

# One-Pass Tree-Shaped Tableau Systems for Timed Temporal Logics

---

Luca Geatti

University of Udine, Italy

joint work with **Nicola Gigante**

University of Udine, Italy

and **Angelo Montanari**

University of Udine, Italy

and **Mark Reynolds**

University of Western Australia, Australia

GandALF, 26 - 28 September 2018, Saarbrücken, Germany

# Introduction

---

A **real-time system** is commonly described as a system that “controls an environment by receiving data, processing them, and returning the results sufficiently quickly to affect the environment at that time”.

- their correctness does not depend only on their logical correctness, but also on their **response time**;
- most of the *mission or safety critical* systems are real-time: their formal correctness is an aspect that cannot be overlooked.

A **real-time system** is commonly described as a system that *“controls an environment by receiving data, processing them, and returning the results sufficiently quickly to affect the environment at that time”*.

- their correctness does not depend only on their logical correctness, but also on their **response time**;
- most of the *mission or safety critical* systems are real-time: their formal correctness is an aspect that cannot be overlooked.

A **real-time system** is commonly described as a system that “controls an environment by receiving data, processing them, and returning the results sufficiently quickly to affect the environment at that time”.

- their correctness does not depend only on their logical correctness, but also on their **response time**;
- most of the *mission* or *safety critical* systems are real-time: their formal correctness is an aspect that cannot be overlooked.

In classical LTL, we can express the request-response property:

$$\varphi := G(\text{request} \rightarrow F \text{response})$$

We do **not** know the exact times at which the request and the response actually take place: the only thing we know is the *temporal ordering* between these two events.

LTL  $\Rightarrow$  qualitative time requirements only,  
not suitable for real-time properties.

# Linear Temporal Logic - LTL

In classical LTL, we can express the request-response property:

$$\varphi := G(\text{request} \rightarrow F \text{response})$$

We do **not** know the exact times at which the request and the response actually take place: the only thing we know is the *temporal ordering* between these two events.

LTL  $\Rightarrow$  **qualitative** time requirements only,  
not suitable for real-time properties.

Timed Propositional Temporal Logic (**TPTL** [AH94]) allows for **quantitative** time requirements.

- Syntax:

$$\begin{aligned}(\text{terms}) \pi &:= x + c \mid c \\(\text{formulae}) \phi &:= p \mid \pi_1 \leq \pi_2 \mid \pi_1 \equiv_d \pi_2 \mid \\ &\quad \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid \\ &\quad X\phi_1 \mid \phi_1 \mathcal{U} \phi_2 \mid \phi_1 \mathcal{R} \phi_2 \mid \\ &\quad x.\phi_1\end{aligned}$$

where  $x$  is a variable,  $p$  is a proposition letter and  $c, d \in \mathbb{N}$ .

- 'x.' is a **freeze quantifier**: 'x.' freezes the variable  $x$  to the time of the local temporal context.



### Definition (Timed state sequence)

*Let  $\sigma = \sigma_0\sigma_1\sigma_2\dots$  be an infinite sequence of states (each state is a subset of proposition letters).*

## Definition (Timed state sequence)

Let  $\sigma = \sigma_0\sigma_1\sigma_2\dots$  be an infinite sequence of states (each state is a subset of proposition letters). A time sequence  $\tau = \tau_0\tau_1\tau_2\dots$  is an infinite sequence of times  $\tau_i \in \mathbb{N}$ , for all  $i \geq 0$ , that satisfies the following conditions:

## Definition (Timed state sequence)

Let  $\sigma = \sigma_0\sigma_1\sigma_2\dots$  be an infinite sequence of states (each state is a subset of proposition letters). A time sequence  $\tau = \tau_0\tau_1\tau_2\dots$  is an infinite sequence of times  $\tau_i \in \mathbb{N}$ , for all  $i \geq 0$ , that satisfies the following conditions:

1. *monotonicity*:  $\tau_i \leq \tau_{i+1}$ , for all  $i \geq 0$ ;
2. *progress*: for all  $t \in \mathbb{N}$ , there exists  $i \geq 0$  such that  $\tau_i > t$ .

## Definition (Timed state sequence)

Let  $\sigma = \sigma_0\sigma_1\sigma_2\dots$  be an infinite sequence of states (each state is a subset of proposition letters). A time sequence  $\tau = \tau_0\tau_1\tau_2\dots$  is an infinite sequence of times  $\tau_i \in \mathbb{N}$ , for all  $i \geq 0$ , that satisfies the following conditions:

1. *monotonicity*:  $\tau_i \leq \tau_{i+1}$ , for all  $i \geq 0$ ;
2. *progress*: for all  $t \in \mathbb{N}$ , there exists  $i \geq 0$  such that  $\tau_i > t$ .

A **timed state sequence**  $\rho = (\sigma, \tau)$  is a pair consisting of a state sequence  $\sigma$  and a time sequence  $\tau$ .

## Definition (Timed state sequence)

Let  $\sigma = \sigma_0\sigma_1\sigma_2\dots$  be an infinite sequence of states (each state is a subset of proposition letters). A time sequence  $\tau = \tau_0\tau_1\tau_2\dots$  is an infinite sequence of times  $\tau_i \in \mathbb{N}$ , for all  $i \geq 0$ , that satisfies the following conditions:

1. *monotonicity*:  $\tau_i \leq \tau_{i+1}$ , for all  $i \geq 0$ ;
2. *progress*: for all  $t \in \mathbb{N}$ , there exists  $i \geq 0$  such that  $\tau_i > t$ .

A **timed state sequence**  $\rho = (\sigma, \tau)$  is a pair consisting of a state sequence  $\sigma$  and a time sequence  $\tau$ .

Let  $\mathcal{E} : \mathcal{V} \rightarrow \mathbb{N}$  be an interpretation for the variables, that we call **environment**.

We inductively define  $\rho^i \models_{\mathcal{E}} \phi$ , as follows:

1.  $\rho^i \models_{\mathcal{E}} p$             iff     $p \in \sigma_i$
2.  $\rho^i \models_{\mathcal{E}} \pi_1 \leq \pi_2$     iff     $\mathcal{E}(\pi_1) \leq \mathcal{E}(\pi_2)$
3.  $\rho^i \models_{\mathcal{E}} \pi_1 \equiv_d \pi_2$     iff     $\mathcal{E}(\pi_1) \equiv_d \mathcal{E}(\pi_2)$
4.  $\rho^i \models_{\mathcal{E}} x.\phi$             iff     $\rho^i \models_{\mathcal{E}[x:=\tau_i]} \phi$

The other operators are interpreted in the same way as in LTL.

**Example** (classical time bounded request-response property):

$$\phi_{BR} \quad := \quad G x.(request \rightarrow F y.(response \wedge y \leq x + 10))$$

We inductively define  $\rho^i \models_{\mathcal{E}} \phi$ , as follows:

1.  $\rho^i \models_{\mathcal{E}} p$             iff     $p \in \sigma_i$
2.  $\rho^i \models_{\mathcal{E}} \pi_1 \leq \pi_2$     iff     $\mathcal{E}(\pi_1) \leq \mathcal{E}(\pi_2)$
3.  $\rho^i \models_{\mathcal{E}} \pi_1 \equiv_d \pi_2$     iff     $\mathcal{E}(\pi_1) \equiv_d \mathcal{E}(\pi_2)$
4.  $\rho^i \models_{\mathcal{E}} x.\phi$             iff     $\rho^i \models_{\mathcal{E}[x:=\tau_i]} \phi$

The other operators are interpreted in the same way as in LTL.

