Petri Nets, Traces, and Local Model Checking

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

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. By 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 labeled 1-safe nets. It turns out that Mazurkiewicz trace theory provides a useful setting in which the progress fairness assumptions can be formalized in a natural way. 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, labeled 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 [18, 11, 22, 20, 23]. This is is done by imposing more structure on models for concurrent systems, in our case an independence relation on the transitions.

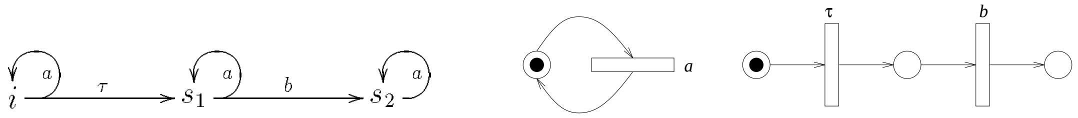
Our main objective is to explore the use of the extra structure of independence in the context of *specification logics*. This paper introduces and studies a *CTL*-like branching time temporal logic, *P-CTL*, interpreted over the reachability graph of labelled 1-safe nets.

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

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

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 let us 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 labeled 1-safe net to the right, containing three transitions labeled a, τ, and b respectively [14, 23].



The net gives us a more concrete model of the process agent. It shows that the transition labeled a is independent of those labeled 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 labeled asynchronous transition system (*l-ATS*) [19, 1, 23, 15]. In general, we can obtain such a labeled asynchronous transition system as the case graph, extended with implicit information about independence, of a labeled 1-safe net. In this paper, we will concentrate on labeled 1-safe nets.

The logic *P-CTL* contains one important feature which is the model-theoretic incorporation of progress. What corresponds to quantified "until" path formulas in *CTL* is in our setting interpreted over firing sequences of labeled 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 $Ev(<b>\, tt)$, "eventually b is enabled" (tt means "True"), is true of the process agent example under the assumption of progress (our interpretation) , but not without (standard *CTL* interpretation). Our interpretation is conservative in the sense that if we interpret *P-CTL* over standard *lts* we get the standard *CTL* interpretation.

Work on expressing progress assumptions and fairness assumptions can be found in for example Manna and Pnueli's book on temporal logic [10]. 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. We are able to avoid this obstacle and treat progress assumptions *uniformally*.

In the standard setting of *CTL*-like logics interpreted over labeled transition systems, model checking has been described in [4] using a state based algorithm, and in [8, 21] using tableaux rules. Model checking in the framework of partial order semantics has been described in [17, 16].

In this paper we present the first, to our knowledge, set of sound and complete tableau rules in the style of [8, 21] for a *CTL*-like logic interpreted in the trace theoretic framework. The rules are a generalization of those in [8, 21] in the sense that if we restrict model checking to labeled 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 [21] 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 [4]. Labeled 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 [2] related work is presented. There, based on the approach in [4], we present a global model checking algorithm for *P-CTL* . By choosing this approach, contrary to a tableau based as in [8, 21], we obtain a polynomial time algorithm (in the size of the reachability graph). Also, different extensions of *P-CTL* in the form of modal operators expressing concurrent or conflicting behaviour are considered.

The rest of the paper is organized as follows. In section 2 we provide the necessary definitions. In section 3 we present the logic and its interpretation. Section 4 contains a motivating example followed by the tableau rules and the definition of tableaux. In section 5 we state the main result, soundness and completeness of the proposed tableau rules. Finally section 6 contains the conclusion and suggestions for future work.

2 Basic Definitions

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

Definition 1 Concurrent alphabet and traces

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

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

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

- *Define concatenation $\circ$ of elements in A^∞ as:*

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

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

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

$$u \preceq v \;\; \textit{iff} \;\; (\forall (a,b) \notin I.\ \pi_{(a,b)}(u) \leq_{pref} \pi_{(a,b)}(v))$$

- *Define an equivalence relation $\equiv$ on A^∞ as $u \equiv v$ if $u \preceq v$ and $v \preceq u$. Let $[u]$ denote the equivalence class containing u.*

- *Fact: $\equiv$ is a congruence with respect to $\circ$.*

- *The elements of $A^\infty/\equiv$ will be called traces.*

- *For* $[u], [v] \in A^\infty/\equiv$ *we define* $[u] \preceq [v]$ *if* $u \preceq v$. *It can be shown that this relation is a partial order. We will write* $[u] \prec [v]$ *if and only if* $u \preceq v$ *and* $u \not\equiv v$.

