Basic Research in Computer Science

BRICS RS-95-39

Petri Nets, Traces, and Local Model Checking

Allan Cheng

BRICS Report Series **RS-95-39**

ISSN 0909-0878 **July 1995**

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@brics.dk**

BRICS publications are in general accessible through WWW and anonymous FTP:

`http://www.brics.dk/`
`ftp ftp.brics.dk (cd pub/BRICS)`

Petri Nets, Traces, and Local Model Checking*

Allan Cheng**

Computer Science Department
Cornell University
Ithaca, New York 14853, USA
e-mail:acheng@cs.cornell.edu

Abstract. It has been observed that the behavioural view of concurrent systems that all possible sequences of actions are relevant is too generous; not all sequences should be considered as likely behaviours. Taking progress fairness assumptions into account one obtains a more realistic behavioural view of the systems. In this paper we consider the problem of performing model checking relative to this behavioural view. We present a CTL-like logic which is interpreted over the model of concurrent systems labelled 1-safe nets. It turns out that Mazurkiewicz trace theory provides a natural setting in which the progress fairness assumptions can be formalized. We provide the first, to our knowledge, set of sound and complete tableau rules for a CTL-like logic interpreted under progress fairness assumptions.

keywords: fair progress, labelled 1-safe nets, local model checking, maximal traces, partial orders, inevitability

1 Introduction

Recently, attention has focused on behavioural views of concurrent systems in which concurrency or parallelism is represented explicitly [Rei85, Maz86, Win86, Sta89, WN94]. This is done by imposing more structure on models for concurrent systems—in our case, an independence relation on the transitions.

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 branching time temporal logic, P-CTL, interpreted over the reachability graph of labelled 1-safe nets.

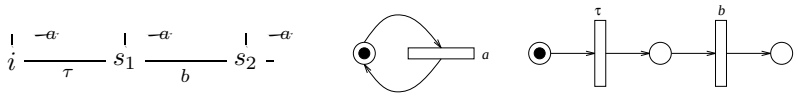
Labelled 1-safe nets are Petri nets whose transitions are labelled by actions from a set *Act* and whose reachable markings have at most one token on any

* Full version of paper appearing in the proceedings of AMAST'95, Springer-Verlag, (*LNCS* 936). This work has been supported by The Danish Research Councils and The Danish Research Academy.

** Visiting from **BRICS**, Basic Research in Computer Science, Centre of the Danish National Research Foundation, Computer Science Department, University of Aarhus, Denmark. e-mail:acheng@daimi.aau.dk

place. Labelled 1-safe nets are for example obtained by translating agents from various process algebras or constructed as the synchronization of finite automata.

As an example, consider the process agent $fix\,(X = a.X)|(\tau.b.\mathbf{0})$. Its transition graph is given below to the left. The initial state is i and s_1 and s_2 are the only other reachable states. The agent can also be represented by the labelled 1-safe net to the right, containing three transitions labelled a, τ, and b, respectively [MN92, WN94].



The net gives us a more concrete model of the process agent. It shows that the transition 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* (*lats*) [Shi85, Bed88, WN94, Old91]. In general, we can obtain such a labelled asynchronous transition system as the case graph, extended with implicit information about independence, of a labelled 1-safe net. In this paper, we will concentrate on labelled 1-safe nets.

The logic P-CTL contains one important feature: the model-theoretic incorporation of progress. Formulas corresponding to quantified "until" path formulas in CTL is in our setting interpreted over firing sequences of labelled 1-safe nets respecting certain *progress assumptions*. This is formalized using maximal traces in the framework of Mazurkiewicz trace-theory, where we make explicit use of the notion of independence between transitions. As an example, the formula $\mathsf{Ev}(<b>tt)$—to be read as"eventually b is enabled"—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 P-CTL interpreted over standard labelled transition systems (*lts*) coincides with the standard CTL interpretation. In process algebraic terms, our notion of fair progress—progress of independent events—intuitively corresponds to a progress fair "parallel operator".

Work on expressing *fairness assumptions* can be found in for example Manna and Pnueli's book on temporal logic [MP92]. Often it involves "coding" these assumptions using linear time temporal logic formulas of the form $\phi_{fair} \Rightarrow \psi$, which require a more detailed knowledge of the particular system. Also, the formula ϕ_{fair} depends on the specific system being analyzed. When handling progress fairness, we are able to avoid this obstacle and treat progress assumptions *uniformly* by using Mazurkiewicz trace-theory.

In the standard setting of CTL-like logics interpreted over *lts*, model checking has been described in [CES86] using a state based algorithm and in [Lar88, SW89] using tableaux rules. Model checking in the framework of partial order semantics has been described in [Pen93, PP90].

In this paper we present the first, to our knowledge, set of sound and complete tableau rules in the style of [Lar88, SW89] for a CTL-like logic interpreted in the trace theoretic framework. The rules are a generalization of those in [Lar88,

SW89] in the sense that if we restrict model checking to labelled 1-safe nets without independent transitions, our tableau rules work in the same way. Using the distinction between "local" and "global" model checking as advocated by Stirling and Walker in [SW89] 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 all the states of the system being investigated. This is, however, necessary for the global model checking algorithm of [CES86]. 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 Sect. 2, we provide the necessary definitions. In Sect. 3, we present the logic and its interpretation. Section 4 contains a motivating example followed by the tableau rules and the definition of tableaux. In Sect. 5, we present the main result, soundness and completeness of the proposed tableau rules, and state the complexity of our model checking problem. Finally, Sect. 6 contains the conclusion and suggestions for future work. The appendix contains various results related to P-CTL.

2 Basic Definitions

In this section we recall some basic definitions and state some facts and lemmas. We start by defining *concurrent alphabets*, the fundamental structure in Mazurkiewicz trace theory [Maz86].

Definition 1. Concurrent alphabet and traces

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

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

- 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 $\leq_{pref}$ be the usual prefix ordering on sequences and $\pi_{(a,b)}$ the projection on $\{a, b\}^\infty$. Define a preorder $\preceq$ on A^∞ which requires the relative order of elements a and b which are in conflict, i.e., $(a, b) \notin I$, to be the same when ignoring other elements of the sequences. Formally:

$$u \preceq v \text{ if and only if } (\forall (a, b) \notin I.\ \pi_{(a,b)}(u) \leq_{pref} \pi_{(a,b)}(v))$$

- Define an equivalence relation $\equiv$ on A^∞ by $u \equiv v$ if and only if $u \preceq v$ and $v \preceq u$. The elements of $A^\infty / \equiv$ are called *traces*. The equivalence class of u—the trace containing u—is denoted $[u]$.

- Fact: $\equiv$ is a congruence with respect to $\circ$.
- For $[u], [v] \in A^\infty/\equiv$ define $[u] \preceq [v]$ if and only if $u \preceq v$. It can be shown that $\preceq$ is a partial order. We write $[u] \prec [v]$ if and only if $u \preceq v$ and $u \not\equiv v$.
- Fact: for $u, v \in A^*$:
  - $[u] \preceq [v]$ if and only if $(\exists u' \in A^*.\ [uu'] = [v])$
  - $u \equiv v$ if and only if $u \equiv_M v$, where $\equiv_M$ is the well known equivalence on finite sequences presented in [Maz86] to define finite traces.

Example 1. Consider the concurrent alphabet (A, I), where $A = \{a, b, c\}$ and $I = \{(a, b)(b, a)\}$. Then, $abc \equiv bac$, $abc \not\equiv acb$, $(abbac)^\infty \equiv (aabbc)^\infty$, and $(abbac)^\infty \not\equiv (abcba)^\infty$.

Remark. We have chosen to present traces using projections $\pi_{(a,b)}$ because finite as well as infinite traces are handled in a uniform way. Similar definitions can be found in, e.g., [Kwi89].

We continue by defining *labelled 1-safe nets*, the labelled version of 1-safe nets.[3]

Definition 2. 1-safe nets

A *1-safe net*, or just a *net*, is a structure $N = (P, T, F, M_0)$ such that

- P and T are finite nonempty disjoint sets; their elements are called *places* and *transitions*, respectively.
- $F \subseteq (P \times T) \cup (T \times P)$; F is called the *flow relation*.
- $M_0 \subseteq P$; M_0 is called the *initial marking* of N; in general, a set $M \subseteq P$ is called a *marking* or a *state* of N.

Given $a \in P \cup T$, the *preset* of a, denoted ${}^\bullet a$, is defined as $\{a' \mid a'Fa\}$; the *postset* of a, denoted $a^\bullet$, is defined as $\{a' \mid aFa'\}$. The union of ${}^\bullet a$ and $a^\bullet$ is denoted ${}^\bullet a^\bullet$. The irreflexive symmetric *independence relation* I over T is defined by $t_1 I t_2$ if and only if ${}^\bullet t_1^\bullet \cap {}^\bullet t_2^\bullet = \emptyset$. Two transitions t_1 and t_2 are said to be *independent* if $t_1 I t_2$ and in conflict otherwise. Notice that (T, I) is a concurrent alphabet. For $D \subseteq T$ and $t \in T$ we define $tID = DIt = \{t' \in D \mid t' I t\}$.

Example 2. The net example from the introduction has three places and three transitions labelled τ, α, and β, respectively.

Definition 3. Firing sequences

Let $N = (P, T, F, M_0)$ be a net.

- A transition $t \in T$ is *enabled* at a marking M of N if ${}^\bullet t \subseteq M$ and $t^\bullet \cap (M - {}^\bullet t) = \emptyset$. Denote the set of transitions enabled at a marking M by *next*(M).

[3] An equivalent definition can be given in terms of Place/Transition nets, see [CEP93].

- Given a transition t, define a relation $\xrightarrow{t}$ between markings as follows: $M \xrightarrow{t} M'$ if and only if t is enabled at M and $M' = (M - {}^\bullet t) \cup t^\bullet$. The transition t is said to *occur* (or *fire*) at M. If $M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \cdots \xrightarrow{t_n} M_n$ for some markings $M_1, M_2, \ldots M_n$, then the sequence $\sigma = t_1 \ldots t_n$ is called an *occurrence sequence*. M_n is the marking *reached* by σ, and this is denoted $M_0 \xrightarrow{\sigma} M_n$. A marking M is *reachable* if it is the marking reached by some occurrence sequence. $M \not\rightarrow$ denotes that there are no enabled transitions at M, i.e., $next(M) = \emptyset$, in which case it is said to be *dead*.
- Given a marking M of N, the set of reachable markings of (P, T, F, M)—the net obtained replacing the initial marking M_0 by M—is denoted by $[M\rangle$.
- A *labelled* 1-safe net $N = (P, T, F, M_0, l)$ is a 1-safe net extended with a labelling function $l : T \to Act$ mapping each transition to an action in Act.

Example 3. The net example from the introduction has three reachable markings.

The behaviour of a net is captured by its reachability graph.

Definition 4. Reachability graph

The *reachability graph* of a net N is the edge-labelled graph $(V, E)_N$, whose set of vertices–or states—V is $[M_0\rangle$. The labelled edges are induced by the firing relations $\xrightarrow{t}$, and hence conveniently and safely denoted $M \xrightarrow{t} M'$.

Example 4. The reachability graph of the net example in the introduction is depicted to the left of the net. For convenience, we didn't label the edges by transitions, but rather theirs labels.

In the following we assume a fixed labelled 1-safe net N and consider its reachability graph $(V, E)_N$. We use the symbols $p, q, \ldots$ to denote states in $(V, E)_N$. If nothing else is mentioned, it is implicitly assumed that (T, I) is used to generate the congruence $\equiv$.

