International Conference on Formal and Applied Practical Reasoning, FAPR-96
June 3–7, 1996, Bonn, Germany
Springer LNCS 1085 (Dov M. Gabbay and Hans Jürgen Ohlbach, Eds.), pp. 399-413

A Tableau Calculus for First-Order Branching Time Logic

Wolfgang May[1] and Peter H. Schmitt[2]

[1] Institut für Informatik, Universität Freiburg, Germany,
may@informatik.uni-freiburg.de,
[2] Institut für Logik, Komplexität und Deduktionssysteme,
Universität Karlsruhe, Germany, pschmitt@ira.uka.de

Abstract. Tableau-based proof systems have been designed for many logics extending classical first-order logic. This paper proposes a sound tableau calculus for temporal logics of the first-order CTL-family. Until now, a tableau calculus has only been presented for the propositional version of CTL. The calculus considered operates with prefixed formulas and may be regarded as an instance of a labelled deductive system. The prefixes allow an explicit partial description of states and paths of a potential Kripke counter model in the tableau. It is possible in particular to represent path segments of finite but arbitrary length which are needed to process reachability formulas. Furthermore, we show that by using prefixed formulas and explicit representation of paths it becomes possible to express and process fairness properties without having to resort to full CTL*. The approach is suitable for use in interactive proof-systems.

1 Introduction

Interactive proof-systems for verification of processes are gaining increasing interest. A very popular approach is to use temporal logic. Following from the observation that most of the specification can be expressed in first-order CTL, an extension of existing proof-systems to temporal logic of the CTL-family seems adequate. This paper presents an intuitive, straightforward extension of the first-order tableau calculus to first-order CTL with additional fairness requirements well-suited for use in an interactive proof-system. The main ideas are

- explicit representation of the "geographical" structure of a fictive model by way of naming of states and paths,
- encoding of this information in a special type of formulas,
- abstraction of path segments of unknown, but finite length in order to process and represent eventualities.

The paper is structured as follows: In section 2 the temporal logic CTL and the notion of a Kripke-structure are reviewed. In section 3 the tableau semantics is presented. Section 4 gives the tableau rules and casts a short glance on correctness and completeness: A complete calculus for first-order CTL cannot be achieved. In section 5 fairness requirements are analyzed and included into the

[1] Most of this work has been done while the first author was student at Universität Karlsruhe. At present his work at Universität Freiburg is supported by grant no. GRK 184/1-96 of the Deutsche Forschungsgemeinschaft.

calculus and some further extensions are pointed out. Section 6 completes the work with some concluding remarks.

2 The Temporal Logic CTL

The base of first-order CTL is a language of first-order predicate logic, including the symbols "(" and ")", the boolean connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$, the quantifiers $\forall$, $\exists$, and an infinite set of variables $\mathsf{Var} := \{x_1, x_2, \ldots\}$. A particular language is given by its *signature* Σ consisting of function symbols and predicate symbols with fixed arities $\mathrm{ord}(f)$ resp. $\mathrm{ord}(p)$. Terms, first-order formulas, and the notions of bound and free variables are defined in the usual way, $\mathsf{free}(\mathcal{F})$ denoting the set of variables occurring free in a set $\mathcal{F}$ of formulas.

A *substitution* (over a signature Σ) is a mapping $\sigma : \mathsf{Var} \rightarrow \mathsf{Term}_\Sigma$ where $\sigma(x) \neq x$ for only finitely many $x \in \mathsf{Var}$. $\sigma : \sigma(x) = t$ is written as $[x \leftarrow t]$. Substitutions are extended to terms and formulas as usual.

A *first-order interpretation* $\mathbf{I} = (I, \mathbf{U})$ over a signature Σ consists of a non-empty set $\mathbf{U}$ (*universe*) and a mapping I which maps every function symbol $f \in \Sigma$ to a function $I(f) : \mathbf{U}^{\mathrm{ord}(f)} \rightarrow \mathbf{U}$ and every predicate symbol $p \in \Sigma$ to a relation $I(p) \subseteq \mathbf{U}^{\mathrm{ord}(p)}$.

A *variable assignment* is a mapping $\chi : \mathsf{Var} \rightarrow \mathbf{U}$. For a variable assignment χ, a variable x, and $d \in \mathbf{U}$, the *modified* variable assignment χ^d_x is identical with χ except that it assigns the element $d \in \mathbf{U}$ to the variable x. Let Ξ denote the set of variable assignments.

The notion of an interpretation is extended to an *evaluation* $\mathbf{I} : \mathsf{Term}_\Sigma \times \Xi \rightarrow \mathbf{U}$:

$$\begin{aligned} &\mathbf{I}(x, \chi) := \chi(x) \quad \text{for } x \in \mathsf{Var}, \\ &\mathbf{I}(f(t_1, \ldots, t_n), \chi) := (I(f))(\mathbf{I}(t_1, \chi), \ldots, \mathbf{I}(t_n, \chi)) \\ &\qquad \text{for } f \in \Sigma,\ \mathrm{ord}(f) = n \text{ and } t_1, \ldots, t_n \in \mathsf{Term}_\Sigma. \end{aligned}$$

To indicate the truth of a formula F in an interpretation $\mathbf{I}$ under a variable assignment χ, the standard notation $\models_{\mathrm{FO}}$ (or simply $\models$) is used: Let s, t be terms, p a predicate symbol, $\mathrm{ord}(p) = n$, $t_1, \ldots, t_n$ terms, x a variable, A and B formulas. Then

$$\begin{aligned} &(\mathbf{I}, \chi) \models \mathsf{true} \quad , \\ &(\mathbf{I}, \chi) \models p(t_1, \ldots, t_n) &&:\Leftrightarrow\ (\mathbf{I}(t_1, \chi), \ldots, \mathbf{I}(t_n, \chi)) \in I(p) \quad , \\ &(\mathbf{I}, \chi) \models \neg A &&:\Leftrightarrow\ \text{not } (\mathbf{I}, \chi) \models A \quad , \\ &(\mathbf{I}, \chi) \models A \vee B &&:\Leftrightarrow\ (\mathbf{I}, \chi) \models A \text{ or } (\mathbf{I}, \chi) \models B \quad , \\ &(\mathbf{I}, \chi) \models \exists x : A &&:\Leftrightarrow\ \text{there is a } d \in \mathbf{U} \text{ with } (\mathbf{I}, \chi^d_x) \models A \quad . \end{aligned}$$

The symbols $A \wedge B := \neg(\neg A \vee \neg B)$, $A \rightarrow B := \neg A \vee B$ and $\forall x : F := \neg\exists x : \neg F$ are defined as usual.

Definition 1. A first-order *Kripke-structure* over a signature Σ is a triple $\mathbf{K} = (\mathbf{G}, \mathbf{R}, \mathbf{M})$ where $\mathbf{G}$ is a set of states, $\mathbf{R} \subseteq \mathbf{G} \times \mathbf{G}$ an *accessibility relation*, and for every $g \in \mathbf{G}$, $\mathbf{M}(g) = (M(g), \mathbf{U}(g))$ is a first-order interpretation of Σ with universe $\mathbf{U}(g)$. $\mathbf{G}$ and $\mathbf{R}$ are called the *frame* of $\mathbf{K}$.

In this paper, only Kripke-structures with constant universe (i.e. $\mathbf{U}(g) = \mathbf{U}(g')$ for all $g, g' \in \mathbf{G}$) are considered. The notion of a *variable assignment* is then defined as in the first-order case.

