A One-Pass Tree-Shaped Tableau for LTL

Nicola Gigante

Ph.D. course on *Temporal Logic: Satisfiability and Model Checking*
13/01/2017 - DiMA - University of Udine

Introduction

Linear Temporal Logic

Recall the syntax and semantics of Linear Temporal Logic (LTL):

$\mathsf{X}\,\alpha$	α will be true at the next state.
$\alpha\,\mathcal{U}\,\beta$	β will eventually be true, and α always holds until then.
$\mathsf{F}\,\beta \equiv \top\,\mathcal{U}\,\beta$	β will eventually be true.
$\mathsf{G}\,\beta \equiv \neg\,\mathsf{F}\,\neg\beta$	β will always be true.
$\alpha\,\mathcal{R}\,\beta \equiv \neg(\neg\alpha\,\mathcal{U}\,\neg\beta)$	β has to be true up to and including when α first comes true (if ever)

LTL Satisfiability

LTL satisfiability is the problem of checking whether there exists a model that satisfies a given LTL formula.

- PSPACE-complete problem.
- A lot of algorithmic solutions exist:
  - (Büchi) Automata-based
  - Tableau methods
  - Temporal resolution
  - Reduction to model checking

Tableau methods for LTL satisfiability

Tableaux were the first methods proposed to solve the LTL satisfiability problem:

- Early tableau methods were graph-shaped and multiple-pass [Wol85] [LP00].
- Subsequently, Schwendimann introduced a *single*-pass tableau with a tree-like shape [Sch98].
  - The tableau structure is still not really a tree, as there is communication between the different branches.

[LP00] Lichtenstein and Pnueli. *"Propositional Temporal Logics: Decidability and Completeness"*

[Sch98] Schwendimann. *"A New One-Pass Tableau Calculus for PLTL"*

A new solution

Today we will describe a tree-shaped one-pass tableau method for LTL satisfiability [Rey16]:

- The structure is an actual tree, built in a single pass.
- Simple to explain and to prove.
- Efficient C++ implementation available [Ber+16]:
  - Extensively compared with other tools (not only tableaux-based)
  - Good speed and memory consumption on average

[Rey16] Reynolds. *"A new rule for LTL tableaux"*

[Ber+16] Bertello, Gigante, Montanari, and Reynolds. *"Leviathan: A New LTL Satisfiability Checking Tool Based on a One-Pass Tree-Shaped Tableau"*

Experimental results

Experimental results with our implementation are promising:

- Performance are generally comparable with other tools
- Consistently competitive with other tableaux
- Good memory usage

All of this despite the simplicity of the algorithm and no additional search heuristics.

How it works

How it works - rules

The tableau for ϕ is a tree where each node is labeled by a set of formulae, with the root labeled with $\{\phi\}$.

- The formula starts in Negated Normal Form.
- At each step, a set of rules is applied to the formulae inside the label of the current node:
  - Static rules replace some formula by something else, looking only at the formula itself.
  - Non-static rules look also at the history of the branch.
  - Some rules can reject the branch (✗), or can accept it (✓).

Static rules

Static rules for boolean connectives work just like in classical propositional tableau.

$$\begin{array}{ccc} \{\alpha \vee \beta\} & & \{\alpha \wedge \beta\} \\ \swarrow \quad \searrow & & \downarrow \\ \{\alpha\} \quad \{\beta\} & & \{\alpha, \beta\} \end{array}$$

Static rules for temporal operators are also quite standard:

$$\begin{array}{ccccccc} \{\alpha \,\mathcal{U}\, \beta\} & & & \{\mathsf{F}\,\beta\} & & \{\mathsf{G}\,\alpha\} \\ \swarrow \quad \searrow & & & \swarrow \quad \searrow & & \downarrow \\ \{\beta\} \quad \{\alpha, \mathsf{X}(\alpha \,\mathcal{U}\, \beta)\} & & & \{\beta\} \quad \{\alpha, \mathsf{X}\,\mathsf{F}\,\beta\} & & \{\alpha, \mathsf{X}\,\mathsf{G}\,\alpha\} \end{array}$$

Static rules

Static rules for boolean connectives work just like in classical propositional tableau.

$$\begin{array}{ccc} \{\alpha \vee \beta\} & & \{\alpha \wedge \beta\} \\ \swarrow \quad \searrow & & \downarrow \\ \{\alpha\} \quad \{\beta\} & & \{\alpha, \beta\} \end{array}$$

Static rules for temporal operators are also quite standard:

$$\begin{array}{ccccccc} \{\alpha \,\mathcal{U}\, \beta\} & & & \{\mathsf{F}\,\beta\} & & \{\mathsf{G}\,\alpha\} \\ \swarrow \quad \searrow & & & \swarrow \quad \searrow & & \downarrow \\ \{\beta\} \quad \{\alpha, \mathsf{X}(\alpha \,\mathcal{U}\, \beta)\} & & & \{\beta\} \quad \{\alpha, \mathsf{X}\,\mathsf{F}\,\beta\} & & \{\alpha, \mathsf{X}\,\mathsf{G}\,\alpha\} \end{array}$$

$\mathsf{X}(\alpha \,\mathcal{U}\, \beta)$ and $\mathsf{X}(\mathsf{F}\,\beta)$ are called X-eventualities.

Step rule

After applying static rules to the label of a node, we will reach a node with a poised label, *i.e.*, containing only literals and tomorrows.

To a poised label we can apply the Step rule, which advances the branch to the next time step:

$$\{p, \neg q, \mathsf{X}\,\alpha, \mathsf{X}\,\beta\}$$
$$\downarrow$$
$$\{\alpha, \beta\}$$

Empty labels and contradictions

Other two static rules are special, because they can accept or reject a branch:

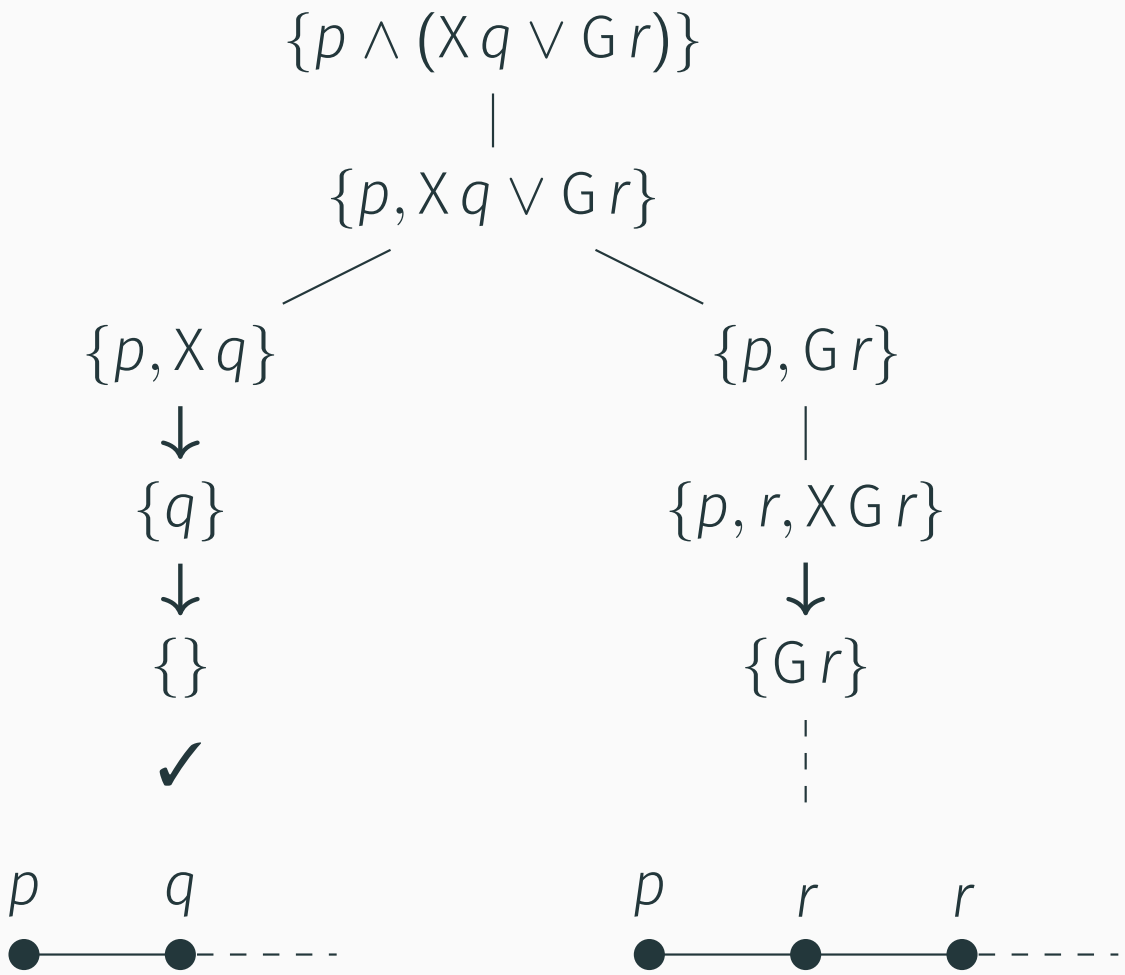
Empty If a label (probably after Step rule) is empty, the branch is accepted.

Contradiction If a label contains a propositional contradiction (*i.e.*, p and $\neg p$), the branch is crossed.

Once a ticked leaf has been found, you can extract a model of the formula from the poised labels before each application of the Step rule.

Example

Let's try to build a tableau for the formula $p \wedge (\mathsf{X}\, q \vee \mathsf{G}\, r)$.



Looping models

In the previous example, what happens if we continue to expand the same node with label $\{\mathsf{G}\, r\}$ again and again?

Looping models

The resulting infinitely repeating model is what we are looking for, but we need an halting condition for the expansion:

Loop rule

If two nodes $u < v$ with labels $\Gamma_u \supseteq \Gamma_v$ are found and all the eventualities in Γ_u are fulfilled inbetween, the branch is accepted and the model loops through u and v.