Definition 5. Paths

- Define a *path* from $p \in V$ as a sequence, finite or infinite, of transitions $t_1, t_2, \ldots$, for which there exists states $p_1, p_2, \ldots$, such that $p \xrightarrow{t_1} p_1 \xrightarrow{t_2} p_2 \cdots$. Notice that the firing rules of the net ensure the uniqueness of the p_i's, if they exist. We therefore also refer to $p \xrightarrow{t_1} p_1 \xrightarrow{t_2} p_2 \cdots$ as a path from p and use the notation $p \xrightarrow{\sigma}$, where $\sigma = t_1 t_2 \cdots$. Define $path(p) \subseteq T^\infty$ to be all paths from p.
  dead state.
- Define $comp(p)$ as the maximal elements of $path(p)/\equiv$ with respect to $\preceq$. For $\sigma \in [\sigma'] \in comp(p)$ we refer to $p \xrightarrow{\sigma}$ as a *computation* from p.

Example 5. In the process agent example from the introduction, τ is cc-enabled along $i \xrightarrow{a^\infty}$, when we use a, b, and τ to refer to the corresponding transitions. Also, $\tau b a^\infty$ is a computation from i while a^∞ is not.

Due to the firing rules of the nets, the congruence $\equiv$ respects the path property.

Lemma 6. *Given a net $N = (P, T, F, M_0)$, and a state p of $(V, E)_N$. Then,*

$$(\forall \sigma \in path(p).\ (\forall \sigma' \in [\sigma].\ p \xrightarrow{\sigma'}))\ .$$

Proof sketch. If σ is finite, the result easily follows from the commutativity of consecutive independent transitions. If σ is infinite, notice that by interchanging consecutive independent transitions of σ we conclude that any finite prefix of σ' is an element of $path(p)$. Since $path(p)$ is limit closed, we conclude $\sigma' \in path(p)$. □

Hence, $path(p)$ can be partitioned into elements of $T^\infty/\equiv$. Moreover, if σ is finite, then $p \xrightarrow{\sigma} q$ implies $(\forall \sigma' \in [\sigma].\ p \xrightarrow{\sigma'} q)$.

Definition 7. Continuously Concurrently Enabled Transitions

Given $\sigma \in path(p), |\sigma| = \infty, \sigma = t_1 t_2 \cdots$. A transition t is said to be *continuously concurrently enabled* (cc-enabled) along $p \xrightarrow{\sigma} = p \xrightarrow{t_1} p_1 \xrightarrow{t_2} p_2 \cdots$ if and only if t is enabled at some point along $p \xrightarrow{\sigma}$ and independent of the remaining transitions of σ from that point. Formally: $(\exists\, n \in I\!N.\ (\forall j \geq n.\ p_j \xrightarrow{t} \wedge\ tIt_{j+1}))$. From the irreflexivity of I, this implies that for the given n, $t \neq t_j$, $j \geq n$. Whenever p is clear from the context, t is said to be cc-enabled along σ.

The next two lemmas state properties of traces.

Lemma 8. *Given a concurrent alphabet (A, I). For $\sigma, \sigma' \in A^\infty$ we have that*

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

Proof. The "if" direction is proved by an easy contradiction argument. For the "only if" direction, first choose a finite prefix σ_1' of σ' such that its Parikh vector (for each $a \in A$ this vector provides the number of occurrences of a's 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'. Then for any $1 \leq j < j_1$ it must be the case that $b_j I b_{j_1}$, 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 $\sigma_1 \preceq \sigma_1'$. □

Lemma 9. *Given a net $N = (P, T, F, M_0)$, a state p of $(V, E)_N$, $\sigma \in path(p)$ such that $|\sigma| = \infty$, and $t \in T$ that is cc-enabled along σ. Then, for any $\sigma' \in [\sigma]$, t is cc-enabled along σ'), i.e., $\equiv$ respects cc-enabledness.*

Proof. Clearly, by definition there exists a finite $\sigma_1 \in path(p)$, a $p' \in S$, and a $\sigma_2 \in path(p')$ such that $p \xrightarrow{\sigma} = p \xrightarrow{\sigma_1} p' \xrightarrow{\sigma_2}$ and t is enabled at p' and independent of all transitions in σ_2. Choose any $\sigma' \in [\sigma]$. Since $\sigma \preceq \sigma'$, we have from the previous lemma that there exists a finite prefix of σ', say σ_1', such that $\sigma_1 \preceq \sigma_1'$.

Using the technique from the proof of the previous lemma we see that there exists a $\gamma \in T^\infty$, such that $\sigma_1\gamma \equiv \sigma_1'$ and all transitions in γ are independent of t. We conclude that t must be enabled at p'', where $p \xrightarrow{\sigma_1'} p''$, since $p \xrightarrow{\sigma_1\gamma} p''$. Choosing σ_2' such that $\sigma' = \sigma_1'\sigma_2'$ we also conclude that all transitions in σ_2' are independent of t, since all transitions in σ_2' are transitions of σ_2. Hence, t is cc-enabled along σ'. □

Hence, based on Lemma 9 we may safely write $t \in T$ is cc-enabled along $[\sigma]$, meaning t is cc-enabled along $\sigma \in path(p)$.

Next, we identify maximal traces as maximal elements in a partial order. The following lemma explains why we focus on these traces. They represent executions—of a concurrent system—which are fair with respect to progress of independent processes. In [MOP89] 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 weak assumption, see [MP92] for a comparison.

Lemma 10. *Given a labelled 1 safe net $N = (P, T, F, M_0, l)$ and a state p of $(V, E)_N$. For $[\sigma] \in comp(p)$ such that $|\sigma| = \infty$ we have*

$$(\exists\, [\sigma'] \in comp(p).[\sigma] \prec [\sigma']) \text{ if and only if}$$

$$(\exists\, t \in T.\ t \text{ is cc-enabled along } \sigma)\ .$$

Proof. The "if" direction is easy, and hence omitted. For the "only if" direction, first we observe the following: since $[\sigma] \prec [\sigma']$, there must exist a $t \in T$ such that $\pi_{(t,t)}(\sigma) < \pi_{(t,t)}(\sigma')$. Clearly, $|\pi_{(t,t)}(\sigma)| = n < \infty$ for some $n \in I\!N$. Let $\sigma = \sigma_1\sigma_2$, where $\#_t(\sigma_1) = n, \#_t(\sigma_2) = 0$ and $|\sigma_1| < \infty$. By Lemma 8 we know that there exists a finite prefix σ_3 of σ' such that $[\sigma_1] \preceq [\sigma_3]$. Furthermore, there must exists a suffix of σ such that all transitions of it are independent of t. To see this, assume that there were infinitely many indexes $i_j \in I\!N$ for $0 \leq j$ such that $(t_{i_j}, t) \notin I$, where $\sigma = t_1 t_2 \cdots$. Since $(\forall j \in I\!N.\ \pi_{(t,t_{i_j})}(\sigma) < \pi_{(t,t_{i_j})}(\sigma'))$, all t_{i_j}'s must occur before the $(n+1)$'th t in $\pi_{(t,t_{i_j})}(\sigma')$. But this clearly means that there must be infinitely many transitions between the n'th and $(n+1)$'th t in σ', which is impossible.

Next, we show that there must exist a transition t' which is cc-enabled along σ. First, choose the first occurrence of a $t' \in T$ along σ' such that $\#_{t'}(\sigma) < \#_{t'}(\sigma')$. Next, split σ into σ_1 (finite) and σ_2 such that $\sigma = \sigma_1\sigma_2$ and all transitions in σ_2 are independent of t'. Then, choose σ_1' as the shortest prefix of σ' such that $\#_{t'}(\sigma_1') = \#_{t'}(\sigma_1) + 1$ 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 transitions and obtain $\sigma_1' \equiv \sigma_1\gamma \in path(p)$. Now $\#_{t'}(\gamma) > 0$. Let $\gamma = t_1' \cdots t_r'\, t'\, t_1'' \cdots t_s''$, where $r, s \geq 0$ and all t_i''s are different from t'. Now assume that $(\exists 1 \leq j \leq r.\, (t_j', t') \notin I)$. Choose the first such j. Then, $\pi_{(t_j',t')}(\sigma_1') > \pi_{(t_j',t')}(\sigma_1)$ and since the relative occurrence of t' and t_j''s in σ_1' and $\sigma_1\gamma$ are the same, a t_j' must occur before the

$(\#_{t'}(\sigma_1)+1)$'th t' in σ'. But $\#_{t'_i}(\sigma) = \#_{t'_i}(\sigma')$ by choice of t'. Then, there must exist a t'_i in σ_2 and this contradicts the assumption that all transitions in σ_2 were independent of t'. By using the properties of $(V,E)_N$ and I (e.g., permutation of consecutive independent transitions: if $M \xrightarrow{t} M' \xrightarrow{t'} M''$ and tIt', then there exists an M''' such that $M \xrightarrow{t'} M''' \xrightarrow{t} M''$),[4] we conclude that t' must be enabled at p' where $p \xrightarrow{\sigma_1} p'$. Hence, t' is cc-enabled along σ. □

3 The Logic P-CTL and its Interpretation

In this section, we assume a fixed labelled 1-safe net $N = (P,T,F,M_0,l)$. P-CTL has the following syntax, where $\alpha \in Act$.

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

tt is an abbreviation for *TRUE*. In Hennessy-Milner logic [Mil89], $<a>A$ expresses the fact that one can perform an action a from a state and, in doing so, reach another state at which A holds. Similarly, the $\bigcirc_\alpha A$ expresses that a transition labelled α can be fired reaching a state where A holds. The "until" operators $U_\exists$ and $U_\forall$ are introduced as generalizations of their counterparts in [CES86], here interpreted over maximal traces, following Mazurkiewicz [Maz86].

The logic is interpreted over the reachability graph $(V,E)_N$ of N as follows, where $p \in V$, $\alpha \in Act$, and we have written $\models$ instead of $\models_N$ since N was fixed.

- $p \models tt$
- $p \models \neg A$ if and only if $p \not\models A$
- $p \models A_1 \ \wedge \ A_2$ if and only if $p \models A_1$ and $p \models A_2$
- $p \models \bigcirc_\alpha A$ if and only if $(\exists t \in T, q \in V.\ l(t) = \alpha \ \wedge \ p \xrightarrow{t} q \ \wedge \ q \models A)$
- $p \models A_1 \, U_\exists \, A_2$ if and only if $(\exists\, [\sigma] \in comp(p),\ p \xrightarrow{\sigma} = p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_2} p_2 \cdots.$ $(\exists\, 0 \le n \le |\sigma|.\, (p_n \models A_2) \ \wedge \ (\forall 0 \le i < n.\, p_i \models A_1)))$
- $p \models A_1 \, U_\forall \, A_2$ if and only if $(\forall [\sigma'] \in comp(p).\ (\forall\, \sigma \in [\sigma'],\ p \xrightarrow{\sigma} = p \xrightarrow{t_1} p_1 \xrightarrow{t_2} p_2 \cdots.$ $(\exists\, 0 \le n \le |\sigma|.\, (p_n \models A_2) \ \wedge \ (\forall 0 \le i < n.\, p_i \models A_1))))$

Furthermore, we define $\mathit{ff} \equiv \neg tt$, $<\alpha> A \equiv \bigcirc_\alpha A$, $[\alpha]A \equiv \neg <\alpha> \neg A$, $\mathsf{F}(A) \equiv tt \, U_\exists \, A$, $\mathsf{G}(A) \equiv \neg \mathsf{F}(\neg A)$, $\mathsf{Ev}(A) \equiv tt \, U_\forall \, A$, and $\mathsf{Al}(A) \equiv \neg \mathsf{Ev}(\neg A)$. The intended meaning of $\mathsf{Ev}(A)$ is that eventually/inevitably A will hold along any computation, while $\mathsf{Al}(A)$ means that along some computation A always holds.