Definition 2. For a Kripke-structure $\mathbf{K}$ over a signature Σ the state-independent portion $\Sigma^c \subseteq \Sigma$ consists of all function symbols $f : (M(g))(f) = (M(g'))(f)$ for all $g, g' \in \mathbf{G}$ and all predicate symbols $p : (M(g))(p) = (M(g'))(p)$ for all $g, g' \in \mathbf{G}$. This induces a state-independent evaluation $\mathbf{K}(t)$ for $t \in \mathsf{Term}_{\Sigma^c}$.

Definition 3. A *path* p in a Kripke-structure $\mathbf{K} = (\mathbf{G}, \mathbf{R}, \mathbf{M})$ is a sequence $p = (g_0, g_1, g_2, \ldots)$, $g_i \in \mathbf{G}$ with $\mathbf{R}(g_i, g_{i+1})$ holding for all i. It induces a mapping $p : \mathbb{N} \to \mathbf{G}$ with $p(i) = g_i$. Let $p|_i := (g_i, g_{i+1}, \ldots)$.

The family CTL of temporal logics of branching time used in this paper is defined in [BMP81], [CE81], and [EH83] in its propositional version. It uses the unary modal operators $\Box$ ("always"), $\Diamond$ ("sometimes"), $\circ$ ("nexttime"), the binary modal operator until, and two path-quantifiers A and E. For this paper, only a short review of CTL – the basic logic of this family – is given. Two classes of formulas are distinguished: *state formulas* holding in states, and *path formulas* holding on paths:

Definition 4. The syntax of CTL-formulas is given as follows:

(S0) Every first-order formula is a CTL-state formula.
(S1) With F and G CTL-state formulas, $\neg F$, $F \wedge G$, and $F \vee G$ are CTL-state formulas.
(P1) With F and G CTL-state formulas, $\circ F$, $\Box F$, $\Diamond F$, and $(F \ \mathsf{until} \ G)$ are CTL-path formulas.
(P2) With P a CTL-path formula, $\neg P$ is a CTL-path formula.
(S2) With P a CTL-path formula, $\mathsf{A}P$ and $\mathsf{E}P$ are CTL-state formulas.
(SQ) With F a CTL-state formula and x a variable, $\forall x : F$ and $\exists x : F$ are CTL-state formulas.
(F) Every CTL-state formula is a CTL-formula.

The definition shows that in CTL every modality (modal operators and negated modal operators) is immediately preceded by a path-quantifier. CTL* is obtained by weakening this requirement [EH83].

Definition 5. The truth of formulas, $\models_{\mathrm{CTL}}$ (or simply $\models$), in a first-order Kripke-structure $\mathbf{K} = (\mathbf{G}, \mathbf{R}, \mathbf{M})$ is defined separately for state- and path formulas:
Let $g \in \mathbf{G}$ be a state, $p = (g_0, g_1, \ldots)$ a path in $\mathbf{K}$, A an atomic formula, F and G CTL-state formulas, P a CTL-path formula, and χ a variable assignment:

(S0) $(g, \chi) \models A \quad :\Leftrightarrow \quad (\mathbf{M}(g), \chi) \models_{\mathrm{FO}} A.$
(S1a) $(g, \chi) \models \neg F \quad :\Leftrightarrow$ not $(g, \chi) \models F.$
(S1b) $(g, \chi) \models F \vee G \quad :\Leftrightarrow \quad (g, \chi) \models F$ or $(g, \chi) \models G.$
(P1a) $(p, \chi) \models \circ F \quad :\Leftrightarrow \quad (g_1, \chi) \models F.$
(P1b) $(p, \chi) \models F \ \mathsf{until} \ G \quad :\Leftrightarrow$ there is an $i \geq 0$ such that $(g_i, \chi) \models G$ and for all $j : 0 \leq j < i \ \ (g_i, \chi) \models F$ holds.
(P2) $(p, \chi) \models \neg P \quad :\Leftrightarrow$ not $(p, \chi) \models P.$
(S2) $(g, \chi) \models \mathsf{E}P \quad :\Leftrightarrow$ there is a path $p = (g_0, g_1, \ldots)$ in $\mathbf{K}$ and an i such that $g_i = g$ and $(p|_i, \chi) \models P.$
(SQ) $(g, \chi) \models \exists x : F \quad :\Leftrightarrow$ there is a $d \in \mathbf{U}(g)$ with $(g, \chi^d_x) \models F.$

The other symbols are defined as $\Diamond F := \mathsf{true\ until\ F}$, $\Box F := \neg\Diamond\neg F$, and $\mathsf{A}P := \neg\mathsf{E}\neg P$. A state formula is *valid in a Kripke-structure* $\mathbf{K} = (\mathbf{G}, \mathbf{R}, \mathbf{M})$ iff it is valid in all states $g \in \mathbf{G}$. A formula is *valid* iff it is valid in all Kripke-structures.

Fairness:
Different kinds of fairness requirements are distinguished [LPS81]. In this paper only the strongest (and most important) type is considered:
Compassion (strong Fairness):
Every action which is enabled infinitely often in the future will be carried out eventually. [La80], [EC80] and [EH83] state that strong fairness cannot be expressed in CTL. The CTL* expression is given as follows:

$$\text{CTL}^*\text{: } \mathsf{A}((\Box\Diamond(\text{action enabled})) \rightarrow \Diamond(\text{action is carried out})) \quad .$$

2.1 Related Work

In [CES86] and [EL85], a model checking procedure for propositional CTL is presented. The inclusion of fairness requirements is done by extensions to the algorithm.

In [BMP81], [EH82], and [Wol85], a tableau semantics and -calculus for propositional CTL is presented. The paths of the tableau represent paths in a fictive model. Cycles in the tableau are allowed. After termination, which is guaranteed, eventuality formulas have to be postprocessed. In case of a non-closable tableau where no inconsistency is found by postprocessing, the whole tableau represents a model of the initial formula. An extension to CTL* or at least to fairness requirements does not exist.

Both methods cannot be extended to first-order variants because the finite number of possible different states is the central point in their concept.

Facing these problems, it seems necessary to make basic changes in the processing of eventualities: it has to be possible to abstract from finitely many states in-between. In turn, it also seems desirable to have a 1:1-correspondence of branches of the tableau to Kripke-structures.

3 A Tableau Semantics for Branching Time

To achieve a strict distinction between the two graph structures "Kripke-structure" and "tableau", the terms "path" and "state" will be used for Kripke-structures whereas the terms "branch" and "node" will be used for tableaux.

Like in traditional tableau proving, for a proof of the validity of a formula F, the inconsistency of the formula $\neg F$ is proven. It is systematically tried to construct a model for $\neg F$, with the intention to show the impossibility of that attempt. So the situation from first-order theorem proving to find a model for a given set of formulas occurs multiply: Every state is such a first-order interpretation. For this purpose the well-known first-order tableau calculus will be embedded in the temporal tableau calculus which is constructed. Moreover, from

these first-order interpretations a branching time temporal Kripke-structure has to be built.

Therefore it is necessary to describe many individual states as well as the relations between them in the tableau. The latter include the ordering of states on a path together with the connections between different paths.

Thus three kinds of entities have to be described: Elements of the universe inside states, states, and paths. In the chosen semantics these will be explicitly named when their existence is stated by a formula:

- Elements of the universe: as in the first-order tableau calculus a new constant resp. function symbol is introduced by a δ-rule when an $\exists$-quantor is processed.
- States: states are named when required by an existence formula (type $\Diamond F$ or $\circ F$). In the chosen semantics a newly named state has to be positioned on an existing path, retaining the linear ordering of all states on this path.
- Paths: paths are named when required by an existence formula of the kind $\mathsf{E}P$. A newly introduced path is assumed to branch off in the state where its existence is claimed.

