Università degli studi di Udine
Laurea Magistrale: Informatica
Lectures for April/May 2014
La verifica del software: temporal logic
Lecture 02 CTL and LTL-SAT

Guest lecturer: Mark Reynolds,
The University of Western Australia

April 30, 2014

Lecture 02

- Branching time temporal logics: CTL* and CTL.
- Checking satisfiability in LTL.

Structures (CORRECTION):

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.$(x, y) \in R$

CTL* Syntax:

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

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

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

CTL* 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 $M, \sigma_{\geq j} \models \alpha$
$M, \sigma \models A\alpha$	iff	for all fullpaths τ with $\tau_0 = \sigma_0$, $M, \tau \models \alpha$

CTL* Abbreviations:

Classical and linear temporal abbreviations as for LTL.

Also,
$E\alpha \equiv \neg A \neg \alpha$ meaning that there is a path on which α holds.

CTL* examples:

$XEXp \rightarrow EXXp$
$AG(p \rightarrow Ap)$
$AXFp \rightarrow XAFp$
$E(pU(E(pUq))) \rightarrow E(pUq)$
$AG(p \rightarrow q) \rightarrow (EFp \rightarrow EFq)$
$AG(p \rightarrow EXFp) \rightarrow (p \rightarrow EGFp)$

CTL Syntax:

CTL is a restriction of CTL* (but it can be presented directly, not via CTL*).
It includes, as wffs, all atoms p, and if α and β are wffs then so are $\neg\alpha$, $\alpha \wedge \beta$, $AX\alpha$, $EX\alpha$, $AF\alpha$, $EF\alpha$, $A(\alpha U \beta)$, and $E(\alpha U \beta)$.
So $AG\alpha$ and $EG\alpha$ are also included.

It uses the same semantics as CTL*.
However, ...

CTL Semantics:

It can be shown that the truth of CTL formulas in a structure only depend on the initial state of the fullpath. They are sometimes called state formulas.
That is, if $\pi_0 = \sigma_0$ then $M, \pi \models \alpha$ iff $M, \sigma \models \alpha$.

Not so for LTL or CTL* formulas in general.

This makes some of our reasoning tasks easier.

Examples:

$AG(p \rightarrow AFq)$
$EFq \rightarrow EFEGq$
$AG(p \rightarrow EXp) \rightarrow AG(p \rightarrow EGp)$
$EG(p \rightarrow A(qUr))$

Expressiveness:

CTL* extends both LTL and CTL.

CTL can say things that LTL can not: eg $AG(p \rightarrow EFq)$

LTL can say things that CTL can not. eg $G(p \rightarrow FGp)$

Example of Satisfiability:

Is the following part of a specification ever possibly true?
$p \wedge G(p \rightarrow EXFp) \wedge AFG\neg p$

Example of Validity:

Does every system satisfy this?
$AG(p \rightarrow EX(qUp)) \rightarrow (p \rightarrow EG(p \vee q))$

Example of Consequence:

If we have a system which ensures the following
$AG(p \rightarrow GXAFq)$
then does it also make the following true?
$GFp \rightarrow GFq$

Example of Synthesis:

Make a structure with a fullpath making the following true.
$AG(p \rightarrow EXFp) \wedge p \wedge EF(p \wedge XEG\neg p)$

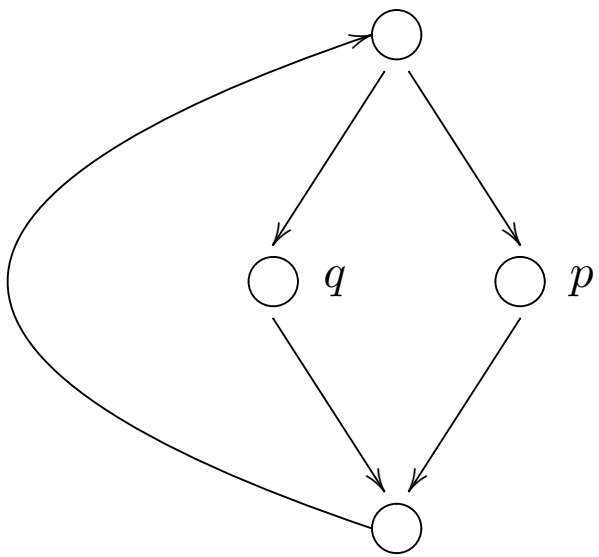
Example of Imperative Programming:

p
$G(p \rightarrow Xp)$
Fr
$G((p \wedge r) \rightarrow Xr)$

Example of Model Checking:

Does the following formula hold (at the top state) in the structure below?