**Example** (classical time bounded request-response property):

$$\phi_{BR} \quad := \quad G x.(request \rightarrow F y.(response \wedge y \leq x + 10))$$

# The TPTL<sub>b</sub>+P logic

TPTL<sub>b</sub>+P is a bounded version of TPTL with past operators:

1.  $\rho^i \models_{\xi} X_w \phi_1$       iff       $\tau_{i+1} \leq \tau_i + w$  and  $\rho^{i+1} \models_{\xi} \phi_1$
2.  $\rho^i \models_{\xi} \phi_1 U_w \phi_2$       iff      there exists  $j \geq i$  such that
  - (i)  $\tau_j \leq \tau_i + w$
  - (ii)  $\rho^j \models_{\xi} \phi_2$
  - (iii)  $\rho^k \models_{\xi} \phi_1$  for all  $i \leq k < j$

The bounds on the operators allow us to know *a priori* the bound between two variables. The bounds are similar to the ones of Metric Temporal Logic.



# The TPTL<sub>b</sub>+P logic

TPTL<sub>b</sub>+P is a bounded version of TPTL with past operators:

1.  $\rho^i \models_{\xi} X_w \phi_1$       iff       $\tau_{i+1} \leq \tau_i + w$  and  $\rho^{i+1} \models_{\xi} \phi_1$
2.  $\rho^i \models_{\xi} \phi_1 U_w \phi_2$       iff      there exists  $j \geq i$  such that
  - (i)  $\tau_j \leq \tau_i + w$
  - (ii)  $\rho^j \models_{\xi} \phi_2$
  - (iii)  $\rho^k \models_{\xi} \phi_1$  for all  $i \leq k < j$

The bounds on the operators allow us to know *a priori* the bound between two variables. The bounds are similar to the ones of Metric Temporal Logic.

## TPTL and $\text{TPTL}_b + P$ - Satisfiability

The **satisfiability problem** for TPTL (resp.  $\text{TPTL}_b + P$ ) is the problem of deciding whether there exists a model satisfying a given TPTL (resp.  $\text{TPTL}_b + P$ ) formula.

Useful in a number of situations:

- sanity check: it allows one to check whether an input formula is satisfiable before running a model checking algorithm;
- monitoring, synthesis ( $\text{UNSAT} \rightarrow \text{UNREALIZABLE}$ ), and, in general, all the steps of a model-based design approach;
- the **timeline-based planning problem** with bounded temporal constraints can be captured by  $\text{TPTL}_b + P$  [Del+17].

## TPTL and $\text{TPTL}_b + P$ - Satisfiability

The **satisfiability problem** for TPTL (resp.  $\text{TPTL}_b + P$ ) is the problem of deciding whether there exists a model satisfying a given TPTL (resp.  $\text{TPTL}_b + P$ ) formula.

Useful in a number of situations:

- sanity check: it allows one to check whether an input formula is satisfiable before running a model checking algorithm;
- monitoring, synthesis ( $\text{UNSAT} \rightarrow \text{UNREALIZABLE}$ ), and, in general, all the steps of a model-based design approach;
- the **timeline-based planning problem** with bounded temporal constraints can be captured by  $\text{TPTL}_b + P$  [Del+17].

## TPTL and $\text{TPTL}_b + P$ - Satisfiability

The **satisfiability problem** for TPTL (resp.  $\text{TPTL}_b + P$ ) is the problem of deciding whether there exists a model satisfying a given TPTL (resp.  $\text{TPTL}_b + P$ ) formula.

Useful in a number of situations:

- sanity check: it allows one to check whether an input formula is satisfiable before running a model checking algorithm;
- monitoring, synthesis ( $\text{UNSAT} \rightarrow \text{UNREALIZABLE}$ ), and, in general, all the steps of a model-based design approach;
- the **timeline-based planning problem** with bounded temporal constraints can be captured by  $\text{TPTL}_b + P$  [Del+17].

## TPTL and $\text{TPTL}_b + P$ - Satisfiability

The **satisfiability problem** for TPTL (resp.  $\text{TPTL}_b + P$ ) is the problem of deciding whether there exists a model satisfying a given TPTL (resp.  $\text{TPTL}_b + P$ ) formula.

Useful in a number of situations:

- sanity check: it allows one to check whether an input formula is satisfiable before running a model checking algorithm;
- monitoring, synthesis ( $\text{UNSAT} \rightarrow \text{UNREALIZABLE}$ ), and, in general, all the steps of a model-based design approach;
- the **timeline-based planning problem** with bounded temporal constraints can be captured by  $\text{TPTL}_b + P$  [Del+17].

Tableau methods are among the most well known techniques used to solve the satisfiability problem for temporal logics:

- **Two-pass** and **graph-shaped** [MP95]:
  - first pass → builds the graph encoding all the candidate models;
  - second pass → prunes the graph by removing the wrong candidates;
  - difficult to implement and impractical because of the huge size of the graph.
- **One-pass** and **tree-shaped** [Ber+16]:
  - in a single pass, we can build a candidate model and decide to either accept or reject it;
  - easy to describe and to implement (since each branch is independent from the others, they are easily parallelisable);

Tableau methods are among the most well known techniques used to solve the satisfiability problem for temporal logics:

- **Two-pass** and **graph-shaped** [MP95]:
  - first pass → builds the graph encoding all the candidate models;
  - second pass → prunes the graph by removing the wrong candidates;
  - difficult to implement and impractical because of the huge size of the graph.
- **One-pass** and **tree-shaped** [Ber+16]:
  - in a single pass, we can build a candidate model and decide to either accept or reject it;
  - easy to describe and to implement (since each branch is independent from the others, they are easily parallelisable);

Main contribution of the paper: the proposal of two original one-pass and tree-shaped tableau systems for the logics

- TPTL
- $\text{TPTL}_b + P$

proving their *soundness* and *completeness* and analyzing their complexity (both algorithms run in *doubly* exponential space).



# The Tableau System

---

## Tableau for TPTL - Structure

- The tableau is a tree where each node is labeled by a set of subformulae and a time point belonging to  $\mathbb{N}$ ;
- The initial tableau for  $z.\phi$  (in Negated Normal Form) is a tree consisting of the following single node (the root):

$$\{z.\phi\}^{TIME=0}$$

## Tableau for TPTL - Structure

- The tableau is a tree where each node is labeled by a set of subformulae and a time point belonging to  $\mathbb{N}$ ;
- The initial tableau for  $z.\phi$  (in Negated Normal Form) is a tree consisting of the following single node (the root):

$$\{z.\phi\}^{TIME=0}$$

## Tableau for TPTL - Structure

- The tableau is built recursively and **top-down** starting from the root, by applying a set of rules to the leaves of the tree (in this order):
  1. **expansion rules**: add one or two children to a leaf of the tree;
  2. **termination rules**: close a branch either by *ticking* a leaf, and thus accepting the branch ( $\checkmark$ ), or by *crossing* a leaf, and thus rejecting the branch ( $\times$ );
  3. **step rule**: force an advancement in time of the model.
- If all the branches of the tableau are closed (that is, either ticked or crossed), we say that the tableau is *complete*.
- Given a complete tableau  $T_\phi$ , the input formula  $\phi$  is satisfiable if and only if there is in  $T_\phi$  at least one accepted branch.

