A tree-shaped tableau for Linear Time Temporal Logic

Mark Reynolds, The University of Western Australia

September 2025

A tree-shaped tableau for Linear Time Temporal Logic

Propositional linear time temporal logic (LTL) is the standard temporal logic for computing applications and many reasoning techniques and tools have been developed for it. Tableaux for deciding satisfiability have existed since the 1980s. However, the tableaux for this logic do not look like traditional tree-shaped tableau systems and their processing is often quite complicated. In this talk we describe a novel style of tableau rule which supports a new simple traditional-style tree-shaped tableau for LTL. We outline the proof that it is sound and complete. As well as being simple to understand, to introduce to students and to use, it is also simple to implement and is competitive against state of the art systems. It is particularly suitable for parallel implementations.

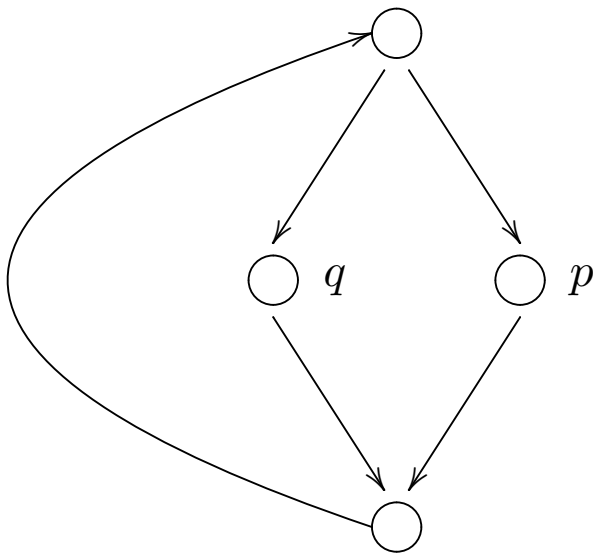
Outline

- The Logic LTL
- A tableau for LTL-Sat
- Soundness
- Completeness

Model of a System:

Thus we suppose that we can model the system as a finite state machine: a set of states and a set of allowed transitions from some states to other states.

In order to allow abstraction of observable basic, or atomic properties, we also suppose that there are a set of atomic properties, and that some properties are true at some states.



Properties:

Examples.
If a process stays in a waiting state then it will eventually be given a resource.
The program will eventually terminate.
The program will never return a null value.
The two processes will never both be in their critical states at the same time.
The program will never be in a logged-in state unless a correct PIN has been entered beforehand.
The system will always dispense a product after the correct money has been inserted.

Structures:

We assume a countable set $\mathcal{L}$ of propositional atoms, or atomic propositions.

A transition structure is a triple (S, R, g) with S a finite set of states, $R \subseteq S \times S$ a binary relation and for each $s \in S$, $g(s) \subseteq \mathcal{L}$.

R is the *transition relation* and *labelling* g tells us which atoms are true at each time.

R is assumed to be total: every state has at least one successor $\forall x \in S. \exists y \in S$ s.t.$(x, y) \in R$

Fullpaths:

Given a structure (S, R, g).
An ω-sequence of states $\langle s_0, s_1, s_2, ... \rangle$ from S is a fullpath (through (S, R, g)) iff for each i, $(s_i, s_{i+1}) \in R$.

If $\sigma = \langle s_0, s_1, s_2, ... \rangle$ is a fullpath then we write $\sigma_i = s_i$, $\sigma_{\geq j} = \langle s_j, s_{j+1}, s_{j+2}, ... \rangle$ (also a fullpath).

LTL Syntax:

Define some strings of symbols as (well formed) formulas, or wffs of LTL.

If $p \in \mathcal{L}$ then p is a wff.

If α and β are wff then so are $\neg\alpha$, $\alpha \wedge \beta$, $X\alpha$, and $\alpha U \beta$.

Read: Not, and(conjunction), tomorrow (or next), and until.

LTL Semantics:

Write $M, \sigma \models \alpha$ iff the formula α is true of the fullpath σ in the structure $M = (S, R, g)$ defined recursively by:

$M, \sigma \models p$ iff $p \in g(\sigma_0)$, for $p \in \mathcal{L}$

$M, \sigma \models \neg\alpha$ iff $M, \sigma \not\models \alpha$

$M, \sigma \models \alpha \wedge \beta$ iff $M, \sigma \models \alpha$ and $M, \sigma \models \beta$

$M, \sigma \models X\alpha$ iff $M, \sigma_{\geq 1} \models \alpha$

$M, \sigma \models \alpha U \beta$ iff there is some $i \geq 0$ such that $M, \sigma_{\geq i} \models \beta$ and for each j, if $0 \leq j < i$ then $M, \sigma_{\geq j} \models \alpha$

Abbreviations:

Classical: $\top \equiv p \vee \neg p$, $\bot \equiv \neg\top$, $\alpha \vee \beta \equiv \neg(\neg\alpha \wedge \neg\beta)$,
$\alpha \to \beta \equiv \neg\alpha \vee \beta$, $\alpha \leftrightarrow \beta \equiv (\alpha \to \beta) \wedge (\beta \to \alpha)$.
Read: truth, falsity, or(disjunction), implication, iff(equivalence).

Temporal: $F\alpha \equiv (\top U \alpha)$, $G\alpha \equiv \neg F(\neg\alpha)$.
Read: eventually, always.

Example Properties:

$G\neg(p \wedge q)$
$G(p \rightarrow (Xq \vee XXq \vee XXXq))$
$G(p \rightarrow Fq)$
$G(Gp \rightarrow Fq)$
$GFp \wedge GFq$
$G(t \rightarrow Gt)$
$Fp \rightarrow ((\neg p)U(q \wedge \neg p))$
$FGp \vee GFq$

LTL satisfiability

We start a detailed look at an algorithm to decide the satisfiability of LTL formulas.

We want to invent an algorithm which solves the LTL-SAT problem. Input should be a formula from LTL. Output is “yes” or “no” depending on whether the formula is satisfiable or not.

Satisfiability:

A formula α is satisfiable iff there is some structure (S, R, g) with some fullpath σ through it such that $(S, R, g), \sigma \models \alpha$.

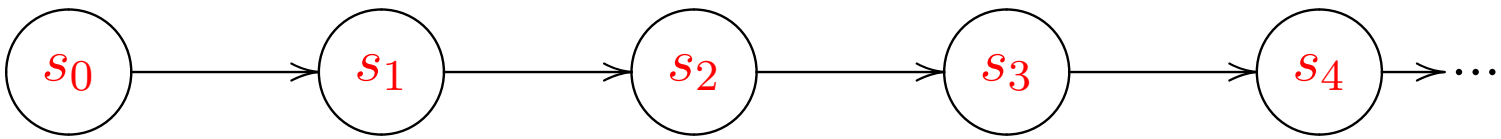
Eg, $\top$, p, Fp, $p \wedge Xp \wedge F\neg p$, Gp are each satisfiable.

Eg, $\bot$, $p \wedge \neg p$, $Fp \wedge G\neg p$, $p \wedge G(p \rightarrow Xp) \wedge F\neg p$ are each not satisfiable.

Can we invent an algorithm for deciding whether an input formula (of LTL) is satisfiable or not?

Build a model:

To test satisfiability of a formula, what about trying to build a model of it?
Eg, suppose that we ask about the satisfiability of
$\neg p \wedge X\neg p \wedge (qUp)$

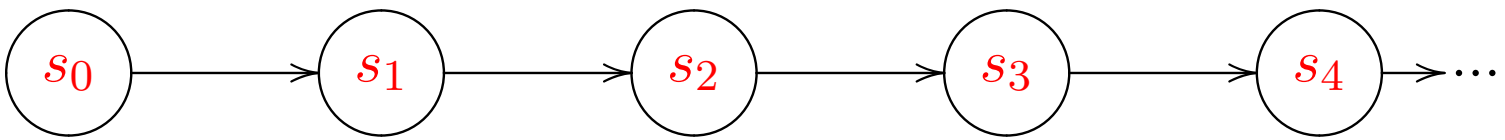


Let's make $\neg p \wedge X\neg p \wedge (qUp)$ true at s_0.

