Model-Checking of Causality Properties

(Appeared in the Proceedings of the Tenth IEEE Symposium on Logic in Computer Science, pp. 90-100, 1995)

Rajeev Alur   Doron Peled
AT&T Bell Laboratories
600 Mountain Ave.
Murray Hill, NJ 07974 , USA
alur, doron @ research.att.com

Wojciech Penczek
Eindhoven University of Technology
Den Dolech 2, Eindhoven
The Netherlands
wojtek @ win.tue.nl

Abstract

A temporal logic for causality (TLC) is introduced. The logic is interpreted over causal structures corresponding to partial order executions of programs. For causal structures describing the behavior of a finite fixed set of processes, a TLC-formula can, equivalently, be interpreted over their linearizations. The main result of the paper is a tableau construction that gives a singly-exponential translation from a TLC formula φ to a Streett automaton that accepts the set of linearizations satisfying φ. This allows both checking the validity of TLC formulas and model-checking of program properties. As the logic TLC does not distinguish among different linearizations of the same partial order execution, *partial order reduction* techniques can be applied to alleviate the state-space explosion problem of model-checking.

1 Introduction

One of the most successful techniques for automatic verification of finite-state systems has been *model-checking*. A model-checking algorithm decides whether a finite-state concurrent system satisfies its specification, given as a formula of a temporal logic [2, 8]. A single execution of a concurrent system can be modeled in two ways. In the *interleaving semantics*, the meaning of a program is a set of linear executions, temporal-logic assertions are interpreted over executions, and a program satisfies a formula φ iff all of its executions satisfy φ. In *partial-order semantics* (or trace semantics), a single execution is a partial order, where the ordering relation over events (or program states) reflects the causal dependency among them [9, 18]. The connection between these two models is that the interleaved execution of a program is a linearization of a partial order execution (i.e., a total order that contains the partial order).

There is a long and rich tradition of research that employs the interleaving semantics, resulting in both theoretical and practical results. The main reason for this is the simplicity of the model providing a natural connection with automata theory. In this framework, a concurrent system P, possibly with fairness requirements, is an ω-automaton M_P that generates all possible total-order executions. The commonly employed specification language is LTL [16]. To check whether the system M_P satisfies an LTL-formula φ, the model-checking algorithm first constructs an ω-automaton $M_{\neg\varphi}$ that accepts all the satisfying models of $\neg\varphi$, and tests emptiness of the intersection of the languages of the two automata M_P and $M_{\neg\varphi}$ [23, 24].

The use of partial-order semantics is less common, but has attracted researchers in concurrency theory for many reasons:

- It does not distinguish among total-order executions that are *equivalent* up to reordering of independent transitions, thereby, resulting in a more abstract and faithful representation of concurrency.
- Logics over partial orders [6, 17, 12, 13, 14, 15] allow a direct representation of properties involving causality and concurrency. These properties are hidden, and sometimes disappear completely when representing the executions as linear interleavings.

In recent years, the partial-order semantics has enjoyed popularity with the model-checking practitioners also due to the following reason.

> The partial-order equivalence among sequences can be exploited to reduce the state-space: the cost of generating at least one representative per equivalence class is typically significantly less than the cost of generating all interleavings [11, 4, 5, 22]. This approach has been shown to be practically useful in verification tools like SPIN [3].

If the specification could distinguish between two sequences of the same equivalence class, as is the case with LTL, the above equivalence cannot be used: the same equivalence class may contain both a sequence that satisfies the specification and a sequence that does not. It is possible to refine the equivalence relation, providing more representatives at the expense of using a bigger state space [11, 4]. The alternative solution is to use a specification logic that is directly interpreted over partial-orders. For this latter approach to be fruitful, we need to examine the following question:

> Is it possible to translate the modal operators interpreted over a partial-order to requirements on its linearizations?

We answer the above question in the affirmative for a partial-order temporal logic called TLC that subsumes many previously defined partial-order logics. The main result of the paper is an automata-theoretic model-checking algorithm for TLC. As the logic does not distinguish among different linearizations of the same partial order execution, the maximum possible partial-order reduction admitted by the program can be used to alleviate the state-space explosion problem of model-checking.

Overview

We first define the temporal logic TLC for expressing properties of partial order executions. The modalities of TLC include both *future* and *past* operators ranging over paths of the partial order. For example, $\exists\Diamond a$ is satisfied in a state if there is a causally dependent a-state in the future, $\forall\boxminus a$ is satisfied in a state s if all the states that s causally depends upon satisfy the predicate a. The logic also includes a temporal operator to express concurrency: $\exists\| a$ is satisfied in a state if there is a concurrent (or causally independent) a-state. Thus, the logic allows expressing causality and concurrency directly, as first class temporal modalities.

The model of TLC formulas is a *causal structure*, i.e., a Kripke structure whose transition relation is antisymmetric and reduced, and thus, can be interpreted as the causal successor relation. The causal structures generated by a concurrent program have a special structure, namely, the causality ordering is fully specified by the propositions that indicate which processes participate in the executions of individual events. Such structures, called *concurrent structures*, are uniquely determined by their linearizations. We exploit this property to construct, given a TLC formula φ, a Streett automaton [20] M_φ that accepts all the linearizations of all the causal structures satisfying φ. The size of M_φ is $2^{\mathcal{O}(nk^2)}$, where n is the size of φ and k is the number of concurrent processes. The crucial step of the construction involves reinterpreting the modal operators over a causal structure using auxiliary operators over linearizations. The decision procedure for the auxiliary (linear) operators is similar in spirit to the tableau-construction for LTL, but there are several interesting points of departure, particularly, in the treatment of the eventualities. The size of the closure is exponential in the size of the formula. Maximality and monotonicity arguments are used to limit the number of consistent subsets of the closure to a single exponential in the size of the formula.

The tableau-construction gives a PSPACE decision procedure to test whether a TLC-formula is satisfiable over concurrent structures. For model-checking of a TLC-specification φ of a concurrent program P, first construct an automaton $M_{\neg\varphi}$ for the negated property, then construct an automaton M_P that generates the program executions, and check if the intersection of the languages of these two automata is empty. Since φ does not distinguish among linearizations of the same partial-order, M_P may generate only one representative per equivalence class, thereby, admitting the benefits of full partial order reduction.

Related work

Several logics [6, 12, 13, 14, 15, 17] allow specifying properties over partial-order executions. The syntax of TLC is reminiscent of the logic POTL [17]. However, none of these works relate the partial-order executions with a representation over interleaving sequences and ω-automata. Also, note that the decision procedure for POTL [17] cannot be used to obtain a decision procedure for TLC—even for the fragment without the parallel operator—since we require the causal ordering relation to be reduced.

The parallel operator $\exists\|$ was first introduced in [10] and interpreted over prime event structures [25]. Our logic is however quite different from the logics defined on prime event structures [10, 13, 14]. First, since event structures are equipped with a conflict relation, they represent the whole branching structure of a program rather than a single execution. Second, since finite acceptors of event structures are not known, no model checking method for event structure logics has been suggested.

A first step towards connecting partial order executions with linearizations and automata was made in the definition of asynchronous Büchi automata [26]. These automata can be seen as recognizers of traces [9, 7] (i.e., equivalence classes of linearizations of partial order executions) or alternatively directly recognizing the linearizations. A logic, called TRPTL, that is translatable to such automata was defined by Thiagarajan in [21]. Our logic is different than the one in [21] in the interpretation of its set of modalities: it has the usual meaning of causality and concurrency over causal structure, while the logic in [21] is strongly based on the notion of prime configurations (or causal histories of events). Our logic—even the fragment without the parallel operator—is strictly more expressive than TRPTL. The translation in [21] has a complexity of $2^{\mathcal{O}(|\varphi|^3 log|\varphi|)}$. Our decision procedure is more efficient in the size of the formula, and hopefully, is much easier to understand as it uses the conventional tableau techniques.