## Tableau for TPTL - Structure

- The tableau is built recursively and **top-down** starting from the root, by applying a set of rules to the leaves of the tree (in this order):
  1. **expansion rules**: add one or two children to a leaf of the tree;
  2. **termination rules**: close a branch either by *ticking* a leaf, and thus accepting the branch (✓), or by *crossing* a leaf, and thus rejecting the branch (✗);
  3. **step rule**: force an advancement in time of the model.
- If all the branches of the tableau are closed (that is, either ticked or crossed), we say that the tableau is *complete*.
- Given a complete tableau  $T_\phi$ , the input formula  $\phi$  is satisfiable if and only if there is in  $T_\phi$  at least one accepted branch.

## Tableau for TPTL - Structure

- The tableau is built recursively and **top-down** starting from the root, by applying a set of rules to the leaves of the tree (in this order):
  1. **expansion rules**: add one or two children to a leaf of the tree;
  2. **termination rules**: close a branch either by *ticking* a leaf, and thus accepting the branch (✓), or by *crossing* a leaf, and thus rejecting the branch (✗);
  3. **step rule**: force an advancement in time of the model.
- If all the branches of the tableau are closed (that is, either ticked or crossed), we say that the tableau is *complete*.
- Given a complete tableau  $T_\phi$ , the input formula  $\phi$  is satisfiable if and only if there is in  $T_\phi$  at least one accepted branch.

## Tableau for TPTL - Structure

- The tableau is built recursively and **top-down** starting from the root, by applying a set of rules to the leaves of the tree (in this order):
  1. **expansion rules**: add one or two children to a leaf of the tree;
  2. **termination rules**: close a branch either by *ticking* a leaf, and thus accepting the branch (✓), or by *crossing* a leaf, and thus rejecting the branch (✗);
  3. **step rule**: force an advancement in time of the model.
- If all the branches of the tableau are closed (that is, either ticked or crossed), we say that the tableau is *complete*.
- Given a complete tableau  $T_\phi$ , the input formula  $\phi$  is satisfiable if and only if there is in  $T_\phi$  at least one accepted branch.

# Tableau for TPTL - Structure

- The tableau is built recursively and **top-down** starting from the root, by applying a set of rules to the leaves of the tree (in this order):
  1. **expansion rules**: add one or two children to a leaf of the tree;
  2. **termination rules**: close a branch either by *ticking* a leaf, and thus accepting the branch (✓), or by *crossing* a leaf, and thus rejecting the branch (✗);
  3. **step rule**: force an advancement in time of the model.
- If all the branches of the tableau are closed (that is, either ticked or crossed), we say that the tableau is *complete*.
- Given a complete tableau  $T_\phi$ , the input formula  $\phi$  is satisfiable if and only if there is in  $T_\phi$  at least one accepted branch.



## Expansion rules

---

Expansion rules are applied to the leaves of the tree, until no expansion rule can be applied anymore.

- $\psi \rightarrow \Delta_1$
- $\psi \rightarrow \Delta_1 \mid \Delta_2$

## Expansion rules

Expansion rules are applied to the leaves of the tree, until no expansion rule can be applied anymore.

- $\psi \rightarrow \Delta_1$        $z.(\alpha \wedge \beta) \rightarrow \{z.\alpha, z.\beta\}$
- $\psi \rightarrow \Delta_1 \mid \Delta_2$      $z.(\alpha \vee \beta) \rightarrow \{z.\alpha\} \mid \{z.\beta\}$

Boolean connectives:

$$\begin{array}{c} \{z.(\alpha \wedge \beta) \dots\} \\ | \\ \{z.\alpha, z.\beta \dots\} \end{array}$$

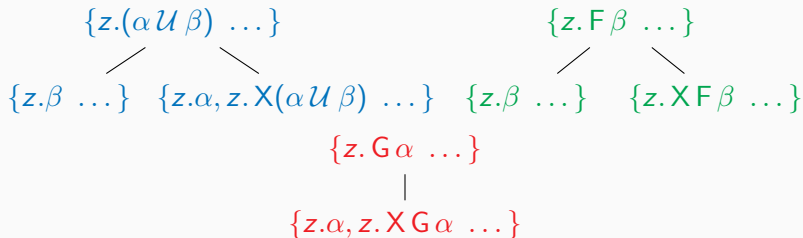
$$\begin{array}{c} \{z.(\alpha \vee \beta) \dots\} \\ / \quad \backslash \\ \{z.\alpha \dots\} \quad \{z.\beta \dots\} \end{array}$$

# Expansion rules

Expansion rules are applied to the leaves of the tree, until no expansion rule can be applied anymore.

- $\psi \rightarrow \Delta_1$                        $z.G\alpha \rightarrow \{z.\alpha, z.XG\alpha\}$
- $\psi \rightarrow \Delta_1 \mid \Delta_2$              $z.(\alpha U \beta) \rightarrow \{z.\beta\} \mid \{z.\alpha, z.X(\alpha U \beta)\}$   
    $z.(F\beta) \rightarrow \{\beta\} \mid \{z.XF\beta\}$

Temporal connectives:



Expansion rules are applied to the leaves of the tree, until no expansion rule can be applied anymore.

- $\psi \rightarrow \Delta_1$        $z.y.\alpha \rightarrow z.\alpha[z/y]$
- $\psi \rightarrow \Delta_1 \mid \Delta_2$

Freeze quantifier:

$$\begin{array}{c} \{z.y.\alpha \dots\} \\ | \\ \{z.\alpha[z/y] \dots\} \end{array}$$

By repeatedly applying expansion rules, we eventually reach a node whose label contains only:

- proposition letters;
- timing constraints;
- formulae of type  $z.X\alpha$ .

Such a node is called a **poised node**.

## Step rule

Once we reach a poised node, we can apply the **STEP** rule and advance in a state of the model.

$$\begin{array}{c} \{z.X\alpha \dots\}^{TIME=i} \\ \swarrow \quad \downarrow \quad \searrow \\ \{z.\alpha^0 \dots\}^{TIME=i} \quad \{z.\alpha^1 \dots\}^{TIME=i+1} \dots \{z.\alpha^{\delta_\phi} \dots\}^{TIME=i+\delta_\phi} \end{array}$$

where  $\delta_\phi$  is a value that we can pre-compute from the initial formula  $\phi$  and that does not affect satisfiability.

$(\cdot)^\delta$  is called a **temporal shift**. For instance:

- $x.XGy.(p \rightarrow y \leq x + 1)^1 = x.XGy.(p \rightarrow y \leq x)$
- $x.XGy.(p \rightarrow y \leq x + 1)^2 = x.XGy.(p \rightarrow \perp)$

## Step rule

Once we reach a poised node, we can apply the **STEP** rule and advance in a state of the model.

$$\begin{array}{c} \{z.X\alpha \dots\}^{TIME=i} \\ \swarrow \quad \downarrow \quad \searrow \\ \{z.\alpha^0 \dots\}^{TIME=i} \quad \{z.\alpha^1 \dots\}^{TIME=i+1} \dots \{z.\alpha^{\delta_\phi} \dots\}^{TIME=i+\delta_\phi} \end{array}$$

where  $\delta_\phi$  is a value that we can pre-compute from the initial formula  $\phi$  and that does not affect satisfiability.