In general, between two known states there can be many other still unknown states. These can be named when needed. Thus, a straightforward dissolving of eventualities at any time is possible.

To allow the naming of states at any position of the model, the descriptions of paths contain, apart from the (partial) ordering of known states, additional information about formulas which have to be true in still unknown states on the segments in-between. These are used when new states are explicitly named.

As a conceptional extension of first-order tableaux, every branch of the tableau (resp. the set of formulas on it) corresponds to a complete Kripke-structure.

3.1 Representation

Starting with a formula F over a signature Σ, it is systematically attempted to create a Kripke-structure satisfying F. As every branch of the tableau represents a complete Kripke-structure, apart from the first-order portion, information about the frame of the Kripke-structure has to be coded in tableau nodes. For distinguishing and naming of states, a tableau calculus based on the free variable tableau calculus from [Ree87],[Fit90] augmented with *prefixes* is used for the first-order portion: A state formula F, assumed to be true in a certain state, occurs in the tableau as *prefixed formula* $\gamma : F$. The paths described in the tableau are named by *path descriptors*. For these, *path information formulas* contain the information about the prefixes situated on this path.

Thus the signature $\Sigma_{\mathcal{T}}$ used in the tableau is partitioned into Σ_L (first-order part) and Σ_F (frame part).

In a first step, Σ is augmented with a countable infinite set of n-ary (skolem) function symbols for every $n \in \mathbb{N}$ and a countable infinite set of variables X_i.

Σ_F consists of a set $\hat{\Gamma}$ of *prefix symbols* and a set $\hat{\Lambda}$ of *path symbols*, each containing a countable infinite set of n-ary prefix- resp. path symbols for every

$n \in \mathbb{N}$. The construction of prefixes and path descriptors from these corresponds to the use of skolem functions in the first-order tableau calculus. Here the prefix- and path symbols take the role of the skolem functions. With this, the free variables resulting from invocations of the γ-rule have to be considered. Thus prefixes γ and path descriptors λ are terms consisting of a prefix symbol $\hat{\gamma}$ resp. a path symbol $\hat{\lambda}$ of an arity n and an n-tuple of terms as arguments. Additionally, there is a 0-ary symbol $\hat{\infty}$ that is not a prefix symbol, but is used in a similar way.

Definition 6. Let $\hat{\Gamma}$ be the set of prefix symbols, $\hat{\Lambda}$ the set of path symbols, Σ^c the state-independent portion of Σ. Then the following sets Σ_L, Γ, and Λ are simultaneously recursively enumerable:

$\Sigma_L := \Sigma \cup \{f : \ f \text{ an } n\text{-ary skolem function symbol}\} \cup \{f_\gamma : f \in \Sigma \backslash \Sigma^c, \gamma \in \Gamma\}$,
with $\mathrm{ord}(f_\gamma) = \mathrm{ord}(f)$ and Skolem functions and all f_γ interpreted state-independently, thus

$\Sigma_L^c = \Sigma^c \cup \{f : \ f \text{ an } n\text{-ary skolem function symbol}\} \cup \{f_\gamma : f \in \Sigma \backslash \Sigma^c, \gamma \in \Gamma\}$.

$$\Gamma := \{\hat{\gamma}(t_1, \ldots, t_n) : \hat{\gamma} \in \hat{\Gamma} \text{ an } n\text{-ary prefix symbol}, t_1, \ldots, t_n \in \mathsf{Term}_{\Sigma_L^c}\}$$

is the set of *prefixes*, and

$$\Lambda := \{\hat{\lambda}(t_1, \ldots, t_n) : \hat{\lambda} \in \hat{\Lambda} \text{ an } n\text{-ary path symbol}, t_1, \ldots, t_n \in \mathsf{Term}_{\Sigma_L^c}\}$$

is the set of *path descriptors*, and $\Sigma_F := \hat{\Gamma} \cup \hat{\Lambda}$.

In both sets Γ, $\Lambda \subset \mathsf{Term}_{\Sigma_\mathcal{T}}$ of terms it is precisely the leading function symbol which is a prefix- resp. path symbol taken from Σ_F and all argument terms are in $\mathsf{Term}_{\Sigma_L^c}$. Those are interpreted state-independently by $\mathbf{K}$.

An interpretation of $\Sigma_\mathcal{T}$ – describing a Kripke-structure – is accordingly partitioned: The interpretation of Σ_L is taken over by a suitable set $\{\mathbf{M}(g) : g \in \mathbf{G}\}$ of first-order interpretations. Complementary to this, an "interpretation" of the prefix- and path symbols in Σ_F is defined. The corresponding evaluations map prefixes and path descriptors to the entities described by them:

Definition 7. A P&P-interpretation ("prefixes and paths interpretation") of the sets $\hat{\Gamma}$ and $\hat{\Lambda}$ to a Kripke-structure $(\mathbf{G}, \mathbf{R}, \mathbf{M})$ with a constant universe $\mathbf{U}$ and a set $\mathbf{P}(\mathbf{K})$ of paths is a triple $\Omega = (\phi, \pi, \psi)$ where

- $\phi : \hat{\Lambda} \to \mathbf{U}^n \to \mathbf{P}(\mathbf{K})$ maps every n-ary $\hat{\lambda} \in \hat{\Lambda}$ to a function $\phi(\hat{\lambda}) : \mathbf{U}^n \to \mathbf{P}(\mathbf{K})$ resp. $\phi(\hat{\lambda}) : \mathbf{U}^n \to \mathbb{N} \to \mathbf{G}$,
- $\pi : \hat{\Lambda} \times (\hat{\Gamma} \cup \{\hat{\infty}\}) \to (\mathbf{U}^n \times \mathbf{U}^m) \to \mathbb{N} \cup \{\infty\}$ is an (in general not total) mapping of pairs of n-ary $\hat{\lambda} \in \hat{\Lambda}$ and m-ary $\hat{\gamma} \in \hat{\Gamma}$ to functions $\pi(\hat{\lambda}, \hat{\gamma}) : \mathbf{U}^n \times \mathbf{U}^m \to \mathbb{N} \cup \{\infty\}$ with $\pi(\lambda, \gamma) = \infty \Leftrightarrow \gamma = \hat{\infty}$, and
- $\psi : \hat{\Gamma} \to \mathbf{U}^n \to \mathbf{G}$ maps every n-ary $\hat{\gamma} \in \hat{\Gamma}$ to a function $\psi(\hat{\gamma}) : \mathbf{U}^n \to \mathbf{G}$.