By propositional reasoning we need to make $\neg p$, $X\neg p$ and qUp true at s_0 as well.

Build a model of $\neg p \wedge X\neg p \wedge (qUp)$:

So $\{\neg p \wedge X\neg p \wedge (qUp), \neg p, X\neg p, qUp\}$ are to hold at s_0.



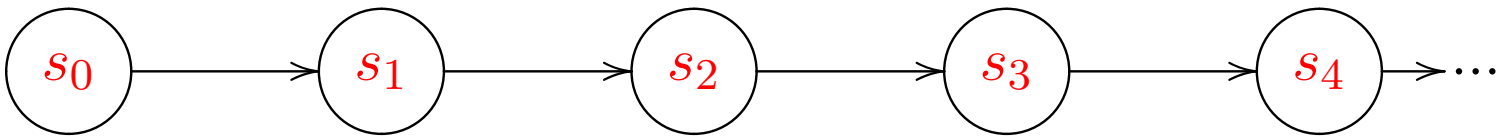
Easy to make $\neg p$ hold there.
Easy to see that we should also make $\neg p$ true at s_1.
But what about qUp?

qUp:

There are two alternative ways to make $\alpha U \beta$ true at a state s.
You can make β true there.
OR
You can make α true there and make $\alpha U \beta$ true at a next state.

In our case, with qUp we can not do the former in s_0 so we need to make q true at s_0 and postpone qUp until s_1.
And again at s_1 we have to make q true there and postpone qUp until s_2.
At s_2 we can use the first case.
Thus we show the formula is satisfiable and we have built a model.

From tableau labels to labelling:



$\{\neg p \wedge X\neg p \wedge (qUp), \neg p, X\neg p, qUp, q\}$ are to hold at s_0.
$\{\neg p, qUp, q\}$ are to hold at s_1.
$\{qUp, p\}$ are to hold at s_2.

Answer: $\{q\}, \{q\}, \{p\}, \{\}, \{\}, ...$

Can this be generalised?:

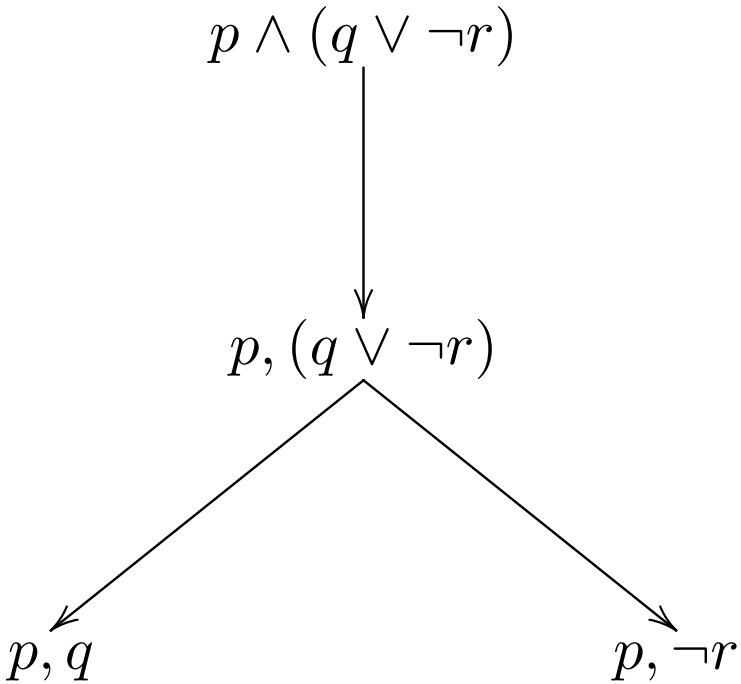
Labelling nodes with formulas is good. Starting from s_0 and working forwards in time is good.

However, some problems:
How to deal with choices (that are not immediately obvious).
What if we need to go on forever building the model?
What if we go on forever making something that is not going to be a model?

The following is my newish tree-shaped tableau for LTL. It builds on work by Sistla and Clarke (1985) and is influenced by LTL tableaux by Wolper (1982) and Schwendimann (1998). There's a GANDALF paper Reynolds (2016) and the idea is also described with an implementation in an IJCAI 2016 paper (BGM 2016).

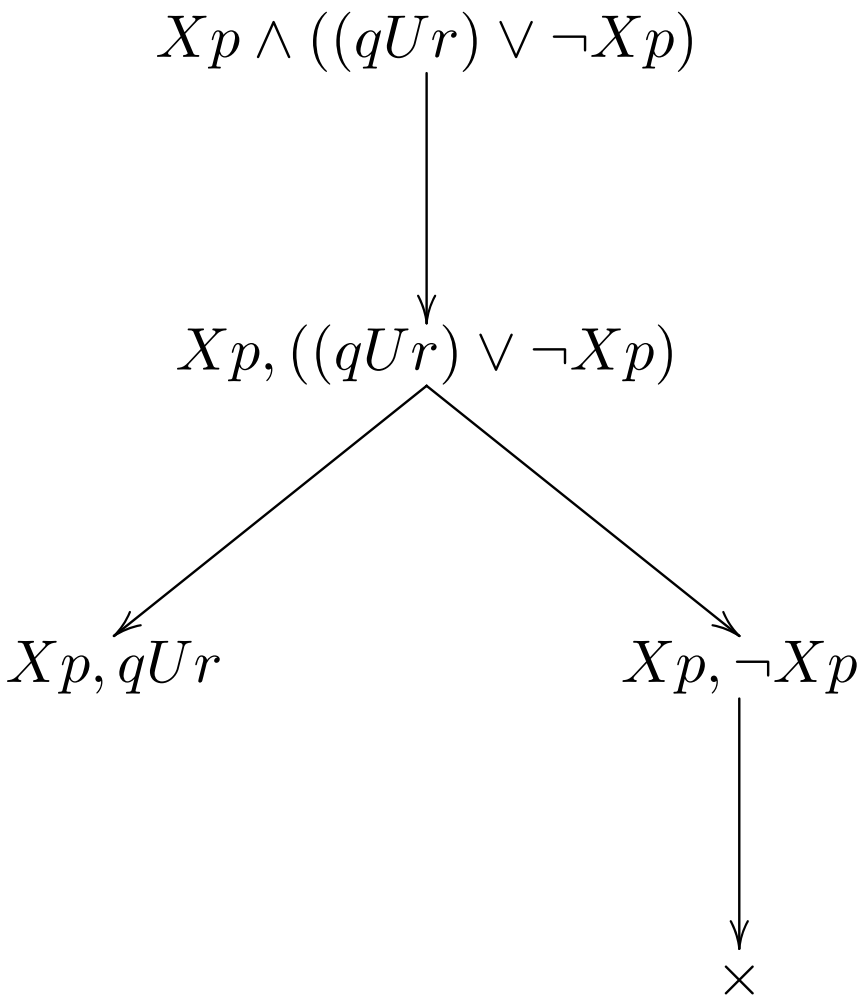
Reminder of Tableau for classical propositional logic:

Possibilities branch into a tree as we work down the page ...



