha_f> tt \wedge <RM> tt \wedge <UM> tt)$ $\qquad$ c_f in upper right corner

- $A_7 = G(<RM> tt \Rightarrow [up] <RM> tt) \wedge$
  $G(<UM> tt \Rightarrow [right] <UM> tt)$ $\qquad$ consistent borders

We claim that $Ac_F = \bigwedge_{i=1}^{7} A_i$ is injectively satisfiable if and only if C_F has a solution. The "if" direction is easy and therefore omitted.

The "only if" direction is nontrivial and we must use another proof technique than [9] since they assume a fixed finite alphabet. We only assume finiteness about the set E.

So assume that we are given a finite injective *l-ATS* $K = (S,i,E,I,Trans,l{:}E \to Act)$ such that $i \models Ac_F$.

We will use the notation $s \xrightarrow{\alpha} s'$ to indicate that there is an event e such that $s \xrightarrow{e} s'$ and $l(e) = \alpha$. This shouldn't lead to any confusion since K is assumed injective. First we prove a property which ensures the existence of a solution given a certain kind of path from i.

Property: If there exists $s_0, \ldots, s_n \in S, e_1, \ldots, e_n \in E, s_0 \xrightarrow{e_1} \ldots \xrightarrow{e_n} s_n, i = s_0$ such that for all $1 \leq j \leq n$ either $l(e_j) = right$ or $l(e_j) = up$ and no state s_j except s_n has an enabled event labelled α_f then C_F has a solution.

Let $\beta_0, \ldots, \beta_n$ denote the unique label among $\{\alpha_0, \ldots, \alpha_k\}$ that by A_3 must label some enabled event at $s_0, \ldots, s_n$.

If $\beta_0 = \alpha_f$ then $n = 0$ and $\beta_0 = \alpha_0$ and obviously we have a solution. So we can assume that we have $n > 0$. By A_2 we must have one of the following.

- $s_0 \xrightarrow{right} s_1$ and $s_0 \xrightarrow{UM}$: By A_2, A_7, and our assumptions we conclude that $s_0 \xrightarrow{right} s_1 \xrightarrow{right} \ldots \xrightarrow{right} s_n$ and $\forall 0 \leq i \leq n.\ s_i \xrightarrow{UM}$. Since $s_n \xrightarrow{\alpha_f}$ we easily obtain a solution by A_4 and A_5.

- $s_0 \xrightarrow{up} s_1$ and $s_0 \xrightarrow{RM}$: Symmetric case of the above.

- $s_0 \xrightarrow{right}$ and $s_0 \xrightarrow{up}$: Without loss of generality we can assume that $l(e_1) = right$, i.e. $s_0 \xrightarrow{right} s_1$. By injectivity and A_2 we must have the existence of $s', s'' \in S$ such that $s_0 \xrightarrow{up} s' \xrightarrow{right} s''$ and $s_0 \xrightarrow{right} s_1 \xrightarrow{up} s''$. Continuing this way as long as $l(e_j) = right$ and exploiting injectivity we obtain the existence of $s', s'', \ldots, s^{(m+1)}$ such that

$$\begin{tikzcd}
s' \arrow[r, "right"] & s'' \arrow[r, "right"] & \cdots \arrow[r, "right"] & s^{(m+1)} \\
s_0 \arrow[r, "right"] \arrow[u, "up"] & s_1 \arrow[r, "right"] \arrow[u, "up"] & \cdots \arrow[r, "right"] & s_m \arrow[u, "up"]
\end{tikzcd}$$

If $m = n$ then since $s_n \xrightarrow{\alpha_f}$ we easily obtain a solution. So assume $m < n$. Then $l(e_{m+1}) = up$. By injectivity we conclude that $s^{(m+1)} = s_{m+1}$. Again, if $\beta_{m+1} = \alpha_f$ we are done by A_4 and A_5, so assume this isn't the case. Then $m+1 < n$. We now show how to expand the above $1 \times (m+1)$ grid to a $2 \times (m+1)$ grid if $l(e_{m+2}) = up$ or to a $1 \times (m+2)$ grid if $l(e_{m+2}) = right$.