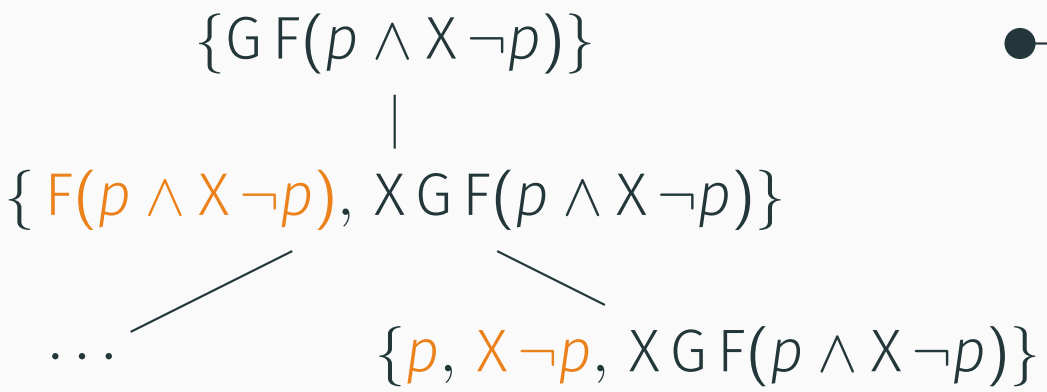
Example - looping model

$\{\mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\,\neg p)\}$

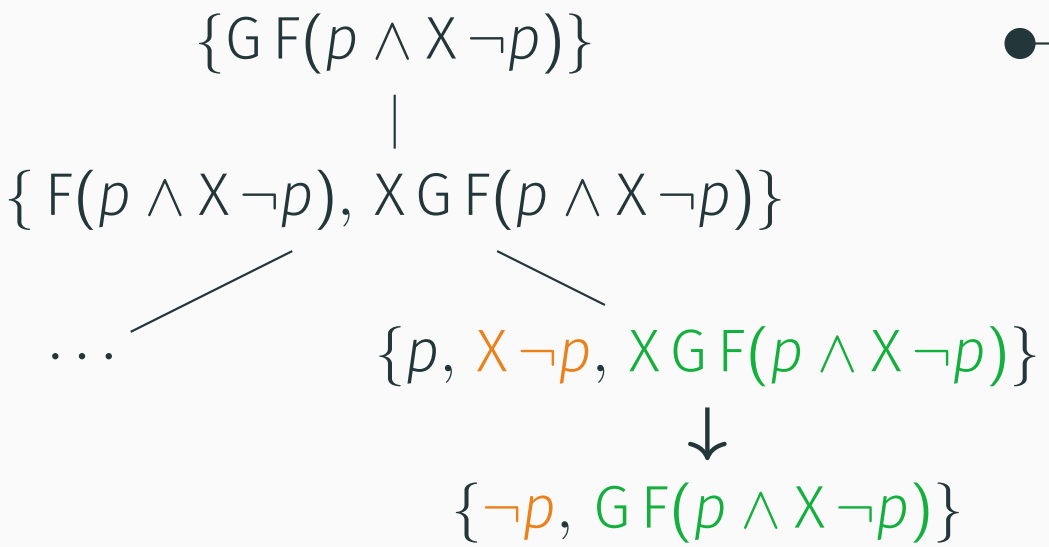
Example - looping model

$$\{\mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\,\neg p)\}$$
$$|$$
$$\{\,\mathsf{F}(p \wedge \mathsf{X}\,\neg p),\ \mathsf{X}\,\mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\,\neg p)\}$$

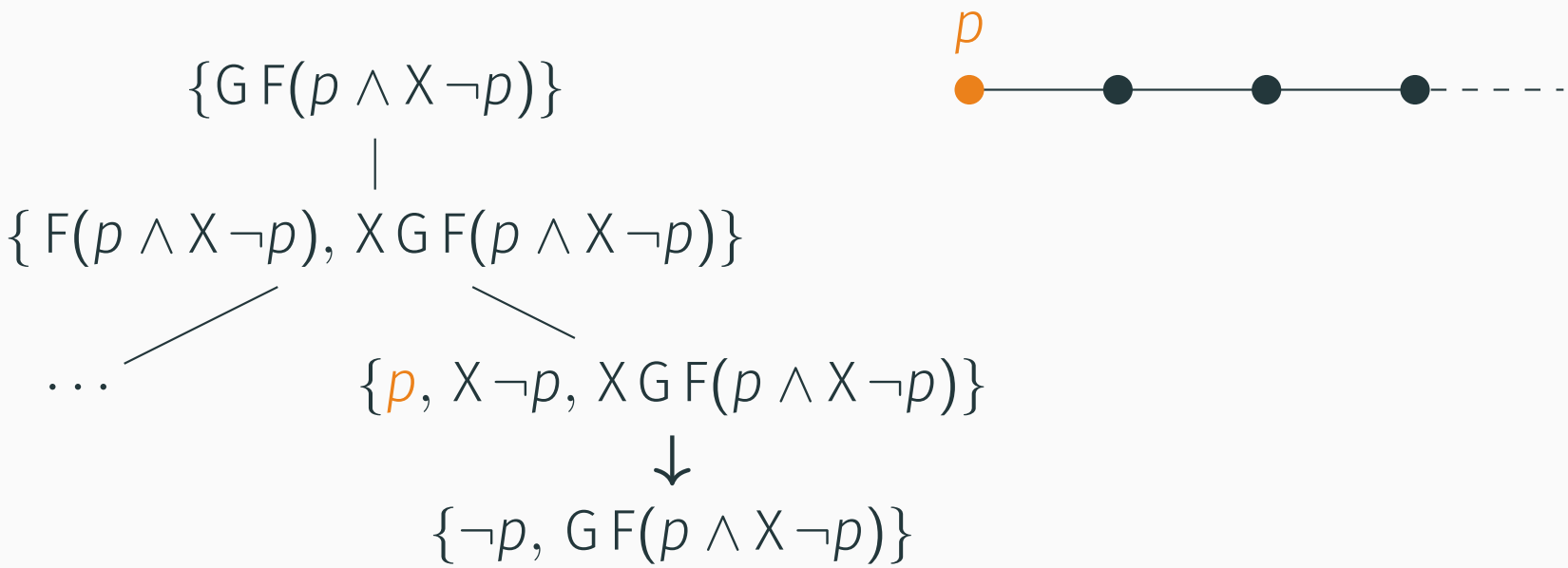
Example - looping model



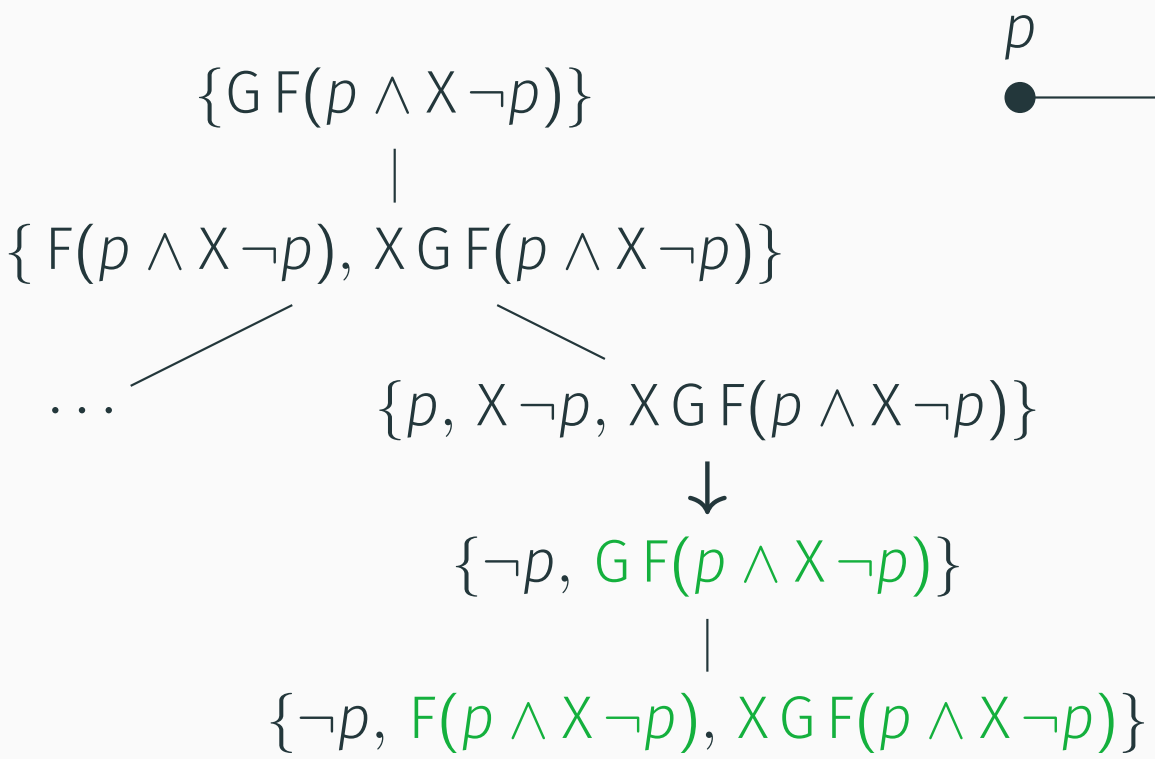
Example - looping model



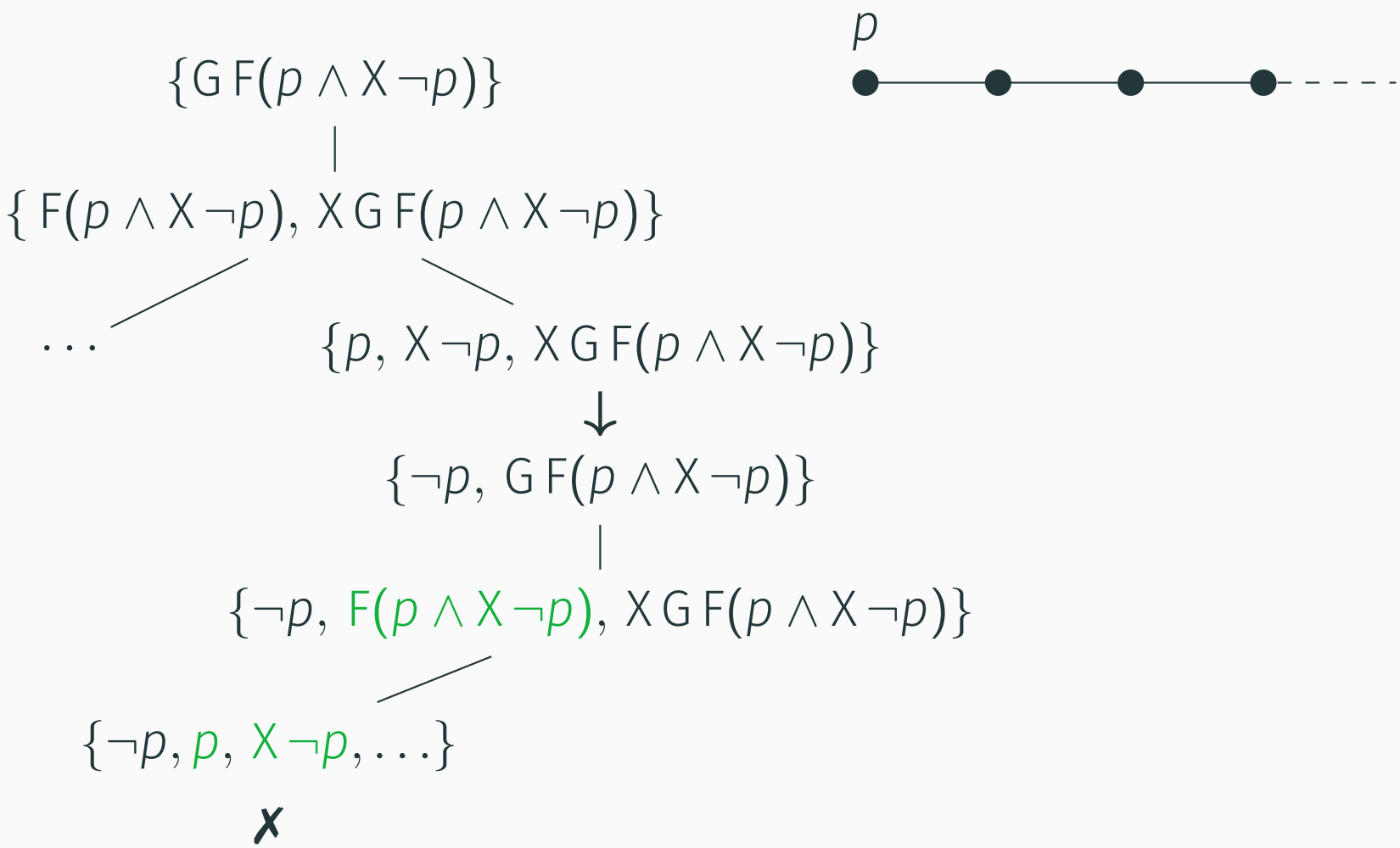
Example - looping model



Example - looping model



Example - looping model



Example - looping model

$\{\mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\,\neg p)\}$

$\{\ \mathsf{F}(p \wedge \mathsf{X}\,\neg p),\ \mathsf{X}\,\mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\,\neg p)\}$

