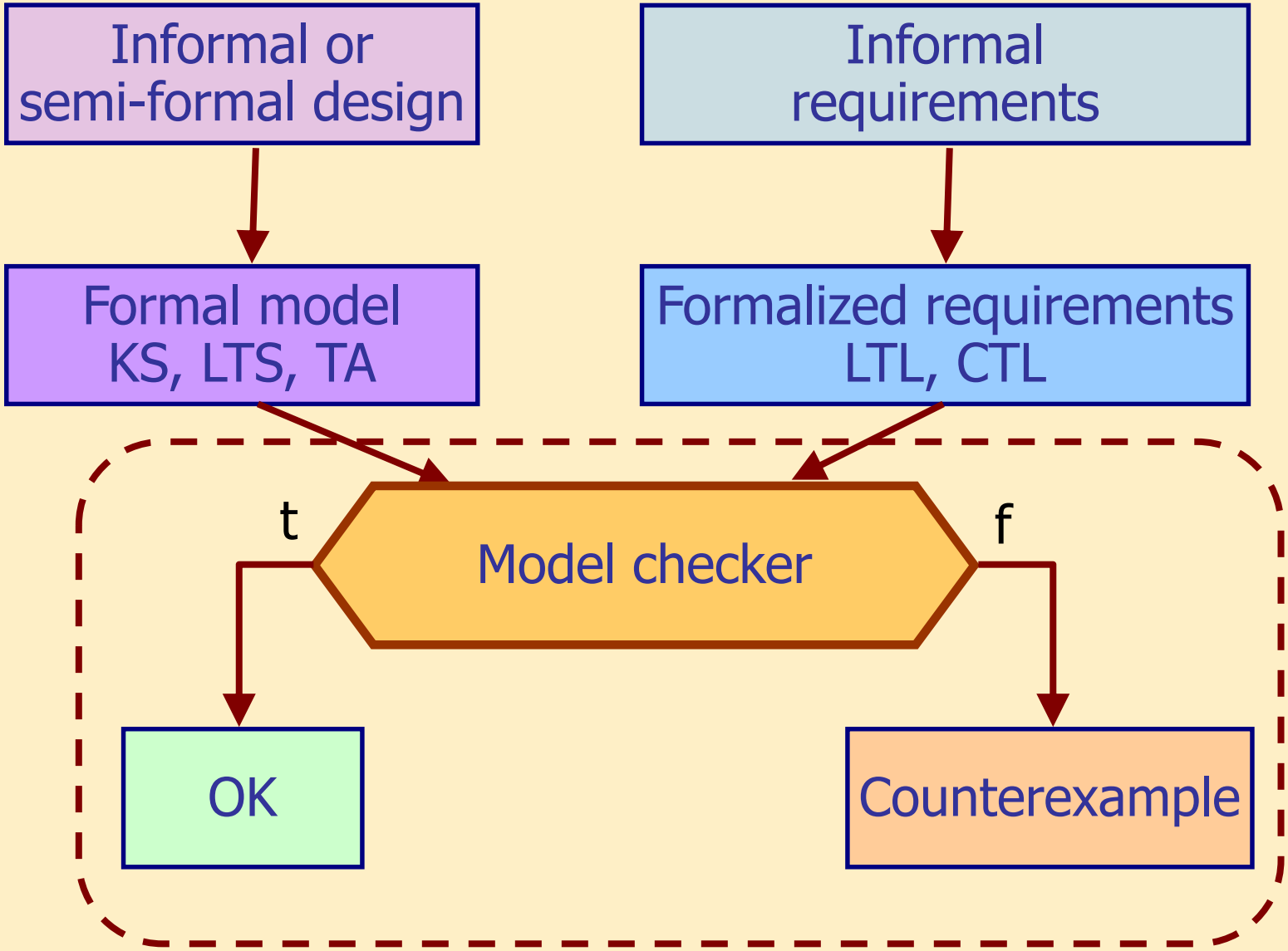
Model Checking

dr. István Majzik
BME Department of Measurement and Information Systems

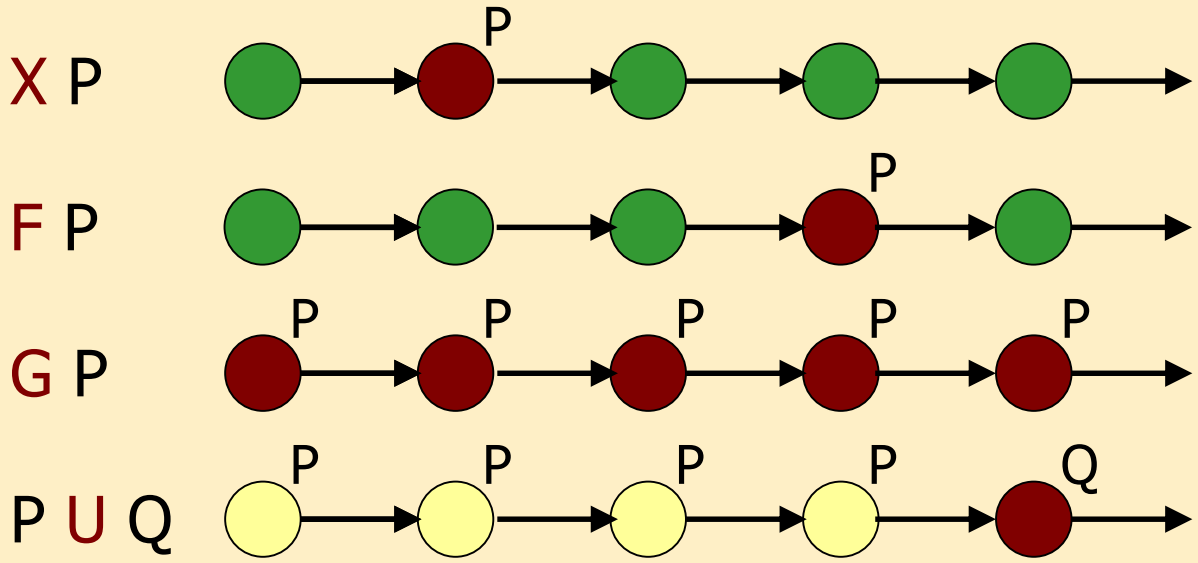
Our goal


Informal or semi-formal design
Informal requirements
Formal model KS, LTS, TA
Formalized requirements LTL, CTL
t
Model checker
f
OK
Counterexample

Recap: linear temporal logic LTL

Elements of LTL:

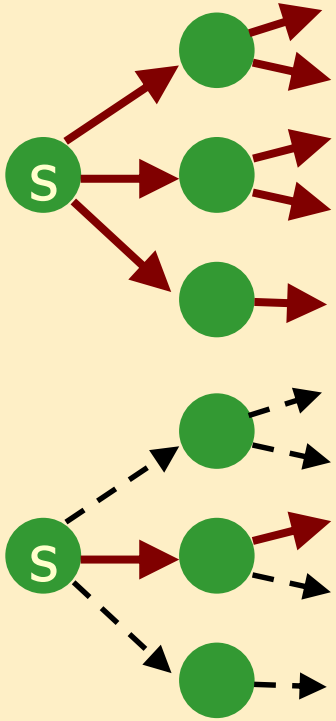
- Atomic propositions (elements of AP): P, Q, ...
- Boolean connectives: $\wedge$, $\vee$, $\neg$, $\Rightarrow$
  $\wedge$: conjunction, $\vee$: disjunction, $\neg$: negation , $\Rightarrow$: implication
- Temporal connectives: X, F, G, U:



Recap: branching time temporal logic CTL*

Elements of CTL*:

- Path quantifiers:
  - A: for All paths
    starting from the current state
  - E: there Exists a path
    starting from the current state
- Path-specific operators (as in LTL):
  - X p, F p, G p, p U q



Recap: branching time temporal logic CTL

Elements of CTL:

Composite operators over states

- EX p: there exists a path where p holds in the next state
- EF p: there exists a path where p holds in the future
- EG p: there exists a path where p holds globally
- E(p U q): there exists a path where p holds until q eventually holds

- AX p: for all paths p holds in the next state
- AF p: for all paths p holds in the future
- AG p: for all paths p holds globally
- A(p U q): for all paths p holds until q eventually holds

Overview

Mechanics of model checking

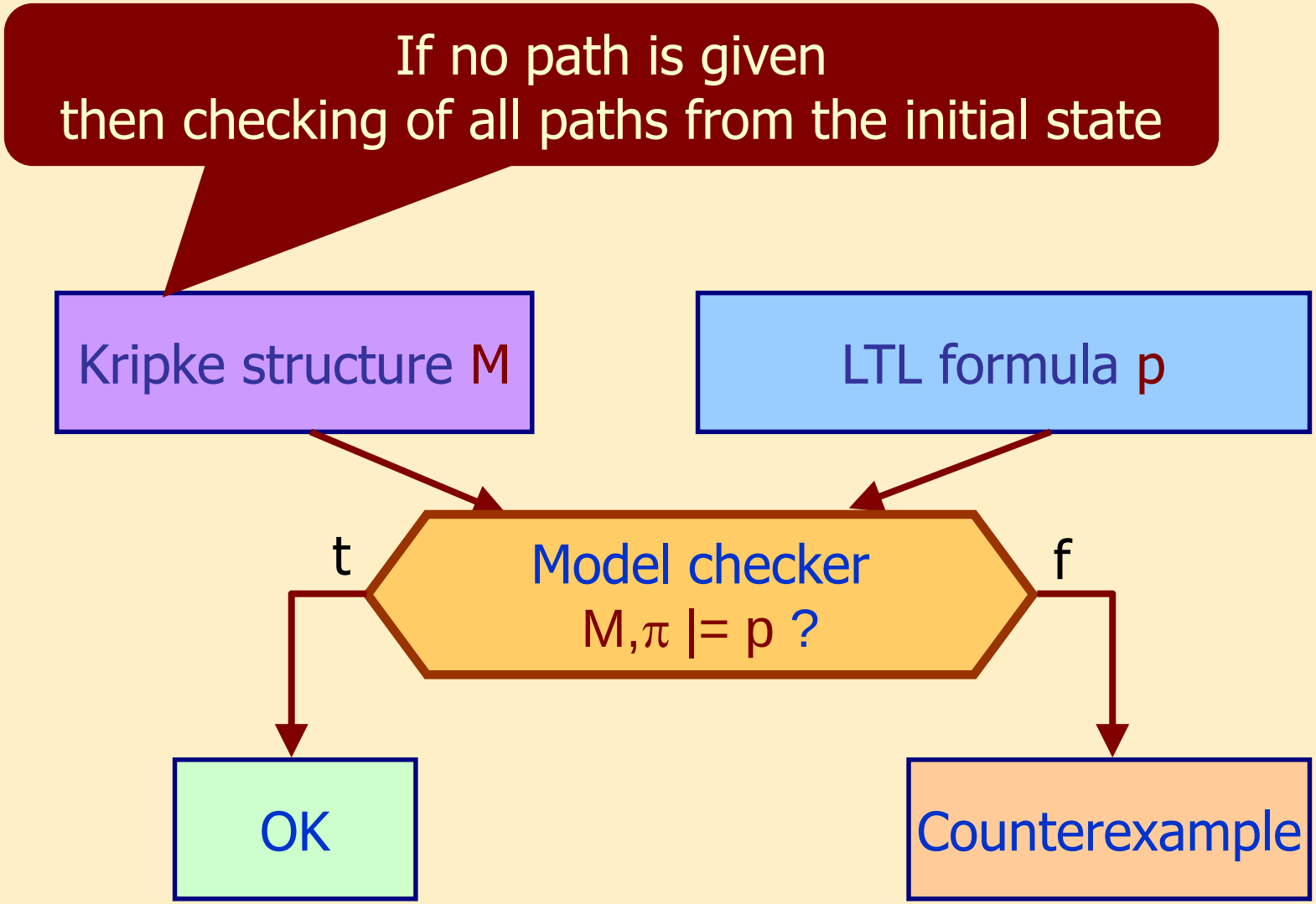
- Techniques for model checking
  - LTL: Semantic tableau
  - CTL: Labeling