Same rules for LTL:

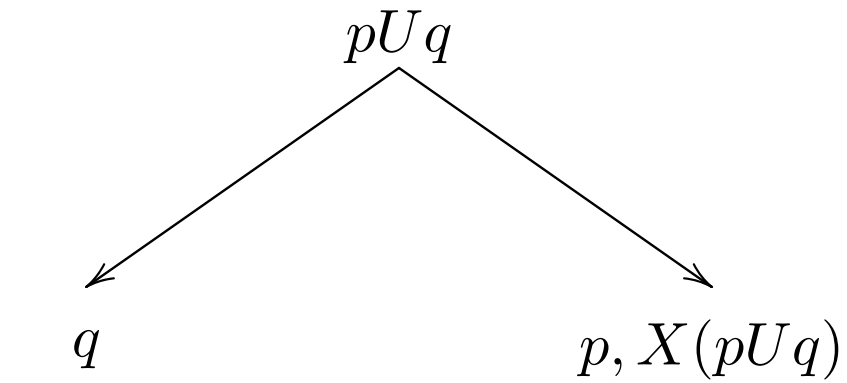
Same things can happen for LTL within a state ...

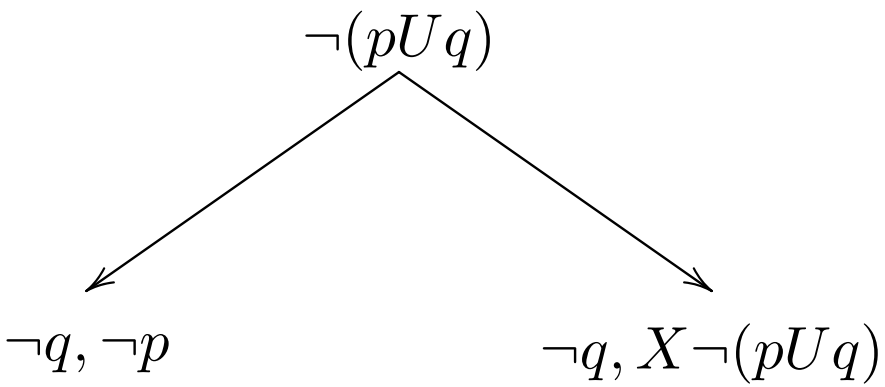


Rules:

[EMP]: If a node is labelled $\{\}$ then this node can be *ticked*.
[X]: If a node is labelled Γ with some α and $\neg\alpha$ in Γ then this node can be *crossed*.
[DNEG]: If a node is labelled $\Gamma \cup \{\neg\neg\alpha\}$ then this node can have one child labelled $\Gamma \cup \{\alpha\}$).
[CON]: If a node is labelled $\Gamma \cup \{\alpha \wedge \beta\}$ in Γ then this node can have one child labelled $\Gamma \cup \{\alpha, \beta\}$.
[DIS]: If a node is labelled $\Gamma \cup \{\neg(\alpha \wedge \beta)\}$ in Γ then this node can have two children labelled $\Gamma \cup \{\neg\alpha\}$ and $\Gamma \cup \{\neg\beta\}$ respectively.
(plus similar for other abbreviations and their negations)

Until also gives us choices:





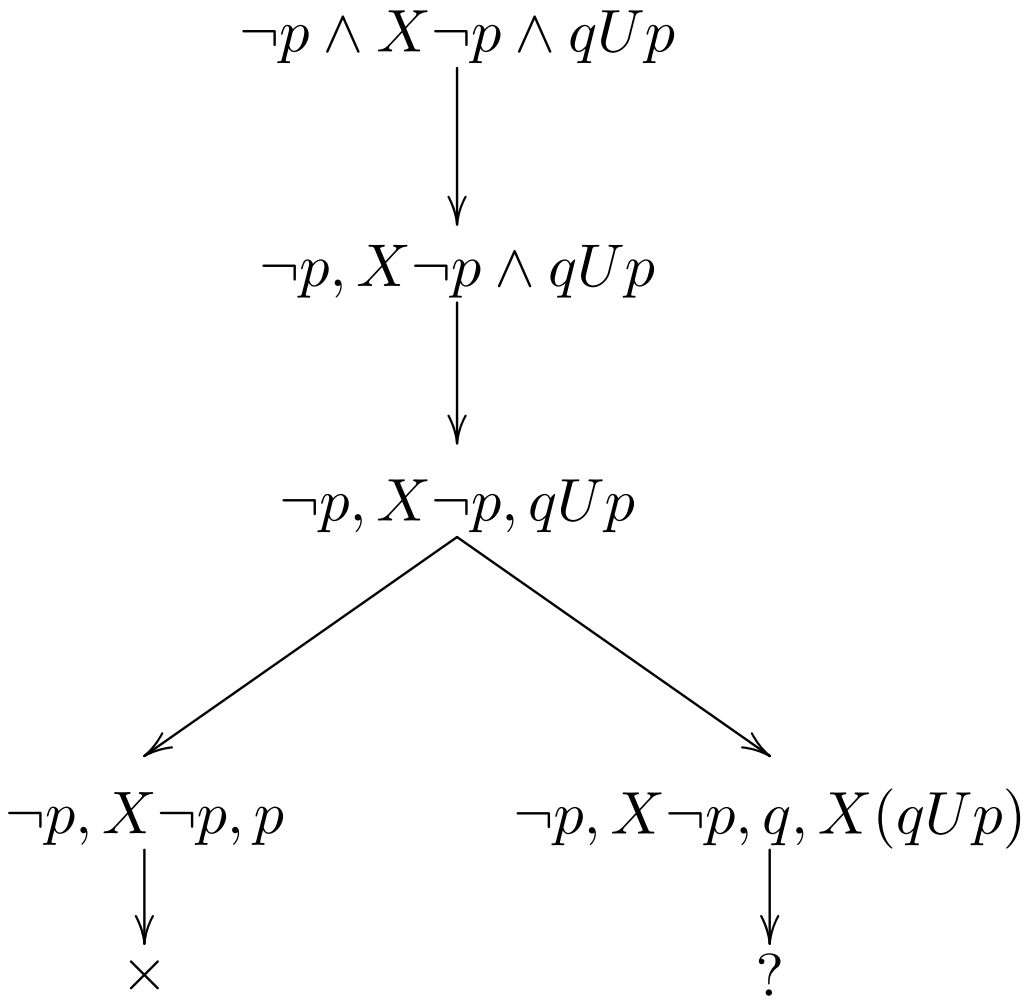
Rules for Until:

[UNT]: If a node is labelled $\Gamma \uplus \{\alpha U \beta\}$ in Γ then this node can have two children labelled $\Gamma \cup \{\alpha, X(\alpha U \beta)\}$ and $\Gamma \cup \{\beta\}$.