Example 6. In the process agent example from the introduction we have $i \models \mathsf{Ev}(<b>tt)$.

[4] To be more precise, we use the axioms of the corresponding l-ATS, which intuitively is $(V,E)_N$ augmented with I [WN94].

Having given the necessary definitions, we end this section by defining the model checking problem.

Definition 11. Given a labelled 1-safe net $N = (P, T, F, M_0, l)$ and a formula A. The *model checking problem of N and A* is the problem of deciding whether or not $M_0 \models A$.

4 A Tableau Method for Model Checking

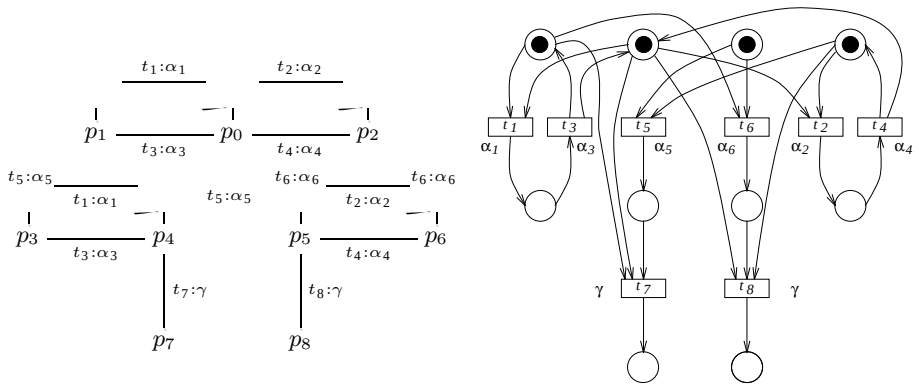
In this section we present a local model checker based on a tableau system for model checking formulas from our logic.

Local model checking based on tableau systems has been presented in [SW89]. As opposed to a global model checker—as the on presented in [CES86]—which checks if all states of the system satisfies 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 (other) 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 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 bigger than the size of the net, a local model checker sometimes has an advantage over a global model checker, since it can perform model checking using less memory.

We begin by considering an example to give 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 when there is no concurrency in the nets, we would also like the tableau system to be a conservative extension of those presented in [Lar88, SW89]. The main difficulty is how to generalize the unfolding of formulas in P-CTL which correspond to minimal fixed-point assertions.

4.1 Unfolding Minimal Fixed-Point Assertions

Below we consider a very simple reachability graph g_1, which is generated by the 1-safe net N_1 to the right.



The t_i's are the transitions, the Greek letters the labels, and p_0 the initial marking. The independence relation is the smallest such containing (t_1, t_5), (t_3, t_5), (t_2, t_6), and (t_4, t_6). Clearly, $p_0 \models_{N_1} \neg\mathsf{Ev}(<\gamma>tt)$ because $[(t_1t_3t_2t_4)^\infty] \in comp(p_0)$ and no state along the computation $(t_1t_3t_2t_4)^\infty$ satisfies $<\gamma>tt$. However, if we drop the transitions t_2, t_4, t_6, and t_8 and call this reduced net N_2, we do indeed have $p_0 \models_{N_2} \mathsf{Ev}(<\gamma>tt)$, since every computation from p_0 must eventually reach p_4 — t_5 cannot be continuously ignored while repeatedly firing t_1 and t_3; they are both *independent* of t_5.

Let us consider what a tableau (proof tree) for $p_0 \models_{g_2} \mathsf{Ev}(<\gamma>tt)$ might look like:

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

The above tree is constructed according to some intuitive tableau rules. Although informal, the example provides the first important observation. The leftmost branch begins and ends with the sequent $p_0 \vdash \mathsf{Ev}(<\gamma>tt)$. In the μ-calculus $\mathsf{Ev}(<\gamma>tt)$ is expressed by the formula $\mu X.\ <\gamma>tt \vee ([Act]X \wedge\ <A>tt)$. Hence, based on the tableau methods from [Lar88, SW89] one might expect that the above tree should be discarded as a tableau since 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, $(p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_3} p_0)^\infty$, is not a computation from p_0 since the transition t_5 is cc-enabled. This example suggests that in certain cases we might allow the unfolding of a minimal fixed-point assertion to reach itself. These cases should include the existence of a transition that is cc-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 transitions which are cc-enabled and update this

information as one unfolds the reachability graph via the tableau rules. So in our case t_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 g_1. Again, we construct in an intuitive and informal manner a tree rooted in the sequent $p_0 \vdash \mathsf{Ev}(<\gamma>tt)$:

$$\dfrac{p_0 \vdash \mathsf{Ev}(<\gamma>tt)}{\text{Two subtrees as above} \quad \dfrac{p_5 \vdash \mathsf{Ev}(<\gamma>tt)}{\dfrac{p_5 \vdash <\gamma>tt}{p_8 \vdash tt}} \quad \dfrac{p_2 \vdash \mathsf{Ev}(<\gamma>tt)}{\dfrac{p_6 \vdash \mathsf{Ev}(<\gamma>tt)}{\dfrac{p_5 \vdash \mathsf{Ev}(<\gamma>tt)}{\dfrac{p_5 \vdash <\gamma>tt}{p_8 \vdash tt}}} \quad p_0 \vdash \mathsf{Ev}(<\gamma>tt)}}$$

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 transitions which are cc-enabled — t_5 for the left branch and t_6 for the right branch. So according to the previous remarks, these branches shouldn't discard the tree from being a tableau. But we do wish to discard the tree as a tableau since $p_0 \models_{g_1} \neg\mathsf{Ev}(<\gamma>tt)$. The problem is that by composing the two loops $(p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_3} p_0)$ and $(p_0 \xrightarrow{t_2} p_2 \xrightarrow{t_4} p_0)$ we can obtain an infinite path $(p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_3} p_0 \xrightarrow{t_2} p_2 \xrightarrow{t_4} p_0)^\infty$. Along this path there is no transition which is cc-enabled, that is, 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 tableau.

One solution to the problem of detecting such "combined" loops is to continue to unfold the minimal fixed-point assertions $p_0 \vdash \mathsf{Ev}(<\gamma>tt)$. 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 transition is cc-enabled. This could discard the tree from being a tableau.

It turns out that the remaining problem is to find some general bound on the number of times we allow the unfolding of a minimal fixed-point assertion. In the next section we provide the necessary definitions. The bound we use is at most $|T|$, the number of transitions in the labelled 1-safe net.

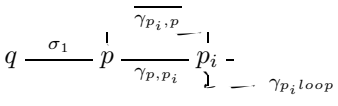
4.2 Tableau Rules

In this section we consider a fixed labelled 1-safe net N and its reachability graph $(V, E)_N$.

We want to perform "local" model checking by unfolding parts of the reachability graph into a tree structure. The tableau rules are supposed to guide this unfolding by imposing constraints which restrict the size and shape of the tree structure. The main difficulty is handling the $U_\forall$ operator.

Consider a state q such that $q \not\models A_1 \; U_\forall \; A_2$. Then either there exists (i) a computation σ such that $A_1 \wedge \neg A_2$ holds at all states along $q \xrightarrow{\sigma}$ and either

a deadlock is reached or a state such that $\neg A_1 \ \wedge \ \neg A_2$ holds reached, or (ii) an infinite computation σ such that $A_1 \ \wedge \ \neg A_2$ holds at all states along $q \xrightarrow{\sigma}$, referred to as an invalidating computation. Since the formulas are interpreted at states and the state space is finite, case (ii) reduces (simply by removing a finite number of loops from σ) to the existence of an infinite computation $\sigma_1\sigma_2$ from q, where σ_1 is finite and all states along $q \xrightarrow{\sigma_1}$ occur only once along $q \xrightarrow{\sigma_1} p \xrightarrow{\sigma_2}$ while all states along $p \xrightarrow{\sigma_2}$ occur infinitely often. Also, $A_1 \ \wedge \ \neg A_2$ holds at all states, as will be the case for the following computations. Using Lemma 10 it is possible to obtain from σ_2 an infinite path σ_3 from p of the form $(\gamma_{p,p_1}\gamma_{p_1 loop}\gamma_{p_1,p}\cdots\gamma_{p,p_k}\gamma_{p_k loop}\gamma_{p_k,p})^\infty$, where all γ's are finite and made up from subsequences of σ_2 and $1 \leq k \leq |T|$. The indices are intended to illustrate the structure of the loops as follows.



Also, since σ_2 was a computation from p, the γ's can be chosen such that for any $t \in \mathit{next}(p)$ one of the $\gamma_{p_i loop}$'s will contain a transition in conflict with t. Hence, σ_3 is a computation from p. We refer to the illustrated loops as *critical loops*. To conclude, $\sigma_1\sigma_3$ is an invalidating computation from q along which all states satisfy $A_1 \ \wedge \ \neg A_2$.

In the example from Sect. 4.1, if we chose p_0 as p, then $p_0 \xrightarrow{t_1}\xrightarrow{t_3} p_0$ and $p_0 \xrightarrow{t_2}\xrightarrow{t_4} p_0$ would constitute critical loops. Actually, we can bound the sizes of the γ's since the state space is finite. The important observation is that together with $|T|$ we have a bound on the length and number of γ's we have to consider. This is what we will encode in the the tableau system. For that purpose we define an annotated logic.

The Annotated Logic. The syntax of the annotated logic which is used in the tableau rules differs only from the previous in that the $U_\exists$ and $U_\forall$ operators are replaced by labelled counterparts. The $U_\exists$ operator is replaced by $U_\exists^{\mathcal{C}}$, where $\mathcal{C} \subseteq V$. The intuition is that $\mathcal{C}$ keeps track of which states have been visited and prevents unnecessary unfolding. For the $U_\forall$ operator we use a more elaborate annotation, $U_\forall^{\mathcal{C}}$, $U_\forall^{(p,n,T')}$, $U_\forall^{(p,n,T',V',\rightarrow)}$, and $U_\forall^{(p,n,T',V',\leftarrow)}$, where $p \in V$, $T' \subseteq T$, $V' \subseteq V$, and $0 \leq n \leq |T|$. V' plays a role similar to $\mathcal{C}$, n bounds the number of critical loops the tableau rules allow to explore, and T' keeps track of which transitions have been concurrently enabled but ignored so far along a path. The emptyness of T' will indicate that an invalidating computation has been found.

Let Ann be the obvious homomorphism which annotates a formula A (generated by the grammar in Sect. 3) by transforming every $U_\exists$ and $U_\forall$ into $U_\exists^\emptyset$ and $U_\forall^\emptyset$, respectively. An annotated formula B is said to be *clean* if there exists a formula A such that B equals $\mathit{Ann}(A)$.

The Tableau Rules. The tableau rules will consist of rules for sequents of the form $p \vdash B$. The rules can be read from top to bottom as: "the top sequent holds (B holds at p) if the bottom sequents and side conditions hold". B, B_1, and B_2 are assumed to be clean annotated formulas.