Ω is organized similarly to a first-order interpretation $\mathbf{I} = (I, \mathbf{U})$ if the corresponding mappings, "universes", and induced evaluations are considered:

$$\Phi = (\phi, \mathbf{P}(\mathbf{K})) \quad , \quad \Pi = (\pi, \mathbb{N} \cup \{\infty\}) \quad , \quad \Psi = (\psi, \mathbf{G})$$

Based on ϕ, π, and ψ, the evaluations

$\Phi : \Lambda \times \Xi \to \mathbf{P}(\mathbf{K})$ of path descriptors,
$\Pi : \Lambda \times (\Gamma \cup \{\hat{\infty}\}) \times \Xi \to \mathbb{N} \cup \{\infty\}$ of pairs of path descriptors and prefixes,
and $\Psi : \Gamma \times \Xi \to \mathbf{G}$ of prefixes

are defined as follows: Let $\lambda = \hat{\lambda}(t_1, \ldots, t_n) \in \Lambda$ and $\gamma = \hat{\gamma}(s_1, \ldots, s_m) \in \Gamma$, thus $t_i, s_i \in \mathsf{Term}_{\Sigma_L^c}$. Then

$$\Phi(\lambda, \chi) := (\phi(\hat{\lambda}))(\mathbf{K}(t_1, \chi), \ldots, \mathbf{K}(t_n, \chi)) \quad ,$$
$$\Pi(\lambda, \gamma, \chi) := (\pi(\hat{\lambda}, \hat{\gamma}))(\mathbf{K}(t_1, \chi), \ldots, \mathbf{K}(t_n, \chi), \mathbf{K}(s_1, \chi), \ldots, \mathbf{K}(s_m, \chi)) \quad ,$$
$$\Psi(\gamma, \chi) := (\psi(\hat{\gamma}))(\mathbf{K}(s_1, \chi), \ldots, \mathbf{K}(s_m, \chi)) \quad .$$

Finally, the interpretation of the derived function symbols f_γ is defined state-independent for all $g \in \mathbf{G}$ as

$$(\mathbf{M}(g))(f_\gamma(t_1, \ldots, t_n), \chi) := (\mathbf{M}(\Psi(\gamma, \chi)))(f(t_1, \ldots, t_n), \chi) \quad .$$

Tableau formulas:
On this foundation the syntax used in the tableaux can be worked out: Let $\mathcal{L}$ be the language of state formulas (CTL or CTL*).
The frame of the Kripke-structure is encoded in *path information formulas* of the form $\lambda : [\gamma_0, L_0, \gamma_1, L_1, \ldots, \gamma_n, L_n, \hat{\infty}]$ with $\lambda \in \Lambda$, $\gamma_i \in \Gamma$ and $L_i \in \mathcal{L} \cup \{\circ\}$. Logical formulas occur in the tableau as *prefixed formulas* of the form $\gamma : F$ with $\gamma \in \Gamma$ and the same branch of the tableau containing a path information formula $\lambda : [\ldots, \gamma, \ldots]$ and $F \in \mathcal{L}$ being a state formula.

Following the explicit naming of paths in the calculus, the formulas used internally to the tableau have a more detailed syntax than ordinary CTL/CTL*-formulas. A syntactic facility to use path descriptors in logical formulas is added: To state the validity of a path formula P on the suffix of a path p (described by a path descriptor λ) beginning in a fixed state g (described by a prefix γ) on that path, the symbol λ can syntactically take the role of a path quantifier. In this role, λ is a *path selector*. This results in the following syntax for node formulas in all tableaux tracing this concept:

Definition 8. (TA) Every atomic formula is a $\mathcal{TK}$-state formula.

(TS1) With F und G $\mathcal{TK}$-state formulas, $\neg F$, $F \wedge G$, $F \vee G$ and $F \to G$ are $\mathcal{TK}$-state formulas.

(TSQ) With F a $\mathcal{TK}$-state formula and x a variable, $\forall x : F$ and $\exists x : F$ are $\mathcal{TK}$-state formulas.

(TP1) With F and G $\mathcal{TK}$-state formulas, $\circ F$, $\Box F$, $\Diamond F$, and $(F \ \mathsf{until} \ G)$ are $\mathcal{TK}$-path formulas.

(TP2) With P a $\mathcal{TK}$-path formula, $\neg P$ is a $\mathcal{TK}$-path formula.

(TS2) With P a $\mathcal{TK}$-path formula, $\mathsf{A}P$ and $\mathsf{E}P$ are $\mathcal{TK}$-state formulas.

(TC1) Every $\mathcal{TK}$-state formula is a $\mathcal{TK}$-pre-node formula.

(TC2) With P a $\mathcal{TK}$-path formula and $\lambda \in \Lambda$, λP is a $\mathcal{TK}$-pre-node formula.

(TK1) Every path information formula is a $\mathcal{TK}$-node formula.

(TK2) With F a $\mathcal{TK}$-pre-node formula and $\gamma \in \Gamma$ a prefix, $\gamma : F$ is a $\mathcal{TK}$-prefixed formula.

(TN) All $\mathcal{TK}$-prefixed formulas are $\mathcal{TK}$-node formulas.

Semantics:

Definition 9. For a P&P-interpretation $\Omega = (\phi, \pi, \psi)$, a path information formula $I = \lambda : [\gamma_0, L_0, \gamma_1, L_1, \ldots, \gamma_n, L_n, \hat{\infty}]$ is *consistent with* Ω for a variable assignment χ, if every $\hat{\gamma}$ occurs in I at most once, and for all i

$$\Pi(\lambda, \gamma_0, \chi) = 0 \quad , \quad \Pi(\lambda, \gamma_i, \chi) < \Pi(\lambda, \gamma_{i+1}, \chi) \quad ,$$
$$\text{and} \quad \Psi(\gamma_i, \chi) = \Phi(\lambda, \chi, \Pi(\lambda, \gamma_i, \chi)) \quad .$$

This means that the path $\Phi(\lambda, \chi) = (g_0, g_1, \ldots)$ of $\mathbf{K}$ begins in state $g_0 = \Psi(\gamma_0, \chi)$ and passes through the other known states $g_{\Pi(\lambda,\gamma_1,\chi)} = \Psi(\gamma_1, \chi)$, ..., $g_{\Pi(\lambda,\gamma_n,\chi)} = \Psi(\gamma_n, \chi)$ in the specified order.

Definition 10. The relation $\models\!\!\!\models$ of a Kripke-structure $\mathbf{K} = (\mathbf{G}, \mathbf{R}, \mathbf{M})$ with a set $\mathbf{P}(\mathbf{K})$ of paths, a P&P-interpretation Ω, a set $\mathcal{F}$ of formulas and a variable assignment χ to $\mathsf{free}(\mathcal{F})$ is defined as follows, based on the truth of formulas in Kripke-structures, $\models_{\mathrm{CTL}}$ resp. $\models_{\mathrm{CTL}^*}$,

1a. for every prefixed formula $\gamma : F$, F not containing a path selector:

$$(\mathbf{K}, \Omega, \chi) \models\!\!\!\models \gamma : F \quad :\Leftrightarrow \quad (\Psi(\gamma, \chi), \chi) \models_{\mathrm{CTL}} F \quad ,$$

i.e. in the state corresponding to the prefix γ under variable assignment χ, the (state) formula F holds.

1b. for every prefixed formula $\gamma : F$, F containing a (leading) path selector:

$$(\mathbf{K}, \Omega, \chi) \models\!\!\!\models \gamma : \lambda P \quad :\Leftrightarrow \quad (\Phi(\lambda, \chi)|_{\Pi(\lambda,\gamma,\chi)}, \chi) \models P \quad ,$$

i.e. on the suffix of the path $\Phi(\lambda, \chi)$ beginning in the $\Pi(\lambda, \gamma, \chi)$th state (which is $\Psi(\gamma, \chi)$ by concistency), the path formula P holds.