2 A Causality Logic

Preliminaries

Let S be a countable set, and let R be a binary relation over S. The inverse of R is denoted by R^{-1}. For an element $s \in S$, the set $pre_R(s)$ contains the elements $t \in S$ such that $(t,s) \in R$, and $post_R(s)$ equals $pre_{R^{-1}}(s)$. An element $s \in S$ is *R-initial* if $pre_R(s)$ is empty. The relation R is *prefinite* if $pre_R(s)$ is finite for all elements $s \in S$. Let R^* be the reflexive-transitive closure of the relation R. The relation R is *reduced* if for each $(s,t) \in R$, $(s,t) \notin (R \setminus (s,t))^*$. Note that if R is reduced then R is irreflexive, and if $(s,t) \in R$ and $(t,u) \in R$ then $(s,u) \notin R$.

An *R-chain* is a finite or infinite sequence $s_0 s_1 \ldots$ of elements $s_i \in S$ such that $(s_i, s_{i+1}) \in R$ for all $i \geq 0$. A *maximal* R-chain is either an infinite R-chain, or a finite R-chain $s_0 \ldots s_m$ such that $post_R(s_m)$ is empty. Two elements s and t are *R-connected* iff there is a finite $(R \cup R^{-1})$-chain from s to t.

The relation R is a *partial order* if R is reflexive, transitive, and antisymmetric. The relation R is a *causal order* if R is antisymmetric and reduced. We use $\prec$ to denote a causal order, and $\leq$ to denote the corresponding partial order, i.e., its reflexive-transitive closure. If $\leq$ is prefinite then the set of $\prec$-initial elements is nonempty.

A partial order $\leq$ is a total order if for every two elements s and t, either $t \leq s$ or $s \leq t$. If $\{\leq_i\}$ is a family of total orders then the intersection $\cap_i \leq_i$ is a partial order. A linearization of a partial order $\leq$ is a total order that contains $\leq$. Every partial order equals the intersection of all its linearizations; every prefinite partial order equals the intersection of all its prefinite linearizations. An enumeration $\overline{s} = s_0 s_1 \ldots$ of S defines the prefinite causal order $\prec_{\overline{s}} = \{(s_i, s_{i+1}) \mid i \geq 0\}$. We will not distinguish between an enumeration and the causal order it defines. A prefinite linearization of a prefinite causal order is an enumeration.

Causal Structures

A *causal structure* K consists of

State-space: a countable infinite set S of states;
Causality relation: a causal order $\prec$ s.t. its corresponding partial order $\leq$ is prefinite;
Propositions: a finite set A of propositions;
Valuation: a mapping λ that assigns each state $s \in S$ a set $\lambda(s) \subseteq A$ of propositions true in s.

An $\prec$-initial state is an initial state of K, and the set of initial states is denoted by S^0. Let s and t be two states. If $s \leq t$ or $t \leq s$, then s and t are said to be (causally) *dependent*. If neither $s \leq t$ nor $t \leq s$, then s and t are said to be *concurrent*, written $s \| t$. The concurrency and dependency relations are symmetric. A *connected* causal structure is a causal structure $(S, \prec, A, \lambda)$ where every pair of two states is $\prec$-connected. A *rooted* causal structure has a single initial state (i.e. $|S^0| = 1$). A rooted causal structure is connected.

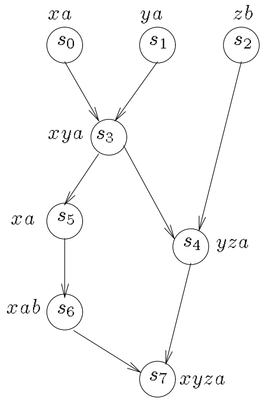

Figure 1: A sample causal structure K_0

Figure 1 shows a fragment of a causal structure over the propositions $\{x, y, z, a, b\}$, where $\{s_0, s_1, s_2\}$ is the set of initial states.

Syntax

The formulas of TLC are defined inductively by the grammar

$$\varphi \;:=\; a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists\bigcirc\, \varphi \mid \exists\Box\varphi \mid \varphi\, \exists\mathcal{U} \varphi \\ \mid \exists\ominus\varphi \mid \varphi\, \exists\mathcal{S}\varphi \mid \exists\|\varphi$$

where $a \in A$ is a proposition.

Apart from the usual boolean connectives and logical constants, we define the following temporal modalities: $\exists\Diamond\varphi$ for $true\, \exists\mathcal{U}\varphi$, $\forall\bigcirc\,\varphi$ for $\neg\exists\bigcirc\,\neg\varphi$, $\forall\Box\varphi$ for $\neg\exists\Diamond\neg\varphi$, $\forall\Diamond\varphi$ for $\neg\exists\Box\neg\varphi$ and $\forall\|\varphi$ for $\neg\exists\|\neg\varphi$. Past-modalities are similarly defined (note that $\exists\boxminus\varphi$ is definable as $(\varphi\, \exists\mathcal{S}(\varphi \wedge \forall\ominus false))$).

Semantics

The formulas of our logic are interpreted over the states of a causal structure. The interpretation is similar to the branching-time operators of CTL (or POTL), but the paths are causal within a single computation. Thus $\exists\bigcirc\,\varphi$ means that φ holds in a state resulting from executing one of the causally-dependent events; $\exists\diamondminus\varphi$ means that φ holds in some state in the causal past; $\exists\|\varphi$ means that there is a causally unrelated (concurrent) φ-state.

For a TLC-formula φ, A_φ denotes the (finite) set of propositions appearing in φ. A *model* of a TLC-formula φ is a causal structure K over the propositions A_φ. Let φ be a TLC-formula, $K = (S, \prec, A_\varphi, \lambda)$ be a model of φ, and s be a state of K. Then, the satisfaction relation $(K, s) \models \varphi$ is defined inductively below (when the structure is understood, $(K, s) \models \varphi$ is abbreviated to $s \models \varphi$):

$s \models a$	iff $a \in \lambda(s)$;
$s \models \neg\varphi$	iff $s \not\models \varphi$;
$s \models \varphi \wedge \psi$	iff $s \models \varphi$ and $s \models \psi$;
$s \models \exists\bigcirc\varphi$	iff $s \prec t$ and $t \models \varphi$ for some state t;
$s \models \exists\ominus\varphi$	iff $t \prec s$ and $t \models \varphi$ for some state t;
$s \models \exists\Box\varphi$	iff there is a maximal $\prec$-chain $s_0 s_1 \ldots$ such that $s_0 = s$ and $s_i \models \varphi$ for all $i \geq 0$;
$s \models \varphi\,\exists\mathcal{U}\,\psi$	iff there is a finite $\prec$-chain $s_0 \ldots s_m$ such that (1) $s_0 = s$, and (2) $s_m \models \psi$, and (3) for all $0 \leq i < m$, $s_i \models \varphi$;
$s \models \varphi\,\exists\mathcal{S}\,\psi$	iff there is a finite K-chain $s_0 \ldots s_m$ such that (1) $s_m = s$, and (2) $s_0 \models \psi$, and (3) for all $0 < i \leq m$, $s_i \models \varphi$;
$s \models \exists\|\varphi$	iff $s\|t$ and $t \models \varphi$ for some state t.

Denote $K \models \varphi$ iff $(K, s) \models \varphi$ for all initial states $s \in S^0$. A formula φ is *valid* iff $K \models \varphi$ for *all* models K of φ. A formula is *satisfiable* iff $K \models \varphi$ for *some* model K.

Since a causal structure has many initial states, it may be the case that both $K \not\models \varphi$ and $K \not\models \neg\varphi$. However, if K is rooted then either $K \models \varphi$ or $K \models \neg\varphi$.