$\ldots$ $\{p,\ \mathsf{X}\,\neg p,\ \mathsf{X}\,\mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\,\neg p)\}$

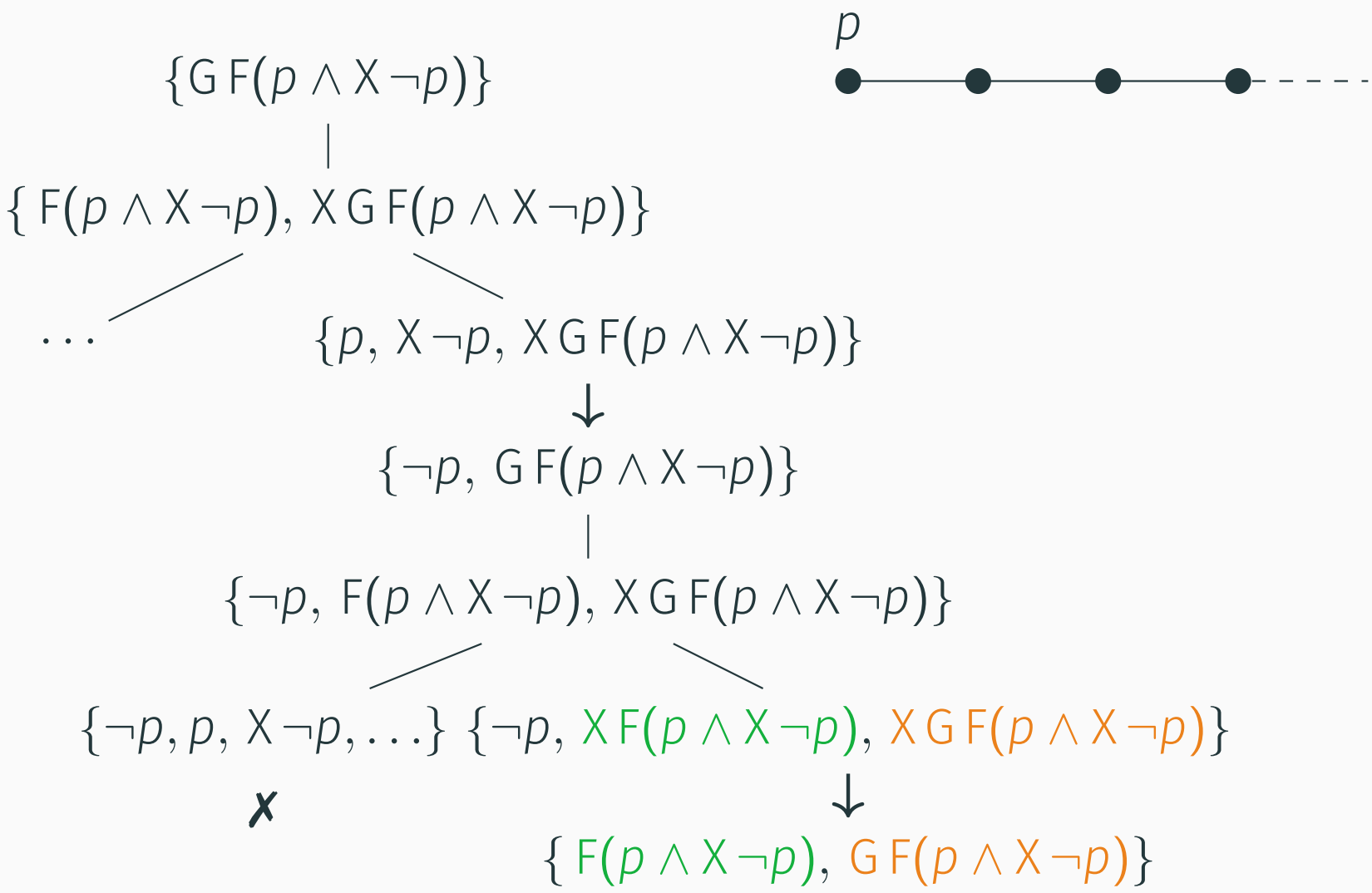
$\{\neg p,\ \mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\,\neg p)\}$

$\{\neg p,\ \mathsf{F}(p \wedge \mathsf{X}\,\neg p),\ \mathsf{X}\,\mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\,\neg p)\}$

$\{\neg p, p,\ \mathsf{X}\,\neg p, \ldots\}$ $\{\neg p,\ \mathsf{X}\,\mathsf{F}(p \wedge \mathsf{X}\,\neg p),\ \mathsf{X}\,\mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\,\neg p)\}$

✗

Example - looping model



Example - looping model

$\{\mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\neg p)\}$

$\{\,\mathsf{F}(p \wedge \mathsf{X}\neg p),\ \mathsf{X}\,\mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\neg p)\}$

$\ldots$ $\{p,\ \mathsf{X}\neg p,\ \mathsf{X}\,\mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\neg p)\}$

$\{\neg p,\ \mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\neg p)\}$

$\{\neg p,\ \mathsf{F}(p \wedge \mathsf{X}\neg p),\ \mathsf{X}\,\mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\neg p)\}$

$\{\neg p, p,\ \mathsf{X}\neg p, \ldots\}$ $\{\neg p,\ \mathsf{X}\,\mathsf{F}(p \wedge \mathsf{X}\neg p),\ \mathsf{X}\,\mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\neg p)\}$

✗

$\{\,\mathsf{F}(p \wedge \mathsf{X}\neg p),\ \mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\neg p)\}$

Example - looping model

$\{\mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\neg p)\}$

$\{\,\mathsf{F}(p \wedge \mathsf{X}\neg p),\ \mathsf{X}\,\mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\neg p)\}$

$\ldots$ $\{p,\ \mathsf{X}\neg p,\ \mathsf{X}\,\mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\neg p)\}$

$\{\neg p,\ \mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\neg p)\}$

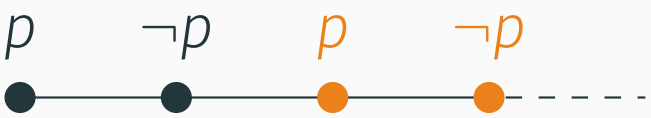
$\{\neg p,\ \mathsf{F}(p \wedge \mathsf{X}\neg p),\ \mathsf{X}\,\mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\neg p)\}$

$\{\neg p, p,\ \mathsf{X}\neg p, \ldots\}$ $\{\neg p,\ \mathsf{X}\,\mathsf{F}(p \wedge \mathsf{X}\neg p),\ \mathsf{X}\,\mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\neg p)\}$

✗

$\{\,\mathsf{F}(p \wedge \mathsf{X}\neg p),\ \mathsf{G}\,\mathsf{F}(p \wedge \mathsf{X}\neg p)\}$

Example - looping model



$\{\mathrm{G}\,\mathrm{F}(p \wedge \mathrm{X}\neg p)\}$

$\{\,\mathrm{F}(p \wedge \mathrm{X}\neg p),\ \mathrm{X}\,\mathrm{G}\,\mathrm{F}(p \wedge \mathrm{X}\neg p)\}$

$\ldots$ $\{p,\ \mathrm{X}\neg p,\ \mathrm{X}\,\mathrm{G}\,\mathrm{F}(p \wedge \mathrm{X}\neg p)\}$

$\{\neg p,\ \mathrm{G}\,\mathrm{F}(p \wedge \mathrm{X}\neg p)\}$

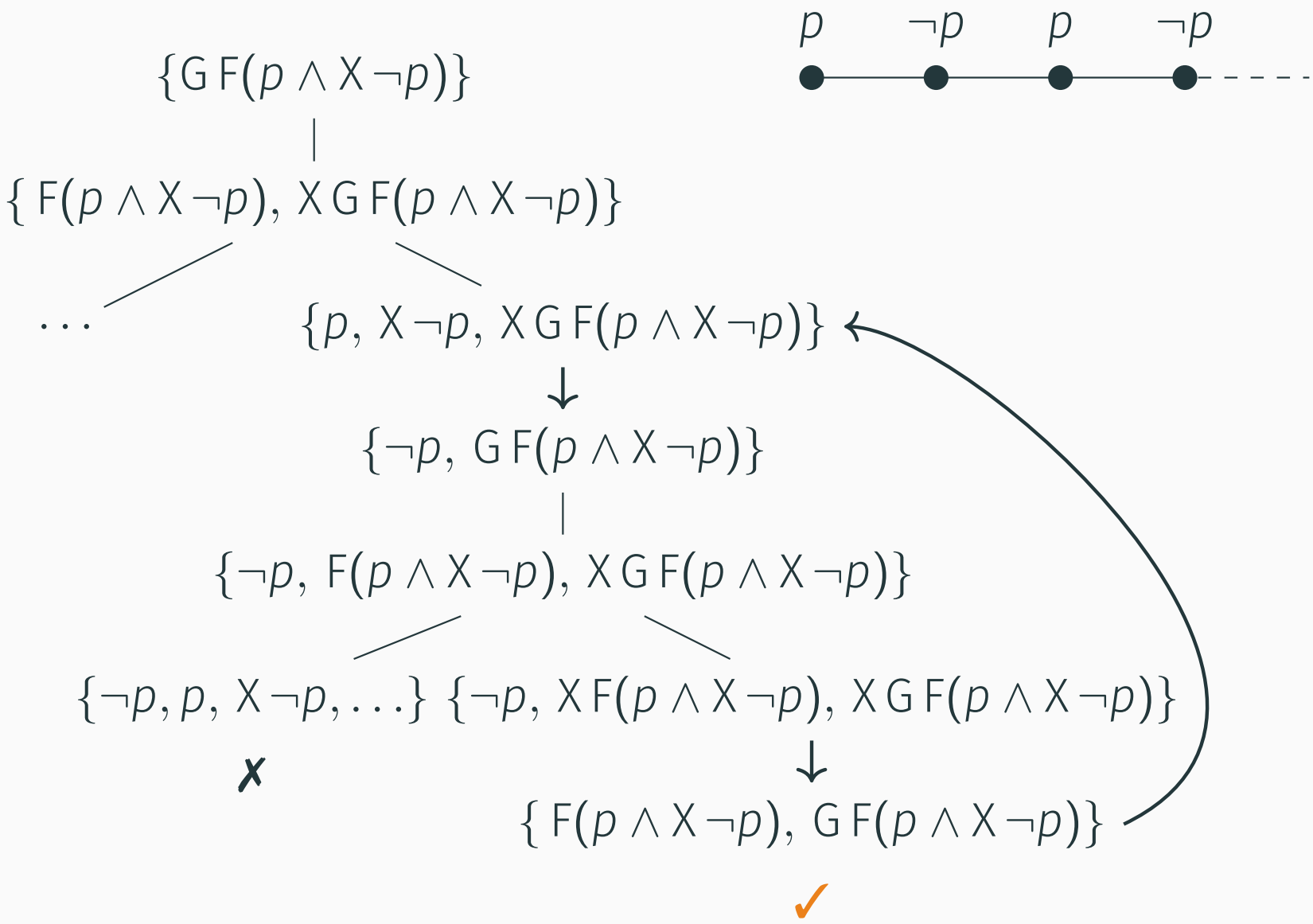
$\{\neg p,\ \mathrm{F}(p \wedge \mathrm{X}\neg p),\ \mathrm{X}\,\mathrm{G}\,\mathrm{F}(p \wedge \mathrm{X}\neg p)\}$