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

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

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

We now define *labeled 1-safe nets*, the labeled version of 1-safe nets[1].

Definition 2 1-safe nets

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

- P *and* T *are finite 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 *of* N*.*

Given $a \in P \cup T$*, the* preset *of* a*, denoted by* ${}^\bullet a$*, is defined as* $\{a' \mid a'Fa\}$*; the* postset *of* a*, denoted by* $a^\bullet$*, is defined as* $\{a' \mid aFa'\}$*. We define the* independence relation I *to be the irreflexive symmetric relation over* T *defined by* $t_1 I t_2$ *iff* ${}^\bullet t_1^\bullet \cap {}^\bullet t_2^\bullet = \emptyset$*.*

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* $(M - {}^\bullet t) \cap t^\bullet = \emptyset$*. We denote the set of enabled transitions at a marking* M *by next(M).*

- *Given a transition* t*, we define a relation* $\xrightarrow{t}$ *between markings as follows:* $M \xrightarrow{t} M'$ *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.*

- *Given a marking* M *of* N*, the set of reachable markings of the net* (P, T, F, M) *(i.e., the net obtained replacing the initial marking* M_0 *by* M*) is denoted by* $[M\rangle$*.*

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

- *A labeled 1-safe net $N = (P, T, F, M_0, l)$ is just a 1-safe net together with a map $l : T \rightarrow Act$, mapping each transition to an action in Act.*

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

Definition 4 Reachability graph

- *The reachability graph of a net N is the edge-labeled graph, $(V, E)_N$, whose vertices, V, are the reachable markings of N; if $M \xrightarrow{t} M'$ for a reachable marking M, then there is an edge from M to M' labeled with t.*

In the following we assume a fixed labeled 1-safe net N and consider its reachability graph $(V, E)_N$. We will use the symbols $p, q, \ldots$ to denote vertices in $(V, E)_N$ and $p \xrightarrow{t} q$ to denote that there is an edge between p and q labeled with t. Notice that (T, I) is a concurrent alphabet. If nothing else is mentioned it is implicitly assumed that (T, I) is used to generate the congruence $\equiv$.

Definition 5 Path

- *Define a* path from $p \in V$ *as a sequence, finite or infinite, of events $t_1, t_2, \ldots$, 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 and use the notation $p \xrightarrow{\sigma}$ to indicate that $\sigma \in path(p)$.*

- *A path from p is* maximal *if it is either infinite or ends in a deadlocked state (or just a deadlock) p_n, that is, a state p_n such that $p_n \not\rightarrow$.*

- *Due to the firing rules of the nets we have that $\equiv$ respect the path property, formally: $(\forall \sigma \in path(p).\ (\forall \sigma' \in [\sigma].\ p \xrightarrow{\sigma'}))$. Hence $path(p)$ can be partitioned into elements of $T^\infty / \equiv$. Moreover if σ is finite then $p \xrightarrow{\sigma} q$ implies $(\forall \sigma' \in [\sigma].\ p \xrightarrow{\sigma'} q)$.*

- *Given $\sigma \in path(p), |\sigma| = \infty, \sigma = t_1 t_2 \cdots$. A transition t is said to be* continuously concurrently enabled along *$p \xrightarrow{\sigma} = p \xrightarrow{t_1} p_1 \xrightarrow{t_2} p_2 \cdots$ if and only if t is enabled from a certain point and independent of the rest of the transitions taken along $p \xrightarrow{\sigma}$, formally: $(\exists n \in \mathbb{N}.\ (\forall j \geq n.\ p_j \xrightarrow{t} \wedge\ t I t_{j+1}))$. Notice that the irreflexivity of I implies that from a certain point t is never taken along the path $p \xrightarrow{\sigma}$. Whenever p is understood we simply say that t is continuously concurrently enabled along σ. In the process agent example from the introduction, τ is continuously concurrently enabled along $i \xrightarrow{a^\infty}$.*