2. for all path information formulas $I = \lambda : [\gamma_0, L_0, \gamma_1, L_1, \ldots, \gamma_n, L_n, \hat{\infty}]$:

$$(\mathbf{K}, \Omega, \chi) \models\!\!\!\models \lambda : [\gamma_0, L_0, \gamma_1, L_1, \ldots, \gamma_n, L_n, \hat{\infty}]$$

iff I is consistent with Ω for the variable assignment χ, and for all $0 \leq i \leq n$: $L_i = \circ \;\Rightarrow\; \Pi(\lambda, \gamma_{i+1}, \chi) = \Pi(\lambda, \gamma_i, \chi) + 1$,
$L_i \neq \circ \;\Rightarrow\;$ for all j with $\Pi(\lambda, \gamma_i, \chi) < j < \Pi(\lambda, \gamma_{i+1}, \chi)$:
$(\Phi(\lambda, \chi, j), \chi) \models L_i$,

i.e. if $L_i = \circ$, then $\Pi(\lambda, \gamma_i, \chi)$ and $\Pi(\lambda, \gamma_{i+1}, \chi)$ are immediately succeeding indices, else for all (finitely, but arbitrary many) states g_j situated between $\Phi(\lambda, \chi, \Pi(\lambda, \gamma_i, \chi))$ and $\Phi(\lambda, \chi, \Pi(\lambda, \gamma_{i+1}, \chi))$ on path $\Phi(\lambda, \chi)$ the relation $(g_j, \chi) \models L_i$ holds.

For a set $\mathcal{F}$ of path information formulas and prefixed formulas, its truth in a Kripke-structure $\mathbf{K} = (\mathbf{G}, \mathbf{R}, \mathbf{M})$ with a set $\mathbf{P}(\mathbf{K})$ of paths under a variable assignment χ to $\mathsf{free}(\mathcal{F})$ is defined as follows:

$$(\mathbf{K}, \chi) \models\!\!\!\models\!\!\!\models \mathcal{F} \;:\Leftrightarrow\; \text{there is a P\&P-interpretation } \Omega = (\phi, \pi, \psi)$$
$$\text{such that } (\mathbf{K}, \Omega, \chi) \models\!\!\!\models \mathcal{F} \text{ holds.}$$

Since a branch of a tableau is a set of formulas like this, $\models\!\!\!\models\!\!\!\models$ is a relation on Kripke-structures and branches.

The construction of Kripke-structures and consistent P&P-interpretations to a given set of formulas plays an important role in the proof of correctness.

4 The Tableau Calculus $\mathcal{TK}$

For proving the validity of a formula F, the inconsistency of $\neg F$ is proven: it is shown that there is no Kripke-structure $\mathbf{K} = (\mathbf{G}, \mathbf{R}, \mathbf{M})$ with any state $g_0 \in \mathbf{G}$ where F does not hold.
Thus the initialization of the tableau is $\boxed{\overline{\hat{0} : \neg F}}$.

The tableau calculus is based on the well-known first-order tableau calculus, consisting of α-, β-, γ- and δ-rules and the atomic closure rule [Ree87, Fit90].

Let F and G be $\mathcal{TK}$-state formulas, A an atomic formula. In the sequel, $F[t/x]$ denotes the formula F with all occurences of x replaced by t.

α: $\dfrac{\gamma : F \wedge G}{\begin{array}{c}\gamma : F \\ \gamma : G\end{array}}$ $\dfrac{\gamma : \neg(F \vee G)}{\begin{array}{c}\gamma : \neg F \\ \gamma : \neg G\end{array}}$ $\qquad$ β: $\dfrac{\gamma : F \vee G}{\gamma : F \mid \gamma : G}$ $\dfrac{\gamma : \neg(F \wedge G)}{\gamma : \neg F \mid \gamma : \neg G}$

γ: $\dfrac{\gamma : \forall x : F}{\gamma : F[X/x]}$ $\dfrac{\gamma : \neg\exists x : F}{\gamma : \neg F[X/x]}$ $\qquad$ with X a new variable.

δ: $\dfrac{\gamma : \exists x : F}{\gamma : F[f(\mathsf{free}(T))/x]}$ $\dfrac{\gamma : \neg\forall x : F}{\gamma : \neg F[f(\mathsf{free}(T))/x]}$ $\qquad$ with f a new function symbol and T the current branch.

Atomic closure rule:
For a substitution σ and a prefix γ, σ_γ is the γ-*localization*, i.e. $\sigma_\gamma(X) = \sigma(X)$ where every function symbol $f \in \Sigma \backslash \Sigma^c$ is replaced by its localized symbol f_γ. So the substitutes in σ_γ contain only function symbols which are interpreted state-independently.

$$\frac{\begin{array}{c}\gamma : A \\ \gamma : A' \\ \sigma(A) = \neg\sigma(A')\end{array}}{\begin{array}{c}\bot \\ \text{apply } \sigma_\gamma \text{ to the whole tableau.}\end{array}}$$

For dissolving modalities, the information about the frame of the Kripke-structure has to be considered. It is encoded in the path information formulas. In one step a prefixed formula is dissolved "along" a path information formula, inducing the following form of tableau rules:

$$\frac{\begin{array}{c}\text{prefixed formula} \\ \text{path information formula}\end{array}}{\begin{array}{c}\text{prefixed formulas} \\ \text{path information formulas}\end{array}}$$

where the premise takes the latest path information formula on the current branch for the path symbol to be considered. The connection between the prefixed formula being dissolved and the path information is established by the prefix and, if exists, the leading path selector of the prefixed formula. For dissolving prefixed formulas, a path quantifier resp. -selector is broken up together with the subsequent modal operator:

- For dissolving a formula of the form $\gamma : \mathsf{E}P$, a path satisfying P is named and the path formula is bound to that path:

$$\frac{\begin{array}{c}\gamma : \mathsf{E}P \\ \lambda : [\hat{0}, \ldots, \gamma, \ldots]\end{array}}{\begin{array}{c}\hat{\kappa}(\mathsf{free}(T)) : [\hat{0}, \ldots, \gamma, \mathsf{true}, \hat{\infty}] \\ \gamma : \hat{\kappa}(\mathsf{free}(T))P\end{array}}$$

- Formulas of the form $\gamma : \mathsf{A}P$ are dissolved once for every path information formula on this branch containing the prefix γ.
- Formulas of the form $\gamma : \lambda P$ are dissolved along the path information formula for λ.

In the latter cases, the claim that the state described by the current prefix satisfies some formula is decomposed in some less complex claims:

- Which formulas hold in the current state?
- Which state should be regarded as the "next relevant state" on the path?
- Which formulas hold in this next relevant state?
- Which formulas hold in all states in-between?

Special Properties of CTL:

For CTL some *propagation theorems* can be stated [May95] which simplify the dissolving of universally path-quantified formulas along branching paths: The validity of a formula $F = \mathsf{A}P$ can be decomposed into the validity of a formula G in the current state and the validity of a formula Q on *all* outgoing paths, concerning only proper successor states. Especially, for parallel paths, only one of them has to be considered.

According to this, for CTL, the rule for $\gamma : \mathsf{E}P$ can be modified: $\triangleright$

$$\frac{\gamma : \mathsf{E}P}{\begin{array}{c}\hat{\kappa}(\mathsf{free}(T)) : [\gamma, \mathsf{true}, \hat{\infty}] \\ \gamma : \hat{\kappa}(\mathsf{free}(T))P\end{array}}$$

Additionally the dissolving of universally path-quantified formulas is divided in two parts. The syntax of tableau formulas is enriched with the syntactic element (A), meaning "on all paths, concerning only proper successor states", which can replace the leading A of a state formula, leading to the following enlargement to Def. 8:

(TC3) With P a $\mathcal{TK}$-path formula, $(\mathsf{A})P$ is a $\mathcal{TK}$-pre-node formula.

The above-mentioned decomposition is formalized as

$$(\mathbf{K}, \Omega, \chi) \models\!\!\!\!\!\!| \; \gamma : \mathsf{A}P \;\Leftrightarrow\; \bigvee((\mathbf{K}, \Omega, \chi) \models\!\!\!\!\!\!| \; \gamma : G_i \text{ and } (\mathbf{K}, \Omega, \chi) \models\!\!\!\!\!\!| \; \gamma : (\mathsf{A})Q_i) \quad ,$$

where $\bigvee$ counts over some possible decompositions (G_i, Q_i).