For the structure K_0 of Figure 1, we have

$$s_0 \models a\,\exists\mathcal{U}\,b,\;\; s_3 \models \forall\|\,b,\;\; s_7 \models \exists\Box\,a,\;\; K_0 \models \exists\Diamond\,b.$$

Concurrent structures

A *process-set* $\Pi \in A$ is a finite set of propositions. Let $K = (S, \prec, A, \lambda)$ be a causal structure. The causal structure K is said to be Π-*concurrent* iff

(1) for all $s \in S$, $\lambda(s) \cap \Pi$ is nonempty;
(2) for all $s, t \in S$, if $\lambda(s) \cap \lambda(t) \cap \Pi$ is nonempty then either $s \leq t$ or $t \leq s$;
(3) whenever $s \prec t$, there exists a process $x \in \Pi$ such that $x \in \lambda(s) \cap \lambda(t)$.

Thus, the causality relation of a Π-concurrent causal structure is the union of total orders, one per each process. The causal structure K_0 of Figure 1 is $\{x, y, z\}$-concurrent.

A process $x \in \Pi$ *participates* in a state s iff $x \in \lambda(s)$. All the states in which x participates are causally related. A state with many participating processes corresponds to a communication event. Since the causality relation $\prec$ is reduced, the causal successors of a state correspond to events that are independent of each other. For instance, in Figure 1, the state s_3 corresponds to a communication event involving x and y. Following this event, the process x can execute the event s_5 on its own, and the process y can participate in the event s_4 (note that for the event s_4 to occur, the process z must have executed s_2). The number of causal successors of each state is bounded by the number of processes.

For a process-set Π, a Π-model for a TLC-formula φ is a Π-concurrent causal structure over the propositions $A_\varphi \cup \Pi$. A TLC-formula φ is Π-valid (Π-satisfiable) iff $K \models \varphi$ for every (some) Π-model K of φ.

From Concurrent Structures to ω-languages

A *linear* structure M consists of

an enumeration $\overline{s} = s_0 s_1 \ldots$ of a state-space;
a finite set A of propositions; and
an ω-word $\overline{w} = w_0 w_1 \ldots$ with each $w_i \subseteq A$.

Note that the state-space is implicitly specified by the enumeration $\overline{s}$ and the valuation is specified by the ω-word $\overline{w}$. A linear structure $M = (\overline{s}, A, \overline{w})$ is a linearization of the causal structure $K = (S, \prec, A, \lambda)$ if (1) $\overline{s}$ is a linearization of $\prec$, and (2) for all $i \geq 0$, $\lambda(s_i) = w_i$. When the causal structure is Π-concurrent, its linearization corresponds to a possible interleaving of all the states. Observe that two linearizations of the same causal structure are equivalent up to commuting concurrent states. This is the so-called trace-equivalence [9, 7] as explained in Section 4.

Proposition 1 *Let $M = (\overline{s}, A, \overline{w})$ be a linear structure, and let Π be a process-set. Then there exists a unique Π-concurrent causal structure $K_{M,\Pi}$ such that M is a linearization of $K_{M,\Pi}$.*

A possible linearization of the causal structure K_0 of Figure 1 is the linear structure M_0 with enumeration

$$s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7$$

and labeling

$$xa, ya, zb, xya, yza, xa, xab, xyza.$$

Given M_0, the causality arrows of K_0 can be inferred correctly using the labeling of states with the processes.

Let φ be a TLC-formula, and Π be a process-set. The formula φ and a process-set Π, then, defines the ω-language $L(\varphi, \Pi)$ over the alphabet $2^{A_\varphi \cup \Pi}$: an ω-word $\overline{w}$ is in $L(\varphi, \Pi)$ iff there exists a linear structure $M = (\overline{s}, A_\varphi \cup \Pi, \overline{w})$ such that $K_{M,\Pi} \models \varphi$.

Examples

Server requests.

Consider a system where users send requests by setting the flags req_a, req_b, and the server grants resources by setting the flags gra_a, gra_b. We can impose various constraints on its executions.

A request is granted only when requested:

$$\forall\Box(\,gra_a \rightarrow \exists\ominus\!\!\Diamond\, req_a).$$

Each request will be eventually granted:

$$\forall\Box(\,req_a \rightarrow \exists\Diamond\, gra_a).$$

An additional request is never asked before a pending causally-dependent request is granted (however, we allow multiple pending concurrent requests):

$$\forall\Box(\,req_a \rightarrow \neg\exists\bigcirc\,(\neg gra_a\,\exists\mathcal{U}\,req_a)).$$

Execution of database transactions.

Consider two finite transactions P and Q that can potentially overlap in time. These can be, for instance, airline reservations, or money transfers between bank accounts. We want each execution that includes the events of both transactions to be equivalent to its serial execution, i.e., an execution in which the events of one transaction precede the events of the other one. One way to ensure this is to disallow the situation when events of one transaction causally precede *and* causally succeed events of the other. This is the kind of serializability that is defined e.g., in [1]. Let p and q be the predicates that hold in states of the transactions P and Q, respectively. Then the desired requirement is specified by the formula:

$$\neg((\exists\Diamond(p\,\wedge\,\exists\Diamond q))\wedge(\exists\Diamond(p\,\wedge\,\exists\overline{\Diamond} q))).$$

One can also express a requirement that the transactions execute more efficiently, i.e., that some overlap is allowed in their executions:

$$\exists\Diamond(p\,\wedge\,\exists\|\,q).$$

3 Decision Procedure

3.1 Computing the causal successors

As a first step towards a decision procedure for TLC, let us try to compute the causality relation from a linearization. Let $M = (\overline{s}, A, \overline{w})$ be a linear structure, and $\Pi \subseteq A$ be a process-set. Let $\prec$ be the causal order of the structure $K_{M,\Pi}$.

Given a position $i \geq 0$, we need to determine the set of indices j such that $s_i \prec s_j$. Clearly, for s_j to be a causal successor of s_i, it is necessary that $j > i$ (i.e. s_j appears after s_i in the linearization), and there exists a process $x \in \Pi$ such that both s_i and s_j are labeled with x. Furthermore, s_j must be the first state following s_i that is labeled with x (i.e. $x \notin w_k$ for $i < k < j$). However, these conditions are not sufficient, and we need to check that there exists no non-trivial (i.e., of length $n > 2$) chain $s_i = s_{l_1} \prec s_{l_2} \prec \ldots \prec s_{l_n} = s_j$; this condition is an artifact of the constraint that $\prec$ is reduced. Consider the two linearizations shown in Figure 2; the edges indicate the causal successors. For the linearization $s_0s_1s_2s_3$, s_3 is a causal successor of s_0, but for the linearization $t_0t_1t_2t_3$, t_3 is not a causal successor of t_0.

For a position $i \geq 0$, let $succ_i \subseteq w_i \cap \Pi$ be the set of processes x such that there exists a position j with $s_i \prec s_j$ and $x \in w_j$. Computing the causal successors of s_i corresponds to computing the set $succ_i$. We define two functions f and g that will be useful in computing this set. Given two indices $0 \leq i < j$, define

$$f(i,j) \;=\; (w_j \cap \Pi) \setminus (\cup_{i<k<j} w_k).$$

The function g is defined inductively:

$$g(i,i+1) \;=\; \Pi \setminus w_{i+1},$$

and if $i+1 < j$ then

$$g(i,j) = \begin{cases} g(i+1,j) & \text{if } w_{i+1} \cap \Pi \subseteq g(i+1,j) \\ g(i+1,j) \setminus w_{i+1} & \text{otherwise} \end{cases}$$

The values of $f(i,3)$ and $g(i,3)$ are shown for $i = 0,1,2$ for both the linearizations of Figure 2. The relation between causal successors and the values of the functions f and g are related as follows.

Proposition 2 *Let i and j be indices such that $i < j$. Then the states s_i and s_j are concurrent iff $(w_i \cap \Pi) \subseteq g(i,j)$.*

Proposition 3 *Given a process x and an index i, let j be the least position after i where x holds. Then, $x \in succ_i$ iff $x \in w_i \cap f(i,j)$ and $g(i,j) \supseteq (w_i \cap \Pi) \setminus f(i,j)$.*

Proposition 2 is useful in checking the operator $\exists\|$ and Proposition 3 is useful in checking the operator $\exists\bigcirc$.