$(\cdot)^\delta$  is called a **temporal shift**. For instance:

- $x.XGy.(p \rightarrow y \leq x + 1)^1 = x.XGy.(p \rightarrow y \leq x)$
- $x.XGy.(p \rightarrow y \leq x + 1)^2 = x.XGy.(p \rightarrow \perp)$



## Termination rules

---

Termination rules decide if

- the current branch has to be accepted (✓) (in this case we have found a model);
- the current branch has to be rejected (✗);
- or the current branch must be further explored (*i.e.*, STEP rule).

# EMPTY rule and CONTRADICTION rule

EMPTY rule:

$$\begin{array}{c} \{\dots, z.p, z.q, \neg z.r, \dots\} \\ | \\ \{\} \\ \checkmark \end{array}$$

CONTRADICTION rule:

$$\begin{array}{c} \{\dots, z.p, \neg z.p, \dots\} \\ \times \end{array}$$

EMPTY rule:

$$\begin{array}{c} \{\dots, z.p, z.q, \neg z.r, \dots\} \\ | \\ \{\} \\ \checkmark \end{array}$$

CONTRADICTION rule:

$$\begin{array}{c} \{\dots, z.p, \neg z.p, \dots\} \\ \times \end{array}$$

SYNC rule:

$$\{\dots, x.(x \leq x + 1), \dots\}$$

✓

We can check the truth of this timing constraint by simply checking if  $0 \leq 1$ .

**Remark:** thanks to the expansion rule  $z.y.\alpha \rightarrow \{z.\alpha[z/y]\}$  and the temporal shift, all the timing constraints that can appear in a label are of the form  $z.(z \sim z + c)$ , for some operator  $\sim$  and some constant  $c \in \mathbb{N}$ .

SYNC rule:

$$\{\dots, x.(x \leq x + 1), \dots\}$$

✓

We can check the truth of this timing constraint by simply checking if  $0 \leq 1$ .

**Remark:** thanks to the expansion rule  $z.y.\alpha \rightarrow \{z.\alpha[z/y]\}$  and the temporal shift, all the timing constraints that can appear in a label are of the form  $z.(z \sim z + c)$ , for some operator  $\sim$  and some constant  $c \in \mathbb{N}$ .

## LOOP rule

Consider the TPTL formula for the bounded request-response property:

$$\phi_{BR} := G x.(\text{request} \rightarrow F y.(\text{response} \wedge y \leq x + 10))$$

Among the models of this formula there are models featuring **infinitely many** requests, and consequently **infinitely many** responses.

### LOOP rule

Let  $v$  be a poised leaf, and let  $u < v$  be a *poised node*, which is a proper ancestor of  $v$ , such that  $\Gamma(u) = \Gamma(v)$  and all the eventualities (i.e.,  $z.X(\alpha \mathcal{U} \beta)$  or  $z.X F \beta$ ) requested in  $u$  are fulfilled between  $u$  and  $v$  (included). Then,

- if  $\text{time}(u) = \text{time}(v)$ , then  $v$  is crossed and the branch rejected;
- if  $\text{time}(u) < \text{time}(v)$ , then  $v$  is ticked and the branch accepted.

## LOOP rule

Consider the TPTL formula for the bounded request-response property:

$$\phi_{BR} := G x.(request \rightarrow F y.(response \wedge y \leq x + 10))$$

Among the models of this formula there are models featuring **infinitely many** requests, and consequently **infinitely many** responses.

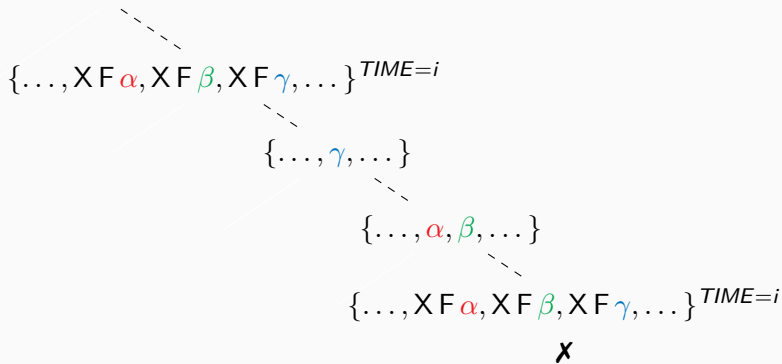
### LOOP rule

Let  $v$  be a poised leaf, and let  $u < v$  be a *poised node*, which is a proper ancestor of  $v$ , such that  $\Gamma(u) = \Gamma(v)$  and all the eventualities (i.e.,  $z.X(\alpha \mathcal{U} \beta)$  or  $z.X F \beta$ ) requested in  $u$  are fulfilled between  $u$  and  $v$  (included). Then,

- if  $\text{time}(u) = \text{time}(v)$ , then  $v$  is crossed and the branch rejected;
- if  $\text{time}(u) < \text{time}(v)$ , then  $v$  is ticked and the branch accepted.



## LOOP rule - Example



We cross the branch because the difference between the two timestamps is 0: the candidate model does *not* satisfy the progress condition.

## PRUNE rule

Consider the formula  $G \neg p \wedge q \mathcal{U} p$ . Even though it does not present any propositional contradiction, it is *unsatisfiable* because the eventuality  $p$  cannot be fulfilled.

### PRUNE rule

Let  $w$  be a poised leaf. If there exist **three poised nodes**  $u < v < w$  such that  $\Gamma(u) = \Gamma(v) = \Gamma(w)$ , and each eventuality requested in  $u$  and fulfilled between  $v$  and  $w$  is also fulfilled between  $u$  and  $v$ , then,  $w$  is crossed and the branch rejected.

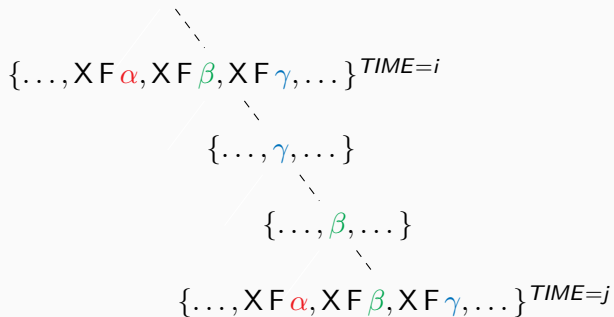
## PRUNE rule

Consider the formula  $G \neg p \wedge q \mathcal{U} p$ . Even though it does not present any propositional contradiction, it is *unsatisfiable* because the eventuality  $p$  cannot be fulfilled.

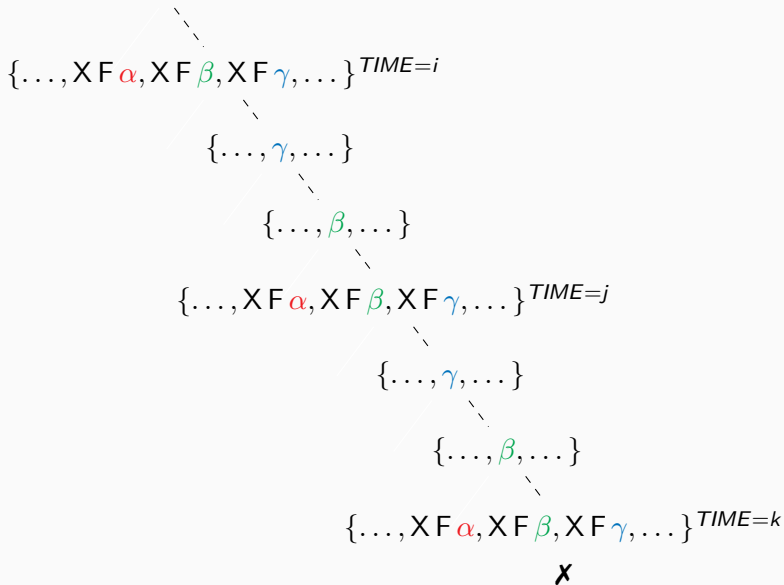
### PRUNE rule

Let  $w$  be a poised leaf. If there exist **three** *poised nodes*  $u < v < w$  such that  $\Gamma(u) = \Gamma(v) = \Gamma(w)$ , and each eventuality requested in  $u$  and fulfilled between  $v$  and  $w$  is also fulfilled between  $u$  and  $v$ , then,  $w$  is crossed and the branch rejected.

