Università degli studi di Udine
Laurea Magistrale: Informatica
Lectures for April/May 2014
La verifica del software: temporal logic
Lecture 03 LTL tableau continued

Guest lecturer: Mark Reynolds,
The University of Western Australia

May 7, 2014

Lecture 03

- Tableau for checking satisfiability in LTL continued.

Lecture 3 LTL satisfiability

We started a detailed look at an algorithm to decide the satisfiability of LTL formulas.
The first slides are repeated from the last lecture so it is easy to look back.

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.

LTL Syntax:

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 $\alpha U \beta$.

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:

$$
\begin{array}{lll}
M, \sigma \models p & \text{iff} & p \in g(\sigma_0), \text{ for } p \in \mathcal{L} \\
M, \sigma \models \neg\alpha & \text{iff} & M, \sigma \not\models \alpha \\
M, \sigma \models \alpha \wedge \beta & \text{iff} & M, \sigma \models \alpha \text{ and } M, \sigma \models \beta \\
M, \sigma \models X\alpha & \text{iff} & M, \sigma_{\geq 1} \models \alpha \\
M, \sigma \models \alpha U \beta & \text{iff} & \text{there is some } i \geq 0 \text{ such that } M, \sigma_{\geq i} \models \beta \\
 & & \text{and for each } j, \text{ if } 0 \leq j < i \text{ then } M, \sigma_{\geq j} \models \alpha
\end{array}
$$

Satisfiability:

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

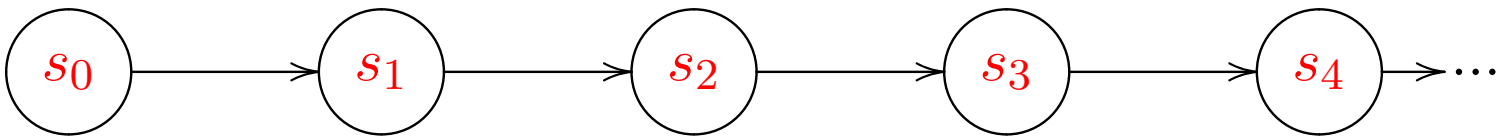
Eg, $\top$, p, Fp, $p \wedge Xp \wedge F\neg p$, Gp are each satisfiable.

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
$\neg p \wedge X \neg p \wedge (qUp)$

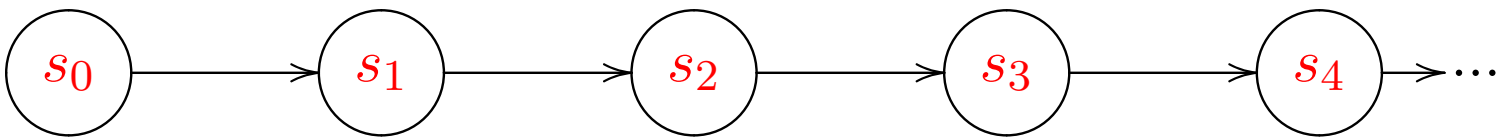


Let's make $\neg p \wedge X \neg p \wedge (qUp)$ true at s_0.

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.



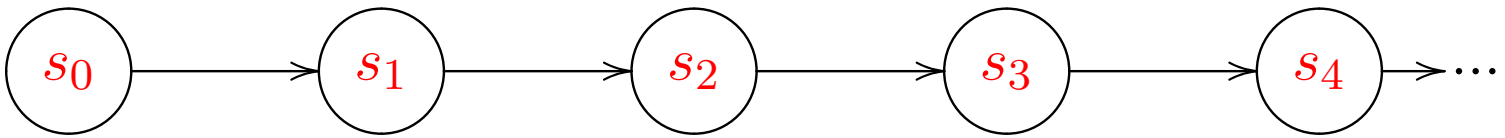
Easy to make $\neg p$ hold there.
Easy to see that we should also make $\neg p$ true at s_1.
But what about qUp?