3.2 An extended language

To translate TLC-modalities over causal structures to requirements over linearizations, we introduce two linear operators. A formula φ^* of the extended logic TLC* has one of the following forms:

$$\varphi^* \;:=\; \varphi \mid F(\varphi,\psi,X,Y) \mid P(\varphi,\psi,X,Y)$$

where φ, ψ are TLC-formulas, and X, Y are process-sets.

The formulas of TLC* are interpreted over linear structures. Consider a linear structure M and a process-set Π. A TLC-formula φ is satisfied at the i-th position of M, if the state s_i satisfies φ in the corresponding causal structure $K_{M,\Pi}$. For the formula $F(\varphi,\psi,X,Y)$ to hold at the i-th position, there are several requirements:

(1) the formula φ must be satisfied at some later position,

(2) if j is the least position after position i where φ holds, then the formula ψ must also be satisfied at position j.

(3) $f(i,j) = X$.

(4) $g(i,j) = Y$.

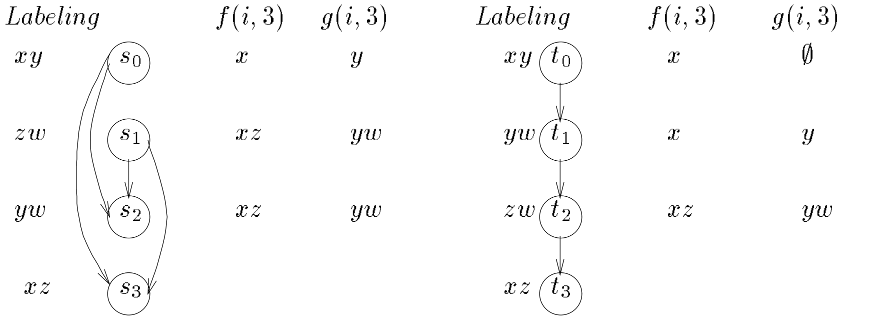

Figure 2: Computing the causal successors

Note that the position j is uniquely determined by the first argument, and uniquely determines the last two arguments X and Y (recall that f and g are well-defined functions). This is crucial in avoiding a doubly-exponential blow-up (in the size of the process-set Π) while building a tableau. The operator P is the past-dual of F. The new operators are useful in deciding whether state s_j is a causal successor of s_i, and also whether s_j is independent of s_i.

Let $M = (\overline{s}, A_{\varphi^*} \cup \Pi, \overline{w})$ be a linear structure, Π be a process-set, and $i \geq 0$. We write $(M,i) \models_\Pi F(\psi,\chi)$ iff there exist process-sets X, Y such that $(M,i) \models_\Pi F(\psi,\chi,X,Y)$. The tableau construction relies on the following properties of the operator F. The operator P has similar properties.

F1. $x \in succ_i$ iff $(M,i) \models_\Pi F(x, true, X, Y)$ for some process-sets X, Y such that $x \in X \cap w_i$ and $Y \supseteq (w_i \cap \Pi) \setminus X$.

Follows from the definitions and Proposition 3.

F2. If $(M,i) \models_\Pi F(\varphi,\psi,X,Y) \wedge F(\varphi,\psi',X',Y')$ then $X = X'$ and $Y = Y'$.

Follows from the requirement that the third argument of F equals $f(i,j)$ and the fourth argument equals $g(i,j)$, where j is the least position greater than i in which φ holds.

F3. $(M,i) \models_\Pi \exists\bigcirc\psi$ iff for some $x \in succ_i$, $(M,i) \models_\Pi F(x,\psi)$.

F4. $(M,i) \models_\Pi \exists\|\varphi$ iff there exists a process $x \notin w_i$ and process-sets X and $Y \supseteq w_i \cap \Pi$ such that $(M,i) \models_\Pi F(x \wedge \varphi, true, X, Y) \vee P(x \wedge \varphi, true, X, Y)$.

The state s_i satisfies $\exists\|\varphi$ iff there exists a state s_j that is independent of s_i and satisfies φ. The state s_j may appear before or after s_i in the linearization. Let us consider only the latter case. From Proposition 2, the equivalent condition is that $w_i \cap \Pi \subseteq g(i,j)$. Observe that if $w_k \cap w_j \cap \Pi$ is nonempty for some $i < k < j$, then s_k is also independent of s_i. Hence, we may assume that there exists a process x labeling s_j such that j is the least position following i that satisfies both x and φ. This translates to the requirement $F(x \wedge \varphi, true, X, Y)$ with $Y \supseteq w_i \cap \Pi$. Notice that φ may not hold in the first state before or after s_i that is concurrent with it. Thus, X may be empty (this happens, for instance, if s_i is concurrent with s_{i+1} and $w_{i+1} \cap \Pi = w_{i+2} \cap \Pi$ and $(M, i+1) \not\models \varphi$ and $(M, i+2) \models \varphi$).

F5. $(M,i) \models_\Pi F(\varphi,\psi,X,Y)$ iff either

$(M,i+1) \models_\Pi \varphi \wedge \psi$ and $X = w_{i+1} \cap \Pi$ and $Y = \Pi \setminus w_{i+1}$, or
$(M,i+1) \models_\Pi F(\varphi,\psi,X',Y')$ and $(M,i+1) \not\models_\Pi \varphi$ and $X = X' \setminus w_{i+1}$ and if $w_{i+1} \cap \Pi \subseteq Y'$ then $Y = Y'$ else $Y = Y' \setminus w_{i+1}$.

This rule dictates the propagation of F-formulas. If j is the least position after i satisfying φ then we require that $X = f(i,j)$ and $Y = g(i,j)$. If φ holds in the position $i+1$ then $j = i+1$. In this case, ψ should hold at position $i+1$, $f(i,j)$ equals $w_{i+1} \cap \Pi$, and $g(i,j)$ equals $\Pi \setminus w_{i+1}$. Otherwise, the F-formula needs to be propagated, and we require that $F(\varphi,\psi,X',Y')$ holds in the position $i+1$ such that $X' = f(i+1,j)$ and $Y' = g(i+1,j)$. The relationship between X' and X, and between Y' and Y, is inferred from the definitions of f and g, respectively.

F6. $(M,i) \models_\Pi \psi\,\exists\mathcal{U}\chi$ iff either $(M,i) \models_\Pi \chi$ or $(M,i) \models_\Pi \psi \wedge F(x, \psi\,\exists\mathcal{U}\chi)$ for some $x \in succ_i$.

This property corresponds to the fixpoint definition of $\psi\,\exists\mathcal{U}\chi$.

3.3 Tableau Construction

Reinterpreting the TLC-operators using the additional TLC*-operators over linearizations allows us to use the decision procedure for TLC*-operators in the style of the tableau construction for LTL [8]. The intuitive description of the construction is given below. The full construction is given in the Appendix.

The *closure* set *cl* of a TLC-formula φ and a fixed process-set Π contains the subformulas of φ, and formulas of the form $F(\psi,\chi,X,Y)$ and $P(\psi,\chi,X,Y)$

with $X, Y \subseteq \Pi$. A *tableau* T is a directed graph $(V, \rightarrow)$. The set V of vertices consists of pairs (u, u'), where $u \subseteq cl$ is a *safety-component*, and $u' \subseteq u$ is an *eventuality-component*. The safety component gives all the formulas that are true at the corresponding state and is a *maximal* consistent subset of the closure. As in the tableau-construction for LTL, each formula implies some constraints on the current vertex, and some constraints on its successors. The difference is that the interpretation of the TLC* operators over sequences distinguishes between the successor of a state s_i in a sequence and its causal successors. The causal successors are computed as explained earlier. For this purpose, the closure contains all formulas of the form $F(x, true, X, Y)$ and $P(x, true, X, Y)$ with $x \in \Pi$ and $X, Y \subseteq \Pi$. Hence, the size of the closure is exponential in the size of the process-set. However, condition **F2** ensures that a consistent set of formulas contains only one such F-formula per process, and thus, the number of maximal consistent subsets of the closure is singly-exponential.