$\{\neg p, p,\ \mathrm{X}\neg p, \ldots\}$ $\{\neg p,\ \mathrm{X}\,\mathrm{F}(p \wedge \mathrm{X}\neg p),\ \mathrm{X}\,\mathrm{G}\,\mathrm{F}(p \wedge \mathrm{X}\neg p)\}$

✗

$\{\,\mathrm{F}(p \wedge \mathrm{X}\neg p),\ \mathrm{G}\,\mathrm{F}(p \wedge \mathrm{X}\neg p)\}$

Example - looping model



Unsatisfiable requests

What if it is not possible to ever satisfy all the required eventualities?

- E.g. $\mathsf{G}\,\neg p \wedge q\,\mathcal{U}\,p$
- It is unsatisfiable, but not for any (direct) propositional contradiction

Unsatisfiable requests - The Prune rule

The Prune rule, the main novelty here, handles this case:

Prune rule

The branch is rejected if all the following conditions are met:

- three nodes $u < v < w$ with the same label Γ are found
- not all the eventualities requested in Γ are fulfilled between the nodes
  (trivial because checked after the Loop rule)
- the set of eventualities fulfilled between u and v is a subset of those between v and w.

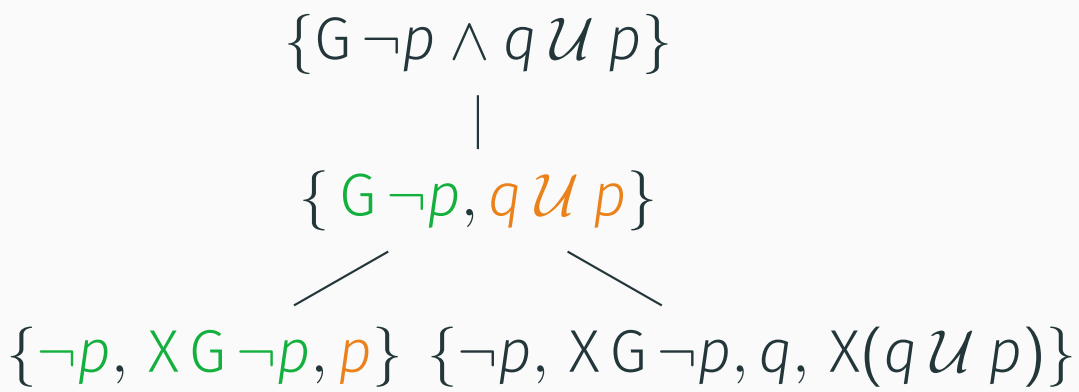
Example - unsatisfiable formula

$$\{\mathsf{G}\,\neg p \wedge q\,\mathcal{U}\,p\}$$

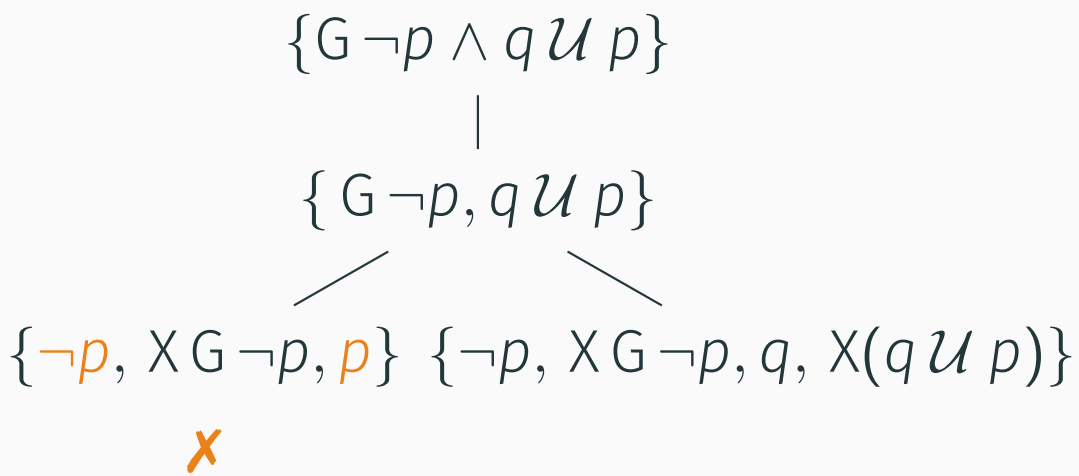
Example - unsatisfiable formula

$$\{\mathsf{G}\,\neg p \wedge q\,\mathcal{U}\,p\}$$
$$|$$
$$\{\,\mathsf{G}\,\neg p, q\,\mathcal{U}\,p\}$$