There is the following survey over the basic types of state formulas extending Def. 10:

(TS2a) $(\mathbf{K}, \Omega, \chi) \models\!\!\!\!\!\!| \; \gamma : \mathsf{A}P$ $:\Leftrightarrow$ For all paths $p = (g_0, g_1, \ldots)$ in $\mathbf{K}$ and all n with $g_n = \Psi(\gamma, \chi)$ $(p|_n, \chi) \models P$ holds.

(TS2b) $(\mathbf{K}, \Omega, \chi) \models\!\!\!\!\!\!| \; \gamma : \mathsf{E}P$ $:\Leftrightarrow$ there is a path $p(\chi) = (g_0, g_1, \ldots)$ in $\mathbf{K}$ and an $n(\chi)$ so that $g_{n(\chi)} = \Psi(\gamma, \chi)$ and $(p(\chi)|_{n(\chi)}, \chi) \models P$ holds.

(TC2) $(\mathbf{K}, \Omega, \chi) \models\!\!\!\!\!\!| \; \gamma : \lambda P$ $:\Leftrightarrow$ $(\Phi(\lambda, \chi)|_{\Pi(\lambda,\gamma,\chi)}, \chi) \models P$.

(TC3) $(\mathbf{K}, \Omega, \chi) \models\!\!\!\!\!\!| \; \gamma : (\mathsf{A})Q_i$ $:\Leftrightarrow$ For all paths $p = (g_0, g_1, \ldots)$ in $\mathbf{K}$ and all n with $g_n = \Psi(\gamma, \chi)$, $(\Psi(\gamma, \chi)) \models G_i$ implies that $(p|_n, \chi) \models P$ holds.

Because of this decomposition, each formula of the form $\gamma : \mathsf{A}P$ is dissolved exactly once, resulting in pairs of formulas $\gamma : G_i$ (for the current state) and $\gamma : (\mathsf{A})Q_i$ (describing a property of all outgoing paths). Thus, for CTL, formulas

of the form $\gamma : (\mathsf{A})P$ are dissolved once for every path information formula on the same branch containing the prefix γ.

The tableau rules for CTL for formulas which are universally path-quantified or explicitly bound to named paths are as follows. In the sequel, T denotes the current branch of the tableau, $\hat{\gamma}$ is a new prefix symbol and $\hat{\kappa}$ is a new path symbol. P is a path formula, F is a state formula.

$$\frac{\alpha : \mathsf{A}\Box F}{\begin{array}{c}\alpha : F \\ \alpha : (\mathsf{A})\Box F\end{array}} \qquad \frac{\begin{array}{c}\alpha : (\mathsf{A})\Box F \\ \lambda : [\ldots, \alpha, L, \beta, \ldots],\ L \neq \circ\end{array}}{\begin{array}{c}\lambda : [\ldots, \alpha, L \wedge F \wedge (\mathsf{A})\Box F, \beta, \ldots] \\ \beta : \mathsf{A}\Box F \quad \text{if } \beta \neq \hat{\circ}\end{array}} \qquad \frac{\begin{array}{c}\alpha : (\mathsf{A})\Box F \\ \lambda : [\ldots, \alpha, \circ, \beta, \ldots]\end{array}}{\beta : \mathsf{A}\Box F}$$

$$\frac{\begin{array}{c}\alpha : \lambda\Box F, \\ \lambda : [\ldots, \alpha, L, \beta, \ldots],\ L \neq \circ\end{array}}{\begin{array}{c}\lambda : [\ldots, \alpha, L \wedge F, \beta, \ldots] \\ \alpha : F \\ \beta : \lambda\Box F \quad \text{if } \beta \neq \hat{\circ}\end{array}} \qquad \frac{\begin{array}{c}\alpha : \lambda\Box F \\ \lambda : [\ldots, \alpha, \circ, \beta, \ldots]\end{array}}{\begin{array}{c}\alpha : F \\ \beta : \lambda\Box F\end{array}}$$

$$\frac{\alpha : \mathsf{A}\Diamond F}{\begin{array}{c|c}\alpha : F & \alpha : \neg F \\ & \alpha : (\mathsf{A})\Diamond F\end{array}} \qquad \frac{\alpha : (\mathsf{A})\Diamond F}{\begin{array}{c}\lambda : [\ldots, \alpha, \circ, \beta, \ldots] \\ \beta : \mathsf{A}\Diamond F\end{array}} \qquad \frac{\alpha : \lambda\Diamond F}{\begin{array}{c}\lambda : [\ldots, \alpha, \circ, \beta, \ldots] \\ \alpha : F \mid \alpha : \neg F \\ \qquad\quad \beta : \lambda\Diamond F\end{array}}$$

$$\frac{\begin{array}{c}\alpha : (\mathsf{A})\Diamond F \\ \lambda : [\ldots, \alpha, L, \beta, \ldots],\ L \neq \circ\end{array}}{\begin{array}{c|c}\lambda : [\ldots, \alpha, L \wedge \neg F \wedge (\mathsf{A})\Diamond F, & \text{if } \beta \neq \hat{\circ}: \\ \quad \hat{\gamma}(\mathsf{free}(T)), L, \beta, \ldots] & \lambda : [\ldots, \alpha, L \wedge \neg F \wedge (\mathsf{A})\Diamond F, \beta, \ldots] \\ \hat{\gamma}(\mathsf{free}(T)) : L & \beta : \mathsf{A}\Diamond F \\ \hat{\gamma}(\mathsf{free}(T)) : F & \end{array}}$$

$$\frac{\begin{array}{c}\alpha : \lambda\Diamond F \\ \lambda : [\ldots, \alpha, L, \beta, \ldots],\ L \neq \circ\end{array}}{\begin{array}{c|c|c}\alpha : F & \lambda : [\ldots, \alpha, L \wedge \neg F, & \text{if } \beta \neq \hat{\circ}: \\ & \quad \hat{\gamma}(\mathsf{free}(T)), L, \beta, \ldots] & \lambda : [\ldots, \alpha, L \wedge \neg F, \beta, \ldots] \\ & \alpha : \neg F & \alpha : \neg F \\ & \hat{\gamma}(\mathsf{free}(T)) : L & \beta : \lambda\Diamond F \\ & \hat{\gamma}(\mathsf{free}(T)) : F & \end{array}}$$

The rules for $\neg\Box F = \Diamond\neg F$ and $\neg\Diamond F = \Box\neg F$ are analogous.