A vertex (u, u') is initial iff u does not contain past-obligations (which are formulas of the form $P(\psi, \chi, X, Y)$), and contains the formula φ. An *accepting path* of the tableau is a path which begins with an initial vertex, and contains infinitely many vertices with empty eventuality-components. Furthermore, whenever $F(\psi, \chi) \in u$ for some vertex (u, u') along the path, there must exist a later vertex (v, v') with $\psi \in v$. This will ensure that all eventualities are satisfied. Whenever the automaton reaches a vertex with the empty eventuality component, it will load the eventualities from the safety-component to the eventuality component in the next step. The tableau construction is now explained for formulas of the form $\exists\bigcirc\,\psi$, $\exists\|\psi$, and $\psi\,\exists\mathcal{U}\chi$.

Consider a formula $\exists\bigcirc\,\psi \in u$ for a tableau-vertex (u, u'). Then, according to **F1**, u must also contain $F(x, true, X, Y)$ with $x \in X \cap u$ and $Y \supseteq (u \cap \Pi) \setminus X$. This ensures that the first vertex (v, v') with $x \in v$ reached after (u, u') corresponds to the x-causal successor of (u, u'). According to **F3**, u must also contain the formula $F(x, \psi, X', Y')$, but $X = X'$ and $Y = Y'$ by **F2**. Hence, for the x-causal successor (v, v'), $\psi \in v$. The condition **F5** is used to propagate F-constraints correctly: for any successor (w, w') of (u, u') either both x and ψ are in w or the constraint $F(x, \psi)$ is propagated. In an accepting path containing (u, u'), since $F(x, \psi) \in u$, a vertex containing x must eventually be reached.

Consider a formula $\exists\|\psi \in u$ for the tableau-vertex (u, u'). Then, according to **F4**, u also contains either a formula $F(x \wedge \psi, true, X, Y)$ or a formula $P(x \wedge \psi, true, X, Y)$ with $x \notin u$ and $Y \supseteq u \cap \Pi$. This ensures that either the first vertex (v, v') before or after (u, u') with both x and ψ in v corresponds to a state that is concurrent with (u, u'). If $F(x \wedge \psi, true) \in u$, then again both x and ψ must be eventually fulfilled in an accepting path. If $P(x \wedge \psi, true) \in u$, then the constraints are propagated to predecessors according to the past counterpart of **F5** for P. The initial vertex of an accepting path has no P-requirements, ensuring the fulfillment of the goal $P(x \wedge \psi, true)$.

The semantics of the until-formula $\psi\,\exists\mathcal{U}\chi$ requires that χ is eventually satisfied along some causal chain. Unlike LTL, where for the formula $\psi\mathcal{U}\chi$, it is sufficient to check that the formula χ appears later in the accepting path after a sequence of nodes containing ψ, we need here to trace causal chains in linearizations. The eventuality causes a sequence of subgoals that need to be achieved: these subgoals are of the form $F(x, \psi\,\exists\mathcal{U}\chi)$, where the associated process x may change as the causal path is followed. Consider a formula $\psi\,\exists\mathcal{U}\chi \in u$ for the tableau-vertex (u, u'). Then, according to **F6**, u contains either χ or both ψ and $F(x, \psi\,\exists\mathcal{U}\chi)$, where (u, u') has x-causal successor. As in the case of $\exists\bigcirc\,\psi$, the x-causal successor (v, v') of the current vertex exists with $\psi\,\exists\mathcal{U}\chi \in v$. If $\chi \notin v$, then v contains a new subgoal $F(y, \psi\,\exists\mathcal{U}\chi)$ where (v, v') has y-causal successor. When the automaton loads the eventuality-component, the unfulfilled subgoals appear also in the eventuality component. It is crucial to maintain consistency between the updates in the safety and eventuality component. A subgoal is removed from the eventuality-component only if χ itself has been achieved.

Now we can associate an ω-language $L(T)$ with the tableau T: an ω-word $\overline{w}$ over the alphabet $2^{A_\varphi \cup \Pi}$ is in $L(T)$ iff there exists an accepting path $\overline{u}$ such that $w_i = u_i \cap (A_\varphi \cup \Pi)$. Since the tableau is finite, it is clear that the language $L(T)$ is ω-regular. It is straightforward to translate the requirements on accepting paths to the standard Streett acceptance conditions.

Proposition 4 *The language $L(T)$ of the tableau equals $L(\varphi, \Pi)$.*

3.4 Complexity of testing satisfiability

To test Π-satisfiability of a given TLC-formula φ, construct the corresponding tableau T, and test for the emptiness of $L(T)$.

Theorem 1 *Let φ be a* TLC*-formula and Π be a process-set. Suppose $|\varphi| = n$, $|\Pi| = k$, and ℓ be the number of occurrences of $\exists\|$ in φ. Then, the Π-satisfiability of φ can be checked in time $2^{\mathcal{O}(nk+k^2+\ell k^2)}$.*

Checking emptiness of the language $L(T)$ corresponds to finding accepting paths in the tableau. Since the number of vertices of the tableau is exponential, a standard argument shows that testing Π-satisfiability is in PSPACE. Checking satisfiability of PTL-formulas is a special case of checking Π-satisfiability of TLC-formulas. It follows that checking Π-satisfiability of TLC-formulas is PSPACE-hard. Since $K \models \varphi$ iff all initial states of K satisfy φ, it is not true that φ is valid iff $\neg\varphi$ is not satisfiable. To test Π-validity of φ, we consider the tableau corresponding to $\neg\varphi$, and check if the tableau has an accepting path along which *some* initial state satisfies $\neg\varphi$.

Theorem 2 *The problem of checking Π-satisfiability (Π-validity) of a* TLC*-formula φ, for a given process-set Π, is* PSPACE*-complete.*

4 Model Checking

Given a concurrent system P and a TLC-specification φ, the model-checking problem is to decide whether all partial order executions of P satisfy φ. The previous section showed how to construct a Streett automaton accepting all the linearizations of the causal structures satisfying $\neg\varphi$. Hence, to solve the model-checking problem, it suffices to build an automaton M_P that accepts the linearizations of the partial order executions of P.

A program is a structure[1] P with the following components:

1. Π is a finite set of processes,
2. for each process $x \in \Pi$, a finite nonempty set S_x of local-states of the process x,
3. for each process $x \in \Pi$, a nonempty set $S_x^0 \subseteq S_x$ of initial states of the process x, and
4. for each nonempty process-set $X \subseteq \Pi$, a transition-relation $\tau_X \subseteq (\prod_{x\in X} S_x)^2$.

The transition-relation τ_X models the events in which all processes in X participate. Now, an automaton accepting the linearizations of the partial order executions of a program can be defined. Since the valuation of a global state in a linearization depends not only on its local state components, but also on the processes involved in the transition leading to this state, we have to include processes in the states of the interpreted program automaton defined below.

The *interpreted program automaton* for P is a structure $M_P = \langle G, \rightarrow, G_0, \lambda\rangle$, where:

1. $G = \prod_{x\in\Pi} S_x \times 2^{\Pi}$,
2. $(s, X) \rightarrow (t, Y)$ iff $(s\,|_Y, t\,|_Y) \in \tau_Y$ and $s\,|_{\Pi\setminus Y} = t\,|_{\Pi\setminus Y}$.
3. $G^0 \subseteq G$ is the set of initial states with $(s, X) \in G_0$ iff $s|_x \in S_x^0$ for all processes x and $X = \Pi$,
4. $\lambda : G \mapsto 2^A$ is the global valuation function over a set of propositions $A \supseteq \Pi$ satisfying (a) $\lambda(\langle s, X\rangle) \cap \Pi = X$, and (b) if $s\,|_X = t\,|_X$ then $\lambda(\langle s, X\rangle) = \lambda(\langle t, X\rangle)$.