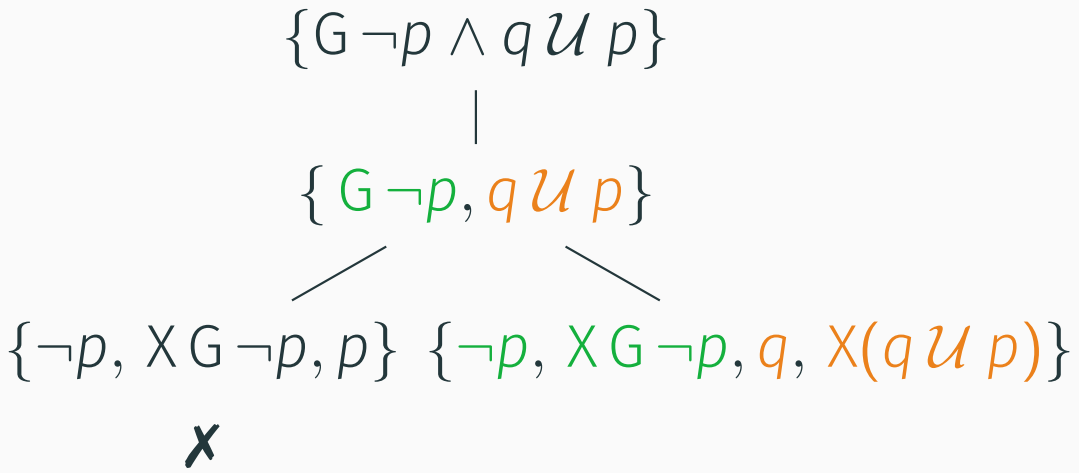
Example - unsatisfiable formula



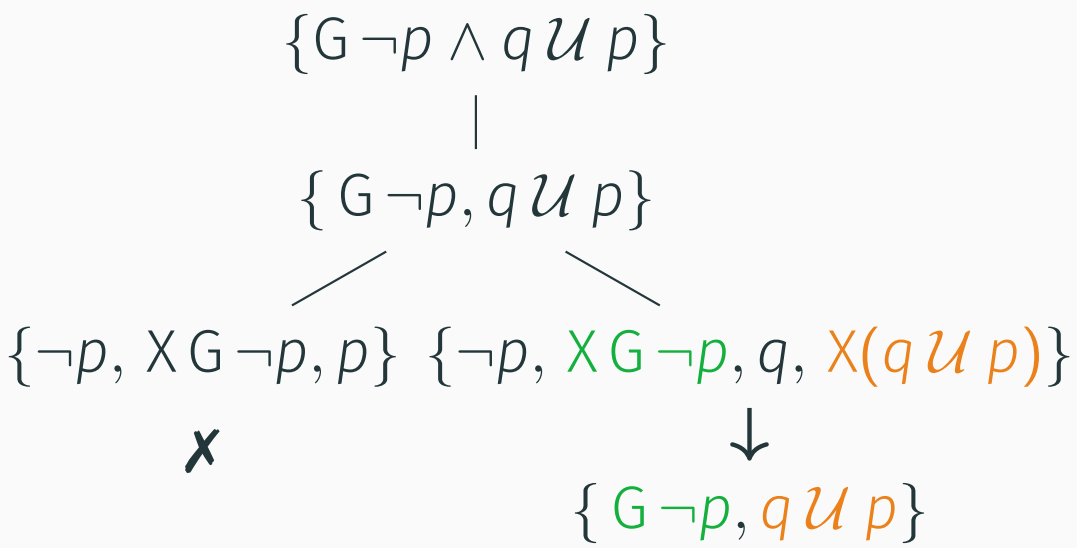
Example - unsatisfiable formula



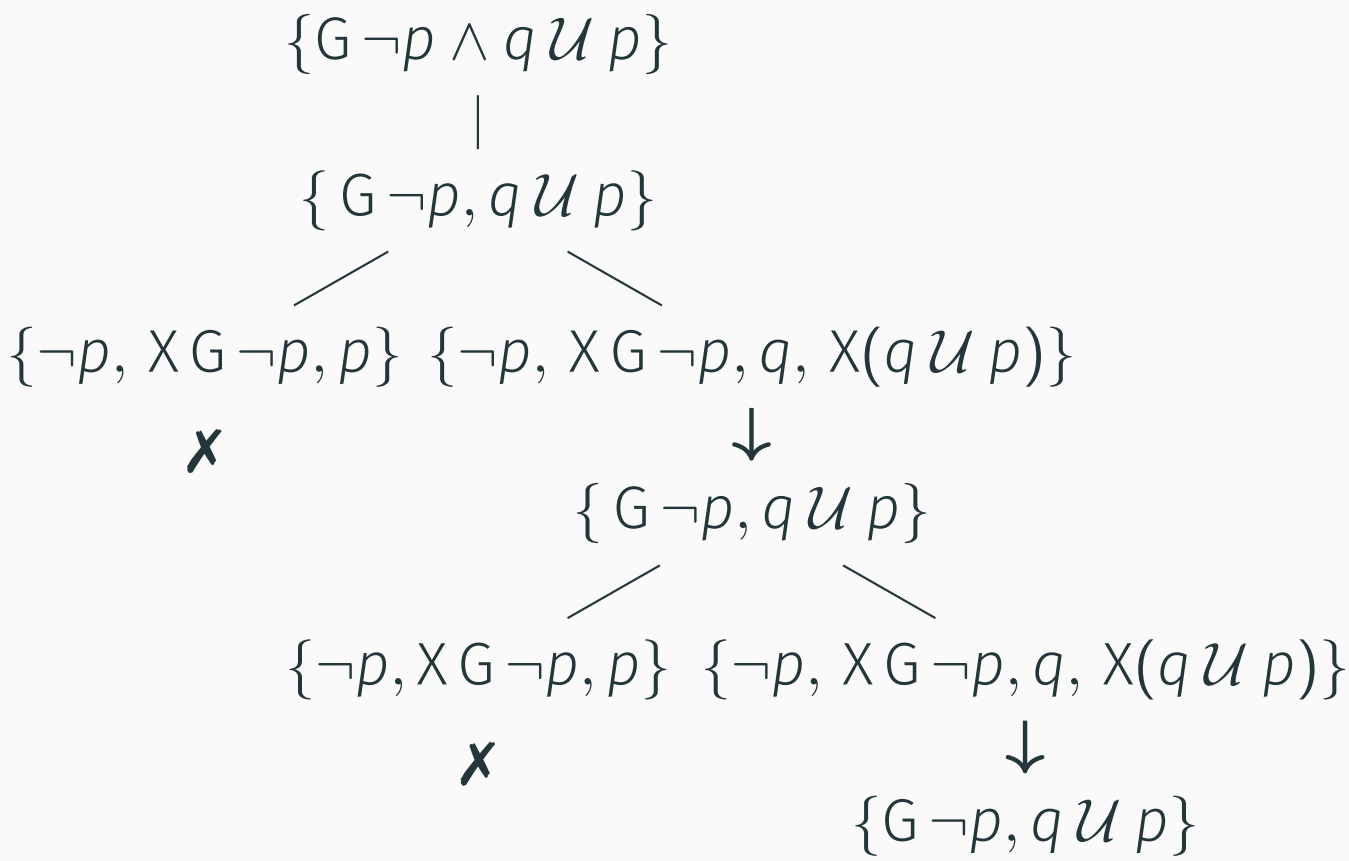
Example - unsatisfiable formula



Example - unsatisfiable formula



Example - unsatisfiable formula



Example - unsatisfiable formula

$$\{\mathsf{G}\neg p \wedge q\,\mathcal{U}\,p\}$$

$$\{\mathsf{G}\neg p,\ q\,\mathcal{U}\,p\}$$

$$\{\neg p,\ \mathsf{X}\,\mathsf{G}\neg p,\ p\} \qquad \{\neg p,\ \mathsf{X}\,\mathsf{G}\neg p,\ q,\ \mathsf{X}(q\,\mathcal{U}\,p)\}$$

✗ $\downarrow$

$$\{\mathsf{G}\neg p,\ q\,\mathcal{U}\,p\}$$

$$\{\neg p,\ \mathsf{X}\,\mathsf{G}\neg p,\ p\} \qquad \{\neg p,\ \mathsf{X}\,\mathsf{G}\neg p,\ q,\ \mathsf{X}(q\,\mathcal{U}\,p)\}$$

✗ $\downarrow$

$$\{\mathsf{G}\neg p,\ q\,\mathcal{U}\,p\}$$

$$\{\neg p,\ \mathsf{X}\,\mathsf{G}\neg p,\ p\} \qquad \{\neg p,\ \mathsf{X}\,\mathsf{G}\neg p,\ q,\ \mathsf{X}(q\,\mathcal{U}\,p)\}$$

✗

Example - unsatisfiable formula

$$\{\mathsf{G}\,\neg p \land q\,\mathcal{U}\,p\}$$

$$\{\mathsf{G}\,\neg p, q\,\mathcal{U}\,p\}$$

$$\{\neg p, \mathsf{X}\,\mathsf{G}\,\neg p, p\} \quad \{\neg p, \mathsf{X}\,\mathsf{G}\,\neg p, q, \mathsf{X}(q\,\mathcal{U}\,p)\}$$

✗ $\downarrow$

$$\{\mathsf{G}\,\neg p, q\,\mathcal{U}\,p\}$$

$$\{\neg p, \mathsf{X}\,\mathsf{G}\,\neg p, p\} \quad \{\neg p, \mathsf{X}\,\mathsf{G}\,\neg p, q, \mathsf{X}(q\,\mathcal{U}\,p)\}$$

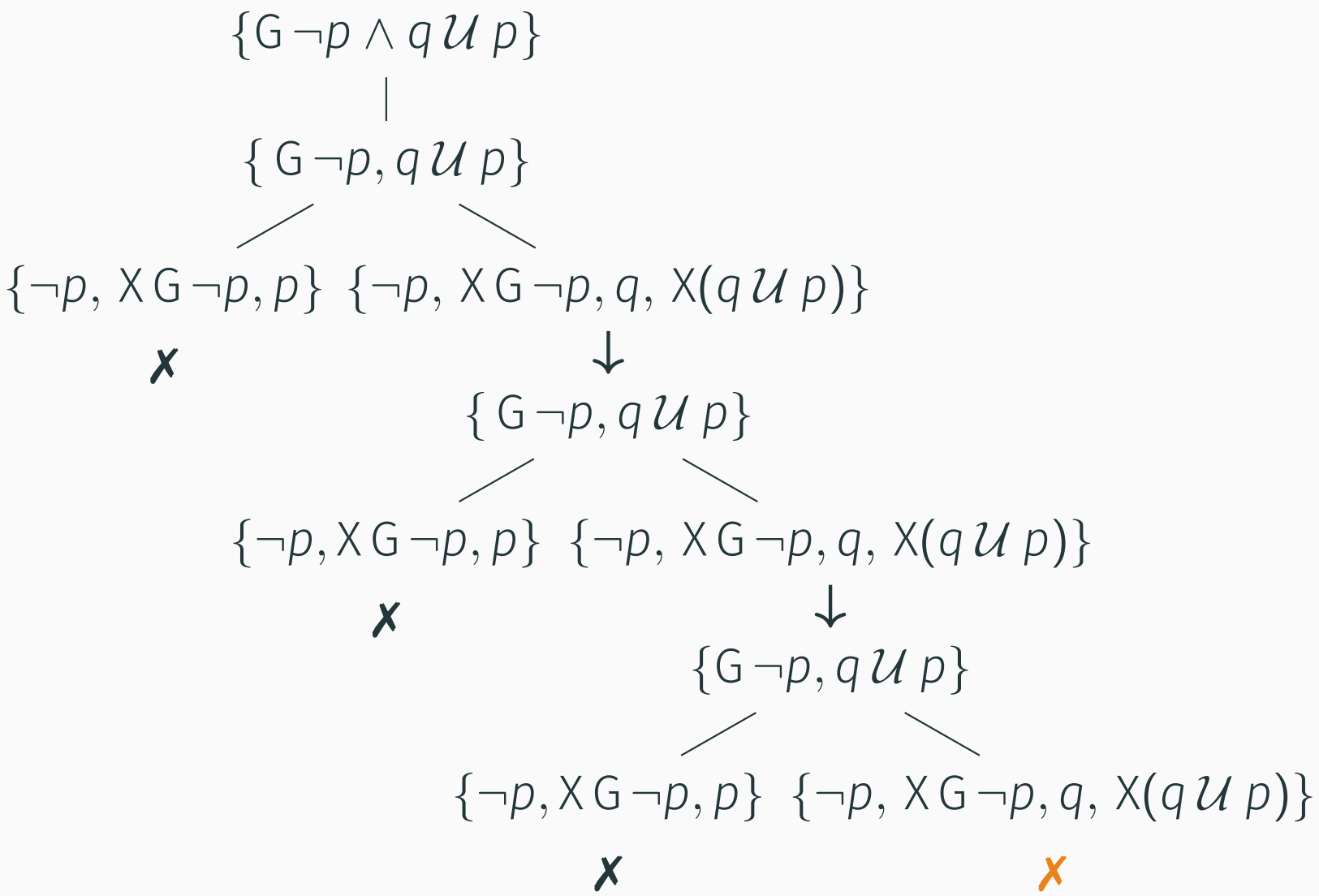
✗ $\downarrow$

$$\{\mathsf{G}\,\neg p, q\,\mathcal{U}\,p\}$$

$$\{\neg p, \mathsf{X}\,\mathsf{G}\,\neg p, p\} \quad \{\neg p, \mathsf{X}\,\mathsf{G}\,\neg p, q, \mathsf{X}(q\,\mathcal{U}\,p)\}$$

✗

Example - unsatisfiable formula



Why three occurrences?

Consider this formula:

$$\phi \equiv p \wedge \mathsf{G}(p \longleftrightarrow \mathsf{X}\,\neg p) \wedge \mathsf{G}\,\mathsf{F}\,q_1 \wedge \mathsf{G}\,\mathsf{F}\,q_2 \wedge$$
$$\mathsf{G}\,\neg(q_1 \wedge q_2) \wedge \mathsf{G}(q_1 \longrightarrow \neg p) \wedge \mathsf{G}(q_2 \longrightarrow \neg p)$$

Why three occurrences?

Consider this formula:

$$\phi \equiv p \wedge \mathsf{G}(p \longleftrightarrow \mathsf{X}\,\neg p) \wedge \mathsf{G}\,\mathsf{F}\,q_1 \wedge \mathsf{G}\,\mathsf{F}\,q_2 \wedge$$
$$\mathsf{G}\,\neg(q_1 \wedge q_2) \wedge \mathsf{G}(q_1 \longrightarrow \neg p) \wedge \mathsf{G}(q_2 \longrightarrow \neg p)$$

And its tableau:

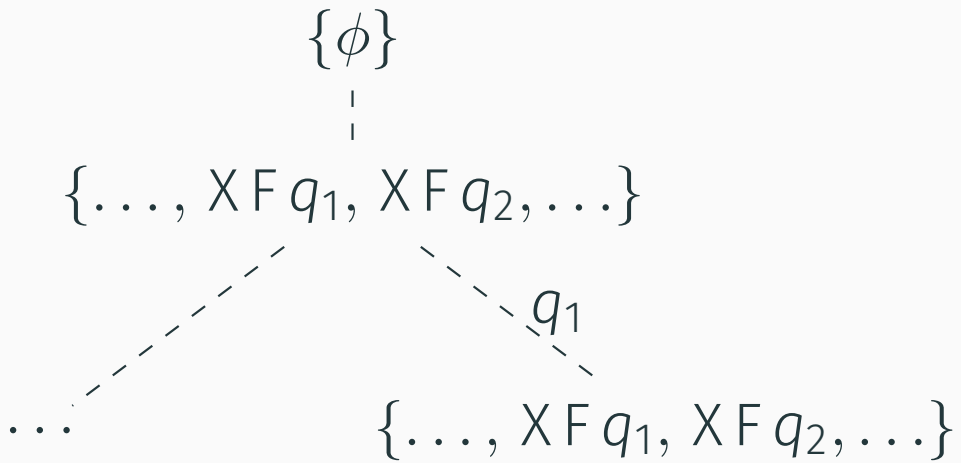
$$\{\phi\}$$

Why three occurrences?