- *Define $comp(p)$ as the maximal elements with respect to $\preceq$ of $path(p)/\equiv$. For $\sigma \in [\sigma'] \in comp(p)$ we refer to $p \xrightarrow{\sigma}$ as a computation from p. In the process agent example, $\tau b a^\infty$ is a computation from i while a^∞ is not, when we use a, b, and τ to refer to the corresponding transitions.*

Lemma 6 *If t is continuously concurrently enabled along $\sigma \in path(p)$ then for any $\sigma' \in [\sigma]$ t is continuously concurrently enabled along σ', that is, $\equiv$ respects* continuously concurrently enabled. *Hence for $\sigma \in path(p)$ we say that $t \in T$ is* continuously concurrently enabled along $[\sigma]$ *if t is* continuously concurrently enabled along *σ.*

Lemma 7 *Given $\sigma \in path(p), |\sigma| = \infty$. Then $(\exists t \in T.\ t$ is continuously concurrently enabled along $\sigma)$ iff $(\exists \sigma' \in path(p).\ [\sigma] \prec [\sigma'])$.*

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

3 The Logic *P-CTL* and its Interpretation

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

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

In Hennessy-Milner logic [13], $<a>\ A$ 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 $\Diamond_\alpha A$ expresses that a transition labeled α can be performed reaching a state where A holds. tt is an abbreviation for *TRUE*. The "until" operators $U_\exists$ and $U_\forall$ are introduced as generalizations of their counterparts in [4], here interpreted over maximal traces, following Mazurkiewicz [11].

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. Only the non trivial cases are presented.

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

Furthermore, we define $ff \equiv \neg tt$, $<\alpha>\ A \equiv \Diamond_\alpha A$, $F\ A \equiv tt\ U_\exists A$, $G\ A \equiv \neg F \neg A$, $Ev\ A \equiv tt\ U_\forall A$, and $Al\ A \equiv \neg Ev\, \neg A$. The meaning of EvA is that eventually/inevitably A will hold along any path that satisfies the progress assumptions (fair progress) while AlA means that along some progress fair path A always holds. In the process agent example from the introduction we have $i \models Ev\ {<}b{>} tt$.

Definition 8 *Given a labeled 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 $M_0 \models A$.*

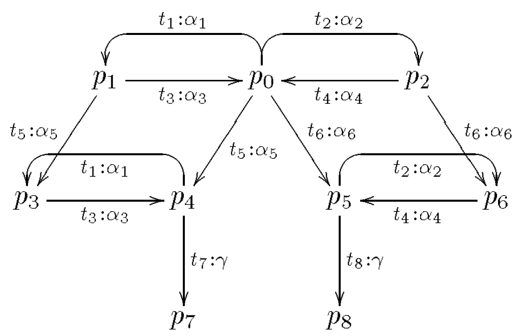
4 A Tableau Method for Model Checking

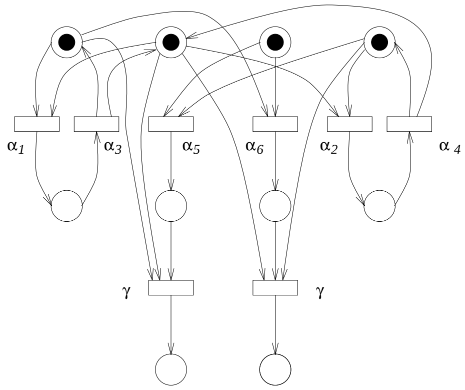
Local model checking as tableau systems has been presented in [21]. As opposed to a global model checker [4] (see also [2], where a global model checker for *P-CTL* is presented), which checks if all states of the system satisfies a formula, a local model checker only checks if a specific state satisfies a given formula. For local model checkers based on tableau systems this is done by only visiting 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.

In this section we present a local model checker based on a tableau system for model checking formulas from our logic. 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 [8, 21]. The main difficulty is how to generalize the unfolding 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 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 $(t_1,t_5),(t_3,t_5),(t_2,t_6),(t_4,t_6)$. Now $p_0 \models_{g_1} \neg Ev <\gamma> tt$ because of e.g. $[(t_1t_3t_2t_4)^\infty] \in comp(p_0)$ and no state along the computation $(t_1t_3t_2t_4)^\infty$ satisfies $<\gamma> tt$. However if we restrict yourselves to the states p_0,p_1,p_3,p_4,p_7 (referred to as g_2) we do indeed have $p_0 \models_{g_2} Ev <\gamma> tt$, since every computation from p_0 must eventually reach p_4 — t_5 cannot be continuously ignored.