$$\frac{\begin{array}{c}\alpha : \mathsf{A}\circ F \\ \lambda : [\ldots, \alpha, L, \beta, \ldots],\ L \neq \circ\end{array}}{\begin{array}{c|c}\lambda : [\ldots, \alpha, \circ, \hat{\gamma}(\mathsf{free}(T)), L, \beta, \ldots] & \text{if } \beta \neq \hat{\circ}: \\ \hat{\gamma}(\mathsf{free}(T)) : L & \lambda : [\ldots, \alpha, \circ, \beta, \ldots] \\ \hat{\gamma}(\mathsf{free}(T)) : F & \beta : F \end{array}} \qquad \frac{\begin{array}{c}\alpha : \mathsf{A}\circ F \\ \lambda : [\ldots, \alpha, \circ, \beta, \ldots]\end{array}}{\beta : F}$$

$$\frac{\alpha : \mathsf{A}(F \text{ until } G)}{\begin{array}{l|l}\alpha : G & \alpha : F \\ & \alpha : \neg G \\ & \alpha : (\mathsf{A})(F \text{ until } G)\end{array}} \qquad \frac{\alpha : \mathsf{A}\neg(F \text{ until } G)}{\begin{array}{l|l}\alpha : \neg G & \alpha : F \\ \alpha : \neg F & \alpha : \neg G \\ & \alpha : (\mathsf{A})\neg(F \text{ until } G)\end{array}}$$

$$\frac{\begin{array}{c}\alpha : (\mathsf{A})(F \text{ until } G) \\ \lambda : [\ldots, \alpha, L, \beta, \ldots],\ L \neq \circ\end{array}}{\begin{array}{l|l}\lambda : [\ldots, \alpha, L \wedge F \wedge \neg G \wedge (\mathsf{A})(F \text{ until } G), & \text{if } \beta \neq \hat{\infty} : \\ \quad \hat{\gamma}(\mathsf{free}(T)), L, \beta, \ldots] & \lambda : [\ldots, \alpha, L \wedge F \wedge \neg G \\ \quad \hat{\gamma}(\mathsf{free}(T)) : L & \qquad \wedge (\mathsf{A})(F \text{ until } G), \beta, \ldots] \\ \quad \hat{\gamma}(\mathsf{free}(T)) : G & \beta : \mathsf{A}(F \text{ until } G)\end{array}}$$

$$\frac{\begin{array}{c}\alpha : (\mathsf{A})(F \text{ until } G) \\ \lambda : [\ldots, \alpha, \circ, \beta, \ldots]\end{array}}{\beta : \mathsf{A}(F \text{ until } G)} \qquad \frac{\begin{array}{c}\alpha : (\mathsf{A})\neg(F \text{ until } G) \\ \lambda : [\ldots, \alpha, \circ, \beta, \ldots]\end{array}}{\beta : \mathsf{A}\neg(F \text{ until } G)}$$

$$\frac{\begin{array}{c}\alpha : (\mathsf{A})\neg(F \text{ until } G) \\ \lambda : [\ldots, \alpha, L, \beta, \ldots],\ L \neq \circ\end{array}}{\begin{array}{l|l}\lambda : [\ldots, \alpha, L \wedge F \wedge \neg G & \text{if } \beta \neq \hat{\infty} : \\ \qquad \wedge (\mathsf{A})\neg(F \text{ until } G), & \lambda : [\ldots, \alpha, L \wedge F \wedge \neg G \\ \quad \hat{\gamma}(\mathsf{free}(T)), L, \beta, \ldots] & \qquad \wedge (\mathsf{A})\neg(F \text{ until } G), \beta, \ldots] \\ \quad \hat{\gamma}(\mathsf{free}(T)) : L & \beta : \mathsf{A}\neg(F \text{ until } G) \\ \quad \hat{\gamma}(\mathsf{free}(T)) : \neg F & \\ \quad \hat{\gamma}(\mathsf{free}(T)) : \neg G & \text{if } \beta = \hat{\infty} : \\ & \lambda : [\ldots, \alpha, L \wedge F \wedge \neg G \\ & \qquad \wedge (\mathsf{A})\neg(F \text{ until } G), \hat{\infty}]\end{array}}$$

The rules for $\lambda \circ F$, $\lambda F \text{ until } G$, and $\lambda \neg(F \text{ until } G)$ are analogous.
Two rules are provided to instantiate states on path segments:

$$\frac{\begin{array}{c}\lambda : [\ldots, \alpha, L, \beta, \ldots], L \neq \circ \\ \beta \neq \hat{\infty}\end{array}}{\begin{array}{l|l}\lambda : [\ldots, \alpha, L, \hat{\gamma}(\mathsf{free}(T)), \circ, \beta, \ldots] & \lambda : [\ldots, \alpha, \circ, \beta, \ldots] \\ \quad \hat{\gamma}(\mathsf{free}(T)) : L & \end{array}}$$

$$\frac{\lambda : [\ldots, \alpha, L, \hat{\infty}],\ L \neq \mathsf{true}}{\begin{array}{c}\lambda : [\ldots, \alpha, L, \hat{\gamma}(\mathsf{free}(T)), L, \hat{\infty}], \\ \hat{\gamma}(\mathsf{free}(T)) : L\end{array}}$$

Definition 11. A tableau $\mathcal{T}$ is *satisfiable* if there is a Kripke-structure $\mathbf{K} = (\mathbf{G}, \mathbf{R}, \mathbf{M})$ and a P&P-interpretation $\Omega = (\phi, \pi, \psi)$ of $\hat{\Lambda}$ and $\hat{\Gamma}$ such that for every variable assignment χ of $\mathsf{free}(\mathcal{T})$ there is a branch T in $\mathcal{T}$ with $(\mathbf{K}, \Omega, \chi) \models T$. A branch T in $\mathcal{T}$ is *closed* if it contains the formula $\bot$. A tableau $\mathcal{T}$ is *closed*, if every branch T in $\mathcal{T}$ is closed.

Theorem 12 ((Substitution Lemma)). *Let* $\mathbf{K}$ *be a Kripke-structure over a signature* Σ, Σ^c *the state-independent portion of* Σ, $\Omega = (\phi, \pi, \psi)$ *a P&P-interpretation,* χ *a variable assignment,* X *a free variable,* $g \in \mathbf{G}$, $s \in \mathsf{Term}_{\Sigma^c}$, $t \in \mathsf{Term}_{\Sigma}$, $\gamma \in \Gamma$ *a prefix,* $\lambda \in \Lambda$ *a path descriptor,* F *a* $\mathcal{TK}$*-state formula,* I *a path information formula, and* $a := \mathbf{K}(s, \chi) \in \mathbf{U}(\mathbf{K})$. *Then*

$$\mathbf{K}([X \leftarrow s]t, \chi) = \mathbf{K}(t, \chi^a_X) \quad , \Phi([X \leftarrow s]\lambda, \chi) = \Phi(\lambda, \chi^a_X) \quad ,$$
$$\Pi([X \leftarrow s]\lambda, \gamma, \chi) = \Pi(\lambda, \gamma, \chi^a_X) \quad , \Psi([X \leftarrow s]\gamma, \chi) = \Psi(\gamma, \chi^a_X) \quad ,$$
$$(g, \chi) \models [X \leftarrow s]F \;\Leftrightarrow\; (g, \chi^a_X) \models F \;,$$
$$(\mathbf{K}, \Omega, \chi) \not\models [X \leftarrow s](\gamma : F) \;\Leftrightarrow\; (\mathbf{K}, \Omega, \chi^a_X) \not\models (\gamma : F) \quad ,$$
$$(\mathbf{K}, \Omega, \chi) \not\models [X \leftarrow s]I \;\Leftrightarrow\; (\mathbf{K}, \Omega, \chi^a_X) \not\models I \qquad \textit{and}$$
$$[X \leftarrow s]I \textit{ consistent with } \Omega \textit{ for } \chi \;\Leftrightarrow\; I \textit{ consistent with } \Omega \textit{ for } \chi^a_X.$$

The proof is done separately for terms and formulas by structural induction. This shows the necessity of the substitutes s of σ_γ in the atomic closure rule being interpreted state-independently (i.e. $s \in \mathsf{Term}_{\Sigma^c}$): Then s has a well-defined global interpretation $a := \mathbf{K}(s, \chi) \in \mathbf{U}(\mathbf{K})$ needed for the modification of χ.

Theorem 13 ((Correctness of $\mathcal{TK}$)).