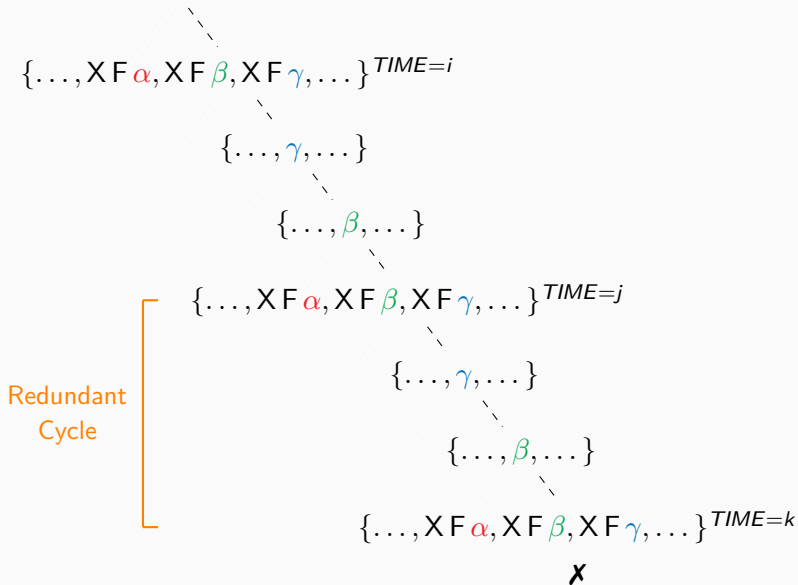
## PRUNE rule - Example



## PRUNE rule - Example



# PRUNE rule - Example



# PRUNE rule - Intuition

## Intuition

The PRUNE rule recognizes and prunes the **redundant cycles**.

Why three occurrences (and not only two)?

- with **two** nodes we identify **one** cycle (e.g., LOOP rule). If this cycle does not fulfill all the eventualities, then it is an **incomplete cycle**, but *not* redundant as it still can fulfill in the future the pending requests;
- with **three** nodes we identify **two** cycles. Therefore, if the second cycles fulfills a subset of the eventualities fulfilled by the first, then it is a **redundant cycle** and we can prune it.

» Conclusions

# PRUNE rule - Intuition

## Intuition

The PRUNE rule recognizes and prunes the **redundant cycles**.

Why three occurrences (and not only two)?

- with **two** nodes we identify **one** cycle (e.g., LOOP rule). If this cycle does not fulfill all the eventualities, then it is an **incomplete cycle**, but *not* redundant as it still can fulfill in the future the pending requests;
- with **three** nodes we identify **two** cycles. Therefore, if the second cycles fulfills a subset of the eventualities fulfilled by the first, then it is a **redundant cycle** and we can prune it.

» Conclusions



# PRUNE rule - Intuition

## Intuition

The PRUNE rule recognizes and prunes the **redundant cycles**.

Why three occurrences (and not only two)?

- with **two** nodes we identify **one** cycle (e.g., LOOP rule). If this cycle does not fulfill all the eventualities, then it is an **incomplete cycle**, but *not* redundant as it still can fulfill in the future the pending requests;
- with **three** nodes we identify **two** cycles. Therefore, if the second cycles fulfills a subset of the eventualities fulfilled by the first, then it is a **redundant cycle** and we can prune it.

▶ Conclusions

## Examples

---

## Tableau for TPTL - Example

$$\{x. G y. (p \rightarrow y \leq x + 2)\}^{TIME=0}$$

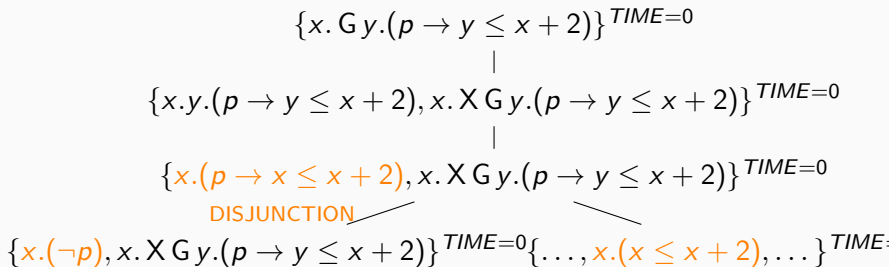
## Tableau for TPTL - Example

$$\begin{aligned} & \{x. G y. (p \rightarrow y \leq x + 2)\}^{TIME=0} \\ & \text{ALWAYS} | \\ & \{x. y. (p \rightarrow y \leq x + 2), x. X G y. (p \rightarrow y \leq x + 2)\}^{TIME=0} \end{aligned}$$

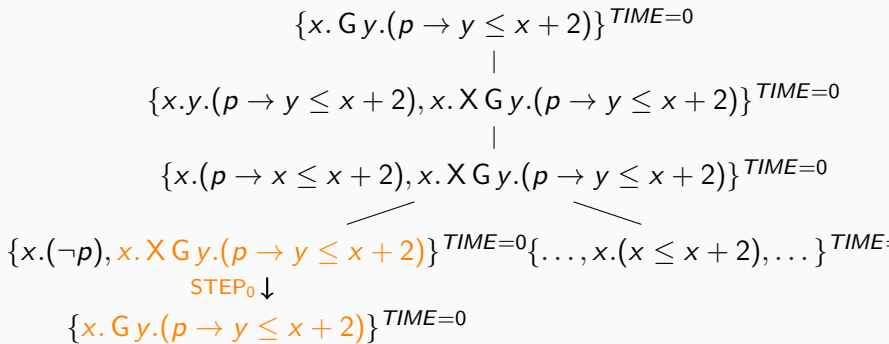
## Tableau for TPTL - Example

$$\begin{array}{c} \{x. G y. (p \rightarrow y \leq x + 2)\}^{TIME=0} \\ | \\ \{x. y. (p \rightarrow y \leq x + 2), x. X G y. (p \rightarrow y \leq x + 2)\}^{TIME=0} \\ \text{FREEZE} | \\ \{x. (p \rightarrow x \leq x + 2), x. X G y. (p \rightarrow y \leq x + 2)\}^{TIME=0} \end{array}$$