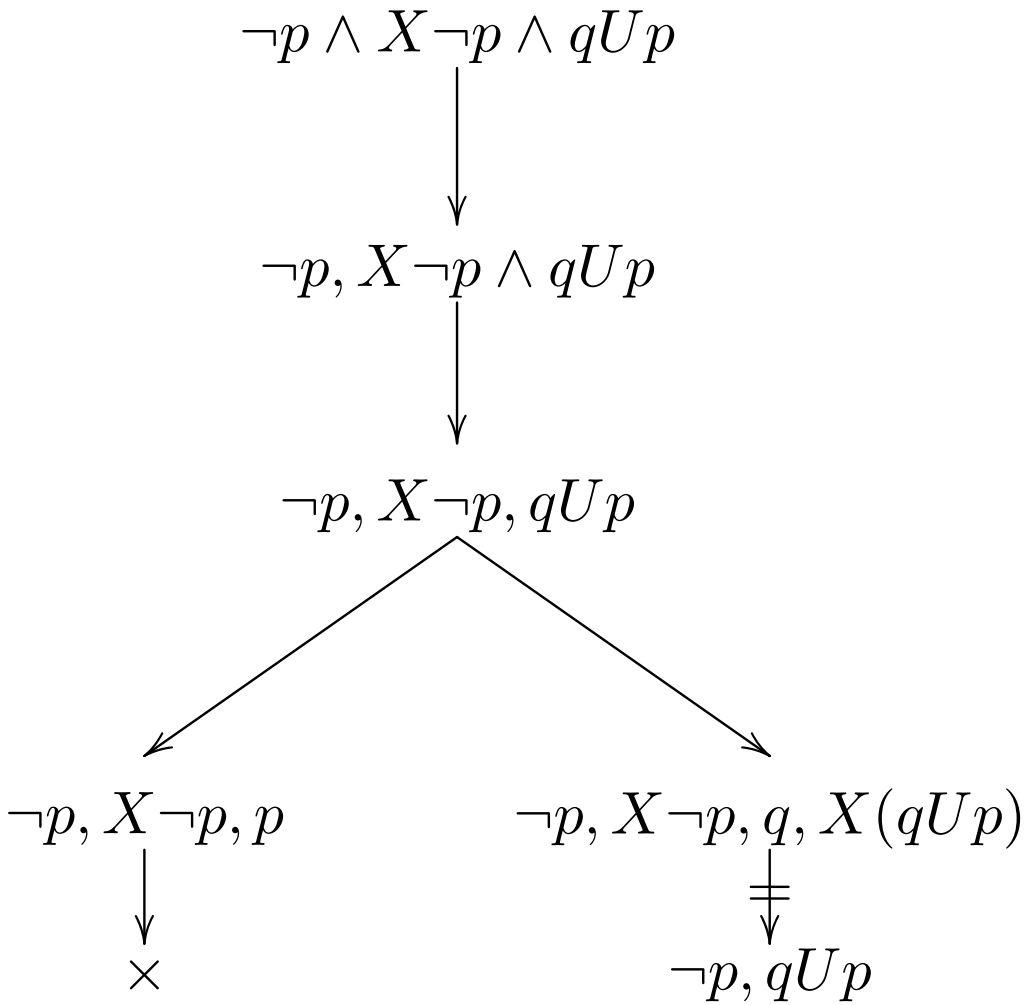
[NUN]: If a node is labelled $\Gamma \uplus \{\neg(\alpha U \beta)\}$ in Γ then this node can have two children labelled $\Gamma \cup \{\neg\beta, X\neg(\alpha U \beta)\}$ and $\Gamma \cup \{\neg\alpha, \neg\beta\}$.

(plus similar for F and G.)

But what to do when we want to move forwards in time?:



Introduce a new type of step:



Step Children:

If none of the above (static) rules are applicable to a node then we say that the node is *propositionally complete* and then, and only then, is the following rule applicable.

[TRANSITION]: The node, labelled by propositionally complete Γ say, can have one child, called a *step-child*, whose label Δ is defined as follows.
$\Delta = \{\alpha | X\alpha \in \Gamma\} \cup \{\neg\alpha | \neg X\alpha \in \Gamma\}$.

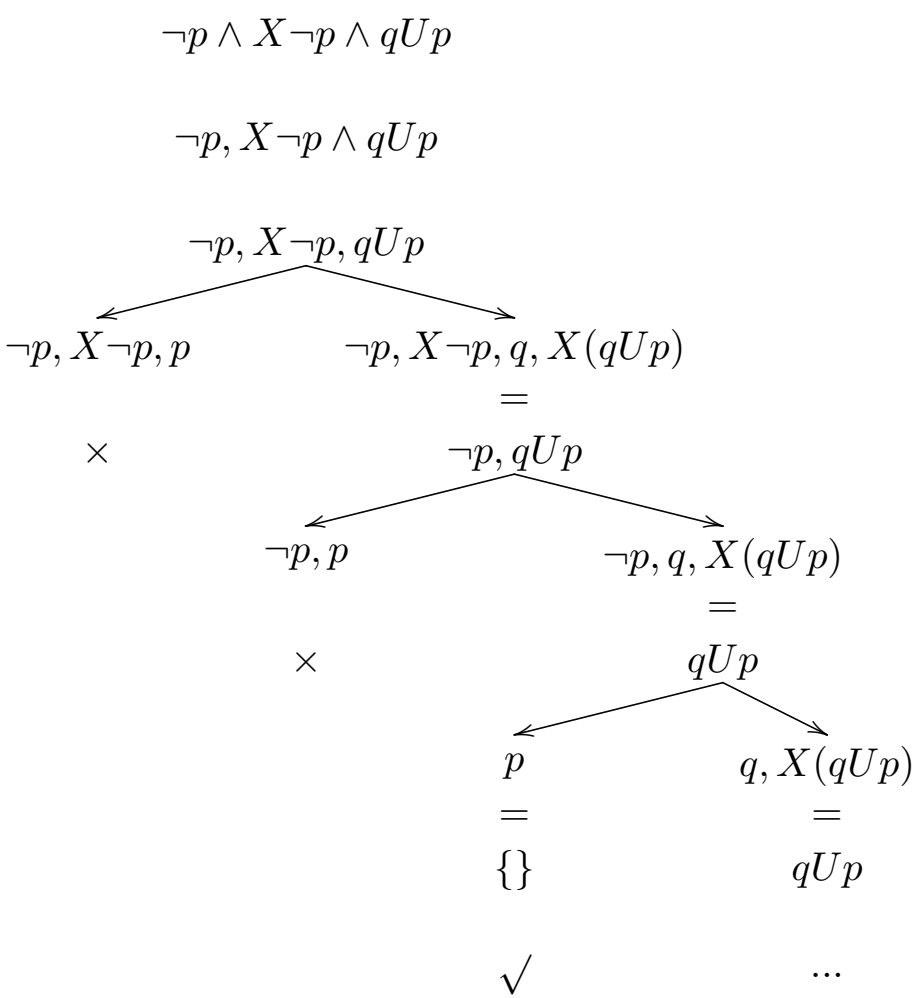
Note that Δ may be empty. After a step rule the try to use the static rules again.

Exercise:

Can now do the whole $\neg p \wedge X \neg p \wedge (qUp)$ example.

Try it.

Exercise answer: $\neg p \wedge X\neg p \wedge (qUp)$ example.



Infinite behaviour:

Still some work to do.

We will try these examples in the next few slides ...

Gp

$G(p \wedge q) \wedge F\neg p$

$p \wedge G(p \leftrightarrow X\neg p) \wedge G(q \rightarrow \neg p) \wedge GF\neg q \wedge GF\neg p$

$p \wedge G(p \rightarrow Xp) \wedge F\neg p$

Example: Gp

Gp gives rise to a very repetitive infinite tableau.

$$Gp$$

$$p, XGp$$
$$=$$
$$Gp$$

$$p, XGp$$
$$=$$
$$Gp$$

$$p, XGp$$

$$\ldots$$

Notice that the infinite fullpath that it suggests is a model for Gp as would a fullpath just consisting of the one state with a self-loop (a transition from itself to itself).

Example: $G(p \wedge q) \wedge F\neg p$

But $G(p \wedge q) \wedge F\neg p$ shows that we can not just accept infinite loops as demonstrating satisfiability....

$$G(p \wedge q) \wedge F\neg p$$

$$G(p \wedge q), F\neg p$$

$$p \wedge q, XG(p \wedge q), F\neg p$$

$$p, q, XG(p \wedge q), F\neg p$$

$$p, q, XG(p \wedge q), \neg p \qquad p, q, XG(p \wedge q), XF\neg p$$

$$\times \qquad =$$

$$G(p \wedge q), F\neg p$$

$$\dots$$

$$XG(p \wedge q), XF\neg p$$

$$=$$

$$G(p \wedge q), F\neg p$$

$$\dots$$

$$XG(p \wedge q), XF\neg p$$

$G(p \wedge q) \wedge F\neg p$ continued

Notice that the infinite fullpath that the tableau suggests is this time not a model for $G(p \wedge q) \wedge F\neg p$.

Constant repeating of p, q being made true does not satisfy the conjunct $F\neg p$.

We have postponed the *eventuality* forever.

This is not acceptable.

Eventualities:

An *eventuality* is just a formula of the form $\alpha U \beta$.

(This includes $F\gamma \equiv \top U \gamma$).