(a) If a tableau $\mathcal{T}$ *is satisfiable and* $\mathcal{T}'$ *is created from* $\mathcal{T}$ *by an application of any of the rules mentioned above, then* $\mathcal{T}'$ *is also satisfiable.*

(b) If there is any closed tableau for $\mathcal{F}$*, then* $\mathcal{F}$ *is unsatisfiable.*

The proof of (a) is done by case-splitting separately for each of the rules. By assumption, there is a Kripke-structure $\mathbf{K}$ and a P&P-Interpretation $\Omega = (\phi, \pi, \psi)$ such that for every variable assignment χ there is a branch T_χ in $\mathcal{T}$ with $(\mathbf{K}, \Omega, \chi) \not\models T_\chi$. In all cases apart from the atomic closure rule, $\mathbf{K}$ and Ω are extended such that they witness the satisfiability of $\mathcal{T}'$. In case of the atomic closure rule the Substitution Lemma guarantees the existence of a branch for every variable assignment to $\mathsf{free}(\mathcal{T}')$. (b) follows directly from (a).

It is well known that the set of first-order tautologies of CTL and even of less expressive systems is not recursively enumerable, see, e.g. [GHR94, Theorem 4.6.1, p. 130]. In [May95] the following is shown:

Theorem 14. *a) First-order CTL is not compact.*
b) Any calculus for first-order CTL cannot be complete.

The calculus is complete modulo inductive properties. For such cases induction rules for temporal properties and well-founded data-structures have to be included. In this setting the notion of completeness has to be relativized to that any proof done in a mathematical way can be completely redone formally.

The calculus is even incomplete for propositional CTL because it cannot use its finite-state-property, so the induction problem remains. For PCTL, the methods mentioned in section 2.1 are complete and efficient. As mentioned there, propositional CTL and first-order CTL require completely different, even contrary, concepts. By introducing abstraction, the presented calculus shows a new concept designed for first-order CTL, accepting not to be optimal for propositional CTL.

5 Fairness and Other Extensions

Fairness is not expressible in CTL. It requires the class of path formulas called "reactivity" [MP92] which is expressible in CTL*. In linear time temporal logic, fairness is expressed as $(\Box\Diamond(\text{action enabled})) \rightarrow \Diamond(\text{action is carried out})$.

A formula P of linear temporal logic can be bound to a path as λP. Complex formulas of linear temporal logic can be processed on single paths by some extensions to the calculus:

- Obvious rules for $\lambda : P \wedge Q$ resp. $\lambda : P \vee Q$.
- All tableau rules copy the leading path selector of the premise in front of the consequent if otherwise the consequent would start with a modal operator not preceded by a path quantifier/selector.

The observation that fairness is a property of a path which is decided "near infinity" makes it tractable in the presented calculus (and intractable in the calculus presented in [BMP81]):

Definition 15. A formula P of linear time temporal logic is of *type* ω iff for every Kripke-structure $\mathbf{K} = (\mathbf{G}, \mathbf{R}, \mathbf{M})$, every path $p \in \mathbf{P}(\mathbf{K})$, every variable assignment χ, and all $n \in \mathbb{N}$

$$(\text{for all } i < n : (p\,|_i, \chi) \models P) \;\Leftrightarrow\; p\,|_n \models P \quad .$$

This establishes the tableau rule $\triangleright$

$$\frac{\begin{array}{c}\gamma_i : \mathsf{A}P, \quad P \text{ of type } \omega \\ \lambda : [\gamma_0, L_0, \gamma_1, L_1, \ldots, \gamma_n, L_n, \hat{\infty}]\end{array}}{\begin{array}{c}\gamma_n : \lambda P \\ \text{for all } j > i\text{: } \gamma_j : \mathsf{A}P\end{array}}$$

Since λP is a linear time formula bound to a single path it can be processed by the calculus on this path.

Theorem 16. *For first-order formulas F and G, $\Box(\Diamond\Box\neg F \vee \Diamond G)$ is of type ω. Fairness is expressible by a formula of type ω.*

The following extensions are pointed out in [May95]:
The handling of state-independent interpreted atomic formulas can be improved. In the pure form, such formulas can only be propagated by frame-axioms which have to be included into the specification and the set of input formulas.
Based on the idea of binding complex formulas of linear time temporal logic to paths the calculus can be used to process CTL*-formulas with only little changes.

6 Conclusion

The presented tableau semantics and -calculus shows new perspectives for formal reasoning in first-order CTL, enabling a formal verification of processes with first-order specifications. Due to the embedding of first-order tableaux all recent techniques such as universal formulas, free variables, liberalized δ-rule, and equality-handling can be made full use of. Because of the complexity, pure computational as well as intellectual, which results in a very large search space

including many occurrences of inductions, interactive proving seems appropriate. This also reflects the point of view that these inductions are part of the specification, and thus are to be proven on one side, and can be exploited on the other.

References

[BMP81] M. Ben-Ari, Z. Manna, A. Pnueli: *The Temporal Logic of Branching Time.* Proc. of the 8th ACM Symp. on Principles of Programming Languages, 1981.

[CE81] E. M. Clarke, E. A. Emerson: *Design and Synthesis of Synchronization Skeletons using Branching Time Temporal Logic.* Proc. of the IBM Workshop on Logics of Programs, Springer LNCS 131, 1981.

[CES86] E. M. Clarke, E. A. Emerson, A. P. Sistla: *Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications.* ACM Tr. on Programming Languages and Systems, Vol. 8, No. 2, 1986.

[EC80] E. A. Emerson, E. M. Clarke: *Characterizing Properties of Parallel Programs as Fixpoints.* Proc. of the 7th Int. Coll. on Automata, Languages and Programming. Springer LNCS 85, 1980.

[EH82] E. A. Emerson, J. Y. Halpern: *Decision Procedures and Expressiveness in the Temporal Logic of Branching Time.* Proc. of the 14th ACM Symp. on Computing, 1982.

[EH83] E. A. Emerson, J. Y. Halpern: *"Sometimes" and "not never" revisited: On Branching Time versus Linear Time in Temporal Logic.* Proc. of the 10th ACM Symp. on Principles of Programming Languages, 1983.

[EL85] E. A. Emerson, C.-L. Lei: *Modalities for Model Checking: Branching Time Strikes Back.* Proc. of the 12th ACM Symp. on Principles of Programming Languages, 1985.

[Fit90] M. Fitting: *First Order Logic and Automated Theorem Proving.* Springer, New York, 1990.

[GHR94] D. M. Gabbay, I. Hodkinson, M. Reynolds: *Temporal Logic. Mathematical Foundations and Computational Aspects. Vol. 1.* Clarendon Press, Oxford Logic Guides No. 28, 1994.

[La80] L. Lamport: *"Sometimes" is Sometimes "Not Never".* Proc. of the 7th ACM Symp. on Principles of Programming Languages, 1980.

[LPS81] D. Lehmann, A. Pnueli, J. Stavi: *Impartiality, Justice, and Fairness: The Ethics of Concurrent Termination.* Proc. of the 8th Int. Coll. on Automata, Languages and Programming. Springer LNCS 115, 1981.

[May95] W. May: *Protokollverifikation in Temporallogik: Evolving Algebras und ein Tableaukalkül.* Diplomarbeit, Universität Karlsruhe, 1995.

[MP92] Z. Manna, A. Pnueli: *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer, 1992.

[Ree87] S. V. Reeves: *Semantic Tableaux as a Framework for Automated Theorem-Proving.* Dept. of Comp. Sc. and Statistics, Queen Mary College, Univ. of London.

[Wol85] P. Wolper: *The Tableau Method for Temporal Logic.* Logique et Analyse, 110-111, vol. 28, 1985.