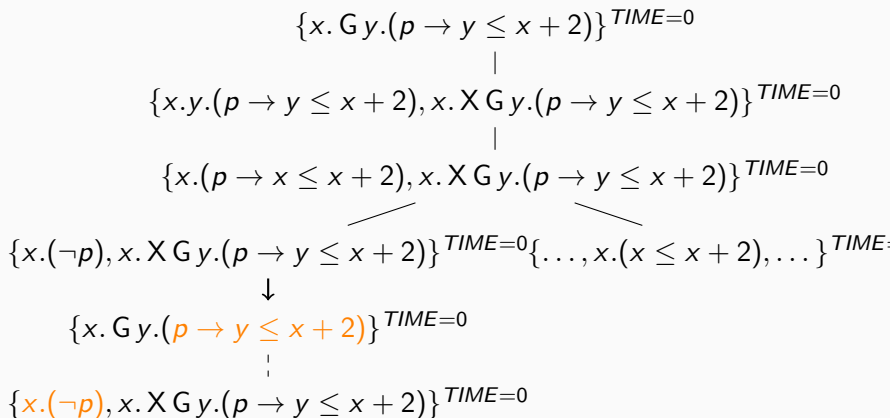
## Tableau for TPTL - Example



## Tableau for TPTL - Example

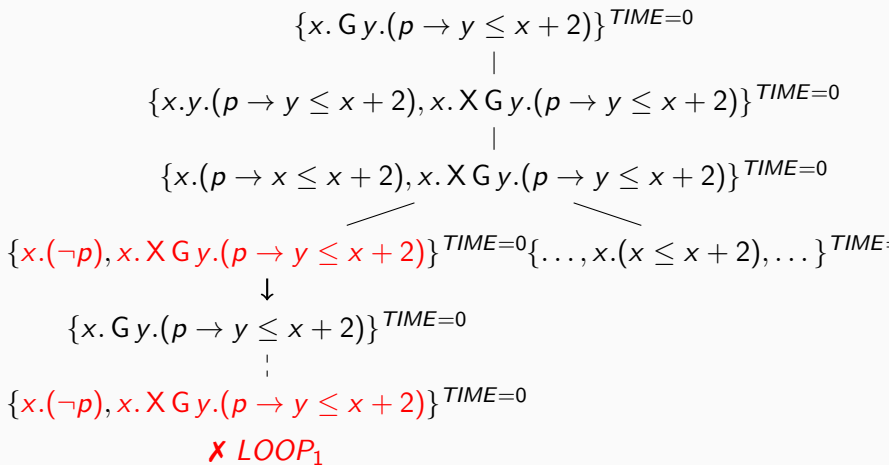


## Tableau for TPTL - Example





## Tableau for TPTL - Example



## Tableau for TPTL - Example

$$\{x.(\neg p), x.XGy.(p \rightarrow y \leq x + 2)\}^{TIME=0}$$

## Tableau for TPTL - Example

$$\{x.(\neg p), x.X G y.(p \rightarrow y \leq x + 2)\}^{TIME=0}$$

STEP<sub>1</sub> ↓

$$\{x. G y.(p \rightarrow y \leq x + 1)\}^{TIME=1}$$

## Tableau for TPTL - Example

$$\begin{array}{c} \{x.(\neg p), x.XGy.(p \rightarrow y \leq x + 2)\}^{TIME=0} \\ \downarrow \\ \{x.Gy.(p \rightarrow y \leq x + 1)\}^{TIME=1} \\ \vdots \\ \{x.(x \leq x + 1), x.XGy.(p \rightarrow y \leq x + 1)\}^{TIME=1} \end{array}$$

## Tableau for TPTL - Example

$$\begin{array}{c} \{x.(\neg p), x.X G y.(p \rightarrow y \leq x + 2)\}^{TIME=0} \\ \downarrow \\ \{x.G y.(p \rightarrow y \leq x + 1)\}^{TIME=1} \\ \vdots \\ \{x.(x \leq x + 1), x.X G y.(p \rightarrow y \leq x + 1)\}^{TIME=1} \\ \text{SYNC} \end{array}$$

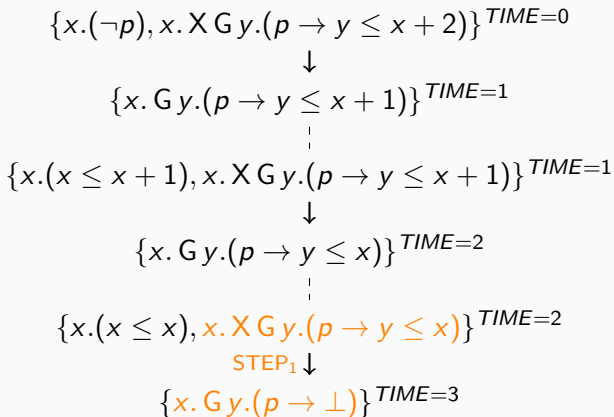
## Tableau for TPTL - Example

$$\begin{aligned} & \{x.(\neg p), x.X G y.(p \rightarrow y \leq x + 2)\}^{TIME=0} \\ & \quad \downarrow \\ & \{x.G y.(p \rightarrow y \leq x + 1)\}^{TIME=1} \\ & \quad \vdots \\ & \{x.(x \leq x + 1), x.X G y.(p \rightarrow y \leq x + 1)\}^{TIME=1} \\ & \quad \text{STEP}_1 \downarrow \\ & \{x.G y.(p \rightarrow y \leq x)\}^{TIME=2} \end{aligned}$$

## Tableau for TPTL - Example

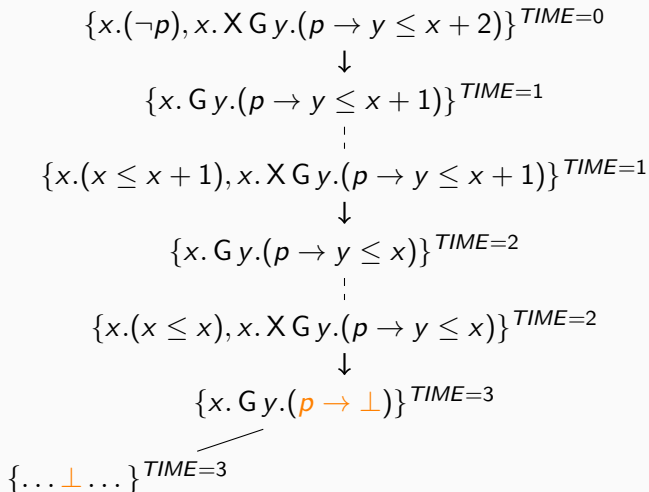
$$\begin{array}{c} \{x.(\neg p), x.X G y.(p \rightarrow y \leq x + 2)\}^{TIME=0} \\ \downarrow \\ \{x.G y.(p \rightarrow y \leq x + 1)\}^{TIME=1} \\ \vdots \\ \{x.(x \leq x + 1), x.X G y.(p \rightarrow y \leq x + 1)\}^{TIME=1} \\ \downarrow \\ \{x.G y.(p \rightarrow y \leq x)\}^{TIME=2} \\ \vdots \\ \{x.(x \leq x), x.X G y.(p \rightarrow y \leq x)\}^{TIME=2} \end{array}$$