$AG(GFEXp \rightarrow GFp)$



Lecture 2, Part 2: 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.

(REMINDER) 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 $(S, R, g), \sigma \models \alpha$.

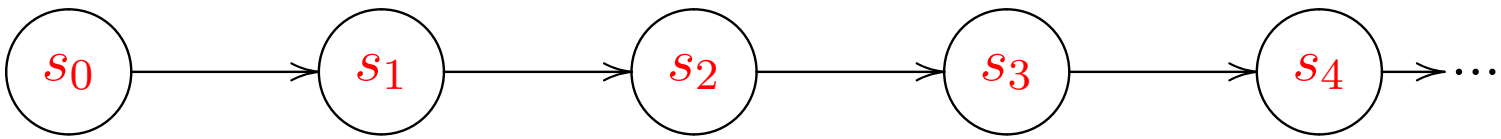
Eg, $\top$, p, Fp, $p \wedge Xp \wedge F\neg p$, Gp are each satisfiable.

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

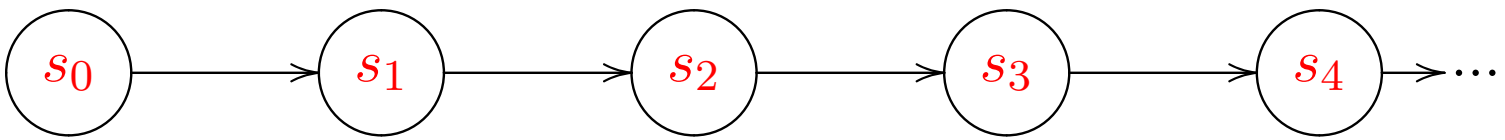


Let's make $\neg p \wedge X \neg p \wedge (qUp)$ true at s_0.

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.



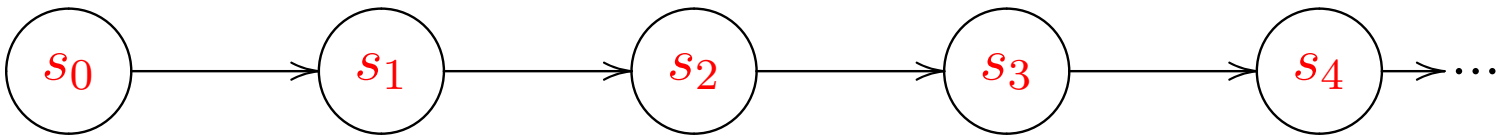
Easy to make $\neg p$ hold there.
Easy to see that we should also make $\neg p$ true at s_1.
But what about qUp?

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?:

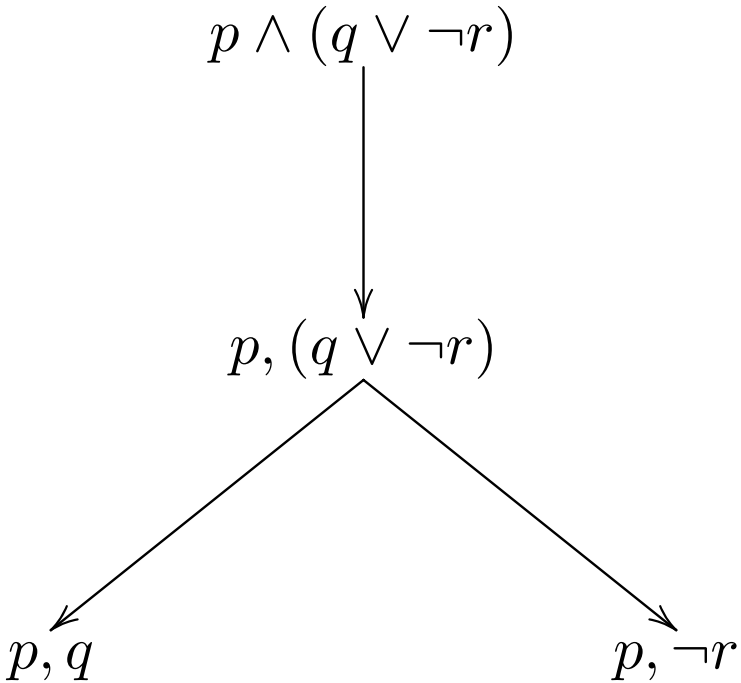
Labelling nodes with formulas is good. Starting from s_0 and working forwards in time is good.