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

2) $$\frac{p \vdash \bigcirc_\alpha B}{q \vdash B}$$ - $t \in T, q \in V, p \xrightarrow{t} q$, - $l(t) = \alpha$.

3) $$\frac{p \vdash B_1 \ U_\exists^{\mathcal{C}} \ B_2}{p \vdash B_2}$$ - $p \notin \mathcal{C}$.

4) $$\frac{p \vdash B_1 \ U_\exists^{\mathcal{C}} \ B_2}{p \vdash B_1 \quad q \vdash B_1 \ U_\exists^{\mathcal{C} \cup \{p\}} \ B_2}$$ - $p \notin \mathcal{C}, t \in T, q \in V, p \xrightarrow{t} q$.

5) $$\frac{p \vdash B_1 \ U_\forall^{\mathcal{C}} \ B_2}{p \vdash B_2}$$ - $p \notin \mathcal{C}$.

6) $$\frac{p \vdash B_1 \ U_\forall^{\mathcal{C}} \ B_2}{p \vdash B_1 \quad q_1 \vdash B_1 \ U_\forall^{\mathcal{C} \cup \{p\}} \ B_2 \cdots q_m \vdash B_1 \ U_\forall^{\mathcal{C} \cup \{p\}} \ B_2}$$ - $next(p) = \{t_1, \ldots, t_m\}$, $0 < m \in I\!N, p \notin \mathcal{C}$, - $(\forall\, 1 \leq i \leq m.\ p \xrightarrow{t_i} q_i)$.

7) $$\frac{p \vdash B_1 \ U_\forall^{\mathcal{C}} \ B_2}{p \vdash B_1 \ U_\forall^{(p, |next(p)|, next(p))} \ B_2}$$ - $p \in \mathcal{C}$.

8) $$\frac{p \vdash B_1 \ U_\forall^{(p,n,T')} \ B_2}{p \vdash B_1 \ U_\forall^{(p,n-1,T',\emptyset,\rightarrow)} \ B_2}$$ - $0 < n \in I\!N, T' \neq \emptyset$.

9) $$\frac{q \vdash B_1 \ U_\forall^{(p,n,T',V',\rightarrow)} \ B_2}{q \vdash B_1 \quad q_i \vdash B_1 \ U_\forall^{(p,n,t_i IT',V' \cup \{q\},\rightarrow)} \ B_2}$$ - $q \notin V', next(q) = \{t_1, \ldots, t_m\}$, $0 < m \in I\!N$, - $(\forall\, 1 \leq i \leq m.\ q \xrightarrow{t_i} q_i)$,

10) $$\frac{q \vdash B_1 \ U_\forall^{(p,n,T',V',\rightarrow)} \ B_2}{q \vdash B_2}$$ - $q \notin V'$.

11) $$\frac{q \vdash B_1 \ U_\forall^{(p,n,T',V',\rightarrow)} \ B_2}{q \vdash B_1 \ U_\forall^{(p,n,T',\emptyset,\leftarrow)} \ B_2}$$ - $q \in V'$.

12) $$\frac{q \vdash B_1 \ U_\forall^{(p,n,T',V',\leftarrow)} \ B_2}{q \vdash B_1 \quad q_i \vdash B_1 \ U_\forall^{(p,n,t_i IT',V' \cup \{q\},\leftarrow)} \ B_2}$$ - $q \notin V', next(q) = \{t_1, \ldots, t_m\}$, $0 < m \in I\!N, q \neq p$, - $(\forall\, 1 \leq i \leq m.\ q \xrightarrow{t_i} q_i)$.

13) $$\frac{q \vdash B_1 \ U_\forall^{(p,n,T',V',\leftarrow)} \ B_2}{q \vdash B_2}$$ - $q \notin V'$.

14) $$\frac{p \vdash B_1 \ U_\forall^{(p,n,T',V',\leftarrow)} \ B_2}{p \vdash B_1 \ U_\forall^{(p,n,T')} \ B_2}$$

Rule 1 to 4 need no further explanation. Referring to the notation from Sect. 4.2, Rule 5 and 6 should detect σ_1, Rule 7 should detect the "switch" to σ_3, Rule

should 8 to 10 detect $\gamma_{p,p_i}\gamma_{p_i loop}$, Rule 11 should detect the "switch" to $\gamma_{p_i,p}$, and Rule 12 to 14 should detect $\gamma_{p_i,p}$.

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

The Derivation Trees and Tableaux. In this section we define the tableaux. 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 of the formulas, to a subclass of derivation trees which will be defined to be the tableaux.

Derivation trees are defined inductively in the usual manner, except perhaps for negation. That is, if $\mathcal{T}_1, \ldots, \mathcal{T}_n$ are derivation trees with roots matching the sequents under the bar of a rule and the side conditions are 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. A tree consisting of a single node labelled with one of the following sequents is a derivation tree.

- $p \vdash tt$
- $p \vdash \neg B$
- $p \vdash B_1\ U_\forall^{(p,n,T')}\ B_2$, where $n = 0$ or $T' = \emptyset$
- $q \vdash B_1\ U_\forall^{(p,n,T',V',\leftarrow)} B_2$, where $q \in V'$

By applying the rules we can obtain new derivation trees, for example:

- If $\mathcal{T}_1$ is a derivation tree with root $p \vdash B_1$, $\mathcal{T}_2$ is a derivation tree with root $q \vdash B_1\ U_\exists^{\mathcal{C}\cup\{p\}}\ B_2$, where $p \notin \mathcal{C}$, and there exists a $t \in T$ such that $p \xrightarrow{t} q$, then $\dfrac{p \vdash B_1\ U_\exists^{\mathcal{C}}\ B_2}{\mathcal{T}_1 \quad \mathcal{T}_2}$ is a derivation tree with root $p \vdash B_1\ U_\exists^{\mathcal{C}}\ B_2$.
- If $\mathcal{T}$ is a derivation tree with root $p \vdash B_2$ and $p \notin \mathcal{C}$, then $\dfrac{p \vdash B_1\ U_\forall^{\mathcal{C}}\ B_2}{\mathcal{T}}$ is a derivation tree with root $p \vdash B_1\ U_\forall^{\mathcal{C}}\ B_2$.

Nothing else is a derivation tree.

We continue by defining the tableaux. In this step we get rid of derivation trees as for example $p \vdash \neg tt$. Sequents of the form $q \vdash B_1\ U_\forall^{(q,n,\emptyset)}\ B_2$, where $n \in I\!N$ and $q \in V$, are called terminal sequents. A tableau is a derivation tree $\mathcal{T}$ with root $p \vdash Ann(A)$ such that either

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

A sequent $p \vdash B$ is *proved* by exhibiting a tableau with root $p \vdash B$.

5 Soundness and Completeness of the Tableau Method

Having given the necessary definitions we are now ready to state the main result.

Theorem 12. *Given a finite labelled net $N = (P, T, F, M_0, l)$. Then for any state p of $(V, E)_N$ ($p \in V$) and any formula A we have:*

$$p \models A \text{ if and only if there exists a tableau with root } p \vdash Ann(A)$$

Proof. The proof proceeds by structural induction in A, showing soundness and completeness simultaneously. The main difficulty is the $U_\forall$ operator. For the soundness part, our observations from Sect. 4.2 provide the basis for a proof by contradiction. For the completeness part, using the induction hypothesis one can give a direct construction of a tableau. Intuitively, if $p \models A_1\ U_\forall\ A_2$, then a tableau will be constructed (top-down from p) by always proving $q \vdash Ann(A_2)$ if $q \models A_2$ for any reached state q. Else, if $q \not\models A_2$, then one proves $q \vdash Ann(A_1)$, starts unfolding the graph from q, and continues by trying to prove $Ann(A_1\ U_\forall\ A_2)$ at the states that are reached.

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

$\underline{\neg}$ We have $p \models \neg A$ if and only if $p \not\models A$ if and only if (induction hypothesis) there exists no tableau with root $p \vdash Ann(A)$ if and only if $p \vdash \neg Ann(A)$ is a tableau.

$\underline{\wedge}$ We have $p \models A_1 \wedge A_2$ if and only if $p \models A_1$ and $p \models A_2$ if and only if (induction hypothesis) there exists tableaux $\mathcal{T}_1$ with root $p \vdash Ann(A_1)$ and $\mathcal{T}_2$ with root $p \vdash Ann(A_2)$ if and only if there exists a tableau with root $p \vdash Ann(A_1 \wedge A_2)$, because $Ann(A_1 \wedge A_2) = Ann(A_1) \wedge Ann(A_2)$.

$\underline{\bigcirc_\alpha}$ We have $p \models \bigcirc_\alpha A$ if and only if $(\exists t \in T, q \in V.\ p \xrightarrow{t} q\ \wedge\ q \models A\ \wedge\ l(t) = \alpha)$ if and only if (induction hypothesis) there exists a tableau $\mathcal{T}$ and $(\exists t \in T, q \in V.\ p \xrightarrow{t} q\ \wedge\ l(t) = \alpha\ \wedge\ \mathcal{T}$ has root $q \vdash Ann(A))$ if and only if there exists a tableau $\mathcal{T}$ with root $p \vdash Ann(\bigcirc_\alpha A)$, since $Ann(\bigcirc_\alpha A) = \bigcirc_\alpha Ann(A)$.

$\underline{U_\exists}$ We have $p \models A_1\ U_\exists\ A_2$ if and only if $(\exists\ p_1, p_2, \ldots, p_n \in V, t_1, t_2, \ldots, t_n \in T, n \geq 0.$
$p = p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_2} p_2 \cdots \xrightarrow{t_n} p_n \wedge p_n \models A_2 \wedge (\forall 0 \leq i < n.\ p_i \models A_1) \wedge (\forall 0 \leq i < j \leq n.\ p_i \neq p_j))$ if and only if (induction hypothesis) there exists tableaux $\mathcal{T}_0, \ldots, \mathcal{T}_{n-1}$ with roots $p_0 \vdash Ann(A_1)\ , \ldots,\ p_{n-1} \vdash Ann(A_1)$ and $\mathcal{T}_n$ with root $p_n \vdash Ann(A_2)$ and transitions $t_1, \ldots, t_n$ such that $p = p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_2} p_2 \cdots \xrightarrow{t_n} p_n$ is loop free if and only if there exists a tableau 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.

"only if" direction (completeness):

We show how to obtain a derivation tree with root $p \vdash Ann(A_1\ U_\forall\ A_2)$. 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 tableau.