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

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

The above tree is constructed according to some intuitive tableau rules. Although informal, the example provides the first important observation. The leftmost branch begins and ends with the sequent $p_0 \vdash Ev <\gamma> tt$. In the μ-calculus $Ev <\gamma> tt$ is expressed by the formula $\mu X. <\gamma> tt \vee [Act]X$. Hence based on the tableau methods from [8, 21], one might expect that the above tree should be discarded as a tableau since in the unfolding of the minimal fixed-point assertion reaches itself. However in the current framework, we interpret the logic over maximal traces, and the detected loop, $(p_0 \xrightarrow{t_1} p_1 \xrightarrow{t_3} p_0)^\infty$, is not a computation from p_0 since the transition t_5 is continuously concurrently enabled. This example suggests that we might allow the unfolding of a minimal fixed-point assertion to reach itself. The cases in which this will be allowed should include the existence of an transition that is continuously concurrently 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 continuously concurrently 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 $p_0 \to p_1 \to p_0$ 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 $p_0 \vdash Ev <\gamma> tt$:

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

Again the interesting parts are the branches that unfold a minimal fixed-point assertion into itself. There are two such branches, the leftmost and the rightmost. However along both of these there are transitions which are continuously concurrently 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 $p_0 \models_{g_1} \neg Ev <\gamma> tt$. 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 continuously concurrently 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 $p_0 \vdash Ev <\gamma> tt$. 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 continuously concurrently 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 subsection we provide the necessary definitions. The bound we use is at most $|T|$, the number of transitions in the labeled 1-safe net.

4.2 Tableau Rules

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

4.2.1 Annotated Logic

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

$$B \ ::= \ tt \mid \neg B \mid B_1 \wedge B_2 \mid \Diamond_\alpha B \mid B_1 \ U_\exists^{\mathcal{C}} \ B_2 \mid B_1 \ U_\forall^{\mathcal{C}} \ B_2$$

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

For the $U_\forall$ operator we need a more elaborate annotation. In the tableau rules we use the following annotations: $B_1 \ U_\forall^{(p,n,T')} \ B_2$, $B_1 \ U_\forall^{(p,n,T',V',\rightarrow)} \ B_2$, and $B_1 \ U_\forall^{(p,n,T',V',\leftarrow)} \ B_2$, where B_1, B_2 are formulas from the annotated logic and $p \in V$, $T' \subseteq T$, $V' \subseteq V$, and $n \in I\!N$. V' plays the same role as $\mathcal{C}$ and T' keeps track of which transitions have been concurrently enabled but ignored along the path corresponding to the current branch in the tree. p is a state from which we try to detect certain "critical loops". In the example from section 4.1 p would correspond to p_0 and the "critical loops" would correspond to $p_0 \xrightarrow{t_1}\xrightarrow{t_3} p_0$ and $p_0 \xrightarrow{t_2}\xrightarrow{t_4} p_0$.

4.2.2 Rules

In this section we present the tableau rules. 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[2]. At the cost of presenting less simple tableau rules we keep the definition of tableaux simple.

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

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

Tableau Rules

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

$\Diamond_\alpha$, 3) $$\frac{p \vdash \Diamond_\alpha B}{q \vdash}$$

- $t \in T, q \in V, p \xrightarrow{t} q$
- $l(t) = \alpha$

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

- $p \notin \mathcal{C}$.

4) $$\frac{p \vdash B_1 \ U_\exists^{\mathcal{C}} \ B_2}{p \vdash B_1 \quad q \vdash B_1 \ U_\exists^{\mathcal{C} \cup \{p\}} \ B_2}$$

- $p \notin \mathcal{C}, t \in T, q \in V, p \xrightarrow{t} q$.

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

- $p \notin \mathcal{C}$.