## Tableau for TPTL - Example





## Tableau for TPTL - Example



## Tableau for TPTL - Example

$$\{x.(\neg p), x.X G y.(p \rightarrow y \leq x + 2)\}^{TIME=0}$$

↓

$$\{x.G y.(p \rightarrow y \leq x + 1)\}^{TIME=1}$$

⋮

$$\{x.(x \leq x + 1), x.X G y.(p \rightarrow y \leq x + 1)\}^{TIME=1}$$

↓

$$\{x.G y.(p \rightarrow y \leq x)\}^{TIME=2}$$

⋮

$$\{x.(x \leq x), x.X G y.(p \rightarrow y \leq x)\}^{TIME=2}$$

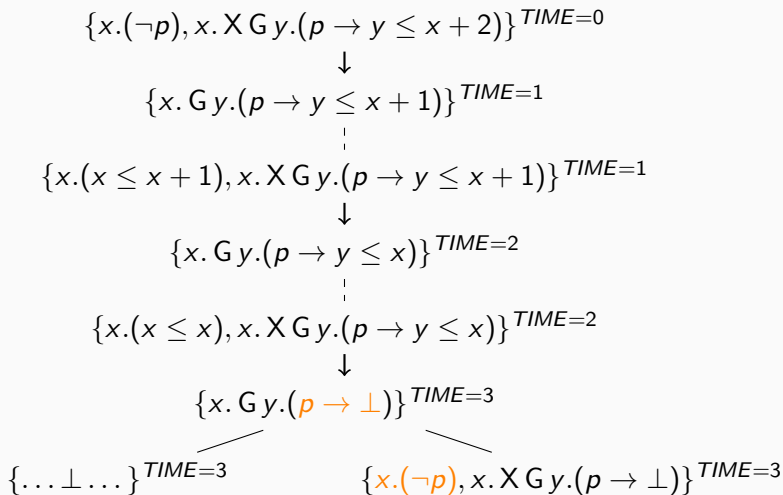
↓

$$\{x.G y.(p \rightarrow \perp)\}^{TIME=3}$$

$$\{\dots \perp \dots\}^{TIME=3}$$

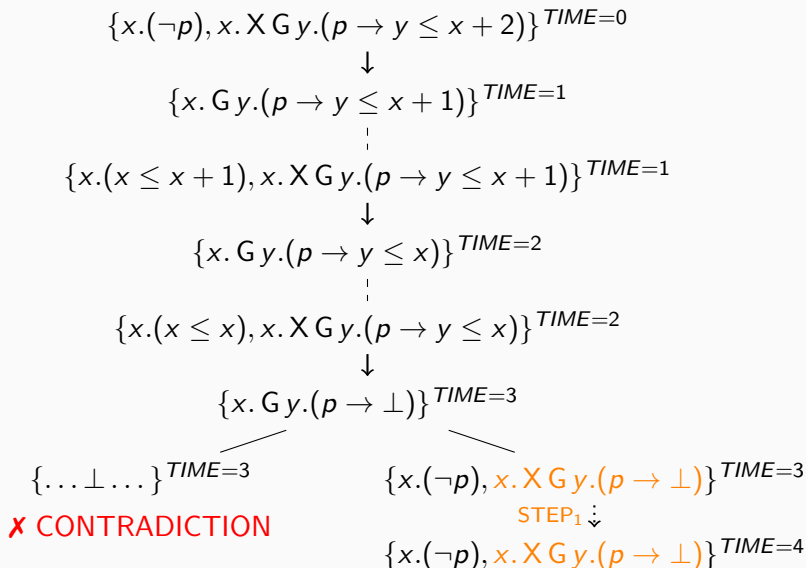
**X CONTRADICTION**

## Tableau for TPTL - Example



**X CONTRADICTION**

## Tableau for TPTL - Example



## Tableau for TPTL - Example

$$\{x.(\neg p), x.X G y.(p \rightarrow y \leq x + 2)\}^{TIME=0}$$

↓

$$\{x.G y.(p \rightarrow y \leq x + 1)\}^{TIME=1}$$

⋮

$$\{x.(x \leq x + 1), x.X G y.(p \rightarrow y \leq x + 1)\}^{TIME=1}$$

↓

$$\{x.G y.(p \rightarrow y \leq x)\}^{TIME=2}$$

⋮

$$\{x.(x \leq x), x.X G y.(p \rightarrow y \leq x)\}^{TIME=2}$$

↓

$$\{x.G y.(p \rightarrow \perp)\}^{TIME=3}$$

$$\{\dots \perp \dots\}^{TIME=3}$$

✗ CONTRADICTION

$$\{x.(\neg p), x.X G y.(p \rightarrow \perp)\}^{TIME=3}$$

⋮

$$\{x.(\neg p), x.X G y.(p \rightarrow \perp)\}^{TIME=4}$$

✓ LOOP<sub>2</sub>

## Tableau for $TPTL_b+P$

- It has the same structure of the previous tableau for  $TPTL$ .
- Now it is **not** true anymore that  $y$  is instantiated always in the future w.r.t.  $x$ , but we can give a priori a bound to the difference between the timestamps of two variables, thanks to the bounds on the temporal operators. This is crucial for simplifying the timed constraints.
- In order to deal with past modalities, the **YESTERDAY rule** has been introduced:

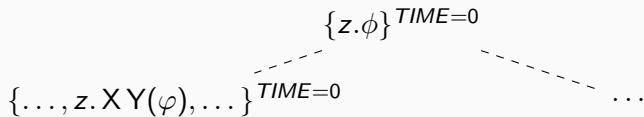
**YESTERDAY** : it checks if all the past requests made by the formulae of the current node are already satisfied by the previous nodes of the current branch;

- if this is not the case, the current branch is rejected and the construction of the tableau restarts from a previous state of the branch, assuming that all the past requests are true.