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 as 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 of the algorithm several claims will be stated. All of them will be shown to be valid in the succeeding paragraph. For convenience, we will write 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. Start by creating the root which 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) $q \vdash B_1\ U_\forall^{\mathcal{C}}\ B_2$
   ii) $q \vdash B_1\ U_\forall^{(q,n,T')}\ B_2$
   iii) $q \vdash B_1\ U_\forall^{(p,n,T',S',\leftrightarrow)}\ B_2$
   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 tableau $\mathcal{T}'$ with root $q \vdash B_2$. Deactivate N and paste $\mathcal{T}'$ below N using rule 5, 10, or 13. None of the added nodes are active. Note that $q \models A_2$ excludes *ii)* because of the way the current tree has been expanded.
4. Else if $q \models \neg A_2$, then necessarily (Claim 1) $q \models A_1$. By induction there exists a tableau $\mathcal{T}'$ with root $q \vdash B_1$.
   * If N is of the form *i*), and $q \notin \mathcal{C}$, then (Claim 2) $next(q) \neq \emptyset$ and apply rule 6, using $\mathcal{T}'$. Deactivate N and activate the new leaves labelled $q_i \vdash B_1\ U_\forall^{\mathcal{C}\cup\{q\}}\ B_2$ that were added by application of rule 6.
   * If N is of the form *i*) and $q \in \mathcal{C}$, then deactivate N and, using rule 7, add a node below N labelled $q \vdash B_1\ U_\forall^{(q,|T|,next(q))}\ B_2$. Using rule 8, because (Claim 3) $next(q) \neq \emptyset$, add yet another node below labelled $q \vdash B_1\ U_\forall^{(q,|T|-1,next(q),\emptyset,\rightarrow)}\ B_2$ which is activated.
   * If N is of the form *ii*), then (Claim 4) $T' \neq \emptyset$. If $n = 0$, then deactivate N. Else if $n > 0$, then deactivate N and apply rule 8, adding a node below N labelled $q \vdash B_1\ U_\forall^{(q,n-1,T',\emptyset,\rightarrow)}$. 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 tableau $\mathcal{T}'$ with root labelled $q \vdash B_1$. Using rule 9 add this tree below N and add nodes labelled $q_i \vdash B_1\ U_\forall^{(p,n,t_iIT',S'\cup\{q\},\rightarrow)}\ B_2$. 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 $q \vdash B_1\ U_\forall^{(p,n,T',\emptyset,\leftarrow)}\ B_2$. Activate this node.
   * If N is of the form *iii*) ($\leftarrow$), $q \notin S'$, and $q \neq p$, then deactivate N. Because (Claim 6) $next(q) \neq \emptyset$, we can use rule 12 and the induction

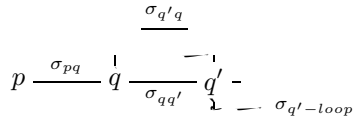
hypothesis to add a tableau $\mathcal{T}'$ with root labelled $q \vdash B_1$. Also, add nodes labelled $q_i \vdash B_1\ U_\forall^{(p,n,t_i IT',S'\cup\{q\},\leftarrow)}\ B_2$. Only these last nodes will be activated.

* 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 $q = p$, then apply rule 14. Deactivate N and activate the added node labelled $q \vdash B_1\ U_\forall^{(q,n,T')}\ B_2$

5. Goto 2.

We now observe the following:

- The above "algorithm" terminates: One only expands *active* nodes and since $(V, E)_N$ 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 to first 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: Since $q \in \mathcal{C}$ we conclude $next(q) \neq \emptyset$.
  * Claim 4 is valid: If $T' = \emptyset$, then because T' keeps track of which transitions 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 Claim 6 are valid: As for Claim 2.

Assume the produced tree is not tableau. Then, using the induction hypothesis, we conclude that the only reason why the tree is not a tableau 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 $p \models A_1\ U_\forall\ A_2$.

"if" direction (soundness):

We show that if there exists a tableau $\mathcal{T}$ 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 one of the following cases hold:

- $|\sigma| < \infty, p = p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_2} \cdots \xrightarrow{t_m} p_m \not\rightarrow, \sigma = t_1 \cdots t_m$, 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 $t_1 \cdots t_{j_0}$ can be made loop free and traceable in $\mathcal{T}$ along nodes q, such that there exists a tableau with root $q \vdash B_1\ U_\forall\ B_2$ (using the induction hypothesis to obtain contradictions). But this gives a contradiction since $\mathcal{T}$ must then have a subtree which is a tableau 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, for any $0 \leq i \leq m$ we have $p_i \models A_1\ \wedge\ \neg A_2$. 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 and this path must be traceable in $\mathcal{T}$. But this means there must exist a leaf labelled $p_m \vdash Ann(A_1)\ U_\forall^{\mathcal{C}}\ Ann(A_2)$ such that $p_m \notin \mathcal{C}$, and since $p_m \not\rightarrow$, $\mathcal{T}$ cannot be a derivation tree.
- $|\sigma| = \infty, p = p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_2} \cdots, \sigma = t_1 t_2 \cdots$, and

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

  As before, we extract two cases:
  - $(\exists i \in I\!N.\ p_i \models A_2)$. 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 above argument, we obtain a contradiction.
  - $(\forall n \in I\!N.\ p_n \models \neg A_2)$. If there is a state which satisfies $\neg A_1$ along the path, the above argument can be applied. Else, we can obtain a path $\sigma' \in [\sigma'] \in comp(p)$ from σ such that $p \xrightarrow{\sigma'}= p'_0 \xrightarrow{t'_1} p'_1 \xrightarrow{t'_2} \cdots \xrightarrow{t'_{m-1}} p'_{m-1} \xrightarrow{t'_m} p''_{k_0} \xrightarrow{t'''_{k_0+1}} p''_{k_0+1} \cdots$, 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 $(V, E)_N$ is finite). The common suffix ensures that no transition is cc-enabled along σ'.

    Since no transition is cc-enabled along σ' there must exist finitely many nonempty loops $\sigma_{loop_1}, \ldots, \sigma_{loop_r}$ starting and ending at p''_{k_0}, such that no transition from $next(p''_{k_0})$ is cc-enabled along $(\sigma_{loop_1} \cdots \sigma_{loop_r})^\infty$, i.e., no enabled transition at p_{k_0} is independent of all transitions taken in the r loops. Notice that these loops might themselves

contain loops. Moreover, since $|next(p''_{k_0})| \leq |T|$ we may assume that $1 \leq r \leq |T|$. Let $next(p''_{k_0}) = \{t'''_{j_1}, \ldots, t'''_{j_r}\}$. We may also assume that σ_{loop_l} corresponds to a loop along which some transition in conflict with t'''_{j_l} is taken.

From each loop σ_{loop_l} we can extract, by deleting inner loops, three paths σ'_{loop_l}, σ''_{loop_l}, and σ'''_{loop_l} such that

- σ''_{loop_l} contains a transition in conflict with t'''_{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 tableau $\mathcal{T}$:

$$p \xrightarrow{t'_1} \cdots \xrightarrow{t'_m} p''_{k_0} \xrightarrow{\sigma} p''_{k_0} \xrightarrow{\sigma'_{loop_1}} \xrightarrow{\sigma''_{loop_1}} \xrightarrow{\sigma'''_{loop_1}} \cdots \xrightarrow{\sigma'_{loop_r}} \xrightarrow{\sigma''_{loop_r}} \xrightarrow{\sigma'''_{loop_r}} p''_{k_0}$$

where σ is a nonempty simple loop obtained from $\sigma'_{loop_1}\sigma''_{loop_1}\sigma'''_{loop_1}$ by deleting inner loops (rule 7 is going to be applied). The path must also 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 keep track (in the annotation) of which transitions have been concurrently enabled. In our case there are no such transitions, so $\mathcal{T}$ cannot be a tableau and we obtain the desired contradiction.

□

As an example, we show that the process agent from the introduction will eventually be able to fire a transition labelled by a b action (assume the transitions are t_1, t_2, and t_3 and are labelled a, τ, and b, respectively). By the previous theorem, to show $i \models \mathsf{Ev}(<b>tt)$ it is sufficient to construct a tableau with root $i \vdash tt\, U_\forall^\emptyset\, (<b>tt)$.

$$\dfrac{i \vdash tt\, U_\forall^\emptyset (<b>tt)}{i \vdash tt \qquad \dfrac{\dfrac{i \vdash tt\, U_\forall^{\{i\}}(<b>tt)}{i \vdash tt\, U_\forall^{(i,2,\{t_1,t_2\})}(<b>tt)}}{\mathcal{T}_1} \qquad \dfrac{\dfrac{s_1 \vdash tt\, U_\forall^{\{i\}}(<b>tt)}{s_1 \vdash <b>tt}}{s_2 \vdash tt}}$$

where $\mathcal{T}_1$ is

$$\dfrac{i \vdash tt\, U_\forall^{(i,1,\{t_1,t_2\},\emptyset,\rightarrow)}(<b>tt)}{\dfrac{\dfrac{\dfrac{i \vdash tt\, U_\forall^{(i,1,\{t_2\},\{i\},\rightarrow)}(<b>tt)}{i \vdash tt\, U_\forall^{(i,1,\{t_2\},\emptyset,\leftarrow)}(<b>tt)}}{i \vdash tt\, U_\forall^{(i,1,\{t_2\})}(<b>tt)}}{\mathcal{T}_2} \qquad \dfrac{\dfrac{s_1 \vdash tt\, U_\forall^{(i,1,\{t_1\},\{i\},\rightarrow)}(<b>tt)}{s_1 \vdash <b>tt}}{s_2 \vdash tt} \qquad i \vdash tt}$$

where $\mathcal{T}_2$ is

$$\dfrac{i \vdash tt\, U_{\forall}^{(i,0,\{t_2\},\emptyset,\rightarrow)}(<b>tt)}{i \vdash tt \quad \dfrac{i \vdash tt\, U_{\forall}^{(i,0,\{t_2\},\{i\},\rightarrow)}(<b>tt)}{\dfrac{i \vdash tt\, U_{\forall}^{(i,0,\{t_2\},\emptyset,\leftarrow)}(<b>tt)}{i \vdash tt\, U_{\forall}^{(i,0,\{t_2\})}(<b>tt)}} \quad \dfrac{s_1 \vdash tt\, U_{\forall}^{(i,0,\emptyset,\{i\},\rightarrow)}(<b>tt)}{\dfrac{s_1 \vdash <b>tt}{s_2 \vdash tt}}}$$

Notice that if we restrict ourselves to labelled 1-safe nets where the independence relation is empty and translate $A_1\ U_{\exists}\ A_2$ into $\mu X.\, A_2 \vee (A_1 \wedge <Act>X)$ and $A_1\ U_{\forall}\ A_2$ into $\mu X.\, A_2 \vee (A_1 \wedge <Act>tt \wedge [Act]X)$ (actually applying this translation recursively on the subformulas A_1 and A_2), our proof rules will work in essentially the same manner as those presented in [Lar88, SW89].

Choosing an instance of the model checking problem to be a pair (N, A) consisting of a labelled 1-safe net and a formula A and defining its size to be the sum of the size of the net and the length of the formula (see e.g., [CEP93, Che95]), we obtain the following complexity result.

Theorem 13. *The model checking problem is PSPACE-complete.*

Proof sketch. The hardness result follows from easy modifications of the results in [CEP93], while the PSPACE upper bound follows from a modification of the results in [Che95] based on the observations in Sect. 4.2 (the bound on the number and length of the γ's). □

6 Conclusion and Future Work

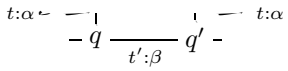
Partial order semantics for concurrent systems have gained interest because interleaving models of concurrency have failed to provide an acceptable interpretation of what it means for events of a concurrent system to be independent. Much work has been devoted to transfer obtained results and notions from the interleaving models to the "true concurrency" models [JNW93, JM93, WN94, LPRT93]. Trying to contribute to the "transferring of results" we have provided proof rules for a CTL-like logic interpreted over maximal traces. The work which we have tried to transfer can be found in [Lar88, SW89]. Our work supports automatic verification of distributed systems whose liveness properties are only provable under the assumption of progress.

There is a trade off between the rules and the definition of tableaux. One can obtain simple rules at the cost of a complicated definition of tableaux.[5] At the cost of presenting less simple rules we have kept the definition of tableaux simple.