- Assume that $l(e_{m+2}) = up$ is the case. Then by A_2, A_7, and injectivity we conclude the existence of an $r^{(m)}$ such that $s^{(m)} \xrightarrow{right} s^{(m+1)} \xrightarrow{up} s_{m+2}$ and $s^{(m)} \xrightarrow{up} r^{(m)} \xrightarrow{right} s_{m+2}$. By repeating this we finally obtain

$$\begin{tikzcd}
r' \arrow[r, "right"] & r'' \arrow[r, "right"] & \cdots \arrow[r, "right"] & r^{(m)} \arrow[r, "right"] & s_{m+2} \\
s' \arrow[r, "right"] \arrow[u, "up"] & s'' \arrow[r, "right"] \arrow[u, "up"] & \cdots \arrow[r, "right"] & s^{(m)} \arrow[r, "right"] \arrow[u, "up"] & s^{(m+1)} \arrow[u, "up"] \\
s_0 \arrow[r, "right"] \arrow[u, "up"] & s_1 \arrow[r, "right"] \arrow[u, "up"] & \cdots \arrow[r, "right"] & s_{m-1} \arrow[r, "right"] \arrow[u, "up"] & s_m \arrow[u, "up"]
\end{tikzcd}$$

- Assume that $l(e_{m+2}) = right$ is the case. Then by similar arguments we get:

$$\begin{tikzcd}
s' \arrow[r, "right"] & s'' \arrow[r, "right"] & \cdots \arrow[r, "right"] & s^{(m+1)} \arrow[r, "right"] & s_{m+2} \\
s_0 \arrow[r, "right"] \arrow[u, "up"] & s_1 \arrow[r, "right"] \arrow[u, "up"] & \cdots \arrow[r, "right"] & s_m \arrow[r, "right"] \arrow[u, "up"] & s^{(m+2)} \arrow[u, "up"]
\end{tikzcd}$$

This procedure can be continued for $e_{m+3}, \ldots, e_n$ giving us a grid from which it is easy to obtain a solution of C_F, by A_1, A_3, A_4, A_5.

This proves the claimed property.

Next we proof the existence of a path of the above mentioned form given the fact that $i \models Ac_F$.

So assume by contradiction that there doesn't exist any path of the above form. Then by A_2 and without loss of generality we can assume $i \xrightarrow{right}$. We now argue by cases.

- Assume $i \xrightarrow{UM}$: We know that $i \not\xrightarrow{\alpha_f}$ and choosing s_1 to be the unique state such that $i = s_0 \xrightarrow{right} s_1$ we also conclude $s_1 \not\xrightarrow{\alpha_f}$. Now by A_2, A_6, A_7, and our assumptions it must be the case that $s_1 \xrightarrow{right}$ and $s_1 \xrightarrow{UM}$. Continuing this way we exhibit an infinite path $s_0 \xrightarrow{right} s_1 \xrightarrow{right} \ldots$ such that for all states s_j along the path we have that $s_j \not\xrightarrow{\alpha_f}$ and $s_j \xrightarrow{UM}$. Since K is finite there must exist a least j_0 such that s_{j_0} is visited twice along $s_0 \xrightarrow{right} \ldots \xrightarrow{right} s_{j_0}$. Let $0 \leq j_1 < j_0$ be the index such that $s_{j_1} = s_{j_0}$.

This provides us with an infinite path $s_0 \xrightarrow{\gamma_1} s_{j_1} \xrightarrow{\gamma_2} s_{j_0} \xrightarrow{\gamma_1} s_{j_0} \xrightarrow{\gamma_2} \cdots$, where γ_1 is the sequence of *right* labelled events leading from s_0 to s_{j_1} and γ_2 from s_{j_1} to s_{j_0}. But by A_6 the above path cannot be a computation from s_0. Hence there must exist an event e' that is continuously enabled along $s_{j_0} \xrightarrow{\gamma_1} s_{j_0} \xrightarrow{\gamma_2} \cdots$. By the axioms of *l-ATS* we can obtain diagram 1, where $s'_{j_1} = s'_{j_0}$ is a state in S.