A transition $\tau = (s, t) \in \tau_X$ is said to be *enabled* from a global state $g = \langle r, Y\rangle \in G$ if $r\,|_X = s$. We also say that τ *involves* the processes X. It follows from 4 above that the valuation of a state depends only on the valuations of local states of the processes involved in the last transition.

A *global execution sequence* $\xi = g_0\, g_1\, g_2\, \ldots$ of M_P is an infinite sequence in G^ω such that $g_0 \in G^0$, and $g_i \rightarrow g_{i+1}$, for each $i \geq 0$. Certain fairness restrictions can limit the set of program executions. One such fairness assumption that we will use in the sequel is the following:

> If some transition τ involving a process-set X is continuously enabled from some state g_i onwards, then some transition τ' involving at least one process from X is executed in some state g_j with $j \geq i$.

This definition can be shown [7, 12] to be equivalent to requiring maximality of partial order executions.

Partial Order Reduction

Partial order reduction techniques allow exploiting the commutativity between transitions to reduce the state space [11, 4, 5, 22]. An important step in using these reductions is defining an irreflexive and symmetric *independence relation* I over the propositional states: for $A_1, A_2 \subseteq A$, $(A_1, A_2) \in I$ iff $A_1 \cap A_2 \cap \Pi = \emptyset$.

The independence relation I induces an equivalence relation over finite sequences. Two sequences $\xi, \xi' \in (2^A)^*$ are equivalent, denoted $\xi \equiv_I \xi'$, iff ξ' can be obtained from ξ by repeatedly commuting independent propositional states. That is, iff there exists a sequence of propositional execution sequences $\xi = \xi_0, \xi_1, \ldots, \xi_n = \xi'$ such that for each $0 < i \leq n$, ξ_{i-1} is a sequence $A_1 A_2 \ldots A_{j-1} A_j \ldots A_m$ and ξ_i is a sequence $A_1 A_2 \ldots A_j A_{j-1} \ldots A_m$, with $(A_{j-1}, A_j) \in I$.

The next step is to extend the equivalence to infinite sequences [7]. Denote by $Pref(\xi)$ the set of finite prefixes of ξ (ξ can be either finite or infinite). Define now an ordering relation over $(2^A)^*$:

$$\xi \preceq_I \xi' \text{ iff } \exists \xi'' \, (\xi' \equiv_I \xi'' \wedge \xi \in Pref(\xi''))$$

The intuitive meaning of $\xi \preceq_I \xi'$ is that ξ is a prefix of some equivalent permutation of ξ'. Now, for two infinite sequences ξ and ξ' in $(2^A)^\omega$, $\xi \equiv_I \xi'$ iff $\forall \mu \in Pref(\xi)\, \exists \mu' \in Pref(\xi')\, \mu \preceq_I \mu'$, and symmetrically, $\forall \mu' \in Pref(\xi')\, \exists \mu \in Pref(\xi)\, \mu' \preceq_I \mu$.

Partial order reductions can be used here, e.g., by applying the algorithm in [11, 4], based on the following two properties:

Proposition 5 *A program is closed under the relation $\equiv_I$. That is, if $\xi \equiv_I \xi'$ and ξ is a propositional execution sequence of a program P, then so is ξ'.*

Proposition 6 *The set of sequences satisfying a* TLC *formula φ is closed under $\equiv_I$.*

This follows from the fact that the semantics of the logic is over partial order executions, and each equivalence class of the relation $\equiv_I$ consists of the linearizations of a single partial order execution.

5 Discussion

Expressive Power of the parallel operator

Let $\text{TLC}^{\|}$ be the fragment of TLC containing formulas that do not use the parallel operator $\exists\|$. Since we admit disconnected structures, it is obvious that $\exists\|$

[1] A similar structure, called an asynchronous automaton, was defined in [26]

is not expressible in $\mathrm{TLC}^{\|}$. The parallel operator is not expressible in $\mathrm{TLC}^{\|}$ even over rooted structures: There are two *rooted* causal structures K and K' such that $K \models \exists\Diamond\exists\|p$ and $K' \not\models \exists\Diamond\exists\|p$ and for all $\mathrm{TLC}^{\|}$-formulas φ, either both K and K' satisfy φ or both satisfy $\neg\varphi$. We conjecture that $\exists\|$ is not expressible in $\mathrm{TLC}^{\|}$ even when the models are restricted to rooted Π-concurrent structures for a fixed set Π of processes with $|\Pi| \geq 3$.

Open problems

There are several open theoretical problems. The complexity of deciding validity of TLC (i.e. validity over all causal structures, not only Π-concurrent) is unknown. Secondly, the parallel operator $\exists\|$ seems to be different in nature than the standard modal operators. An axiomatization of the logic may give further insights to understand $\exists\|$. Thirdly, the relationship between Asynchronous Büchi Automata, recognizable word languages closed under trace equivalence, and the partial order logics such as TLC is yet not fully explored.

References

[1] P. A. Bernstein, V. Hadzilacos, N. Goodman, Concurrency control and Recovery in Database Systems, Addison-Wesley, 1987.

[2] E.M. Clarke, E.A. Emerson, A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal-logic specifications. *ACM TOPLAS* **8** (2), 244–263 1986.

[3] G. J. Holzmann, *Design and Validation of Computer Protocols.* Prentice Hall Software Series, 1992.

[4] G.J. Holzmann, D. Peled, An improvement in formal verification. *Proc. FORTE'94*, 177–194, 1994.

[5] P. Godefroid, P. Wolper, A Partial Approach to Model Checking. *Information and Computation* **110** (2), 305–326 1994.

[6] S. Katz, D. Peled, Interleaving Set Temporal Logic. *Theoretical Computer Science* **75** (3), 21–43, 1992.

[7] M. Z. Kwiatkowska, Event Fairness and Non-Interleaving Concurrency, *Formal Aspects of Computing* **1**, 213–228, 1989.

[8] O. Lichtenstein, A. Pnueli, Checking that finite-state concurrent programs satisfy their linear specification. *Proc. 11th ACM POPL*, 97–107, 1984.

[9] A. Mazurkiewicz, Trace Theory.. in: W. Brauer, W. Reisig, G. Rozenberg (eds.), *Advances in Petri Nets 1986*, LNCS 255, 279–324, 1987.

[10] M. Mukund, P.S. Thiagarajan, An Axiomatization of Well Branching Prime Event Structures. *Theoretical Computer Science* **96**, 35–72, 1992.

[11] D. Peled, Combining Partial Order Reductions with On-the-fly Model-Checking. *Proc. 6th Conference on Computer Aided Verification*, LNCS 818, 377–390, 1994.

[12] D. Peled, A. Pnueli, Proving partial order properties. *Theoretical Computer Science* **126**, 143–182, 1994.

[13] W. Penczek, A Temporal Logic for Event Structures. *Fundamenta Informaticae* **11** (3), 297–326, 1988.

[14] W. Penczek, A Temporal Logic for the Local Specification of Concurrent Systems. *Information Processing IFIP-89*, 857– 862, 1989.

[15] W. Penczek, Temporal Logics on Trace Systems: On automated verification, *Intl. J. Foundations of Computer Science* **4** 1, 31–67, 1993.

[16] A. Pnueli. The temporal logic of programs. *Proc. 18th FOCS*, 46–77, 1977.

[17] S. Pinter and P. Wolper, A temporal logic for reasoning about partially ordered computations. *Proc. 3rd ACM PODC*, 28-37, 1984.

[18] V.R. Pratt. Modeling concurrency with partial orders. *Intl. J. of Parallel Programming* **15** (1), 33–71, 1986.

[19] W. Reisig, Partial order semantics versus interleaving semantics for CSP like languages and its impact on fairness. *Proc. 11th ICALP*, LNCS 172, 403–413, 1984.