[5] The set of simple rules we have identified requires a global side condition in the definition of tableaux.

Future research might consider how to handle a more expressive logic (perhaps one containing a recursion operator) in a similar way, i.e., define the interpretation of the formulas over maximal traces and proving soundness and completeness of some set of proof rules.

The general satisfiability problem for the logic is still an open problem. Consider the following example



where t is independent of t'. Let a be an atomic proposition that holds at q and but not at q' (a can be simulated by having an a labelled enabled transition at all states where a should hold). Then, the formula $(\neg \mathsf{AI}(a)) \wedge (a \wedge \mathsf{Inv}(a \Rightarrow <\alpha> a))$ is satisfied at q. But under the usual CTL-interpretation the formula (where $<\alpha>$ is read as the "next" operator) is unsatisfiable. The important observation is that our interpretation of AI does not quantify over the path $q \xrightarrow{\sigma}$, where $\sigma = t^{\infty}$.

We have investigated extensions of the logic with modal operators expressing concurrent behaviour. It turns out that restricted versions of the satisfiability problem for these logics is undecidable, see the appendix. Axiomatizations of these logics remain to be investigated.

Acknowledgments: I thank Mogens Nielsen for inspiring discussions and Nils Klarlund for comments on an early draft of this paper.

References

[Bed88] M. A. Bednarczyk. *Categories of asynchronous systems.* PhD thesis, University of Sussex, 1988. PhD in computer science, report no.1/88.

[Ber66] R. Berger. The undecidability of the domino problem. *Memoirs American Math. Sco.*, 66, 1966.

[CEP93] Allan Cheng, Javier Esparza, and Jens Palsberg. Complexity results for 1-safe nets. In *Proc. FST&TCS 13, Thirteenth Conference on the Foundations of Software Technology & Theoretical Computer Science*, pages 326–337. Springer-Verlag (*LNCS* 761), Bombay, India, December 1993. To appear in TCS, volume 148.

[CES86] 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.

[Che95] Allan Cheng. Complexity Results for Model Checking. Research Series RS-95-18, BRICS, Department of Computer Science, University of Aarhus, 1995.

[Har85] David Harel. Recurring dominoes: making the highly undecidable highly understandable. *Ann. Disc. Math.*, 24:51–72, 1985.

[JM93] Lalita Jategaonkar and Albert Meyer. Deciding true concurrency equivalences on finite safe nets. In *Proc. ICALP'93*, pages 519–531, 1993.

[JNW93] 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.

[Kwi89] Marta Z. Kwiatkowska. Event fairness and non-interleaving concurrency. *Formal Aspects of Computing*, 1:213–228, 1989.

[Lar88] 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.

[LPRT93] 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.

[Maz86] Antoni Mazurkiewicz. Trace theory. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, pages 279–324. Springer-Verlag (*LNCS* 255), 1986.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice Hall International Series In Computer Science, C. A. R. Hoare series editor, 1989.

[MN92] 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.

[MOP89] Antoni Mazurkiewicz, Edward Ochmański, and Wojciech Penczek. Concurrent systems and inevitability. *Theoretical Computer Science*, 64():281–304, 1989.

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

[Old91] Ernst R. Olderog. *Nets, Terms and Formulas*. Cambridge University Press, 1991. *Number 23 Tracts in Theoretical Computer Science*.

[Pen93] Wojciech Penzcek. Temporal logics for trace systems: On automated verification. *International Journal of Foundations of Computer Science*, 4 (1):31–67, 1993.

[PP90] Doron Peled and Amir Pnueli. Proving partial order liveness properties. In *Proc. ICALP'90*, pages 553–571. Springer-Verlag (*LNCS* 443), 1990.

[Rei85] Wolfgang Reisig. *Petri Nets – An Introduction*. EATCS Monographs in Computer Science Vol.4, 1985.

[Shi85] M. W. Shields. Concurrent machines. *Computer Journal*, 28:449–465, 1985.

[Sta89] Eugene W. Stark. Concurrent transition systems. *Theoretical Computer Science*, 64():221–269, 1989.

[SW89] 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.

[Win86] Glynn Winskel. Event structures. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, pages 325–390. Springer-Verlag (*LNCS* 255), 1986.

[WN94] Glynn Winskel and Mogens Nielsen. Models for concurrency. Research Series RS-94-12, BRICS, Department of Computer Science, University of Aarhus, May 1994. 144 pp. To appear as a chapter in the *Handbook of Logic and the Foundations of Computer Science*, Oxford University Press.

Appendix

Model Checking P-CTL by Labelling States

In this section we present a state labelling based algorithm that solves the model checking problem. The algorithm essentially works as the one presented for CTL in [CES86] except for the $U_\forall$ operator.

Theorem 14. *Given a net N and a formula A. Let $(V, E)_N$ denote the reachability graph of $N = (P, T, F, M_0, l)$. The following state labelling based algorithm solves the model checking problem for N and A in time $O(|A|(|V| + |E||T|))$.*

Proof. Given a formula A and a net N, 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 A' of A (or its negation $\neg A'$) if and only if it is satisfied in that state. Hence, after the $|A|$'th stage all states in V will have been labelled with either A or $\neg A$.

The data structures needed to perform the labelling are essentially those described for the CTL model checker in [CES86]. The only exception is 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 $A' = A_1 \, U_\forall \, A_2$. 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 A', and states labelled with $\neg A_1$ and $\neg A_2$ are labelled with $\neg A'$. 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 (V, E) 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 V. As long as there is a terminal node n in G', repeatedly do the following:

1) If there is a state $p \in n$, a transition $t \in T$, and a state $p' \in V$ such that $p \xrightarrow{t} p'$ and p' is labelled with $\neg A'$, then label all states in n with $\neg A'$. Furthermore, for all nodes m in G', if n can be reached from m then label all states in m with $\neg A'$. Remove all processed nodes from G' and let G' denote the new DAG.
2) Else, if there is a state p in n but no transition t and state $p' \notin n$ such that $p \xrightarrow{t} p'$, then label all states in n (and m's above n, as described just above) with $\neg A'$ (there must exist an invalidating computation in n from p). Update G' as above.
3) Else, all states of n have successor states not in n. Moreover, these successor states are all labelled by A'. Assume $T = \{t_1, \ldots, t_k\}$.

- Initialize a boolean array B of length k such that all its entries are set to *False*. Then, for each edge $\xrightarrow{t_i}$ between any two states in n, set all entries $B[j]$ such that $\neg(t_i I t_j)$ to *True*.
- If there is an entry $B[l]$ which is *False* and t_l is enabled at any state in n, then label all states in n with A'. 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 $\neg A'$ and update G' as above.

It should be obvious that case 1) labels the states in n correctly. Case 2) is also correct because we can exhibit a computation in n whose states are labelled with A_1 and $\neg A_2$. Case 3) is correct because of the following observation: there exists a computation inside n if and only if there is no transition t_l that is (i) independent of all transition labelling edges between states in n, and (ii) enabled at (necessarily all) a state in n.

An analysis of the algorithm yields the time complexity $O(|A|(|V|+|E||T|))$. Hence, our algorithm is comparable to the one presented in [CES86]. □

Extensions of P-CTL and Undecidability Results

In this section we present different extensions of P-CTL with modal operators expressing concurrent or conflicting behaviour. We prove that the satisfiability problem for some of these logics is undecidable for finite as well as infinite labelled nets, if we impose an "injectivity" constraint on their reachability graphs.

Remark. Infinite labelled nets are a generalization of finite labelled nets, where the sets P, T, and F may be at most countably infinite. Hence, for any state p in the reachability graph of the nets we consider, $comp(p) \neq \emptyset$.

The New Modal Operators

Assume a fixed labelled net $N = (P, T, F, M_0, l)$ and let $(V, E)_N$ denote its reachability graph and I the independence relation.

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

$$A ::= \lll\ggg((\alpha_1, A_1), \ldots, (\alpha_n, A_n)) \mid \sharp((\alpha_1, A_1), \ldots, (\alpha_n, A_n)) \mid \langle\alpha_1 \cdots \alpha_n\rangle A \mid \rangle\alpha_1 \cdots \alpha_n\langle A \ .$$

The interpretation is:

- $p \models \lll\ggg((\alpha_1, A_1), \ldots, (\alpha_n, A_n))$ if and only if $(\exists\, t_1, \ldots, t_n \in T, q_1, \ldots, q_n \in V.$
  $(\forall\, 1 \leq i \leq n.\ p \xrightarrow{t_i} q_i \ \wedge\ l(t_i) = \alpha_i \ \wedge\ q_i \models A_i) \ \wedge\ (\forall\, 1 \leq i < j \leq n.\ t_i I t_j))$,

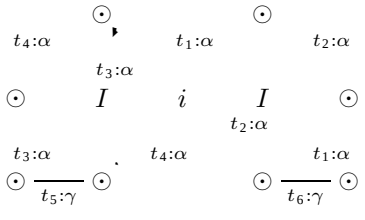
– $p \models \sharp((\alpha_1, A_1), \ldots, (\alpha_n, A_n))$ if and only if $(\exists\, t_1, \ldots, t_n \in T, q_1, \ldots, q_n \in V.$ $(\forall\, 1 \leq i \leq n.\ p \xrightarrow{t_i} q_i \ \wedge\ l(t_i) = \alpha_i \ \wedge\ q_i \models A_i) \ \wedge\ (\forall\, 1 \leq i < j \leq n.\ (t_i, t_j) \notin I))$,

– $p \models \langle \alpha_1 \cdots \alpha_n \rangle A$ if and only if $(\exists\, t_1, \ldots, t_n \in T, q \in V.\ (\forall\, 1 \leq i \leq n.\ l(t_i) = \alpha_i) \ \wedge$
$(\forall\, 1 \leq i < j \leq n.\ t_i I t_j) \ \wedge\ p \xrightarrow{t_1 \cdots t_n} q \ \wedge\ q \models A)$, and

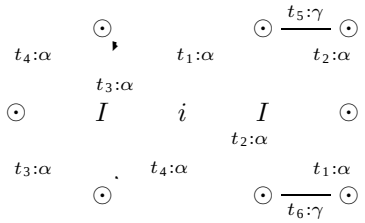
– $p \models\rangle \alpha_1 \cdots \alpha_n \langle A$ if and only if $(\exists\, t_1, \ldots, t_n \in T, q \in V, p \xrightarrow{t_1 \cdots t_n} q.$
$(\forall\, 1 \leq i \leq n.\ l(t_i) = \alpha_i) \ \wedge\ (\forall\, 1 \leq i < n.\ (t_i, t_{i+1}) \notin I) \ \wedge\ q \models A)$.

The $\lll\ggg$ and $\langle\rangle$ operators specify concurrent behaviour.[6] The difference between them is that $\langle\rangle$ requires a property A to hold *after* the execution of a set of mutually independent (labelled) transitions, while $\lll\ggg$ requires properties A_i to hold after the execution of (labelled) transitions t_i from a set of mutually independent transitions. In a similar way, the $\sharp$ and $\rangle\langle$ operators specify conflicting behaviour.