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?:

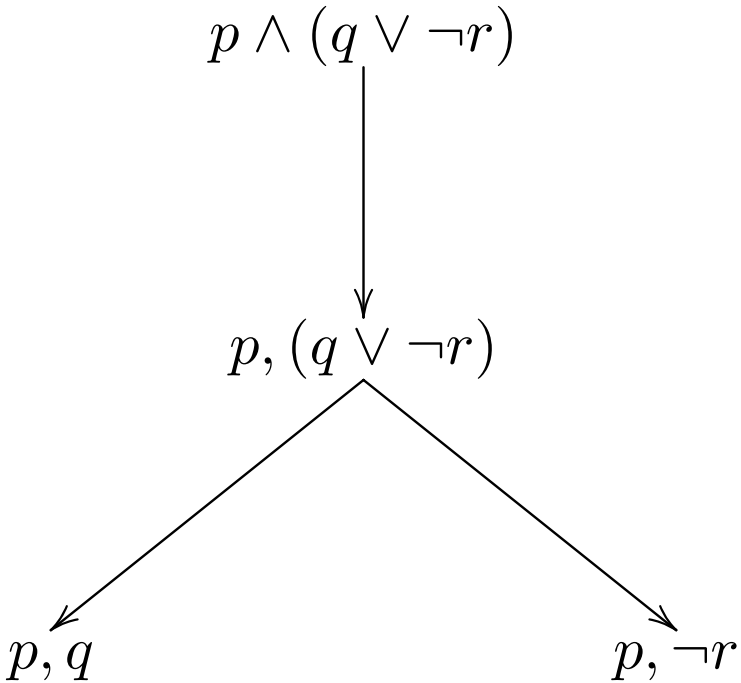
Labelling nodes with formulas is good. Starting from s_0 and working forwards in time is good.

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 a new (unpublished) simpler tableau for LTL. It builds on work by Sistla and Clarke (1985) and is influenced by LTL tableaux by Wolper (1982) and Schwendimann (1998).

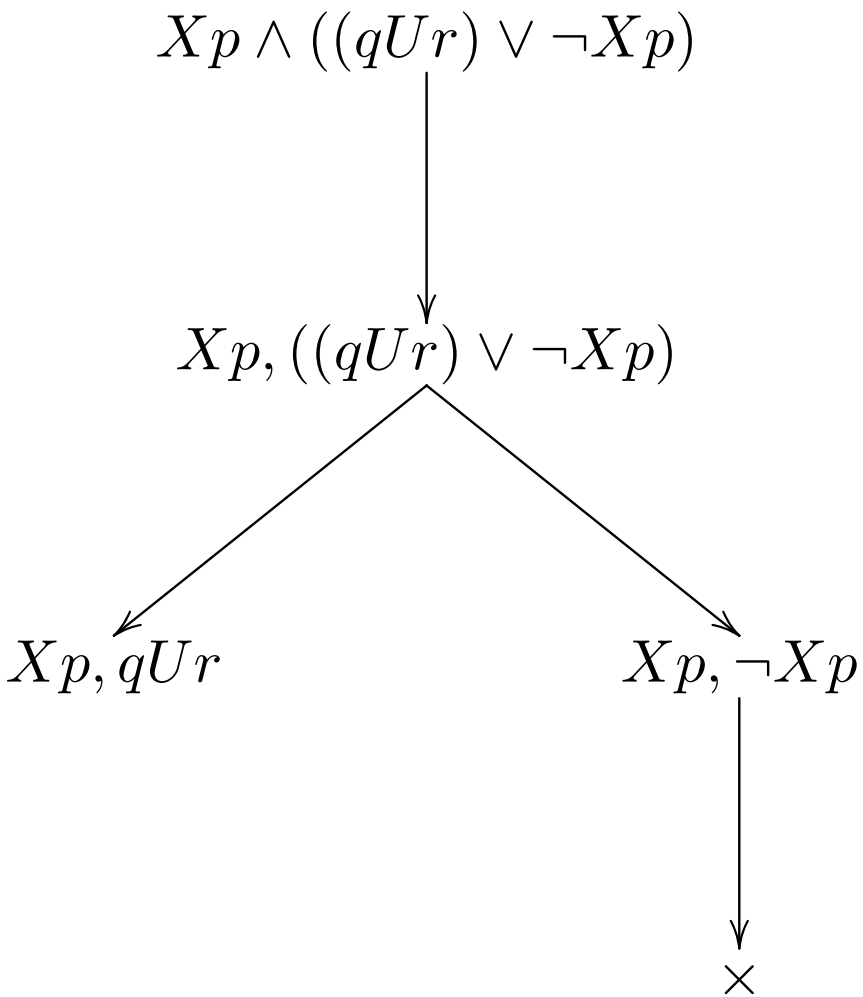
Reminder of Tableau for classical propositional logic:

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



Same rules for LTL:

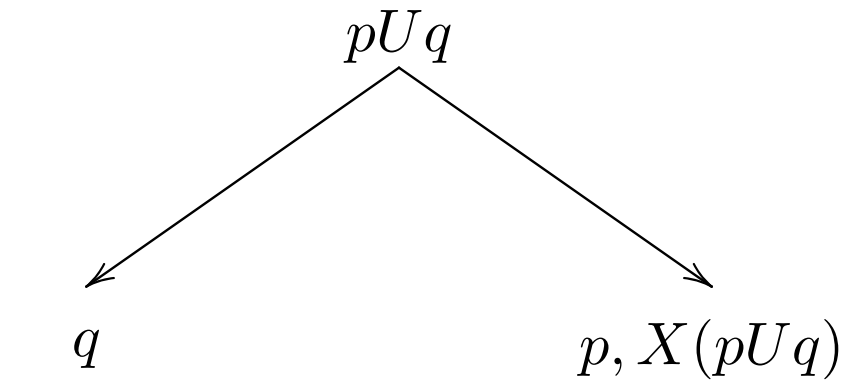
Same things can happen for LTL within a state ...

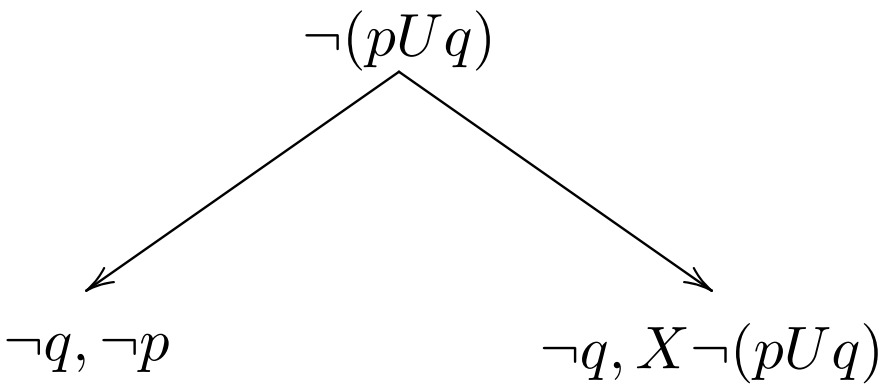


Rules:

[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 Γ with some $\neg\neg\alpha$ in Γ then this node can have one child labelled $\Gamma \cup \{\alpha\}$ (provided this is not Γ itself).
[CON]: If a node is labelled Γ with some $\alpha \wedge \beta$ in Γ then this node can have one child labelled $\Gamma \cup \{\alpha, \beta\}$ (provided this is not Γ itself).
[DIS]: If a node is labelled Γ with some $\neg(\alpha \wedge \beta)$ in Γ then this node can have two children labelled $\Gamma \cup \{\neg\alpha\}$ and $\Gamma \cup \{\neg\beta\}$ respectively (provided neither is Γ itself).