[20] R. S. Streett, Propositional Dynamic Logic of Looping and Converse is Elementary Decidable. *Information and Control* **54**, 121–141, 1982.

[21] P.S. Thiagarajan, A Trace Based Extension of Linear Time Temporal Logic. *Proc. 10th IEEE LICS*, 1994.

[22] A. Valmari, A Stubborn attack on state explosion. *Proc. 2nd Conference on Computer-Aided Verification*, LNCS 531, 156–165, 1990.

[23] M.Y. Vardi, P. Wolper, An automata-theoretic approach to automatic program verification. *Proc. 1st LICS*, 332–344, 1986.

[24] P. Wolper, M.Y. Vardi, A.P. Sistla, Reasoning about infinite computation paths. *Proc. 24th FOCS*, 185–194, 1983.

[25] G. Winskel, Event structures. in: W. Brauer, W. Reisig, G. Rozenberg (eds.), *Advances in Petri Nets 1986*, LNCS 255, 279–324, 1987.

[26] W. Zielonka, Notes on finite asynchronous automata. *R.A.I.R.O.-Informatique Théorique et Applications* **21**, 99–135, 1987.

Appendix: Tableau Construction

For this appendix, we consider a fixed TLC-formula φ, and a fixed process-set Π. For the clarity of presentation, we assume that the formula uses the operator $\forall\Diamond$ instead of $\exists\Box$.

We formally define the semantics of TLC*. Consider a linear structure $M = (\overline{s}, A, \overline{w})$. First, let us define the past-counterparts of the functions f and g: given two indices $0 \leq j < i$, define

$$f'(i,j) \;=\; (w_j \cap \Pi) \setminus (\cup_{j<k<i} w_k).$$

The function g' is defined inductively:

$$g'(i, i-1) \;=\; \Pi \setminus w_{i-1},$$

and if $i-1 > j$ then

$$g'(i,j) = \begin{cases} g'(i-1,j) & \text{if } w_{i-1} \cap \Pi \subseteq g'(i-1,j) \\ g'(i-1,j) \setminus w_{i-1} & \text{otherwise} \end{cases}$$

The set of atomic propositions appearing in φ^* is denoted by A_{φ^*}, and the set of processes appearing in φ^* is denoted by Π_{φ^*}. Let φ^* be a formula of TLC*, and $\Pi \supseteq \Pi_{\varphi^*}$ be a process-set. Given a linear structure $M = (\overline{s}, A_{\varphi^*} \cup \Pi, \overline{w})$, and a position $i \geq 0$, the satisfaction relation $(M,i) \models_\Pi \varphi^*$ is defined as follows:

$(M,i) \models_\Pi \psi$ iff $(K_{M,\Pi}, s_i) \models \psi$, where ψ is a TLC-formula;

$(M,i) \models_\Pi F(\psi,\chi,X,Y)$ iff there exists $j > i$ s.t.
(1) $(M,j) \models_\Pi \psi \wedge \chi$, and
(2) for all $i < k < j$, $(M,k) \not\models_\Pi \psi$, and
(3) $f(i,j) = X$, and
(4) $g(i,j) = Y$;

$(M,i) \models_\Pi P(\psi,\chi,X,Y)$ iff there exists $j < i$ s.t.
(1) $(M,j) \models_\Pi \psi \wedge \chi$, and
(2) for all $i > k > j$, $(M,k) \not\models_\Pi \psi$, and
(3) $f'(i,j) = X$, and
(4) $g'(i,j) = Y$.

Closure

The *closure* set cl contains formulas of TLC*. The *eventuality-closure* set cl' is a subset of cl. The *closure* sets cl and cl' are the smallest sets that satisfy the following conditions:

- cl contains all the subformulas of φ;
- for all $\psi \in cl$, $\neg\psi \in cl$. $\neg\neg\psi$ is identified with ψ.
- $\Pi \subseteq cl$ and $\exists\ominus true \in cl$ and $true \in cl$;
- for all $x \in \Pi$ and all $X, Y \subseteq \Pi$, $F(x, true, X, Y) \in cl$ and $P(x, true, X, Y) \in cl$;
- for all $\exists\bigcirc\psi \in cl$, $x \in \Pi$, and $X, Y \subseteq \Pi$, $F(x,\psi,X,Y) \in cl$;
- for all $\exists\ominus\psi \in cl$, $x \in \Pi$, and $X, Y \subseteq \Pi$, $P(x,\psi,X,Y) \in cl$;
- for all $\forall\Diamond\psi \in cl$, $x \in \Pi$, and $X, Y \subseteq \Pi$, $F(x, \forall\Diamond\psi, X, Y)$ is in both cl and cl';
- for all $\psi\,\exists\mathcal{U}\chi \in cl$, $x \in \Pi$, and $X, Y \subseteq \Pi$, $F(x, \psi\,\exists\mathcal{U}\chi, X, Y)$ is in both cl and cl';
- for all $\psi\,\exists\mathcal{S}\chi \in cl$, $x \in \Pi$, and $X, Y \subseteq \Pi$, $P(x, \psi\,\exists\mathcal{S}\chi, X, Y) \in cl$;
- for all $\exists\|\psi \in cl$, $x \in \Pi$, and $X, Y \subseteq \Pi$, $x \wedge \psi$ and $F(x\wedge\psi, true, X, Y)$ and $P(x \wedge \psi, true, X, Y)$ are in cl;

Tableau Vertices

For a subset $u \subseteq cl$, let $succ(u) \subseteq u \cap \Pi$ contain processes x such that $F(x, true, X, Y) \in u$ for some process-sets $X \ni x$ and $Y \supseteq (u \cap \Pi) \setminus X$; and let $pred(u) \subseteq u \cap \Pi$ contain processes x such that $P(x, true, X, Y) \in u$ for some process-sets $X \ni x$ and $Y \supseteq (u \cap \Pi) \setminus X$. Also, we write $F(\psi,\chi) \in u$ iff $F(\psi,\chi,X,Y) \in u$ for some process-sets X, Y.

A subset $u \subseteq cl$ is *consistent* iff it satisfies all the following:

- *Initiality:* if $\varphi \notin u$ then $\exists\ominus true \in u$.
- *Boolean connectives:*

  $true \in u$;
  if $\psi \in cl$ then $\neg\psi \in u$ iff $\psi \notin u$;
  if $\psi \wedge \chi \in cl$ then $\psi \wedge \chi \in u$ iff both $\psi \in u$ and $\chi \in u$.
- *Process-set:* $u \cap \Pi$ is nonempty.
- *Future operators:*

  if $\exists\bigcirc\psi \in cl$ then $\exists\bigcirc\psi \in u$ iff $F(x,\psi) \in u$ for some $x \in succ(u)$;
  if $\forall\Diamond\psi \in cl$ then $\forall\Diamond\psi \in u$ iff either $\psi \in u$, or $F(x, \forall\Diamond\psi) \in u$ for all $x \in succ(u)$;
  if $\psi\,\exists\mathcal{U}\chi \in cl$ then $\psi\,\exists\mathcal{U}\chi \in u$ iff either $\chi \in u$, or both $\psi \in u$ and $F(x, \psi\,\exists\mathcal{U}\chi) \in u$ for some $x \in succ(u)$;
  if both $F(\psi,\chi,X,Y)$ and $F(\psi,\chi',X',Y')$ are in u then $X' = X$ and $Y' = Y$.
- *Past operators:*

  if $\exists\ominus\psi \in cl$ then $\exists\ominus\psi \in u$ iff $P(x,\psi) \in u$ for some $x \in pred(u)$;
  if $\psi\,\exists\mathcal{S}\chi \in cl$ then $\psi\,\exists\mathcal{S}\chi \in u$ iff either $\chi \in u$, or both $\psi \in u$ and $P(x, \psi\,\exists\mathcal{U}\chi) \in u$ for some $x \in pred(u)$;
  if both $P(\psi,\chi,X,Y)$ and $P(\psi,\chi',X',Y')$ are in u then $X' = X$ and $Y' = Y$.