The $\lll\ggg$, $\langle\rangle$, $\sharp$, and $\rangle\langle$ operators might be replaced by others. We have chosen to present them because they can distinguish the following situations. All the depicted transition systems are reachability graphs of nets. Transitions that are independent are indicated by an I in their "independence square". States—except the initial state—are indicated by $\odot$ and states labelled by A or $\neg A$ indicate that one has to extend the reachability graph in a trivial manner such that a property A, e.g., $<\gamma>tt$, either holds or doesn't, as indicated by A or $\neg A$:



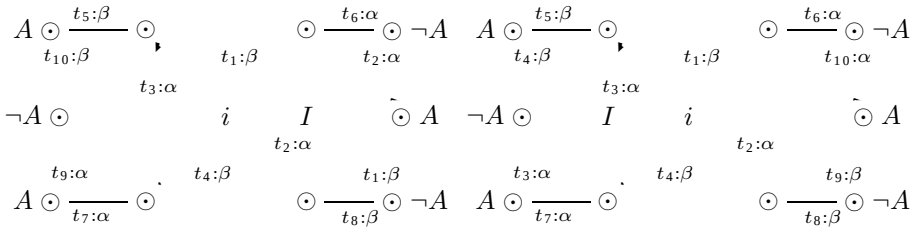
$\neg \lll\ggg ((\alpha, <\gamma>tt), (\alpha, <\gamma>tt))$



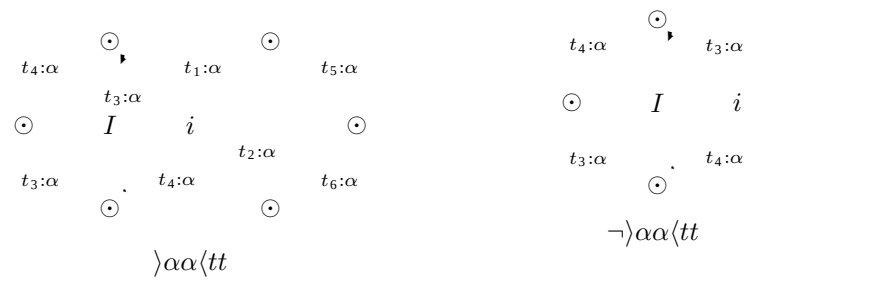
$\lll\ggg ((\alpha, <\gamma>tt), (\alpha, <\gamma>tt))$

The above two reachability graphs cannot be distinguished by the $\langle\rangle$, $\rangle\langle$, or $\sharp$ operator. To see this, notice that all states having α and β labelled transitions satisfy the same formulas, as is the case for states having exactly one α labelled transition and states having no transitions. By induction it can then be shown that the two initial states satisy the same formulas (not containing the $\lll\ggg$ operator. The same reasoning applies to the three following examples.

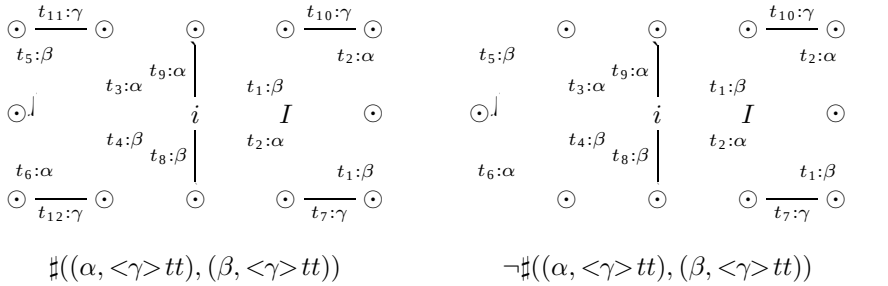
[6] The $\langle\rangle$ operator resembles the one proposed in [LPRT93].



The above two reachability graphs cannot be distinguished by the $\lll\ggg$, $\rangle\langle$, or $\sharp$ operator.



The above two reachability graphs cannot be distinguished by the $\lll\ggg$, $\langle\rangle$, or $\sharp$ operator.



The above two reachability graphs cannot be distinguished by the $\lll\ggg$, $\langle\rangle$, or $\rangle\langle$ operator.

The Undecidability Results

We will concentrate on the extensions of P-CTL which contains the operators $\lll\ggg$. Actually, for any extension of P-CTL with any of the other three presented operators the following undecidability results hold.

Definition 15. N is said to be *injective* if for all $p \in V$ and $t, t' \in T$ it is the case that $p \xrightarrow{t}, p \xrightarrow{t'}$, and $l(t) = l(t')$ implies $t = t'$.

Definition 16. The *(finite) injective satisfiability problem* is the problem of deciding, given a formula A, whether there exists a (finite) injective labelled net N such that $M_0 \models A$. If this is the case, A will be said to be *(finitely) injectively satisfiable*.

The following problems are known to be undecidable, see [Ber66, Har85] and [LPRT93].

Definition 17. The *colouring problem*, CP. An *instance* of the problem is a quadruple $\mathcal{C} = (C, R, U, c_0)$, 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 $\mathcal{C}$ is a function $Col : \mathbb{N} \times \mathbb{N} \to C$ such that:

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

Definition 18. The *finite colouring problem*, FCP. An *instance* of the problem is a quintuple $\mathcal{C}_F = (C, R, U, c_0, c_f)$, 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 $\mathcal{C}_F$ 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:

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

For the logics containing the $\ll\gg$ operator we have the following result.

Theorem 19. *The set of formulas that are injectively satisfiable is non-recursive.*

Proof. We reduce the colouring problem to the injective satisfiability problem. Given $\mathcal{C} = (C, R, U, c_0)$, we construct a formula $A_{\mathcal{C}}$, a conjunct of five formulas given below, that is injectively satisfiable if and only if $\mathcal{C}$ has a solution.

Assume that the labels $\alpha_0, \ldots, \alpha_k$, *up*, and *right* are distinct symbols. The five conjuncts are the following:

- $A_1 = <\alpha_0> tt$, $Col(0,0) = c_0$
- $A_2 = G(\ll\gg ((<right>, tt), (<up>, tt)))$, 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 $A_{\mathcal{C}} = \bigwedge_{i=1}^{5} A_i$ is injectively satisfiable if and only if $\mathcal{C}$ has a solution. The "if" direction is easy and therefore omitted. The "only if" direction follows the lines in [LPRT93] and makes essential use of injectivity of the solution to $A_{\mathcal{C}}$ and the following *"diamond"* and *"commutativity"* properties of the reachability graph of a labelled net N:

- If $p \in V$, $t, t' \in T$, $p \xrightarrow{t} q$, $p \xrightarrow{t'} q'$, and tIt', then there exists $q'' \in V$ such that $q \xrightarrow{t'} q''$ and $q' \xrightarrow{t} q''$.
- If $p \in V$, $t, t' \in T$, $p \xrightarrow{t} q \xrightarrow{t'} q'$, adn tIt', then there exists $q'' \in V$ such that $p \xrightarrow{t'} q'' \xrightarrow{t} q'$.

□

Theorem 20. *The set of formulas that are finitely injectively satisfiable is non-recursive.*

Proof. We reduce the finite colouring problem to the finite injective satisfiability problem. Given $\mathcal{C}_F = (C, R, U, c_0, c_f)$, we construct a formula $A_{\mathcal{C}_F}$, a conjunct of seven formulas given below, that is finitely injectively satisfiable if and only if $\mathcal{C}_F$ has a solution.

Again, assume that the labels $\alpha_0, \ldots, \alpha_k$, *UM*, *RM*, *up*, and *right* are distinct symbols. The seven conjuncts are the following:

- $A_1 = <\alpha_0> tt$, $Col(0,0) = c_0$
- $A_2 = G((\lll ((right, tt), (up, tt)) \wedge [RM]ff \wedge [UM]ff) \vee$
  $(<up> tt \wedge <RM> tt \wedge [right]ff \wedge [UM]ff) \vee$
  $(<right> tt \wedge <UM> tt \wedge [up]ff \wedge [RM]ff) \vee$
  $(<c_f> tt \wedge <RM> tt \wedge <UM> tt \wedge [right]ff \wedge [up]ff))$, grid structure
- $A_3 = G(\bigwedge_{i=0}^{k}(<\alpha_i> tt \Leftrightarrow \bigwedge_{j \neq i}[\alpha_j]ff))$, 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
- $A_6 = EV(<\alpha_f> tt \wedge <RM> tt \wedge <UM> tt)$, c_f in upper right corner
- $A_7 = G(<RM> tt \Rightarrow [up] <RM> tt) \wedge$
  $G(<UM> tt \Rightarrow [right] <UM> tt)$, consistent borders

We claim that $A_{\mathcal{C}_F} = \bigwedge_{i=1}^{7} A_i$ is injectively satisfiable if and only if $\mathcal{C}_F$ has a solution. The "if" direction is easy and therefore omitted. The "only if" direction is nontrivial and follows from the next two lemmas. We use another proof technique than [LPRT93] since they assume a fixed finite alphabet. We only assume finiteness about the set of transitions T of the solution to $A_{\mathcal{C}_F}$. □

Assume that we have fixed $\mathcal{C}_F$ and a finite injective net N such that $M_0 \models A_{\mathcal{C}_F}$, where $A_{\mathcal{C}_F}$ is defined in the proof of Theorem 20. We will use the notation $p \xrightarrow{\alpha} p'$ to indicate that there is a transition $t \in T$ such that $p \xrightarrow{t} p'$ and $l(t) = \alpha$. This shouldn't lead to any confusion since N is assumed injective.

Lemma 21. *Assume N is a net such that $M_0 \models A_{\mathcal{C}_F}$. If there exist $p_0, \ldots, p_n \in V$ and $t_1, \ldots, t_n \in T$ such that $p_0 \xrightarrow{t_1} \ldots \xrightarrow{t_n} p_n, p_0 = M_0$, for all $1 \leq j \leq n$ either $l(t_j) = right$ or $l(t_j) = up$, and no state p_j except p_n has an enabled transition labelled α_f, then $\mathcal{C}_F$ has a solution.*

Proof. Let $\beta_0, \ldots, \beta_n$ denote the unique labels among $\{\alpha_0, \ldots, \alpha_k\}$ that by A_3 must label some enabled transition at $p_0, \ldots, p_n$, respectively.