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 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).

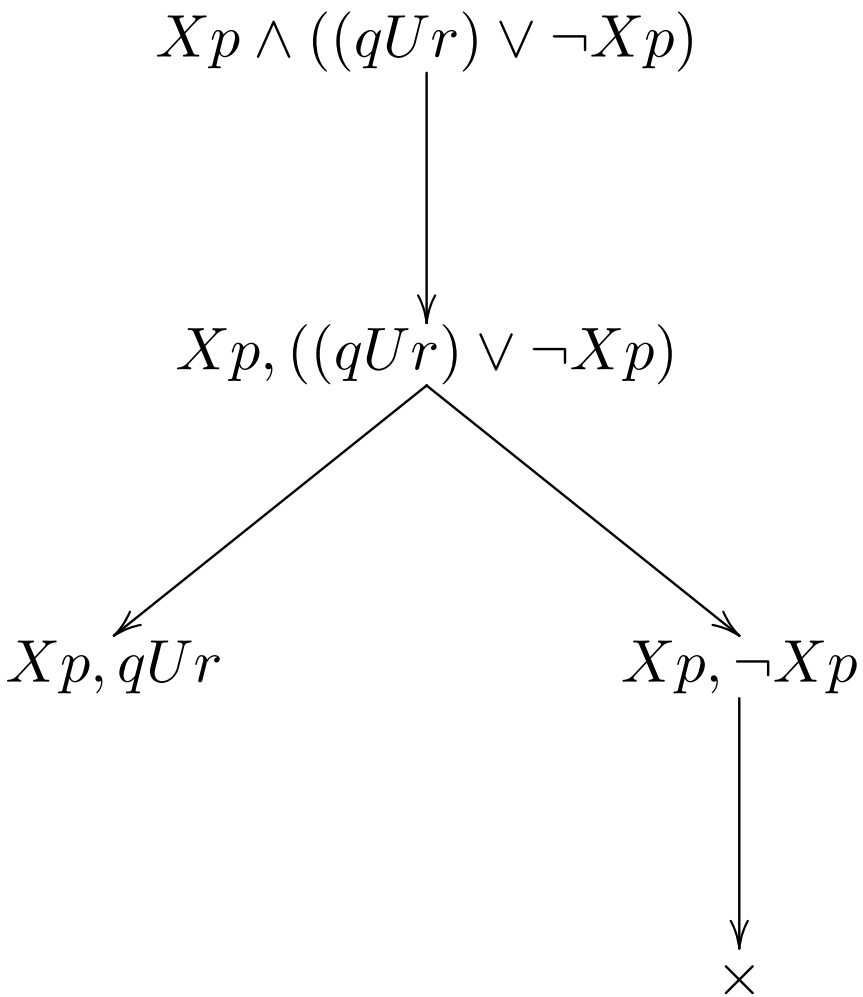
Reminder of Tableau for classical propositional logic:

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



Same rules for LTL:

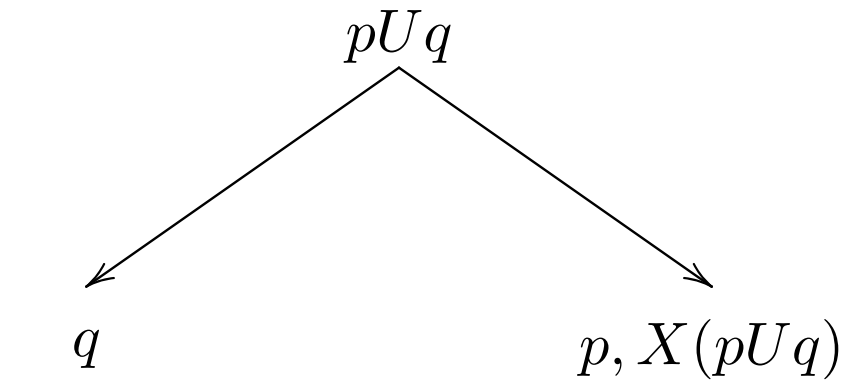
Same things can happen for LTL within a state ...