Consider this formula:

$$\phi \equiv p \wedge \mathsf{G}(p \longleftrightarrow \mathsf{X}\neg p) \wedge \mathsf{G}\,\mathsf{F}\,q_1 \wedge \mathsf{G}\,\mathsf{F}\,q_2 \wedge$$
$$\mathsf{G}\neg(q_1 \wedge q_2) \wedge \mathsf{G}(q_1 \longrightarrow \neg p) \wedge \mathsf{G}(q_2 \longrightarrow \neg p)$$

And its tableau:

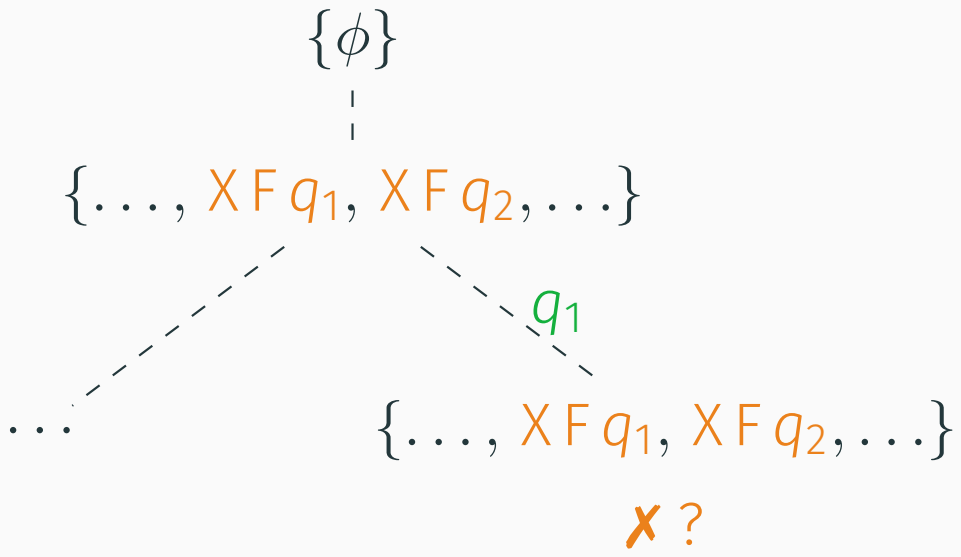


Why three occurrences?

Consider this formula:

$$\phi \equiv p \wedge \mathsf{G}(p \longleftrightarrow \mathsf{X}\neg p) \wedge \mathsf{G}\,\mathsf{F}\,q_1 \wedge \mathsf{G}\,\mathsf{F}\,q_2 \wedge$$
$$\mathsf{G}\neg(q_1 \wedge q_2) \wedge \mathsf{G}(q_1 \longrightarrow \neg p) \wedge \mathsf{G}(q_2 \longrightarrow \neg p)$$

And its tableau:

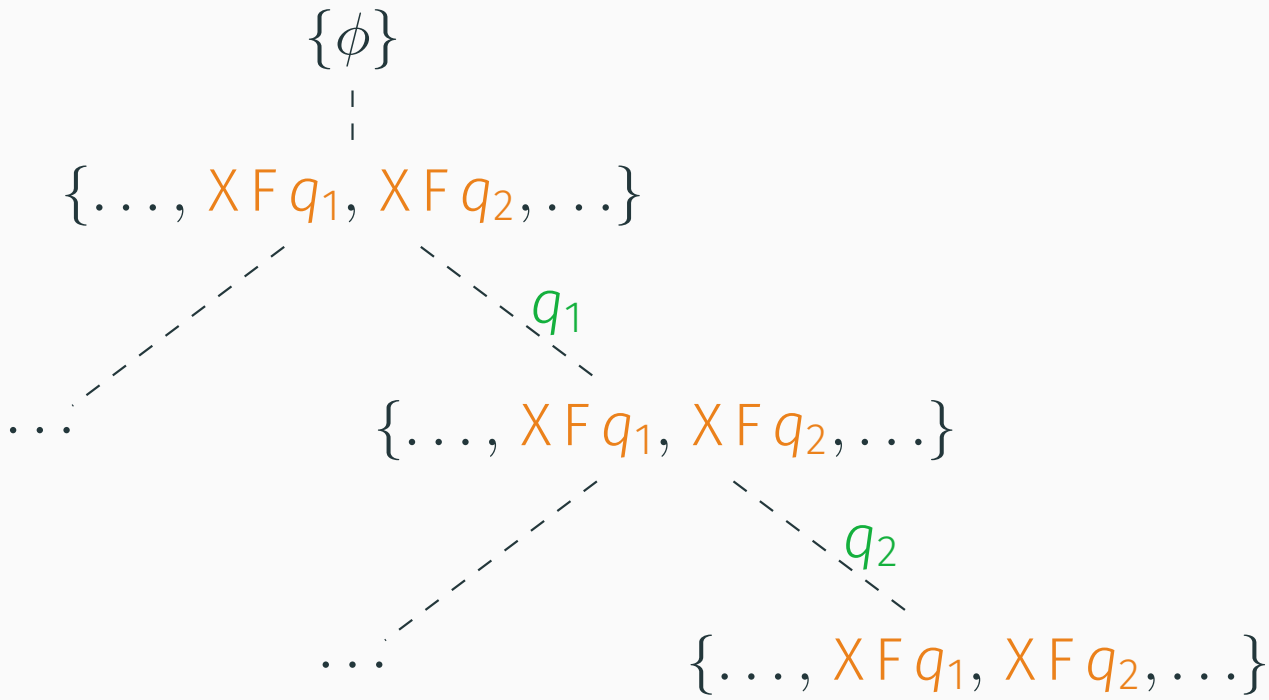


Why three occurrences?

Consider this formula:

$$\phi \equiv p \wedge \mathsf{G}(p \longleftrightarrow \mathsf{X}\neg p) \wedge \mathsf{G}\,\mathsf{F}\,q_1 \wedge \mathsf{G}\,\mathsf{F}\,q_2 \wedge$$
$$\mathsf{G}\neg(q_1 \wedge q_2) \wedge \mathsf{G}(q_1 \longrightarrow \neg p) \wedge \mathsf{G}(q_2 \longrightarrow \neg p)$$

And its tableau:

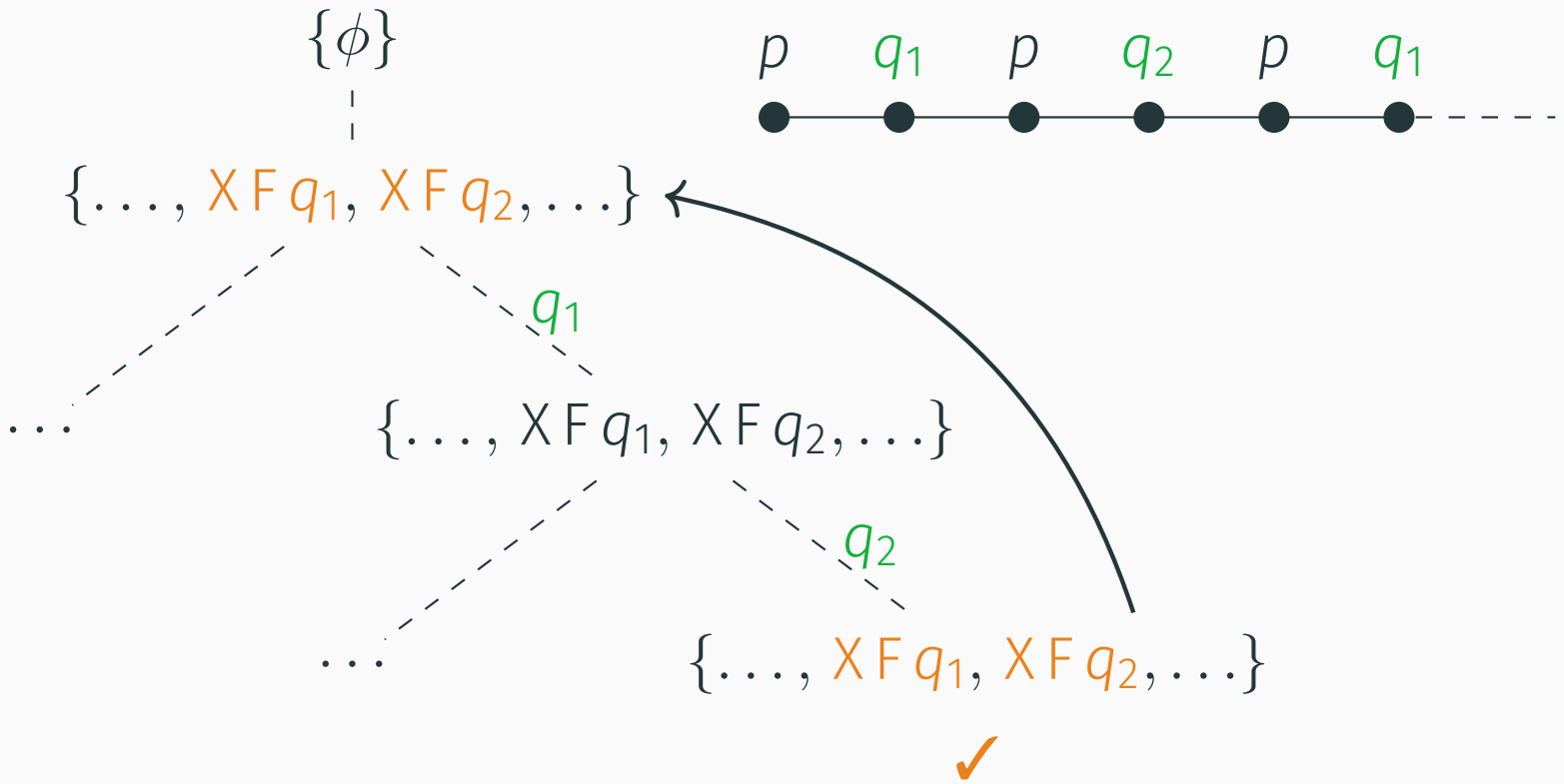


Why three occurrences?

Consider this formula:

$$\phi \equiv p \wedge \mathsf{G}(p \longleftrightarrow \mathsf{X}\neg p) \wedge \mathsf{G}\,\mathsf{F}\,q_1 \wedge \mathsf{G}\,\mathsf{F}\,q_2 \wedge$$
$$\mathsf{G}\neg(q_1 \wedge q_2) \wedge \mathsf{G}(q_1 \longrightarrow \neg p) \wedge \mathsf{G}(q_2 \longrightarrow \neg p)$$

And its tableau:



Soundness and Completeness

Soundness

The proof of correctness of the tableau is split into two parts as usual. The first is soundness:

Soundness

If the complete tableau for a formula ϕ contains an accepted branch, then the formula is satisfiable.

To prove that, we have to show that the model that we extract from the branch as shown before is indeed a correct model for the formula.

Soundness

Soundness is easy. Here we sketch the argument:

- Consider $b = \langle b_0, \ldots, b_n \rangle$ a successfull branch.
- Extend b infinitely by repeating the looping suffix, if accepted by the Loop rule, or by infinitely repeating the last state if accepted by the Empty rule.
- Let $\langle v_0, \ldots \rangle$ be the infinite sequence of all poised nodes in the looping branch.
- Let Δ_i, be the union of all the labels of the ancestors of v_i up to v_{i-1}, excluded (or up to the root for v_0).
- Note that $\Gamma(v_i) \models \Delta_i$, by construction, e.g. $\{\alpha\} \models \{\alpha \vee \beta\}$.
- Thus Δ_i is the set of formulae that will hold at the ith state of the constructed model.