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

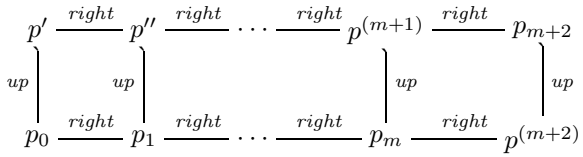
- $p_0 \xrightarrow{right}$ p_1 and $p_0 \xrightarrow{UM}$: By A_2, A_7, and our assumptions we conclude that $p_0 \xrightarrow{right} p_1 \xrightarrow{right} \ldots \xrightarrow{right} p_n$ and $\forall 0 \leq i \leq n.\ p_i \xrightarrow{UM}$. Since $p_n \xrightarrow{\alpha_f}$ we easily obtain a solution by A_4 and A_5.
- $p_0 \xrightarrow{up}$ p_1 and $p_0 \xrightarrow{RM}$: Symmetric to the above case.
- $p_0 \xrightarrow{right}$ and $p_0 \xrightarrow{up}$: Without loss of generality we can assume that $l(t_1) = right$, i.e., $p_0 \xrightarrow{right} p_1$. By injectivity and A_2 there must exist $p', p'' \in V$ such that $p_0 \xrightarrow{up} p' \xrightarrow{right} p''$ and $p_0 \xrightarrow{right} p_1 \xrightarrow{up} p''$. Continuing this way as long as $l(t_j) = right$ and exploiting injectivity we conclude there must exist $p', p'', \ldots, p^{(m+1)}$ such that

$$\begin{array}{ccccccc} p' & \xrightarrow{right} & p'' & \xrightarrow{right} & \cdots & \xrightarrow{right} & p^{(m+1)} \\ {\scriptstyle up}\uparrow & & {\scriptstyle up}\uparrow & & & & {\scriptstyle up}\uparrow \\ p_0 & \xrightarrow{right} & p_1 & \xrightarrow{right} & \cdots & \xrightarrow{right} & p_m \end{array}$$

  If $m = n$, then we easily obtain a solution, since $p_n \xrightarrow{\alpha_f}$. So assume $m < n$. Then $l(t_{m+1}) = up$ and by injectivity we conclude that $p^{(m+1)} = p_{m+1}$. 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 continue by showing how to expand the above $1 \times (m + 1)$ grid to a $2 \times (m + 1)$ grid if $l(t_{m+2}) = up$ or to a $1 \times (m + 2)$ grid if $l(t_{m+2}) = right$.
  - Assume that $l(t_{m+2}) = up$. By A_2, A_7, and injectivity we conclude there must exist a $q^{(m)}$ such that $p^{(m)} \xrightarrow{right} p^{(m+1)} \xrightarrow{up} p_{m+2}$ and $p^{(m)} \xrightarrow{up} q^{(m)} \xrightarrow{right} p_{m+2}$. By repeating this we obtain:

$$\begin{array}{ccccccccc} q' & \xrightarrow{right} & q'' & \xrightarrow{right} & \cdots & \xrightarrow{right} & q^{(m)} & \xrightarrow{right} & p_{m+2} \\ {\scriptstyle up}\uparrow & & {\scriptstyle up}\uparrow & & & & {\scriptstyle up}\uparrow & & {\scriptstyle up}\uparrow \\ p' & \xrightarrow{right} & p'' & \xrightarrow{right} & \cdots & \xrightarrow{right} & p^{(m)} & \xrightarrow{right} & p^{(m+1)} \\ {\scriptstyle up}\uparrow & & {\scriptstyle up}\uparrow & & & & {\scriptstyle up}\uparrow & & {\scriptstyle up}\uparrow \\ p_0 & \xrightarrow{right} & p_1 & \xrightarrow{right} & \cdots & \xrightarrow{right} & p_{m-1} & \xrightarrow{right} & p_m \end{array}$$

- Assume that $l(t_{m+2}) = \mathit{right}$. By similar arguments we get:



This procedure can be continued for $t_{m+3}, \ldots, t_n$ giving us a grid from which it is easy to obtain a solution to $\mathcal{C}_F$, by A_1, A_3, A_4, and A_5.

□

Next, we proof there exist a path of the form mentioned in Lemma 21 if $M_0 \models A_{\mathcal{C}_F}$.

Lemma 22. *If N is a net such that $M_0 \models A_{\mathcal{C}_F}$, then there exist $p_0, \ldots, p_n \in V$ and $t_1, \ldots, t_n \in T$ such that $p_0 \xrightarrow{t_1} \ldots \xrightarrow{t_n} p_n, p_0 = M_0$, for all $1 \leq j \leq n$ either $l(t_j) = \mathit{right}$ or $l(t_j) = \mathit{up}$, and no state p_j except p_n has an enabled transition labelled α_f.*

Proof. Assume by contradiction that there doesn't exist any path of the above form. By A_2 either $M_0 \xrightarrow{\mathit{right}}$ or $M_0 \xrightarrow{\mathit{left}}$ Without loss of generality we may assume $M_0 \xrightarrow{\mathit{right}}$. We continue be a case analysis.

– Assume $M_0 \xrightarrow{UM}$: We know that $M_0 \overset{\alpha_f}{\not\rightarrow}$, and choosing p_1 to be the unique state such that $M_0 = p_0 \xrightarrow{\mathit{right}} p_1$ we also conclude $p_1 \overset{\alpha_f}{\not\rightarrow}$. Now by A_2, A_6, A_7, and our assumptions it must be the case that $p_1 \xrightarrow{\mathit{right}}$ and $p_1 \xrightarrow{UM}$. Continuing this way we exhibit an infinite path $p_0 \xrightarrow{\mathit{right}} p_1 \xrightarrow{\mathit{right}} \cdots$ such that $p_j \overset{\alpha_f}{\not\rightarrow}$ and $p_j \xrightarrow{UM}$ for all states p_j along the path. Since K is finite there must exist a least j_0 such that p_{j_0} is visited twice along $p_0 \xrightarrow{\mathit{right}} \cdots \xrightarrow{\mathit{right}} p_{j_0}$. Let $0 \leq j_1 < j_0$ be the index such that $p_{j_1} = p_{j_0}$. This gives us an infinite path $p_0 \xrightarrow{\gamma_1} p_{j_1} \xrightarrow{\gamma_2} p_{j_0} \xrightarrow{\gamma_2} p_{j_0} \xrightarrow{\gamma_2} \cdots$, where γ_1 (γ_2) is the sequence of *right* labelled transitions leading from p_0 to p_{j_1} (from p_{j_1} to p_{j_0}). But by A_6 the above path cannot be a computation from p_0. Hence there must exist a transition t' that is cc-enabled along $p_{j_0} \xrightarrow{\gamma_2} p_{j_0} \xrightarrow{\gamma_2} \cdots$. By the diamond and commutativity properties of $(V, E)_N$ we obtain Diagram 1, where $p'_{j_1} = p'_{j_0}$ is a state in V.

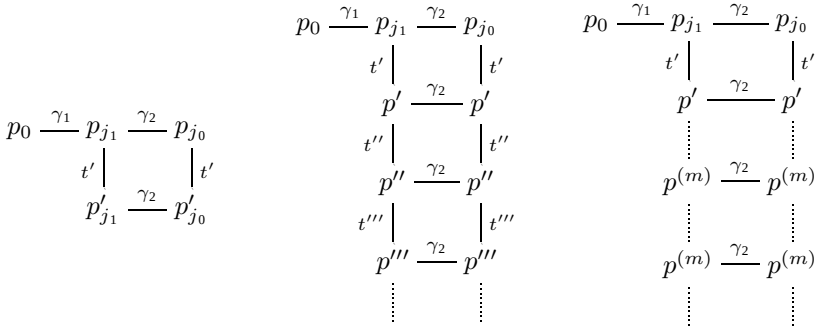


Diagram 1 Diagram 2 Diagram 3

Now by A_2, γ_2 being labelled by *right*, and A_6, we conclude that there must exist a transition t'' which is cc-enabled along the loop obtained by repeating $p'_{j_1} \xrightarrow{\gamma_2} p'_{j_0}$. By repeating this argument we obtain ($p' = p'_{j_1} = p'_{j_0}$) Diagram 2.

Since N is finite, the set T is finite. All of the reached states have the property that $<c_f>tt \wedge <RM>tt \wedge <UM>tt$ doesn't hold, since $<right>tt$ and A_2 hold. One can now repeat the above argument, observing that finiteness of V implies that some $p^{(m)}$ must occur twice along the leftmost vertical path in Diagram 2. From this observation we can construct Diagram 3. Again, we conclude there must exist a transition that is cc-enabled along the looping part of Diagram 3. This transition must be independent of all the transitions on the looping parts between $p^{(m)}$ shown in Diagram 3, especially the transitions labelled *right*. Also, all states along these loops have an enabled transition labelled *right*.

Let (V', E') denote the transition system obtained by restricting $(V, E)_N$ to the states satisfying $\neg(<c_f>tt \wedge <RM>tt \wedge <UM>tt)$. It should now be clear that one can produce an infinite computation from p_0 which stays in (V', E'). One modifies the above infinite path by choosing transitions that are cc-enabled following the above scheme. It is important to notice that because it is always possible to insert loops containing transitions labelled *right* (the γ_2 loops) the states reached by "taking a cc-enabled transition" also contain a self loop of transitions labelled *right*. Since $(V, E)_N$ is finite there are only a finite number of maximal strongly connected components in (V', E'). Hence, by repeating the above procedure one will only be able to proceed towards terminal maximal strongly connected components. This eventually produces a computation along which $\neg(<c_f>tt \wedge <RM>tt \wedge <UM>tt)$ holds, contradicting A_6.

– Assume $M_0 \xrightarrow{up}$: A similar way of reasoning leads to the desired contradiction. To sketch the argument: Choose consecutive *right* labelled transitions 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 transitions

labelled *up* and *RM* (else contradicting our main assumption by A_2). From this state choose the infinite path labelled *up* (else contradicting our main assumption by A_2 and A_7). Apply the above argument in a symmetric way. □

For the remaining operators we have the following corollary.

Lemma 23. *For the logics containing at least one of the operators* $\sharp$, $\langle\rangle$, *or* $\rangle\langle$, *Theorem 19 and 20 remain true.*

Proof sketch. Replace $\ll\gg((<right>, tt), (<up>, tt))$ in A_2 in the proof of Theorem 19 and 20 by either

- $(<right> tt \wedge\ <up> tt\ \wedge\ \neg\sharp((right, tt), (up, tt)))$,
- $(\langle right\, up\rangle tt)$, or
- $(<right><up> tt\ \wedge\ \neg\rangle right\, up\langle tt)$

depending on which operator is available. □

This article was processed using the LaTeX macro package with LLNCS style

Recent Publications in the BRICS Report Series

RS-95-39 **Allan Cheng.** ***Petri Nets, Traces, and Local Model Checking*****. July 1995. 32 pp. Full version of paper appearing in Proceedings of AMAST '95, LNCS 936, 1995.**

RS-95-38 **Mayer Goldberg.** ***Gödelisation in the λ-Calculus*****. July 1995. 7 pp.**

RS-95-37 **Sten Agerholm and Mike Gordon.** ***Experiments with ZF Set Theory in HOL and Isabelle*****. July 1995. 14 pp. To appear in** ***Proceedings of the 8th International Workshop on Higher Order Logic Theorem Proving and its Applications,*** **LNCS, 1995.**

RS-95-36 **Sten Agerholm.** ***Non-primitive Recursive Function Definitions*****. July 1995. 15 pp. To appear in** ***Proceedings of the 8th International Workshop on Higher Order Logic Theorem Proving and its Applications,*** **LNCS, 1995.**

RS-95-35 **Mayer Goldberg.** ***Constructing Fixed-Point Combinators Using Application Survival*****. June 1995. 14 pp.**

RS-95-34 **Jens Palsberg.** ***Type Inference with Selftype*****. June 1995. 22 pp.**

RS-95-33 **Jens Palsberg, Mitchell Wand, and Patrick O'Keefe.** ***Type Inference with Non-structural Subtyping*****. June 1995. 22 pp.**

RS-95-32 **Jens Palsberg.** ***Efficient Inference of Object Types*****. June 1995. 32 pp. To appear in** ***Information and Computation*****. Preliminary version appears in** ***Ninth Annual IEEE Symposium on Logic in Computer Science,*** **LICS '94 Proceedings, pages 186–195.**

RS-95-31 **Jens Palsberg and Peter Ørbæk.** ***Trust in the λ-calculus*****. June 1995. 32 pp. To appear in** ***Static Analysis: 2nd International Symposium,*** **SAS '95 Proceedings, 1995.**

RS-95-30 **Franck van Breugel.** ***From Branching to Linear Metric Domains (and back)*****. June 1995. 30 pp. Abstract appeared in Engberg, Larsen, and Mosses, editors,** ***6th Nordic Workshop on Programming Theory,*** **NWPT '6 Proceedings, 1994, pages 444-447.**