## Yesterday rule - Example

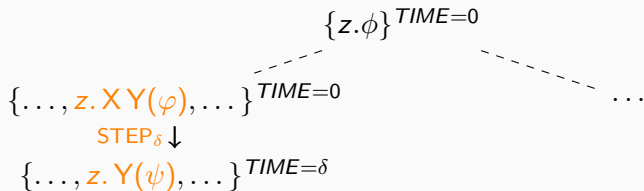
$$\{z.\phi\}^{TIME=0}$$

## Yesterday rule - Example

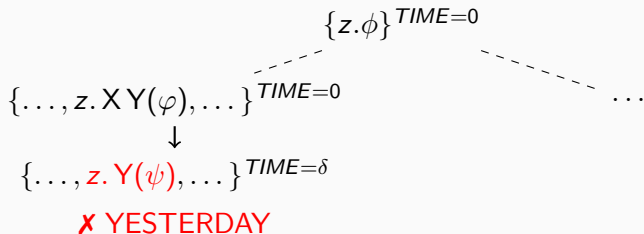




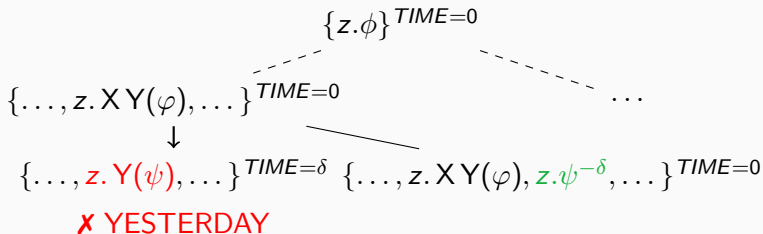
## Yesterday rule - Example



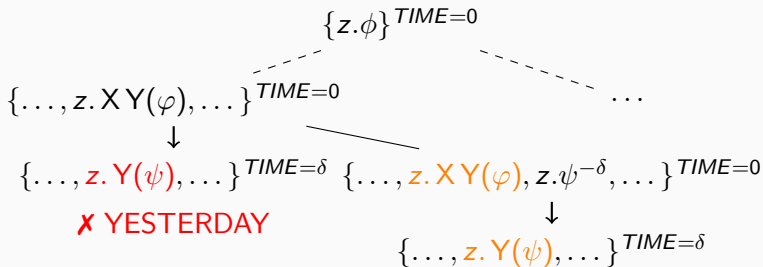
## Yesterday rule - Example



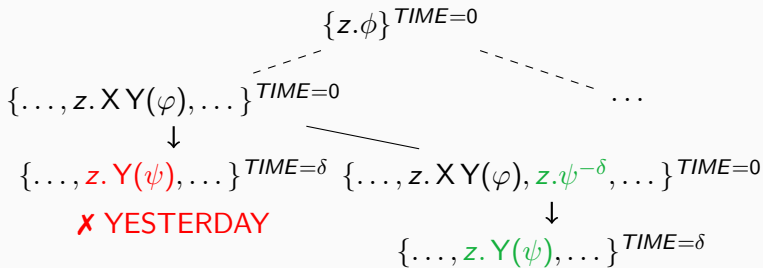
## Yesterday rule - Example



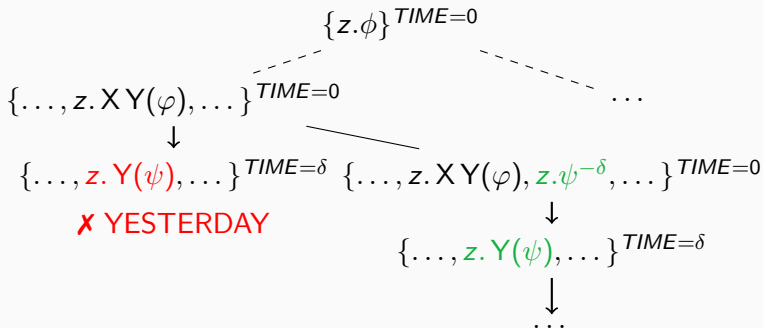
## Yesterday rule - Example



## Yesterday rule - Example



# Yesterday rule - Example



## Conclusions

---

**Results:** we developed two original *one-pass* and *tree-shaped* tableau systems for the logics TPTL and TPTL<sub>b</sub>+P.

- easy to implement and well suited for parallel implementations;
- no optimality: although the satisfiability problem for these two logics is EXPSPACE-complete, our tableau systems run in *doubly* exponential space (logarithmic encoding for the constants).



**Results:** we developed two original *one-pass* and *tree-shaped* tableau systems for the logics TPTL and TPTL<sub>b</sub>+P.

- easy to implement and well suited for parallel implementations;
- no optimality: although the satisfiability problem for these two logics is EXPSPACE-complete, our tableau systems run in *doubly* exponential space (logarithmic encoding for the constants).

**Results:** we developed two original *one-pass* and *tree-shaped* tableau systems for the logics TPTL and TPTL<sub>b</sub>+P.

- easy to implement and well suited for parallel implementations;
- no optimality: although the satisfiability problem for these two logics is EXPSPACE-complete, our tableau systems run in *doubly* exponential space (logarithmic encoding for the constants).

### Future developments:

- SAT-based encoding of the tableau for LTL, based on bounded satisfiability;
- SMT-based encoding of the tableau for TPTL, using Difference Logic (DL) as the underlying theory;
- tableau system for TPTL+P:
  - there is a heavy price to pay for the addition of past modalities to TPTL: the satisfiability problem for TPTL+P is *nonelementary*;
  - at the moment, there exists no direct procedure for deciding its satisfiability;
  - the main problem to solve is how to recognize a period.
- extending the tableau systems of [Ber+16] to other linear time temporal logics, e.g., *metric* LTL.

### Future developments:

- SAT-based encoding of the tableau for LTL, based on bounded satisfiability;
- SMT-based encoding of the tableau for TPTL, using Difference Logic (DL) as the underlying theory;
- tableau system for TPTL+P:
  - there is a heavy price to pay for the addition of past modalities to TPTL: the satisfiability problem for TPTL+P is *nonelementary*;
  - at the moment, there exists no direct procedure for deciding its satisfiability;
  - the main problem to solve is how to recognize a period.
- extending the tableau systems of [Ber+16] to other linear time temporal logics, e.g., *metric* LTL.

### Future developments:

- SAT-based encoding of the tableau for LTL, based on bounded satisfiability;
- SMT-based encoding of the tableau for TPTL, using Difference Logic (DL) as the underlying theory;
- tableau system for TPTL+P:
  - there is a heavy price to pay for the addition of past modalities to TPTL: the satisfiability problem for TPTL+P is *nonelementary*;
  - at the moment, there exists no direct procedure for deciding its satisfiability;
  - the main problem to solve is how to recognize a period.
- extending the tableau systems of [Ber+16] to other linear time temporal logics, e.g., *metric* LTL.

### Future developments:

- SAT-based encoding of the tableau for LTL, based on bounded satisfiability;
- SMT-based encoding of the tableau for TPTL, using Difference Logic (DL) as the underlying theory;
- tableau system for TPTL+P:
  - there is a heavy price to pay for the addition of past modalities to TPTL: the satisfiability problem for TPTL+P is *nonelementary*;
  - at the moment, there exists no direct procedure for deciding its satisfiability;
    - the main problem to solve is how to recognize a period.
- extending the tableau systems of [Ber+16] to other linear time temporal logics, e.g., *metric* LTL.

### Future developments:

- SAT-based encoding of the tableau for LTL, based on bounded satisfiability;
- SMT-based encoding of the tableau for TPTL, using Difference Logic (DL) as the underlying theory;
- tableau system for TPTL+P:
  - there is a heavy price to pay for the addition of past modalities to TPTL: the satisfiability problem for TPTL+P is *nonelementary*;
  - at the moment, there exists no direct procedure for deciding its satisfiability;
  - the main problem to solve is how to recognize a period.
- extending the tableau systems of [Ber+16] to other linear time temporal logics, e.g., *metric* LTL.

### Future developments:

- SAT-based encoding of the tableau for LTL, based on bounded satisfiability;
- SMT-based encoding of the tableau for TPTL, using Difference Logic (DL) as the underlying theory;
- tableau system for TPTL+P:
  - there is a heavy price to pay for the addition of past modalities to TPTL: the satisfiability problem for TPTL+P is *nonelementary*;
  - at the moment, there exists no direct procedure for deciding its satisfiability;
  - the main problem to solve is how to recognize a period.
- extending the tableau systems of [Ber+16] to other linear time temporal logics, e.g., *metric* LTL.



**Thank you for your attention!**

## References

---



Rajeev Alur and Thomas A. Henzinger. “A Really Temporal Logic”. In: *J. ACM* 41.1 (Jan. 1994), pp. 181–203. ISSN: 0004-5411. DOI: 10.1145/174644.174651. URL: <http://doi.acm.org/10.1145/174644.174651>.



Matteo Bertello, Nicola Gigante, Angelo Montanari, and Mark Reynolds. “Leviathan: A New LTL Satisfiability Checking Tool Based on a One-Pass Tree-Shaped Tableau”. In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*. 2016, pp. 950–956. URL: <http://www.ijcai.org/Abstract/16/139>.



Dario Della Monica, Nicola Gigante, Angelo Montanari, Pietro Sala, and Guido Sciavicco. “Bounded Timed Propositional Temporal Logic with Past Captures Timeline-based Planning with Bounded Constraints”. In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*. 2017, pp. 1008–1014. DOI: 10.24963/ijcai.2017/140. URL: <https://doi.org/10.24963/ijcai.2017/140>.



Zohar Manna and Amir Pnueli. *Temporal Verification of Reactive Systems: Safety*. New York, NY, USA: Springer-Verlag New York, Inc., 1995. ISBN: 0-387-94459-1.