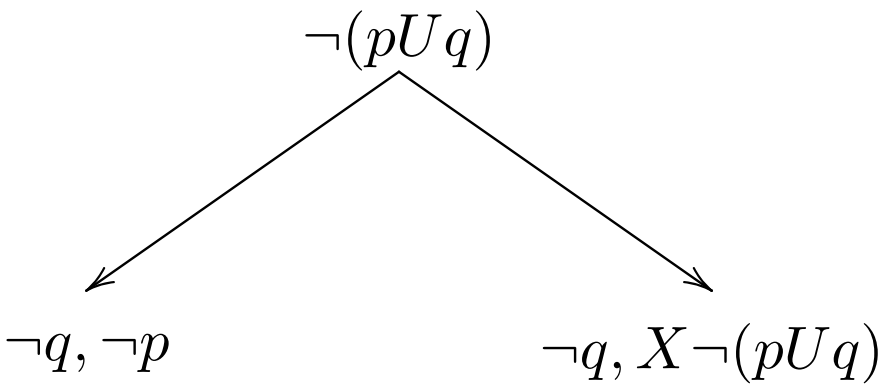


Rules:

[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:



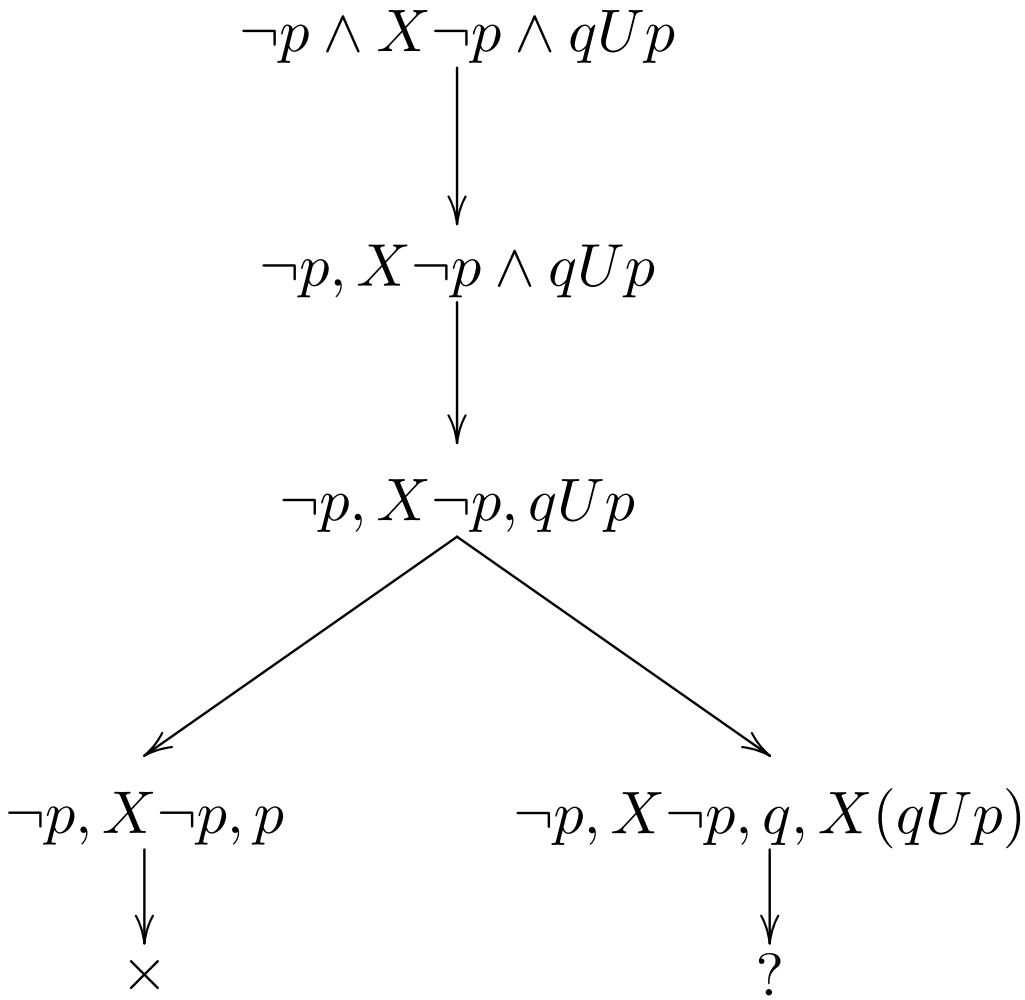


Rules for Until:

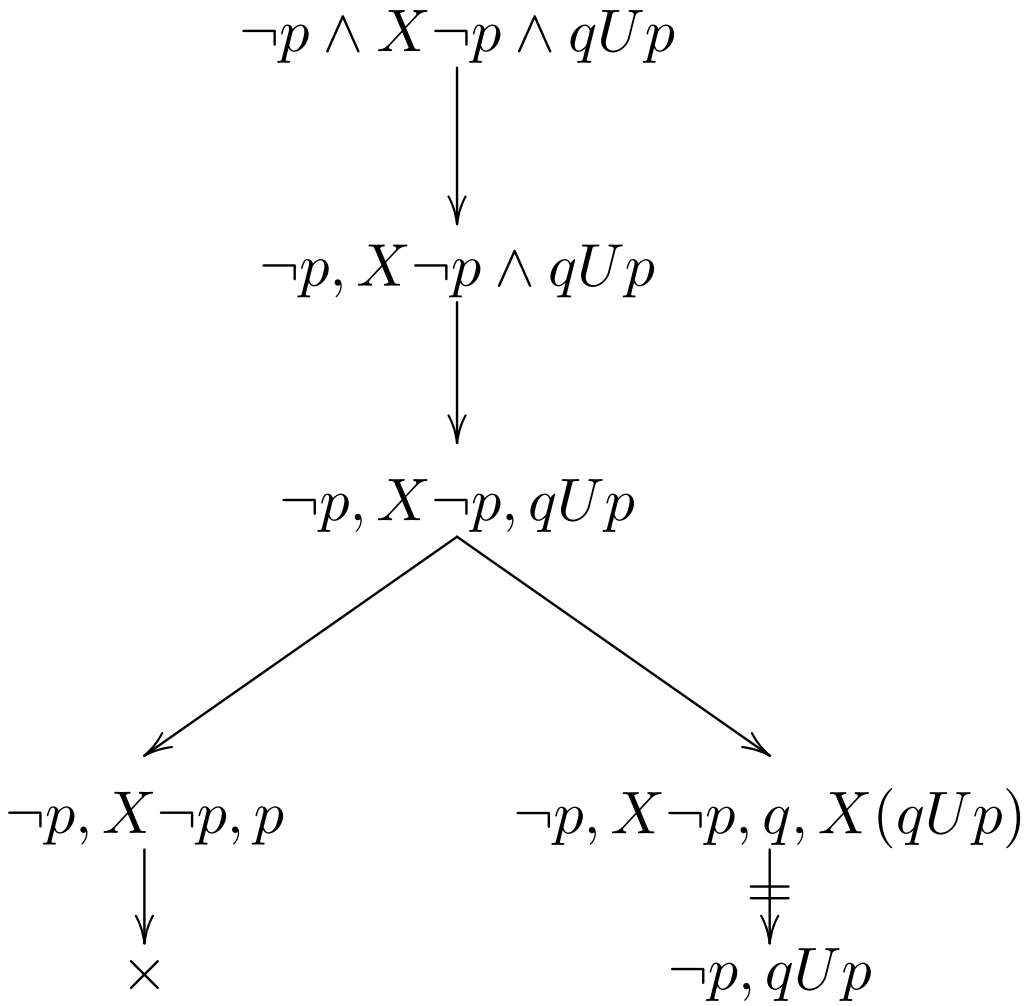
[UNT]: If a node is labelled Γ with some $\alpha U \beta$ in Γ then this node can have two children labelled $\Gamma \cup \{\alpha, X(\alpha U \beta)\}$ and $\Gamma \cup \{\beta\}$ (provided neither is Γ itself).

[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 the try to use the static rules again.

Example:

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

Homework.

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 $\not\not$).
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.

And that's all for the second lecture.

See you next Tuesday.