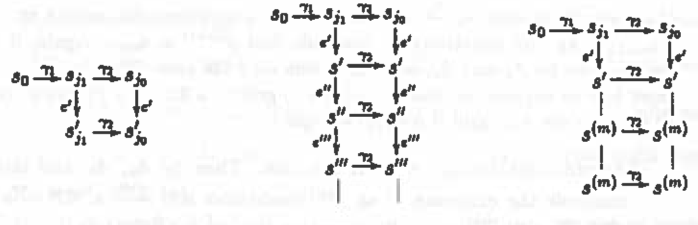


Diagram 1 Diagram 2 Diagram 3

Now again by A_2, γ_2 being labelled by *right*, and A_6 we conclude, that there must be an event e'' which is continuously enabled along the loop obtained by repeating $s'_{j_1} \xrightarrow{\gamma_2} s'_{j_0}$. By repeating this argument we obtain ($s' = s'_{j_1} = s'_{j_0}$) diagram 2.

Since K is finite, the set E must be finite. All of the reached states have the property that $<c_j> tt \wedge <RM> tt \wedge <UM> tt$ doesn't hold, since $<right> tt$ and A_2 hold. One can now repeat the argument observing that finiteness of S implies that some $s^{(m)}$ must occur along the leftmost vertical path in the above diagram. Again one must have the existence of an event that is continuously among the looping part of diagram 3.

Again the event must be independent of all the events in the looping part, especially the events labelled *right*. Also all states reached have an enabled transition labelled *right*.

Let K' denote the transition system obtained by restricting K to the states satisfying $\neg(<c_j> tt \wedge <RM> tt \wedge <UM> tt)$.

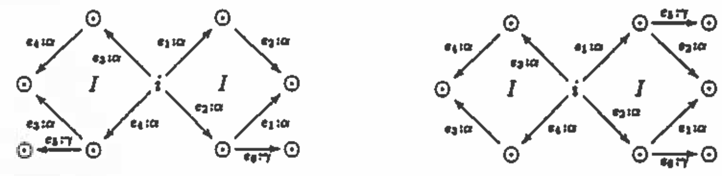
It should now be clear that one can produce an infinite computation from s_0 which stays in K'. One modifies the above infinite path by choosing events that are continuously enabled following the above scheme. It is important to notice that because it is always possible to insert loops containing events labelled *right* (the γ_2 loops) the states reached by "taking a continuously enabled event" also contains a self loop of events labelled *right*. Since K is finite there are only a finite number of maximal strongly connected components in K'. Hence by repeating the above procedure one will only be able to proceed towards terminal maximal strongly connected components. So eventually one is able to produce a computation along which $\neg(<c_j> tt \wedge <RM> tt \wedge <UM> tt)$ holds, contradicting A_6.

- Assume $i \xrightarrow{up}$: A similar way of reasoning leads to the desired contradiction. To sketch the argument: Choose *right* labelled events as far as possible. If one produces an infinite path labelled *right* the argument is as above. Else one must eventually reach a state with enabled events labelled *up* and *RM* (else contradicting A_6). From this state choose the infinite path labelled *up* (else contradicting A_6). Apply the above argument in a symmetric way.

□

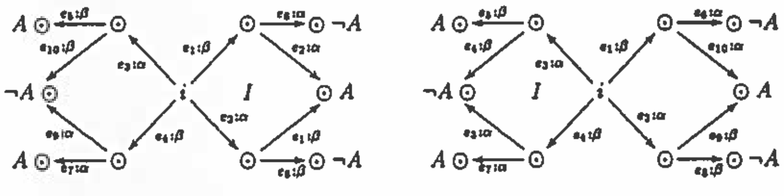
D Incomparability of Modal Operators

The $\Leftrightarrow$, $\langle\rangle$, $\sharp$, and $)($ operators might be replaced by others. We have chosen to present them because they distinguish *l-ATS* 's in the following situations (events that are independent are indicated by an I in their "independence square", states except the initial state are indicated by $\odot$, and finally states labelled by A or $\neg A$ indicate that one has to expand the *l-ATS* in a trivial manner, for e.g. $A \equiv <\gamma> tt$):