6) $$\frac{p \vdash B_1 \ U_\forall^{\mathcal{C}} \ B_2}{p \vdash B_1 \quad q_1 \vdash B_1 \ U_\forall^{\mathcal{C} \cup \{p\}} \ B_2 \cdots q_m \vdash B_1 \ U_\forall^{\mathcal{C} \cup \{p\}} \ B_2}$$

- $p \notin \mathcal{C}, next(p) = \{t_1, \ldots, t_m\}$, $0 < m \in I\!N$,
- $(\forall 1 \leq i \leq m. \ p \xrightarrow{t_i} q_i)$

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) $$\frac{p \vdash B_1 \ U_\forall^{(p,n,T')} \ B_2}{p \vdash B_1 \ U_\forall^{(p,n-1,T',\emptyset,\rightharpoonup)} \ B_2}$$

- $0 < n \in I\!N, T' \neq \emptyset$

9) $$\frac{q \vdash B_1 \ U_\forall^{(p,n,T',V',\rightharpoonup)} \ B_2}{q \vdash B_1 \quad q_i \vdash B_1 \ U_\forall^{(p,n,e_i I T', V' \cup \{q\}, \rightharpoonup)} \ B_2}$$

- $q \notin V', next(q) = \{t_1, \ldots, t_m\}$, $0 < m \in I\!N$,
- $(\forall 1 \leq i \leq m. \ q \xrightarrow{t_i} q_i)$

and for $D \subseteq T$, $t \in T$, we define $tID = DIt = \{t' \in D \,|\, t'It\}$.

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

- $q \notin V'$

11) $$\frac{q \vdash B_1 \ U_\forall^{(p,n,T',V' \cup \{q\},\rightharpoonup)} \ B_2}{q \vdash B_1 \ U_\forall^{(p,n,T',\emptyset,\leftharpoondown)} \ B_2}$$

12) $$\frac{q \vdash B_1 \ U_\forall^{(p,n,T',V',\leftharpoondown)} \ B_2}{q \vdash B_1 \quad q_i \vdash B_1 \ U_\forall^{(p,n,e_i I T', V' \cup \{q\}, \leftharpoondown)} \ B_2}$$

- $q \notin V', next(q) = \{t_1, \ldots, t_m\}$, $0 < m \in I\!N, q \neq p$
- $(\forall 1 \leq i \leq m. \ q \xrightarrow{t_i} q_i)$

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

- $q \notin V'$

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

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

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

$$\begin{tikzcd}
p \arrow[r, "\sigma_{pp'}"] & p' \arrow[r, "\sigma_{p'q_i}"'] & q_i \arrow[l, bend right, "\sigma_{q_ip'}"'] \arrow[loop right, "\sigma_{q_i-loop}"]
\end{tikzcd}$$

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

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

4.2.3 Derivation Trees and 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 formulae, 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 $T_1, \ldots, T_n$ 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 labeled by the sequent above the bar.

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

- $p \vdash tt$ is a derivation tree.
- $p \vdash \neg B$ is a derivation tree.
- $p \vdash B_1\ U_\exists^{\mathcal{C}}\ B_2$, where $p \in \mathcal{C}$ is a derivation tree.
- $p \vdash B_1\ U_\forall^{(p,n,T')}\ B_2$, 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 T_1 is a derivation tree with root $p \vdash B_1$ and T_2 is a derivation tree with root $q \vdash B_1\ U_\exists^{\mathcal{C}\cup\{p\}}\ B_2$, where $p \notin \mathcal{C}$ and $\exists t \in T.p \xrightarrow{t} q$ then $\dfrac{p \vdash B_1\ U_\exists^{\mathcal{C}}\ B_2}{T_1 \quad T_2}$ is a derivation tree with root $p \vdash B_1\ U_\exists^{\mathcal{C}}\ B_2$.

- If T is a derivation tree with root $p \vdash B_2$ and $p \notin \mathcal{C}$ then $\dfrac{p \vdash B_1\, U_\forall^{\mathcal{C}}\, B_2}{T}$ is a derivation tree with root $p \vdash B_1\, U_\forall^{\mathcal{C}}\, B_2$.