If $\alpha U \beta$ appears in the tableau label of a node u then we want β to appear in the label of some later (or equal node) v. In that case we say that the eventuality is *satisfied* by v.

Eventualities are eventually satisfied in any (actual) model of a formula: by the semantics of until.

Loops:

If a label is repeated along a branch and all eventualities are satisfied in between then we can build a model by looping states. In fact, the ancestor can have a superset and it will work.

[LOOP]: If a node n has a proper ancestor (i.e. not itself) m such that $\Gamma(m) \supseteq \Gamma(n)$, m has a STEP-child, and all eventualities in $\Gamma(m)$ are satisfied by labels between m and n (including m itself) then n can be ticked.

Nice example to try:

$$p \wedge G(p \leftrightarrow X\neg p) \wedge G(q \rightarrow \neg p) \wedge G(r \rightarrow \neg p) \wedge G(q \rightarrow \neg r) \wedge GFq \wedge GFr$$

Are we done yet?

No.

Examples like $G(p \wedge q) \wedge F\neg p$ may have branches that go on forever without a tick. We need to stop and fail branches so that we can answer "no" correctly and terminate and so that we do not get distracted when another branch may be successful. In fact, no infinite branches should be allowed.

Try also: $p \wedge G(p \rightarrow Xp) \wedge F\neg p$

Can't we see that these infinite branches are just getting repetitive without making a model?

Closure set:

As we construct the tableau model we only need to record, in the labels, which formulas we want to be true at that time from a finite set of interesting formulas.
The *closure set* for a formula ϕ is as follows:

$$\{\psi, \neg\psi | \psi \leq \phi\} \cup \{X(\alpha U \beta), \neg X(\alpha U \beta) | \alpha U \beta \leq \phi\}$$

(Where $\psi \leq \phi$ means that ψ is a subformula of ϕ.)

Size of closure set is $\leq 4n$ where n is the length of the initial formula.

This shows that only formulas from a finite set will appear in labels.

...and only $\leq 2^{4n}$ possible labels.

Don't go on further than you need to:

This gives us the idea of *useless* intervals on branches in the tableau.

If a node at the end of a branch (of a partially complete tableau) has a label which has appeared already twice above, and between the second and third appearance there are no new eventualities satisfied then that whole interval of states has been useless!

The **REP**etition rule:

[REP]:
(later called PRUNE rule)
Suppose that $u = u_0, u_1, ..., u_{j-1}, u_j = v, u_{j+1}, ..., u_k = w$ is a sequence of consecutive descendants in order.
Suppose that $\Gamma(u) = \Gamma(v) = \Gamma(w)$, u and v have STEP-children and w is propositionally complete.
Suppose also that for all eventualities $\alpha U \beta \in \Gamma(u)$, if β is satisfied between v and w then β is satisfied between u and v anyway.
Then w can be crossed.

(We assume that there are some unsatisfied eventualities from $\Gamma(u)$ left. Otherwise you should have used the LOOP rule earlier to tick the branch.)

Examples:

Now try $G(p \wedge q) \wedge F\neg p$ and $p \wedge G(p \rightarrow Xp) \wedge F\neg p$.

LTL Tableau Summary:

Is a finite tree of nodes labelled by subsets of the closure set of ϕ such that:

- the root is labelled with $\{\phi\}$
- the labels of children of each node are according to one of the tableau rules

Successful if some leaf is ticked.

Failed if all leaves are crossed.

(Not really a proper description of an algorithm but we will see that the further details of which formula to consider at each step in building the tableau are unimportant).

Proof of Correctness:

This will consist of three parts.

Proof of soundness. If a formula has a successful tableau then it has a model.

Proof of completeness: If a formula has a model then building a tableau will be successful.

Proof of termination. Show that the tableau building algorithm will always terminate.

Part One:, the Proof of Termination:

(Sketch only)

Any reasonable tableau search algorithm will always terminate because there can be no infinitely long branches.

We know this because the REP rule will cross any that go on too long.

Thus there will either be at least one tick or all crosses.

Termination is also why we require that other rules are not used repeatedly in between STEP rules.

Part Two: Proof of Soundness:

(Overview)
Use a successful tableau to make a model of the formula, thus showing that it is satisfiable.

Use a successful branch. Each STEP tells us that we are moving from one state to the next.

Within a particular state we can make all the formulas listed true there (as evaluated along the rest of the fullpath). Atomic propositions listed tell us that they are true at that state.

An induction deals with most of the rest of the formulas.

Eventualities either get satisfied and disappear in a terminating branch or have to be satisfied if the branch is ticked by the LOOP rule.

PART 3: Proof of Completeness:

We have to show that if a formula has a model then it has a successful tableau.

This time we will use the model to find the tableau.

Proof of Completeness:

The basic idea is to use a model (of the satisfiable formula) to show that *in any tableau* there will be a branch (i.e. a leaf) with a tick.

A weaker result is to show that there is some tableau with a leaf with a tick.

Such a weaker result may actually be ok to establish correctness and complexity of the tableau technique.
However, it raises questions about whether a "no" answer from a tableau is correct and it does not give clear guidance for the implementer.

Completeness Proof:

Suppose that ϕ is a satisfiable formula of LTL.
It will have a model. Choose one, say $(S, R, g), \sigma \models \phi$. In what follows we (use standard practice when the model is fixed and) write $\sigma_{\geq i} \models \alpha$ when we mean $(S, R, g), \sigma_{\geq i} \models \alpha$.
Also, build a tableau T for ϕ in any manner as long as the rules are followed. Let $\Gamma(x)$ be the formula set label on the node x in T.
We will show that T has a ticked leaf.
To do this we first construct a sequence $x_0, x_1, x_2, \ldots.$ of nodes, with x_0 being the root. This sequence may terminate at a tick (and then we have succeeded) or it may hypothetically go on forever (and more on that later).
In general the sequence will head downwards from a parent to a child node but occasionally it may jump back up to an ancestor.

Invariant

As we go we will also make sure that each node x_i is associated with a state $\sigma_{j(i)}$ in S.
We will ensure that for each i, for each $\alpha \in \Gamma(x_i)$, $\sigma_{\geq j(i)} \models \alpha$.
Start by putting $j(0) = 0$ when x_0 is the tableau root node.
Note that the only formula in $\Gamma(x_0)$ is ϕ and that $\sigma_{\geq 0} \models \phi$.
Good start.

Now suppose that we have identified the x sequence up until x_i.

Consider the rule that is used in T to extend a tableau branch from x_i to some children.

[EMP] If $\Gamma(x_i) = \{\}$ then we are done. T is a successful tableau as required.

[X] Consider if it is possible for the branch to stop at x_i with a cross because of a contradiction. So there is some α with α and $\neg\alpha$ in $\Gamma(x_i)$. But this can not happen as then $\sigma_{\geq j(i)} \models \alpha$ and $\sigma_{\geq j(i)} \models \neg\alpha$.

[DNEG]

So $\neg\neg\alpha$ is in $\Gamma(x_i)$ and there is one child, which we will make $x(i+1)$ and we will put $j(i+1) = j(i)$. Because $\sigma_{\geq j(i)} \models \neg\neg\alpha$ we also have $\sigma_{\geq j(i+1)} \models \alpha$. Also for every other $\beta \in \Gamma(x_{i+1}) = \Gamma(x_i) \cup \{\alpha\}$, we still have $\sigma_{\geq j(i+1)} \models \beta$. So we have the invariant holding.