Soundness

Soundness is easy. Here we sketch the argument:

- It is easy to see that each single temporal step in the model is sound, *i.e.*:
  - for each $\mathsf{X}\,\alpha \in \Gamma(v_i)$, $\alpha \in \Gamma(v_{i+1})$,
  - for each $\alpha \,\mathcal{U}\, \beta \in \Delta_i$, either $\beta \in \Delta_i$ or $\alpha \in \Delta_i$ and $\alpha \,\mathcal{U}\, \beta \in \Delta_{i+1}$,
  - for each $\mathsf{F}\,\beta \in \Delta_i$, either $\beta \in \Delta_i$ or $\mathsf{F}\,\beta \in \Delta_{i+1}$,
  - and vice versa.
- The Loop rule guarantees that the repeating part of the model fulfills all the eventualities, *i.e.*:
  - for all $\alpha \,\mathcal{U}\, \beta \in \Delta_i$, there exists $j \geq i$ such that $\beta \in \Delta_j$.
  - for all $\mathsf{F}\,\beta \in \Delta_i$, there exists $j \geq i$ such that $\beta \in \Delta_j$.

Completeness

The other direction is more involved:

Completeness

If a formula ϕ is satisfiable, then a complete tableau for ϕ contains at least a successfull branch.

Note that this holds no matter the order of application of static rules to non-poised labels.

Redundant cycles

The main part of the proof is to justify the Prune rule.

Consider a formula ϕ and a fullpath $\sigma = \langle \sigma_0, \sigma_1, \ldots \rangle$ for ϕ.

- Let Δ_i a set of formulae such that for all $\psi \in \Delta_i$
  - $\sigma_{\geq i} \models \psi$,
  - ψ of the form p or $\neg p$ for some propositional letter,
  - ψ is a subformulae of ϕ of the form $\mathsf{X}\,\alpha$.
  - ψ is of the form $\mathsf{X}\,\psi'$, where ψ' is a subformula of ϕ of the form $\alpha \,\mathcal{U}\, \beta$, $\mathsf{F}\,\beta$ or $\mathsf{G}\,\alpha$.

  Notice these are the kind of formulae that can appear in a poised label.

Redundant cycles

Definition (Redundant cycles)

Let σ_i, σ_j and σ_k, $i < j < k$, be three states of σ that satisfy the prune condition, *i.e.*:

- $\Delta_i = \Delta_j = \Delta_k$,
- Not all the eventualities requested in Δ_i were fulfilled between σ_i and σ_k,
- The set of eventualities requested in Δ_i and fulfilled between σ_j and σ_k is a subset of those fulfilled between σ_i and σ_{j-1}.

The segment between σ_{j+1} and σ_k is called a redundant cycle.

Redundant cycles

You can see this is the counterpart on the model of the triggering condition of the Prune rule.

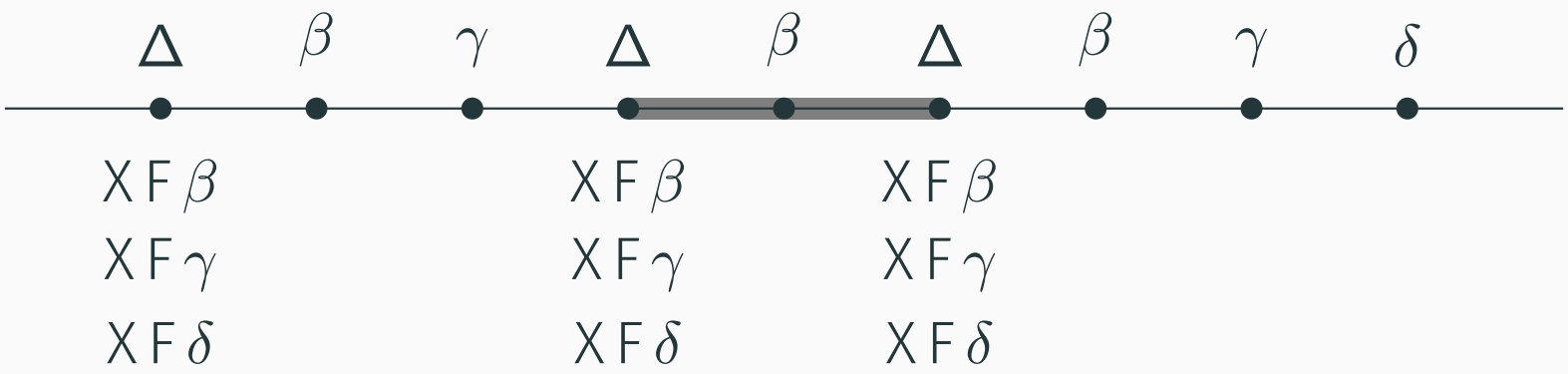
- So the Prune rule avoids models with redundant cycles.
- We have to show that we cannot lose all the models.

Theorem

If ϕ is satisfiable by a model σ, then there exists another model σ' without any redundant cycle.

Pruning of a redundant cycle

Consider a model σ with a redundant cycle:

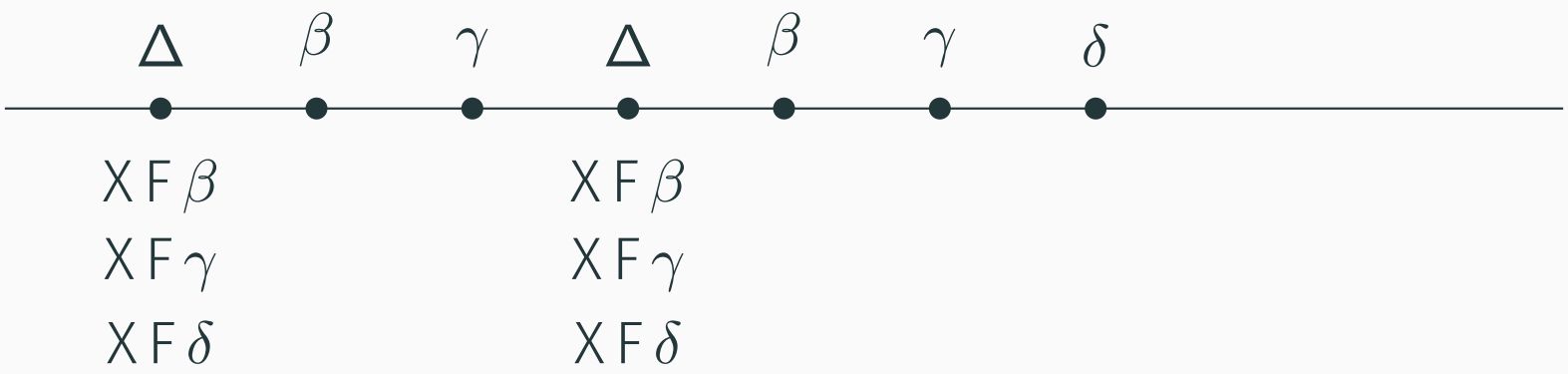


It is easy to see that pruning the redundant cycle results into another correct model.

Important fact: if an eventuality is requested in Δ_i and not fulfilled between i and $j > i$, then it is also requested at Δ_j.

Pruning of a redundant cycle

Consider a model σ with a redundant cycle:



It is easy to see that pruning the redundant cycle results into another correct model.

Important fact: if an eventuality is requested in Δ_i and not fulfilled between i and $j > i$, then it is also requested at Δ_j.

Pruning all the redundant cycles

So we may obtain the non-redundant model by pruning, one after the other, all the redundant cycles:

- We prune them in order of appearance of the final point, until there are no redundant cycle before the first repetition of the period.
- Note that a single prune preserves the satisfaction of the formula, but may push ahead the beginning of the periodic part of the model, exposing other cycles to prune.
- So we have to show that we can remove all of them in a finite number of steps.

Pruning all the redundant cycles

So suppose that there are an infinite numbr of redundant cycles in the model:

- As the number of possible labels is finite, there must be some labels that are involved in an infinite number of prunes. Call them recurring labels.
- There is a step z, after which all the non-recurring labels will not cause a prune anymore.
- So let Δ be the recurring label that causes the first prune after the zth step.
- Let σ_i,σ_j and σ_k the states involved in the prune.
- Observe that no other prunes will ever involve nodes before σ_i from now on.

Pruning all the redundant cycles

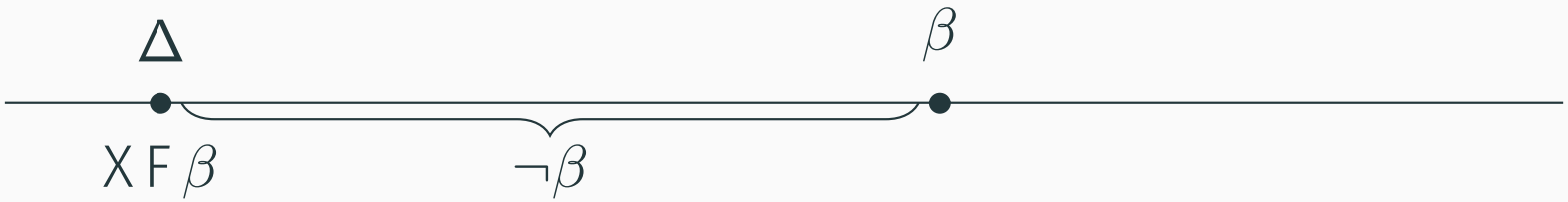
The picture shows Δ on the model

Pruning all the redundant cycles

Pruning all the redundant cycles

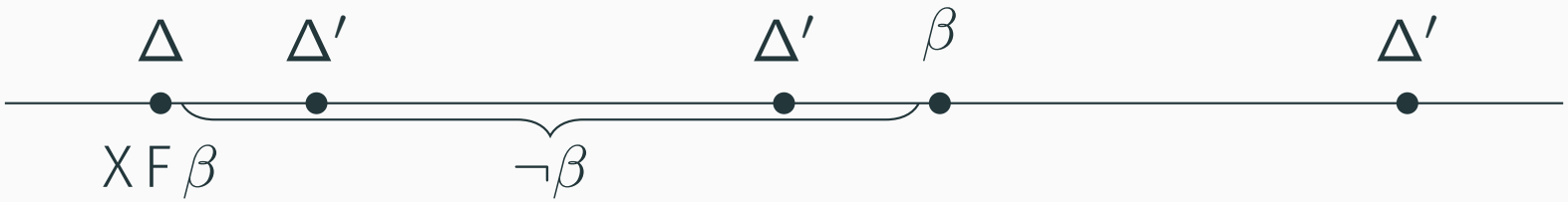
Consider an eventuality X F β requested in Δ.