Nothing else is a derivation tree. Next, using the annotation of the formulae, we obtain two useful definitions:

- Let Ann be the obvious function which takes a formula A from the first grammar and annotates all its $U_\exists$ and $U_\forall$ operators with $\mathcal{C} = \emptyset$, giving a formula $Ann(A)$ from the second grammar. A formula B from the second grammar is said to be *clean* if there exists a formula A from the first grammar such that B equals $Ann(A)$.
- Sequents of the form $q \vdash B_1\, U_\forall^{(q,n,\emptyset)}\, B_2$, where $n \in I\!N$ and $q \in V$, and of the form $q \vdash B_1\, U_\exists^{\mathcal{C}}\, B_2$, where $q \in \mathcal{C}$, are called terminal sequents.

We can now define tableaux and will hereby restrict our attention to meaningful derivation trees. We get rid of meaningless derivation trees as for example $p \vdash \neg tt$.
A tableau is a derivation tree 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
  - every proper subtree T' of T whose root is labeled with a clean formula is itself a tableau and
  - T has no leaves labeled with terminal sequents.

5 Soundness and Completeness

Having given the necessary definitions we are now ready to state the main result, which holds for the reachability graph of a labeled 1-safe net N, where p is a reachable marking of N:

Theorem 9

Soundness:

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

Completeness*:*

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

Proof. The proof proceeds by structural induction, showing soundness and completeness simultaneously. The main difficulty is the $U_\forall$ operator. The proof is constructed using the technique from [2]. □

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

$$\dfrac{i \vdash tt\, U_\forall^{\emptyset}(<b>tt)}{i \vdash tt \qquad \dfrac{i \vdash tt\, U_\forall^{\{i\}}(<b>tt)}{\dfrac{i \vdash tt\, U_\forall^{(i,2,\{t_1,t_2\})}(<b>tt)}{T_1}} \qquad \dfrac{s_1 \vdash tt\, U_\forall^{\{i\}}(<b>tt)}{\dfrac{s_1 \vdash <b>tt}{s_2 \vdash tt}}}$$

where T_1 is

$$\dfrac{i \vdash tt\, U_\forall^{(i,1,\{t_1,t_2\},\emptyset,\rightarrow)}(<b>tt)}{\dfrac{i \vdash tt\, U_\forall^{(i,1,\{t_2\},\{i\},\rightarrow)}(<b>tt)}{\dfrac{i \vdash tt\, U_\forall^{(i,1,\{t_2\},\emptyset,\leftarrow)}(<b>tt)}{\dfrac{i \vdash tt\, U_\forall^{(i,1,\{t_2\})}(<b>tt)}{T_2}}} \qquad \dfrac{s_1 \vdash tt\, U_\forall^{(i,1,\{t_1\},\{i\},\rightarrow)}(<b>tt)}{\dfrac{s_1 \vdash <b>tt}{s_2 \vdash tt}} \qquad i \vdash tt}$$

where T_2 is

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

Notice that if we restrict ourselves to labeled 1-safe nets where the independence relation is empty and translate $A_1\ U_\exists\ A_2$ as $\mu X.\, A_2 \vee (A_1 \wedge <Act>X)$ and $A_1\ U_\forall\ A_2$ as $\mu X.\, A_2 \vee (A_1 \wedge <Act>tt \wedge [Act]X)$ (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 [8, 21].

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 translate obtained results and notions from the interleaving models to the "true concurrency" models [6, 5, 23, 9]. Trying to contribute to the "translation of results" we have provided proof rules for a *CTL*-like logic interpreted over maximal traces. The work which we have tried to "translate" can be found in [8, 21]. Our work supports automatic verification of distributed systems whose liveness properties are only provable under the assumption of progress.

A modal operator specifying concurrent behaviour have been presented in [9]. In [2] we present other modal operators specifying concurrent or conflicting behaviour. By extending *P-CTL* with such modal operators one can also obtain undecidability results such as those in [9]. Also, the operators presented in [2] are all incomparable. The choice of the modal operators has partly been made because we want to draw attention to the fact that there is no obvious choice. This leaves open the problem of identifying a (set of) more general modal operator(s).

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

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

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

References

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

[2] Allan Cheng. Local model checking and traces. Technical report, Daimi, Computer Science Department, Aarhus University, May 1994. BRICS Report Series RS-94-17.

[3] 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.

[4] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[21] 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.

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

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