(CON, DIS etc are similar)

[UNT]

So $\alpha U \beta$ is in $\Gamma(x_i)$ and there are two children. One y is labelled $\Gamma(x_i) \cup \{\beta\}$ and the other, z, is labelled $\Gamma(x_i) \cup \{\alpha, X(\alpha U \beta)\}$.
We know $\sigma_{\geq j(i)} \models \alpha U \beta$. Thus, there is some $k \geq j(i)$ such that $\sigma_{\geq k} \models \beta$ and for all l, if $0 \leq l < k$ then $\sigma_{\geq j(i)+l} \models \alpha$.
If $\sigma_{\geq j(i)} \models \beta$ then we can choose $k = j(i)$ (even if other choices as possible) and otherwise choose any such $k > j(i)$. Again there are two cases, either $k = j(i)$ or $k > j(i)$.
In the first case, when $\sigma_{\geq j(i)} \models \beta$, we put $x_{i+1} = y$ and otherwise we will make $x_{i+1} = z$. In either case put $j(i+1) = j(i)$.
Let us check the invariant. Consider the first case.
We know that we have $\sigma_{\geq j(i+1)} \models \beta$.
In the second case, we know that we have $\sigma_{\geq j(i+1)} \models \alpha$ and $\sigma_{\geq j(i+1)+1} \models \alpha U \beta$. Thus $\sigma_{\geq j(i+1)} \models X(\alpha U \beta)$.
Also, in either case, for every other $\gamma \in \Gamma(x_{i+1})$ we still have $\sigma_{\geq j(i+1)} \models \gamma$.
So we have the invariant holding.

[STEP]

So $\Gamma(x_i)$ is propositionally complete and there is one child, which we will make x_{i+1} and we will put $j(i+1) = j(i) + 1$.
Consider a formula
$\gamma \in \Gamma(x_{i+1}) = \{\alpha | X\alpha \in \Gamma(x_i)\} \cup \{\neg\alpha | \neg X\alpha \in \Gamma(x_i)\}$.
CASE 1: Say that $X\gamma \in \Gamma(x_i)$. Thus, by the invariant, $\sigma_{\geq j(i)} \models X\gamma$. Hence, $\sigma_{\geq j(i)+1} \models \gamma$. But this is just $\sigma_{\geq j(i+1)} \models \gamma$ as required.
CASE 2: Say that $\gamma = \neg\delta$ and $\neg X\delta \in \Gamma(x_i)$. Thus, by the invariant, $\sigma_{\geq j(i)} \models \neg X\delta$. Hence, $\sigma_{\geq j(i)+1} \not\models \delta$. But this is just $\sigma_{\geq j(i+1)} \models \gamma$ as required.
So we have the invariant holding.

[LOOP]

If, in T, the node x_i is a leaf just getting a tick via the LOOP rule then we are done.

T is a successful tableau as required.

[REP]

Now the tricky case.
Suppose that x_i is a node which gets a cross in T via the REP rule.
So there is a sequence
$u = x_h, x_{h+1}, ..., x_{h+a} = v, x_{h+a+1}, ..., x_{h+a+b} = x_i = w$ such that $\Gamma(u) = \Gamma(v) = \Gamma(w)$ and no extra eventualities of u are satisfied between v and w that were not already satisfied between u and v.
What we do now is to choose some such u, v and w, there may be more than one triple, and proceed with the construction as if x_i was v instead of w.
That is we move on to look at the rule (as above, and the rule will not be REP) that is used to get from v to its children.
However, we use $\sigma_{\geq i}$ to make the choice of child x_{i+1} (if there is a choice).
All the reasoning above works because $\Gamma(v) = \Gamma(x_i)$ and so the invariant holds for v instead of x_i as well.
Thus we keep going.

The above construction may end finitely with us finding a ticked leaf and succeeding.
However, at least in theory, it may seem possible that the construction keeps going forever even though the tableau will be finite.
The rest of the proof is to show that this actually can not happen.
The construction can not go on forever. It must stop and the only way that we have shown that that can happen is by finding a tick.

We suppose for contradiction that the construction does go on forever.
Thus, because there are only a finite number of nodes in the tableau, we must meet the REP rule and jump back up the tableau infinitely often.
The proof by contradictions shows that as eventualties are witnessed the path of x_is can never jump back up higher again.
Have a read!

You might like to read up on other approaches to deciding satisfiability in LTL.
Wolper [Wol85]
Schwendiman [Sch98]
Sistla and Clarke [SC85]
Schmitt and Goubault-Larrecq [SGL97]
Automata-based approaches
Resolution-based approaches.
Others, e.g. Small model theorems with axiom systems.
Implementation races, and benchmarking: [GKS10]

What about complexity?

Deciding LTL satisfiability is in PSPACE [SC85].
In fact our tableau approach can be used to show that.
Easiest to use the tableau search to directly show that the problem is in NPSPACE and then Savitch tells us also in PSPACE.
We are allowed to guess the right choices and need to show that "yes" answers can be guessed and checked using memory space bounded by a polynomial in the size of the input. To do this, just guess the right branch and remember at each step: the label here, the previous label (to check you do the STEP rule properly), the label back at an ancestor that you want to LOOP to, and the eventualities that you still have to satisfy from that. (Size of memory usage just linear in size of input even though branch length may be exponential).

Questions

And that's all!

Thank you.

Any questions.

Extended Proof of Completeness:

The following slides are only to be used if there is extra time or questions about the completness proof.

Proof of Completeness:

The basic idea is to use a model (of the satisfiable formula) to show that *in any tableau* there will be a branch (i.e. a leaf) with a tick.

A weaker result is to show that there is some tableau with a leaf with a tick.

Such a weaker result may actually be ok to establish correctness and complexity of the tableau technique.
However, it raises questions about whether a "no" answer from a tableau is correct and it does not give clear guidance for the implementer.

Completeness Proof:

Suppose that ϕ is a satisfiable formula of LTL.
It will have a model. Choose one, say $(S, R, g), \sigma \models \phi$. In what follows we (use standard practice when the model is fixed and) write $\sigma_{\geq i} \models \alpha$ when we mean $(S, R, g), \sigma_{\geq i} \models \alpha$.
Also, build a tableau T for ϕ in any manner as long as the rules are followed. Let $\Gamma(x)$ be the formula set label on the node x in T.
We will show that T has a ticked leaf.
To do this we first construct a sequence $x_0, x_1, x_2, \ldots.$ of nodes, with x_0 being the root. This sequence may terminate at a tick (and then we have succeeded) or it may hypothetically go on forever (and more on that later).
In general the sequence will head downwards from a parent to a child node but occasionally it may jump back up to an ancestor.

Invariant

As we go we will also make sure that each node x_i is associated with a state $\sigma_{j(i)}$ in S.
We will ensure that for each i, for each $\alpha \in \Gamma(x_i)$, $\sigma_{\geq j(i)} \models \alpha$.
Start by putting $j(0) = 0$ when x_0 is the tableau root node.
Note that the only formula in $\Gamma(x_0)$ is ϕ and that $\sigma_{\geq 0} \models \phi$.
Good start.

Now suppose that we have identified the x sequence up until x_i.

Consider the rule that is used in T to extend a tableau branch from x_i to some children.

[EMP] If $\Gamma(x_i) = \{\}$ then we are done. T is a successful tableau as required.

[X] Consider if it is possible for the branch to stop at x_i with a cross because of a contradiction. So there is some α with α and $\neg\alpha$ in $\Gamma(x_i)$. But this can not happen as then $\sigma_{\geq j(i)} \models \alpha$ and $\sigma_{\geq j(i)} \models \neg\alpha$.

[DNEG]

So $\neg\neg\alpha$ is in $\Gamma(x_i)$ and there is one child, which we will make $x(i+1)$ and we will put $j(i+1) = j(i)$. Because $\sigma_{\geq j(i)} \models \neg\neg\alpha$ we also have $\sigma_{\geq j(i+1)} \models \alpha$. Also for every other $\beta \in \Gamma(x_{i+1}) = \Gamma(x_i) \cup \{\alpha\}$, we still have $\sigma_{\geq j(i+1)} \models \beta$. So we have the invariant holding.