- *Parallel operator:*

  if $\exists\|\psi \in cl$ then $\exists\|\psi \in u$ iff for some $x \notin u$ and $Y \supseteq u \cap \Pi$, either $F(x \wedge \psi, true, X, Y)$ or $P(x \wedge \psi, true, X, Y)$ is in u.

The *tableau* T is a directed graph $(V, \rightarrow)$. The set V of vertices consists of tableau-states: a tableau-state is a pair (u, u'), where the safety-component $u \subseteq cl$ is consistent and the eventuality-component u' is a subset of $u \cap cl'$.

Tableau Edges

For two tableau-states (u, u') and (v, v'), $(u, u') \rightarrow (v, v')$ iff all the following conditions are satisfied

- *Propagation of next-constraints:*

  if $F(\psi, \chi, X, Y) \in cl$, then $F(\psi, \chi, X, Y) \in u$ iff either (1) $\psi \in v$ and $\chi \in v$ and $X = v \cap \Pi$ and $Y = \Pi \setminus v$, or (2) $\psi \notin v$ and $F(\psi, \chi, X', Y') \in v$ and $X = X' \setminus v$ and if $v \cap \Pi \subseteq Y'$ then $Y = Y'$ else $Y = Y' \setminus v$;

  if $P(\psi, \chi, X, Y) \in cl$, then $P(\psi, \chi, X, Y) \in v$ iff either (1) $\psi \in u$ and $\chi \in u$ and $X = u \cap \Pi$ and $Y = \Pi \setminus u$, or (2) $\psi \notin u$ and $P(\psi, \chi, X', Y') \in u$ and $X = X' \setminus u$ and if $u \cap \Pi \subseteq Y'$ then $Y = Y'$ else $Y = Y' \setminus u$.

- *Propagation of eventuality-constraints:*

  if $F(x, \psi) \in u'$ and $x \notin v$ then $F(x, \psi) \in v'$;

  if $F(x, \forall\Diamond\psi) \in u'$ and $x \in v$ and $\psi \notin v$ then for all $y \in succ(v)$, $F(y, \forall\Diamond\psi) \in v'$;

  if $F(x, \psi\, \exists\mathcal{U}\chi) \in u'$, $x \in v$, and $\chi \notin v$ then for some $y \in succ(v)$, $F(y, \psi\, \exists\mathcal{U}\chi) \in v'$.

- *Loading of eventuality-constraints:* if u' is empty then $v' = v \cap cl'$.

From tableau to ω-automaton

A tableau vertex (u, u') is *initial* iff it has no past-obligations: for all $P(\psi, \chi, X, Y) \in cl$, $P(\psi, \chi, X, Y) \notin u$. A tableau-path $\overline{u} = (u_0, u'_0)(u_1, u'_1)\ldots$ is an infinite path through the tableau T: for all $i \geq 0$, $(u_i, u'_i) \in V$ and $(u_i, u'_i) \rightarrow (u_{i+1}, u'_{i+1})$. The path $\overline{u}$ is *initialized* iff (u_0, u'_0) is an initial vertex. The path $\overline{u}$ is *accepting* iff the following constraints are satisfied:

- for infinitely many indices i, u'_i is empty.

- if $F(\psi, \chi) \in u_i$ for infinitely many i then $\psi \in u_j$ for infinitely many j.

Now we can associate an ω-language $L(T)$ with the tableau T: an ω-word $\overline{w}$ over the alphabet $2^{A_\varphi \cup \Pi}$ is in $L(T)$ iff there exists an initialized accepting path $\overline{u}$ such that $w_i = u_i \cap (A_\varphi \cup \Pi)$ for all $i \geq 0$.

Correctness

Consider a linear structure $M = (\overline{s}, A_\varphi \cup \Pi, \overline{w})$ such that $K_{M,\Pi} \models \varphi$. Let, for all $i \geq 0$, u_i be the subset of cl consisting of formulas ψ such that $(M, i) \models_\Pi \psi$. For all $i \geq 0$, u_i is consistent. Let u'_0 be the empty set. For all $i \geq 0$, given u'_i and u_{i+1}, define u'_{i+1} as follows:

- If u'_i is empty, then $u'_{i+1} = u_{i+1} \cap cl'$.

- When u'_i is nonempty, each $F(x, \psi\, \exists\mathcal{U}\chi) \in u'_i$ contributes at most one eventuality-obligation to u'_{i+1}:

  - if $x \notin u_{i+1}$ then u_{i+1} contains $F(x, \psi\, \exists\mathcal{U}\chi, X, Y)$ for some (uniquely defined) X, Y, and u'_{i+1} also contains this obligation $F(x, \psi\, \exists\mathcal{U}\chi, X, Y)$;

  - if $x \in u_{i+1}$ and $\chi \in u_{i+1}$ then $F(x, \psi\, \exists\mathcal{U}\chi)$ has no contribution to u'_{i+1};

  - Suppose $x \in u_{i+1}$ and $\chi \notin u_{i+1}$. Due to consistency, there exists at least one $y \in succ(u_{i+1})$ such that $F(y, \psi\, \exists\mathcal{U}\chi, X, Y) \in u_{i+1}$ (if there are many such choices for y, choose one nondeterministically). In this case, the contribution is $F(y, \psi\, \exists\mathcal{U}\chi, X, Y)$.

- When u'_i is nonempty, the contribution of each $F(x, \forall\Diamond\psi) \in u'_i$ is the following:

  - if $x \notin u_{i+1}$ then u_{i+1} contains $F(x, \forall\Diamond\psi, X, Y)$ for some (uniquely defined) X, Y, and u'_{i+1} also contains this obligation $F(x, \forall\Diamond\psi, X, Y)$;

  - if $x \in u_{i+1}$ and $\psi \in u_{i+1}$ then $F(x, \forall\Diamond\psi)$ has no contribution to u'_{i+1};

  - Suppose $x \in u_{i+1}$ and $\psi \notin u_{i+1}$. Then u'_{i+1} contains every $F(y, \forall\Diamond\psi, X, Y)$ in u_{i+1} such that $y \in succ(u_{i+1})$.

For all $i \geq 0$, $(u_i, u'_i) \rightarrow (u_{i+1}, u'_{i+1})$ in the tableau T.

Lemma 1 *The sequence $\overline{u} = (u_0, u'_0)(u_1, u'_1)\ldots$ is an initialized accepting path of the tableau T.*

In the other direction, let $\overline{u} = (u_0, u'_0)(u_1, u'_1)\ldots$ be an initialized accepting path in the tableau. We unfold this path to obtain a model for φ. For all $i \geq 0$, let s_i be the prefix $u_0 u_1 \ldots u_i$. Thus, $s_0 s_1 s_2 \ldots$ is an enumeration of a countable state-space. For all $i \geq 0$, let $w_i = u_i \cap (A_\varphi \cup \Pi)$. Consider the linear structure $M_{\overline{u}} = (\overline{s}, A_\varphi \cup \Pi, \overline{w})$.

Lemma 2 *For all $i \geq 0$ and all formulas $\psi \in cl$, $(M_{\overline{u}}, i) \models_\Pi \psi$ iff $\psi \in u_i$.*

It follows that $K_{M_{\overline{u}},\Pi} \models \varphi$, and the language $L(T)$ of the tableau equals $L(\varphi, \Pi)$.

Complexity

Suppose $|\varphi| = n$, $|\Pi| = k$, and ℓ be the number of occurrences of $\exists\|$ in φ.

Lemma 3 *The number of consistent subsets of cl is* $2^{\mathcal{O}(nk+k^2+\ell k^2)}$.

Lemma 4 *The emptiness of the language $L(T)$ can be checked in time* $2^{\mathcal{O}(nk+k^2+\ell k^2)}$.