Pruning all the redundant cycles



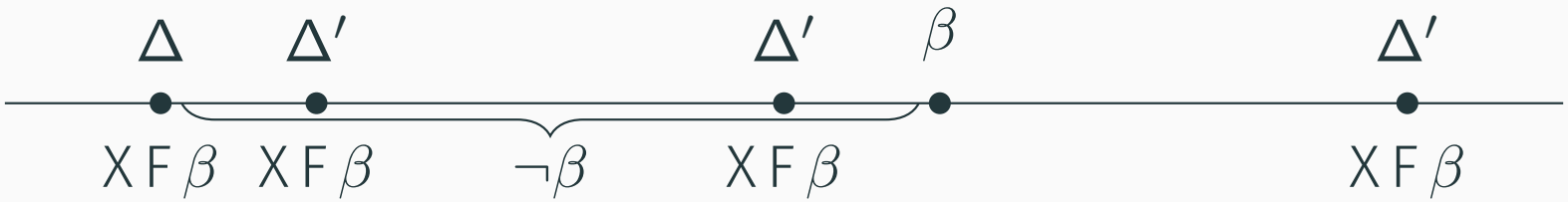
Consider an eventuality $\mathsf{X}\,\mathsf{F}\,\beta$ requested in Δ,
and its first fulfillment.

Pruning all the redundant cycles



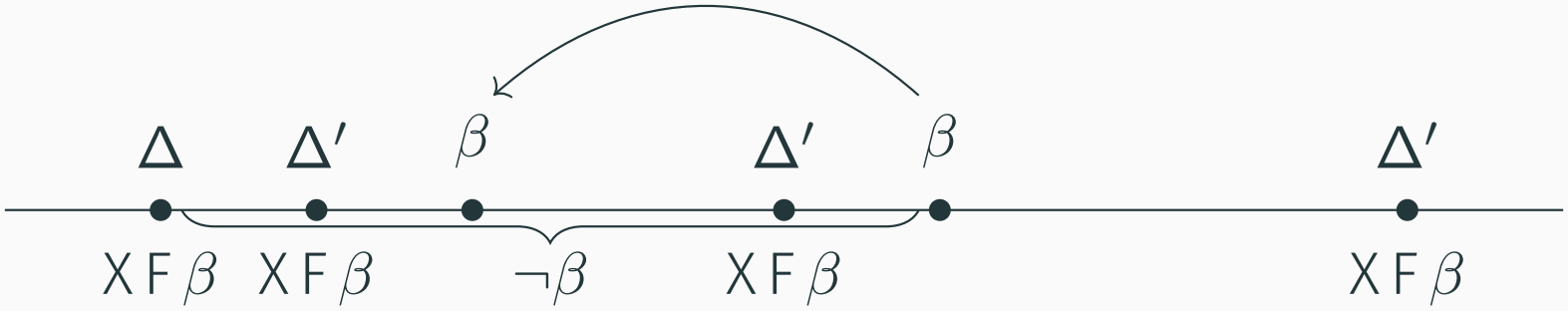
Now consider another prune happening, and suppose the redundant cycle contains the point where β is fulfilled first.

Pruning all the redundant cycles



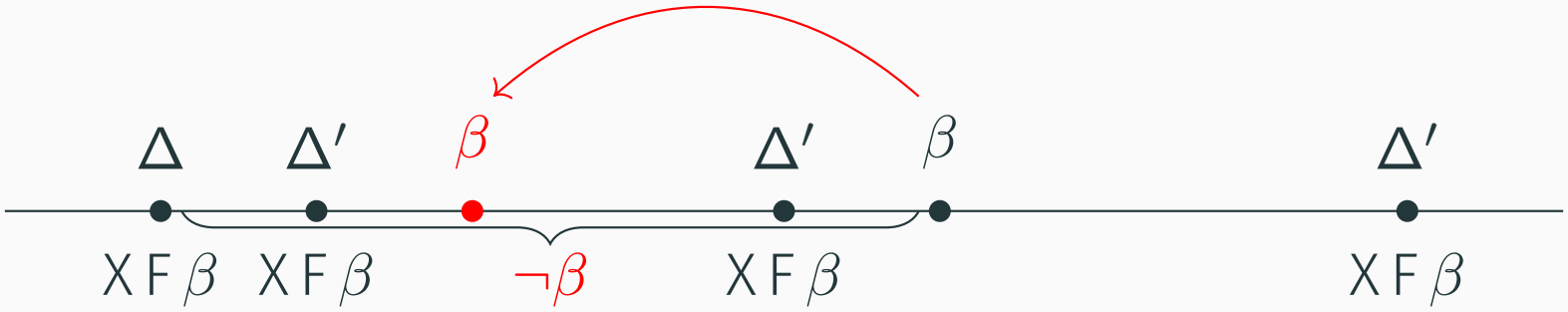
Since X F β is not fulfilled before the first occurrence of Δ', it has to be requested also by Δ'.

Pruning all the redundant cycles



Thus, β is involved in the redundant cycle, and it must appear also between the first two occurrences of Δ'

Pruning all the redundant cycles

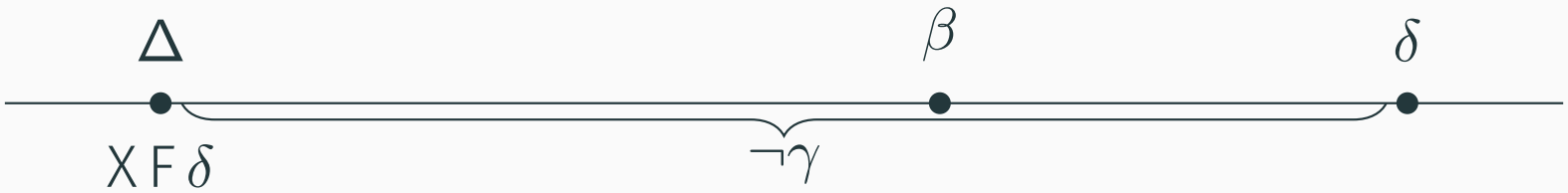


But this contraddicts the assumption that we had found the first occurrence of β.

Pruning all the redundant cycles

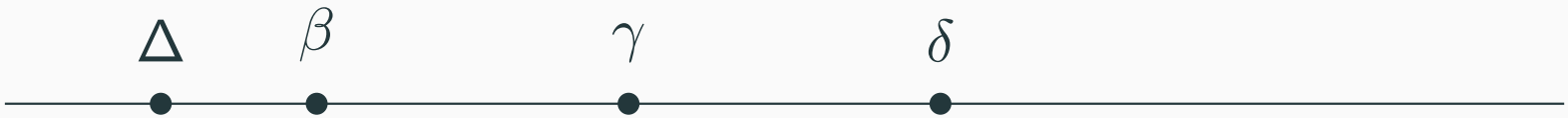
Thus, any following prune cannot remove the first witness of β from the model, and can only involve later states.

Pruning all the redundant cycles



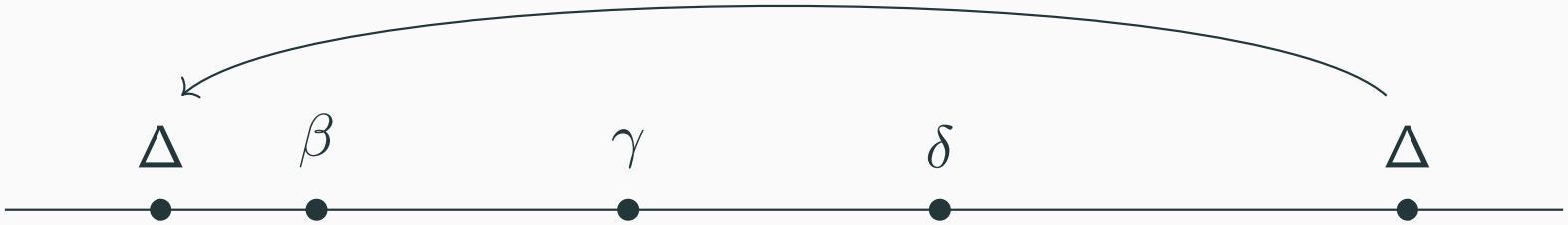
The same argument holds for the first occurrence of the next eventuality requested in Δ, and so on...

Pruning all the redundant cycles



Thus, after a finite number of steps, the prunes must have passed over the fulfillment of all the eventualities requested in Δ.

Pruning all the redundant cycles



Then, at the next repetition of Δ (there are infinite), we have found a fulfilling loop.

Finding a successful branch

Now we can finally prove the completeness.

- Consider a formula ϕ, a complete tableau T for ϕ, and a model $\sigma = \langle \sigma_0, \sigma_1, \ldots \rangle$ that satisfies ϕ.
- We proved we can restrict ourselves to a model σ without redundant cycles.
- The model is going to guide us through T to find a successful branch $b = \langle b_0, \ldots, b_n \rangle$, *i.e.*, with b_n a ticked leaf.

Finding a successful branch

So we build the branch starting from the root b_0, and the first state of the model σ_0.

- We keep a mapping $J(i)$ from our position i in the tree to our current position in the model. So at the start, $J(0) = 0$.
- We keep the invariant that for all $\alpha \in \Gamma(b_i)$, $\sigma_{\geq J(i)} \models \alpha$.
- At the start the invariant holds since σ satisfies ϕ and the root node is labelled $\{\phi\}$.

Finding a successful branch

At each step, from node b_i we find the successor b_{i+1}:

- If b_i is ticked we found the successful branch and we stop.
- Depending on which rule was applied to b_i during the construction, we chose one of its children:
  - E.g. if to b_i was applied the rule for disjunctions to $\alpha \vee \beta$, then if $\sigma_{\geq J(i)} \models \alpha$ we chose the left child, if $\sigma_{\geq J(i)} \models \beta$ we chose the right one.
  - One of the two must be the case because $\sigma_{\geq J(i)} \models \alpha \vee \beta$ by our invariant.
  - If to b_i was applied the Step rule, we move to the child and we step also in the model, *i.e.*, $J(i+1) = i+1$.

Finding a successful branch

Following this procedure we will always eventually find a ticked branch, because:

- We cannot find a node that was crossed by contradiction, because then our invariant would imply a contradiction in the model.
- We cannot find a node that was crossed by the Prune rule, because then our invariant would imply that there were a redundant cycle in the model.
- We cannot go on forever because at each step we move to a child, and the tree is finite.

Conclusions

Future Works

The simplicity of the tableau rules allow to easily reason about possible improvements:

- Search heuristics
- Parallelism
- Extension to other logics (past operators, metric extensions, forgettable past, freeze quantifiers, etc...)

Thank you!

Questions?

Bibliography

[Ber+16] M. Bertello et al. “Leviathan: A New LTL Satisfiability Checking Tool Based on a One-Pass Tree-Shaped Tableau.” In: *Proc. of the 25th Int. Joint Conference on Artificial Intelligence*. 2016.

[LP00] Orna Lichtenstein and Amir Pnueli. “Propositional Temporal Logics: Decidability and Completeness.” In: *Logic Journal of the IGPL* 8.1 (2000), pp. 55–85.

[Rey16] M. Reynolds. “A new rule for LTL tableaux.” In: *Proc. of the 7th Int. Symposium on Games, Automata, Logics and Formal Verification*. 2016.

[Sch98] S. Schwendimann. “A New One-Pass Tableau Calculus for PLTL.” In: *Proc. of the 4th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*. 1998, pp. 277–292.