[CON]

So $\alpha \wedge \beta$ is in $\Gamma(x_i)$ and there is one child, which we will make $x(i+1)$ and we will put $j(i+1) = j(i)$. Because $\sigma_{\geq j(i)} \models \alpha \wedge \beta$ we also have $\sigma_{\geq j(i+1)} \models \alpha$ and $\sigma_{\geq j(i+1)} \models \beta$. Also for every other $\gamma \in \Gamma(x_{i+1}) = \Gamma(x_i) \cup \{\alpha, \beta\}$, we still have $\sigma_{\geq j(i+1)} \models \gamma$.
So we have the invariant holding.

[DIS]

So $\neg(\alpha \wedge \beta)$ is in $\Gamma(x_i)$ and there are two children. One y is labelled $\Gamma(x_i) \cup \{\neg\alpha\}$ and the other, z, is labelled $\Gamma(x_i) \cup \{\neg\beta\}$.
We know $\sigma_{\geq j(i)} \models \neg(\alpha \wedge \beta)$. Thus, $\sigma_{\geq j(i)} \not\models \alpha \wedge \beta$ and it is not the case that both $\sigma_{\geq j(i)} \models \alpha$ and $\sigma_{\geq j(i)} \models \beta$. So either $\sigma_{\geq j(i)} \models \neg\alpha$ or $\sigma_{\geq j(i)} \models \neg\beta$.
If the former, i.e. that $\sigma_{\geq j(i)} \models \neg\alpha$ we will make $x_{i+1} = y$ and otherwise we will make $x_{i+1} = z$. In either case put $j(i+1) = j(i)$.
Let us check the invariant. Consider the first case. The other is exactly analogous.
We already know that we have $\sigma_{\geq j(i+1)} \models \neg\alpha$. Also for every other $\gamma \in \Gamma(x_{i+1}) = \Gamma(y) = \Gamma(x_i) \cup \{\neg\alpha\}$, we still have $\sigma_{\geq j(i+1)} \models \gamma$.
So we have the invariant holding.

[UNT]

So $\alpha U \beta$ is in $\Gamma(x_i)$ and there are two children. One y is labelled $\Gamma(x_i) \cup \{\beta\}$ and the other, z, is labelled $\Gamma(x_i) \cup \{\alpha, X(\alpha U \beta)\}$.
We know $\sigma_{\geq j(i)} \models \alpha U \beta$. Thus, there is some $k \geq j(i)$ such that $\sigma_{\geq k} \models \beta$ and for all l, if $0 \leq l < k$ then $\sigma_{\geq j(i)+l} \models \alpha$.
If $\sigma_{\geq j(i)} \models \beta$ then we can choose $k = j(i)$ (even if other choices as possible) and otherwise choose any such $k > j(i)$. Again there are two cases, either $k = j(i)$ or $k > j(i)$.
In the first case, when $\sigma_{\geq j(i)} \models \beta$, we put $x_{i+1} = y$ and otherwise we will make $x_{i+1} = z$. In either case put $j(i+1) = j(i)$.
Let us check the invariant. Consider the first case.
We know that we have $\sigma_{\geq j(i+1)} \models \beta$.
In the second case, we know that we have $\sigma_{\geq j(i+1)} \models \alpha$ and $\sigma_{\geq j(i+1)+1} \models \alpha U \beta$. Thus $\sigma_{\geq j(i+1)} \models X(\alpha U \beta)$.
Also, in either case, for every other $\gamma \in \Gamma(x_{i+1})$ we still have $\sigma_{\geq j(i+1)} \models \gamma$.
So we have the invariant holding.

[NUN]

So $\neg(\alpha U \beta)$ is in $\Gamma(x_i)$ and there are two children. One y is labelled $\Gamma(x_i) \cup \{\neg\alpha, \neg\beta\}$ and the other, z, is labelled $\Gamma(x_i) \cup \{\neg\beta, X\neg(\alpha U \beta)\}$.
We know $\sigma_{\geq j(i)} \models \neg(\alpha U \beta)$.
So for sure $\sigma_{\geq j(i)} \models \neg\beta$.
Furthermore, possibly $\sigma_{\geq j(i)} \models \neg\alpha$ as well, but otherwise if $\sigma_{\geq j(i)} \models \alpha$ then we can show that we can not have $\sigma_{\geq j(i)+1} \models \alpha U \beta$. Suppose for contradiction that $\sigma_{\geq j(i)} \models \alpha$ and $\sigma_{\geq j(i)+1} \models \alpha U \beta$.

[NUN] continued

We have supposed that $\sigma_{\geq j(i)} \models \alpha$ and $\sigma_{\geq j(i)+1} \models \alpha U \beta$ while also $\sigma_{\geq j(i)} \not\models \alpha U \beta$.
Then there is some $k \geq 0$ such that $\sigma_{\geq j(i)+1+k} \models \beta$ and for all l, if $0 \leq l < k$ then $\sigma_{\geq j(i)+1+l} \models \alpha$.
But then for all l, if $0 \leq l < k+1$ then $\sigma_{\geq j(i)+l} \models \alpha$.
Thus $\sigma_{\geq j(i)} \models \alpha U \beta$.
Contradiction.

[NUN] continued

So we can conclude that there are two cases when the NUN rule is used.
CASE 1: $\sigma_{\geq j(i)} \models \neg\beta$ and $\sigma_{\geq j(i)} \models \neg\alpha$.
CASE 2: $\sigma_{\geq j(i)} \models \neg\beta$ and $\sigma_{\geq j(i)+1} \models \neg(\alpha U \beta)$.
In the first case, when $\sigma_{\geq j(i)} \models \neg\beta$, we put $x_{i+1} = y$ and otherwise we will make $x_{i+1} = z$. In either case put $j(i+1) = j(i)$.
Let us check the invariant. In both cases we know that we have $\sigma_{\geq j(i+1)} \models \neg\beta$.
Now consider the first case. We also have $\sigma_{\geq j(i)} \models \neg\alpha$.
In the second case, we know that we have $\sigma_{\geq j(i)+1} \models \neg(\alpha U \beta)$.
Thus $\sigma_{\geq j(i+1)} \models X\neg(\alpha U \beta)$.
Also, in either case, for every other $\gamma \in \Gamma(x_{i+1})$ we still have $\sigma_{\geq j(i+1)} \models \gamma$.
So we have the invariant holding.

[STEP]

So $\Gamma(x_i)$ is propositionally complete and there is one child, which we will make x_{i+1} and we will put $j(i+1) = j(i) + 1$.
Consider a formula
$\gamma \in \Gamma(x_{i+1}) = \{\alpha | X\alpha \in \Gamma(x_i)\} \cup \{\neg\alpha | \neg X\alpha \in \Gamma(x_i)\}$.
CASE 1: Say that $X\gamma \in \Gamma(x_i)$. Thus, by the invariant, $\sigma_{\geq j(i)} \models X\gamma$. Hence, $\sigma_{\geq j(i)+1} \models \gamma$. But this is just $\sigma_{\geq j(i+1)} \models \gamma$ as required.
CASE 2: Say that $\gamma = \neg\delta$ and $\neg X\delta \in \Gamma(x_i)$. Thus, by the invariant, $\sigma_{\geq j(i)} \models \neg X\delta$. Hence, $\sigma_{\geq j(i)+1} \not\models \delta$. But this is just $\sigma_{\geq j(i+1)} \models \gamma$ as required.
So we have the invariant holding.

[LOOP]

If, in T, the node x_i is a leaf just getting a tick via the LOOP rule then we are done.

T is a successful tableau as required.

[REP]

Now the tricky case.
Suppose that x_i is a node which gets a cross in T via the REP rule.
So there is a sequence
$u = x_h, x_{h+1}, ..., x_{h+a} = v, x_{h+a+1}, ..., x_{h+a+b} = x_i = w$ such that $\Gamma(u) = \Gamma(v) = \Gamma(w)$ and no extra eventualities of u are satisfied between v and w that were not already satisfied between u and v.
What we do now is to choose some such u, v and w, there may be more than one triple, and proceed with the construction as if x_i was v instead of w.
That is we move on to look at the rule (as above, and the rule will not be REP) that is used to get from v to its children.
However, we use $\sigma_{\geq i}$ to make the choice of child x_{i+1} (if there is a choice).
All the reasoning above works because $\Gamma(v) = \Gamma(x_i)$ and so the invariant holds for v instead of x_i as well.
Thus we keep going.