Until also gives us choices:



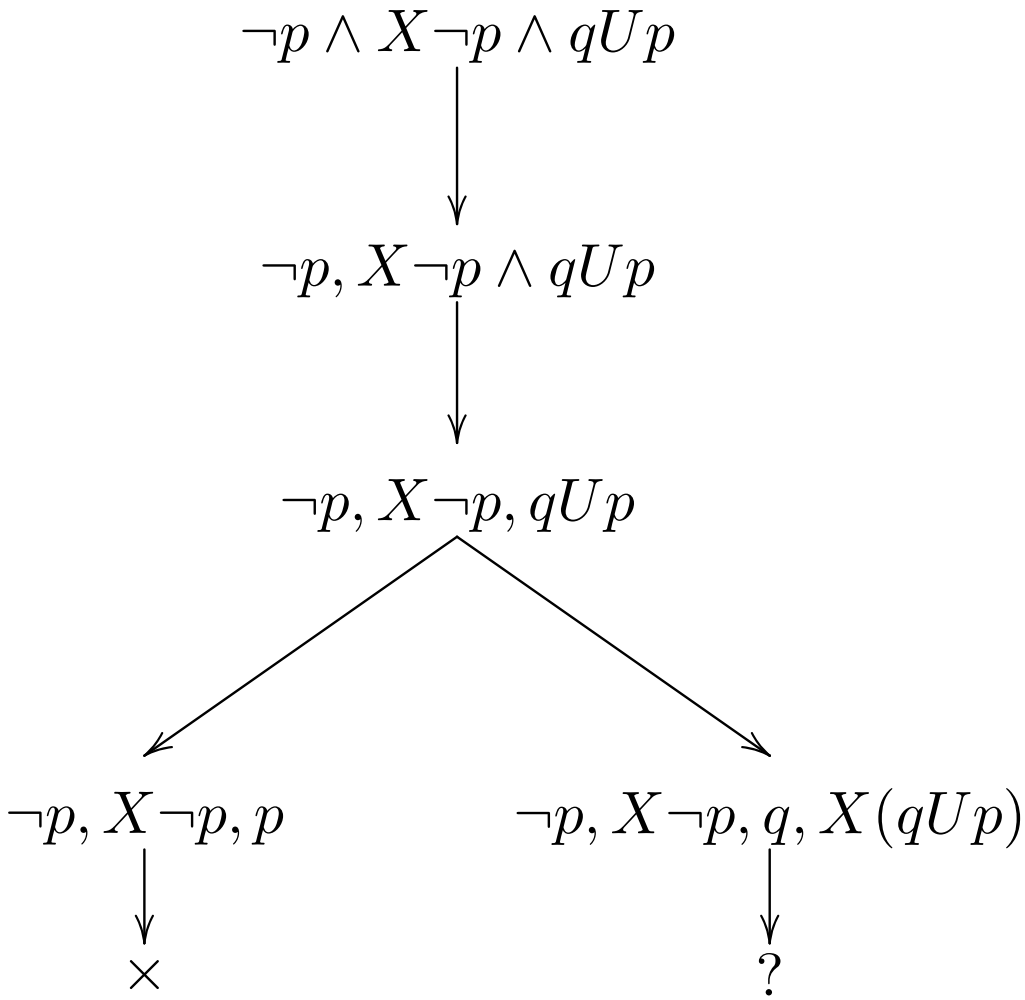


Rules for Until:

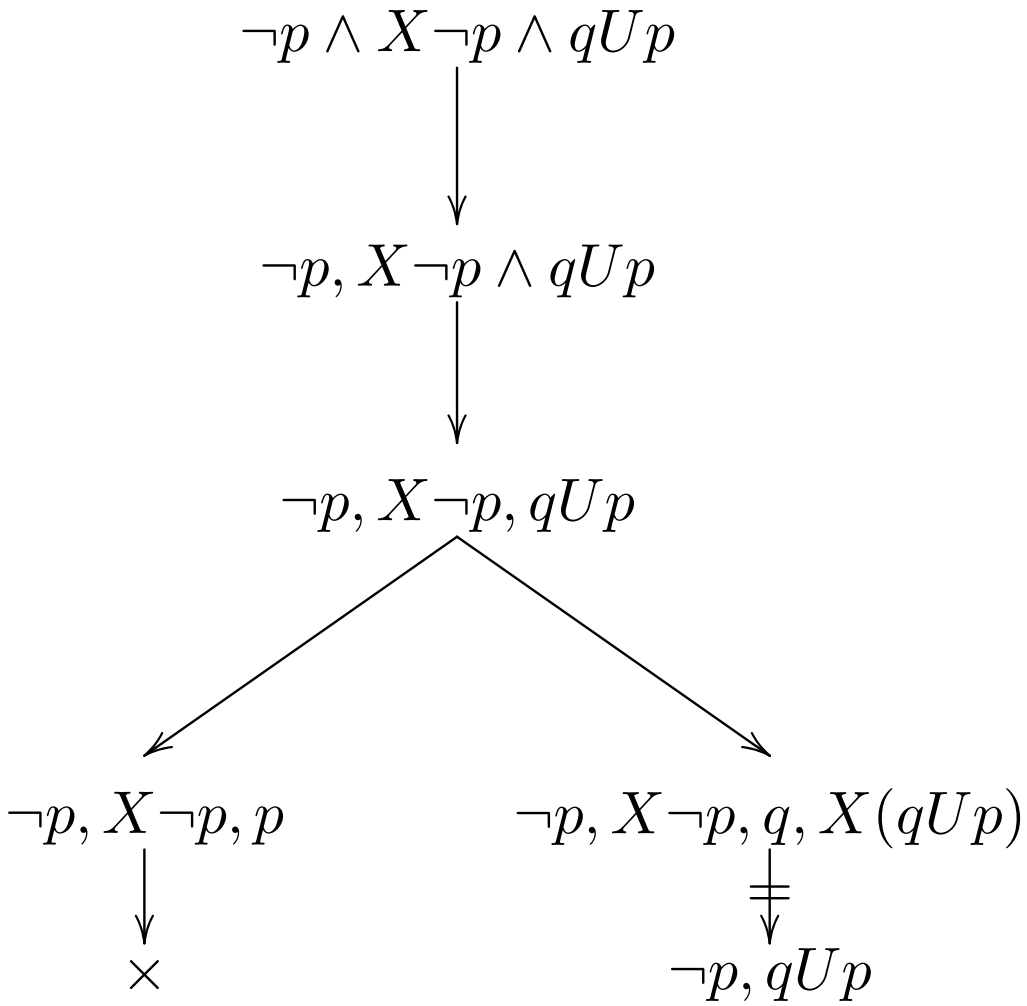
[UNT]: If a node is labelled Γ with some $\alpha U \beta$ in Γ then this node can have two children labelled $\Gamma \cup \{\alpha, X(\alpha U \beta)\}$ and $\Gamma \cup \{\beta\}$ ***provided that you have not already used this rule on $\alpha U \beta$ since the last use of the rule STEP (see below)***.

[NUN]: If a node is labelled Γ with some $\neg(\alpha U \beta)$ in Γ then this node can have two children labelled $\Gamma \cup \{\neg\beta, X\neg(\alpha U \beta)\}$ and $\Gamma \cup \{\neg\alpha, \neg\beta\}$ (provided neither is Γ itself).

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



Introduce a new type of step:



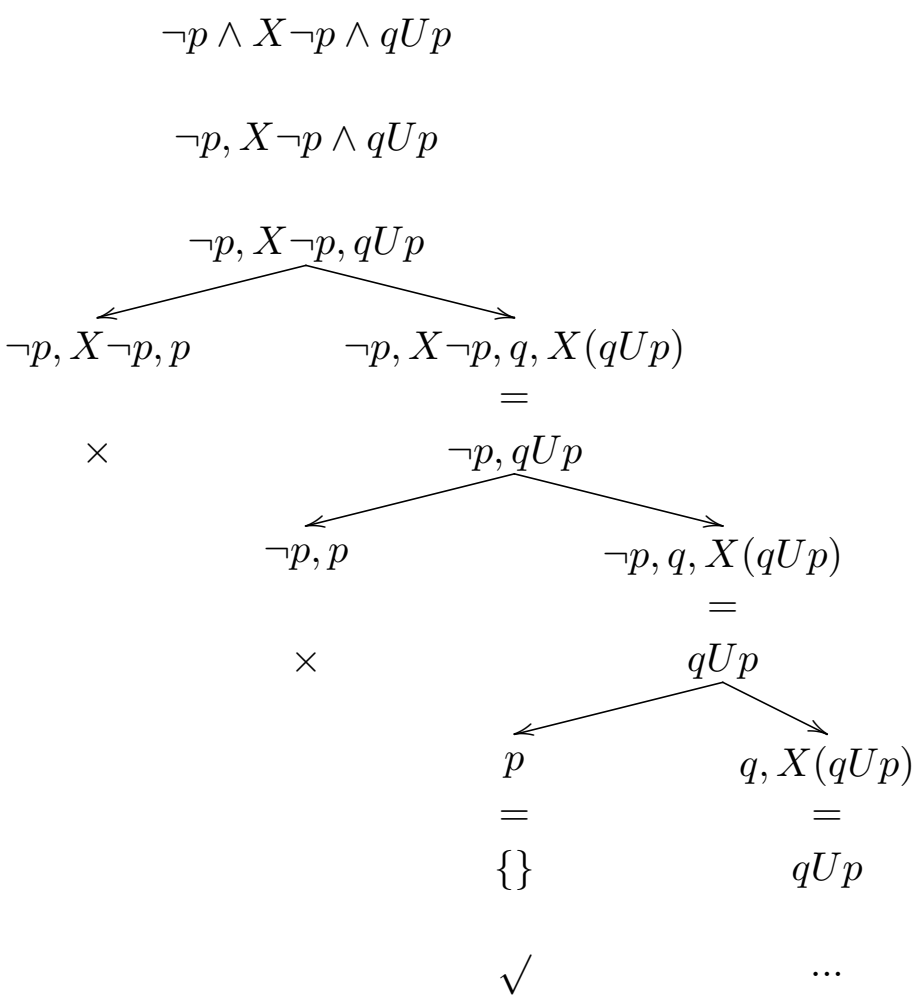
Step Children:

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.

[STEP]: 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 then try to use the static rules again.

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



Notation for tableaux:

Note that we do not always write the full set of formulas at every stage.
Eg, $\neg p \wedge X\neg p \wedge qUp$ should be $\{\neg p \wedge X\neg p \wedge qUp\}$ and it should be followed by $\{\neg p, X\neg p \wedge qUp, \neg p \wedge X\neg p \wedge qUp\}$ instead of just $\neg p, X\neg p \wedge qUp$.
In fact, you may notice that each formula only has one rule which is able to act on it. Once it has been used, the formula becomes uninteresting.
This is traditional for tableaux and does not matter much.
However, for our "looping" rules coming up we need to remember that we are officially using the whole set of formulas. The displayed set may only lose formulas at temporal steps.

General Idea of the LTL Tableau:

The tableau for ϕ is a tree of nodes (going down the page from a root) each labelled by a set of formulas.
The root is labelled $\{\phi\}$.
Some of the child relations are step-children (indicated by $-\!\!\!\!\!\rightarrow$).
Some of the branches end in crossed leaves and some in ticked leaves.
The whole tableau is successful (if there is a ticked branch), failed (if all branches are crossed) and otherwise unfinished. Note that you can stop the algorithm if you tick a branch!
In what follows we will sometimes denote by $\Gamma(u)$, the tableau label on a node u.

Before we continue, some shortcuts:

You can show that certain common formulas always lead to a similar sequence of steps. Thus we can instead use some shortcuts.

For example, $\top$ can be ignored in a list of formulas and $\bot$ crosses a branch.

Also, $G\alpha$ generates $\alpha, XG\alpha$ while $F\alpha$ leads to two branches, one with α and the other with $XF\alpha$.

Useful for manual tableaux and implementations to take note of.

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.

$$
\begin{array}{c}
Gp \\
\\
p, XGp \\
= \\
Gp \\
\\
p, XGp \\
= \\
Gp \\
\\
p, XGp \\
\\
\ldots
\end{array}
$$

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 my 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]:
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 use the LOOP rule 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.

First, 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.

Proof of Soundness:

(Overview first)
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.

And that's all for the third lecture.

See you tomorrow.