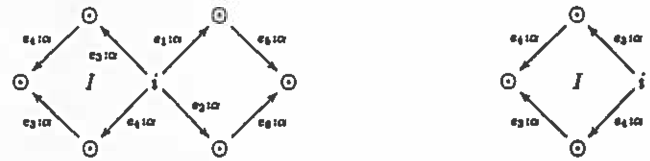
$\neg \Leftrightarrow ((\alpha, <\gamma> tt), (\alpha, <\gamma> tt))$ $\Leftrightarrow ((\alpha, <\gamma> tt), (\alpha, <\gamma> tt))$

The two above *l-ATS* 's cannot be distinguished by the $\langle\rangle$, $)($, or $\sharp$ operator.



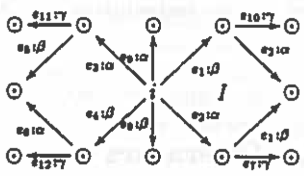
$\langle\alpha\beta\rangle A$ $\neg\langle\alpha\beta\rangle A$

The two above *l-ATS* 's cannot be distinguished by the $\Leftrightarrow$, $)($, or $\sharp$ operator.

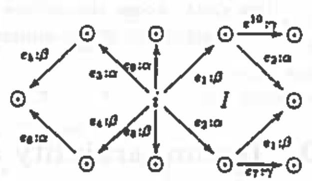


$)\alpha\alpha(tt$ $\neg)\alpha\alpha(tt$

The two above *l-ATS* 's cannot be distinguished by the $\Leftrightarrow$, $\langle\rangle$, or $\sharp$ operator.



$\sharp((\alpha, <\gamma> tt), (\beta, <\gamma> tt))$



$\neg\sharp((\alpha, <\gamma> tt), (\beta, <\gamma> tt))$

The two above *I-ATS* 's cannot be distinguished by the ⟺, ⟨⟩, or ⟩⟨ operator.

Recent Publications in the BRICS Report Series

RS-94-7 André Joyal, Mogens Nielsen, and Glynn Winskel. *Bisimulation from Open Maps*. May 1994, 42 pp. Journal version of LICS '93 paper.

RS-94-8 Javier Esparza and Mogens Nielsen. *Decidability Issues for Petri Nets*. 1994, 23 pp.

RS-94-9 Gordon Plotkin and Glynn Winskel. *Bistructures, Bidomains and Linear Logic*. May 1994, 16 pp. To appear in the proceedings of ICALP '94, LNCS, 1994.

RS-94-10 Jakob Jensen, Michael Jørgensen, and Nils Klarlund. *Monadic second-order Logic for Parameterized Verification*. May 1994, 14 pp.

RS-94-11 Nils Klarlund. *A Homomorphism Concept for ω-Regularity*. May 1994.

RS-94-12 Glynn Winskel and Mogens Nielsen. *Models for Concurrency*. May 1994, 144 pp. To appear as a chapter for the *Handbook of Logic and the Foundations of Computer Science*, Oxford University Press.

RS-94-13 Glynn Winskel. *Stable Bistructure Models of PCF*. May 1994, 26 pp. *Preliminary draft*. Invited lecture for MFCS 94. To appear in the LNCS proceedings of MFCS 94.

RS-94-14 Nils Klarlund. *The Limit View of Infinite Computations*. May 1994, 16 pp. To appear in proceedings of Concur '94.

RS-94-15 Mogens Nielsen and Glynn Winskel. *Petri Nets and Bisimulations*. May 1994, 36 pp.

RS-94-16 Lars Arge. *External-Storage Data Structures for Plane-Sweep Algorithms*. June 1994, 37 pp.

RS-94-17 Allan Cheng. *Local Model Checking and Traces*. June 1994, 30 pp.