The above construction may end finitely with us finding a ticked leaf and succeeding.
However, at least in theory, it may seem possible that the construction keeps going forever even though the tableau will be finite.
The rest of the proof is to show that this actually can not happen.
The construction can not go on forever. It must stop and the only way that we have shown that that can happen is by finding a tick.

Suppose for contradiction that the construction does go on forever.
Thus, because there are only a finite number of nodes in the tableau, we must meet the REP rule and jump back up the tableau infinitely often.
When we do apply the REP rule with triple (u, v, w) call that a jump triple.
There are only a finite number of jump triples so there must be some that cause us to jump infinitely often.
Say that (u_0, v_0, w_0) is one such.
We can choose u_0 so that for no other infinite jump triple (u_1, v_1, w_1) do we have u_1 being a proper ancestor of u_0.

As we proceed through the construction of $x_0, x_1, ..$ and see a jump every so often, eventually all the jump triples who only cause a jump a finite number of times stop causing jumps.
After that time, (u_0, v_0, w_0) will still cause a jump every so often.
Thus after that time u_0 will never appear again as the x_i that we choose and all the x_is that we choose will be descendants of u_0.
This is because we will never jump up to u_0 or above it (closer to the root).
Say that x_N is the very last x_i that is equal to u_0.

Now consider any $\alpha U \beta$ that appears in $\Gamma(u_0)$. (There must be at least one eventuality in $\Gamma(u_0)$ as it is used to apply rule REP).
A simple induction shows that $\alpha U \beta$ will appear in every $\Gamma(x_i)$ from $i = N$ up until at least when β appears in some $\Gamma(x_i)$ after that (if that ever happens).
This is because if $\alpha U \beta$ is in $\Gamma(x_i)$ and β is not there and does not get put there then $X(\alpha U \beta)$ will also be put in before the next temporal step rule.
Each temporal step rule will thus put $\alpha U \beta$ into the new label.

Now $j(i)$ just increases by 0 or 1 with each increment of i, We also know that $\sigma_{\geq j(i)} \models \alpha U \beta$ from $i = N$ onwards until (and if) β gets put in $\Gamma(x_i)$.
Since σ is a fullpath we will eventually get to some i with $\sigma_{\geq j(i)} \models \beta$.
In that case our construction makes us put β in the label.
Thus we do eventually get to some $i \geq N$ with $\beta \in \Gamma(x_i)$.
Let N_β be the first such $i \geq N$.
Note that all the nodes between u_0 and x_{N_β} in the tableau also appear as x_i for $N < i < N_\beta$ so that they all have $\alpha U \beta$ and not β in their labels $\Gamma(x_i)$.

Now let us consider if we ever jump up above x_{N_β} at any step of our construction (after N_β).
In that case there would be tableau nodes u, v and w arranged according to the jump situation.
Since u is not above u_0 and v is above x_{N_β}, we must have $\Gamma(u) = \Gamma(v)$ with $\alpha U \beta$ in them and not satisfied in between.
But w will be below x_{N_β} at the first such jump, meaning that β is satisfied between v and w.
Contradiction.

The above reasoning applies to all eventualities in $\Gamma(u_0)$.
Thus, after they are all satisfied, the construction x_i does not jump up above any of them.
When the next supposed jump involving u_0 with some v and w happens after that it is clear that all of the eventualities in $\Gamma(u_0)$ are satisfied above v.
This is a contradiction to such a jump ever happening.
Thus we can conclude that there are not an infinite number of jumps after all.
The construction must finish with a tick.
END of completeness proof.


Rajeev Alur and Thomas A. Henzinger.
Real-time logics: Complexity and expressiveness.
Inf. Comput., 104(1):35–77, 1993.


M. Bertello, N. Gigante, A. Montanari and M. Reynolds.
Leviathan: A New LTL Satisfiability Checking Tool Based on a One-Pass Tree-Shaped Tableau.
In *Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016, Proceedings*, pages 950–956. IJCAI/AAAI Press, 2016.


Burgess, J. P. Axioms for Tense Logic I: "Since" and "Until", Notre Dame J. Formal Logic, 23(2):367–374, 1982.


J. P. Burgess and Y. Gurevich.
The decision problem for linear temporal logic.
Notre Dame J. Formal Logic, 26(2):115–128, 1985.


E. Emerson and E. C. Clarke.

Using branching time temporal logic to synthesise synchronisation skeletons.
Sci. of Computer Programming, 2, 1982.


E. Emerson and A. Sistla.
Deciding branching time logic.
In *Proc. 16th ACM Symposium on Theory of Computing*, 1984.


M. Fischer and R. Ladner.
Propositional dynamic logic of regular programs.
J. Computer and System Sciences, 18:194–211, 1979.


Oliver Friedmann, Markus Latte, and Martin Lange.
A decision procedure for CTL* based on tableaux and automata.
In *IJCAR'10*, pages 331–345, 2010.


R. Goré.
Tableau methods for modal and temporal logics.

In M. D'Agostino, D. Gabbay, R. Hähnle, and J. Posegga, editors, *Handbook of Tableau Methods*, pages 297–396. Kluwer Academic Publishers, 1999.

H. Kamp.
Tense logic and the theory of linear order.
PhD thesis, University of California, Los Angeles, 1968.

Y. Kesten, Z. Manna, and A. Pnueli.
Temporal verification of simulation and refinement.
In *A decade of concurrency: reflections and perspectives: REX school/symposium, Noordwijkerhout, the Netherlands, June 1–4, 1993*, pages 273–346. Springer–Verlag, 1994.

A. Pnueli.
The temporal logic of programs.
In *Proceedings of the Eighteenth Symposium on Foundations of Computer Science*, pages 46–57, 1977.
Providence, RI.


V. R. Pratt.
Models of program logics.
In *Proc. 20th IEEE. Symposium on Foundations of Computer Science, San Juan*, pages 115–122, 1979.


Mark Reynolds.
A tableau-based decision procedure for CTL*.
Journal of Formal Aspects of Computing, pages 1–41, August 2011.


A. Sistla and E. Clarke.
Complexity of propositional linear temporal logics.
J. ACM, 32:733–749, 1985.


Valentin Goranko, Angelo Kyrilov, and Dmitry Shkatov.
Tableau tool for testing satisfiability in ltl: Implementation and experimental analysis.
Electronic Notes in Theoretical Computer Science, 262(0):113 – 125, 2010.

Proceedings of the 6th Workshop on Methods for Modalities (M4M-6 2009).

M. Reynolds.

A New Rule for LTL Tableaux.

In *Seventh International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2016, Catania, Italy, 14-16 September 2016, Proceedings*, pages 287–301. EPTCS, 2016.

A. Sistla and E. Clarke.

Complexity of propositional linear temporal logics.

J. ACM, 32:733–749, 1985.

S. Schwendimann.

A new one-pass tableau calculus for PLTL.

In Harrie C. M. de Swart, editor, *Proceedings of International Conference, TABLEAUX 1998, Oisterwijk*, LNAI 1397, pages 277–291. Springer, 1998.

P. Schmitt and J. Goubault-Larrecq.
A tableau system for linear-time temporal logic.
In *TACAS 1997*, pages 130–144, 1997.

M. Vardi and L. Stockmeyer.
Improved upper and lower bounds for modal logics of programs.
In *17th ACM Symp. on Theory of Computing, Proceedings*, pages 240–251. ACM, 1985.

P. Wolper.
The tableau method for temporal logic: an overview.
Logique et Analyse, 28:110–111, June–Sept 1985.