Why is this useful?

- Possibilities, determining boundaries
  - Discovering boundaries (e.g. size of verifiable models)
  - Efficient implementation (10^{69000} states? – next lecture)
- Interesting applications (later)
  - Automatic test case generation
  - Synthesis of runtime monitors

LTL Model Checking using Semantic Tableau

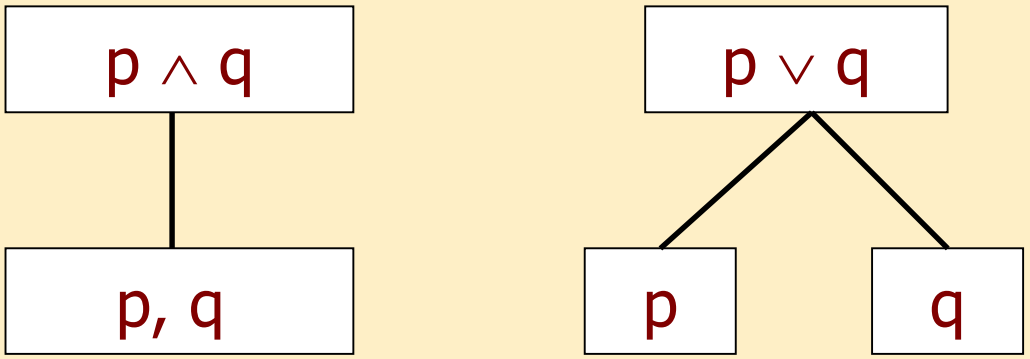
LTL model checking



Introuduction: Semantic Tableau for Propositional Logic

Problem: satisfiability in propositional logic

- Idea: decomposition of the formula to a tree (the tableau)
  - Nodes: formulas to satisfy
  - Adding edges: decomposition rules based on the semantics of connectives
    Branching: more than one ways to satisfy a formula
- Before decomposition: negation normal form (NNF):
  negation only appears on atoms
  - de Morgan's law: $\neg(p \vee q)=(\neg p)\wedge(\neg q)$, $\neg(p \wedge q)=(\neg p)\vee(\neg q)$
- Decomposition rules for PL:



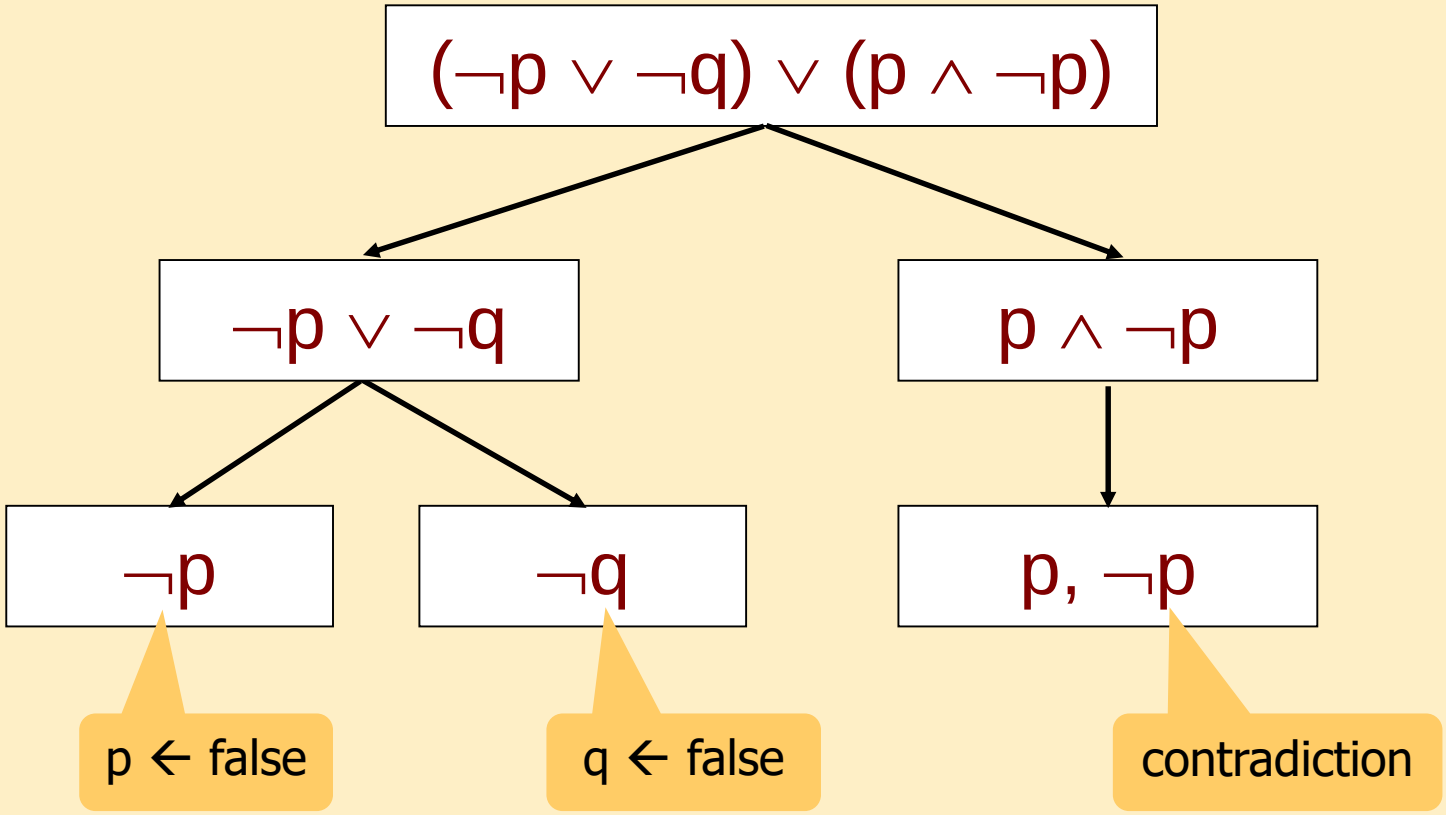
Introduction: Semantic Tableau for Propositional Logic

When to stop decomposing?

- Terminating a branch:
  - Only literals left
  - Each literal has to be satisfied by assigning values to variables
- After terminating a branch:
  - Contradiction: opposite literals
    - E.g. p, $\neg$p is contradicting, no possible satisfying assignment
  - Successful branch: no contradiction
    - E.g.: for p, $\neg$q: p ← true, q ← false
    - This assignment is a model of the original formula
- Each successful branch corresponds to a satisfying assignment

Introduction: An example tableau for PL

- Original formula: $\neg(p \wedge q) \vee \neg(\neg p \vee p)$
- Pushing $\neg$ inwards: $(\neg p \vee \neg q) \vee (p \wedge \neg p)$
- Tableau construction:



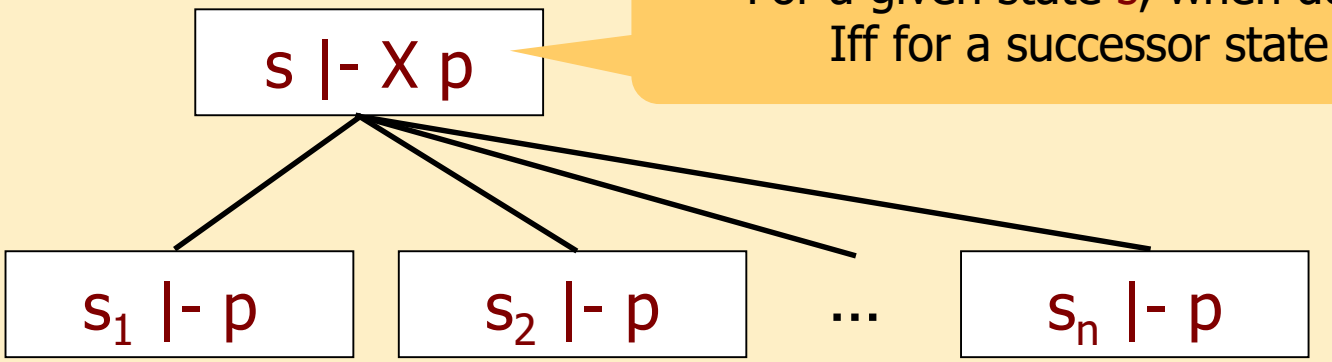
Generalizing tableau construction to LTL

- Model checking: searches for a counterexample, thus

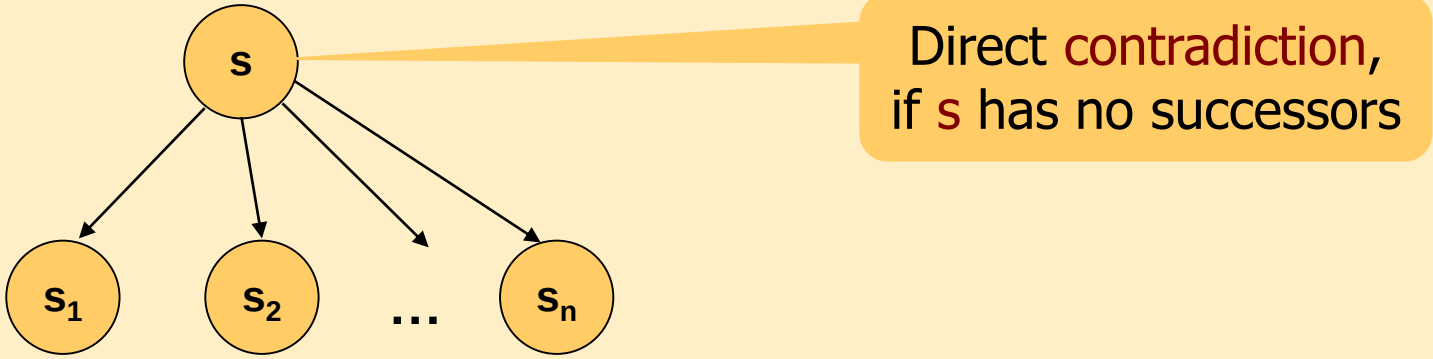
  The tableau is constructed for the negated formula!

  - The negated formula is transformed to NNF
  - If there exists a successful (not contradicting) branch, it induces a counterexample!
  - If all branches are contradicting, then the original property holds!
- Decomposition rules for temporal connectives
  - Novelty: Decomposition is performed based on the model
  - Notation: s |- p denotes that we evaluate p starting from state s
- Handling literals:
  - s |- P holds iff $P \in L(s)$
  - s |- $\neg P$ holds iff $P \notin L(s)$
- Temporal operators:
  - Rules for X and U are sufficient (others can be derived)

Decomposition for operator X



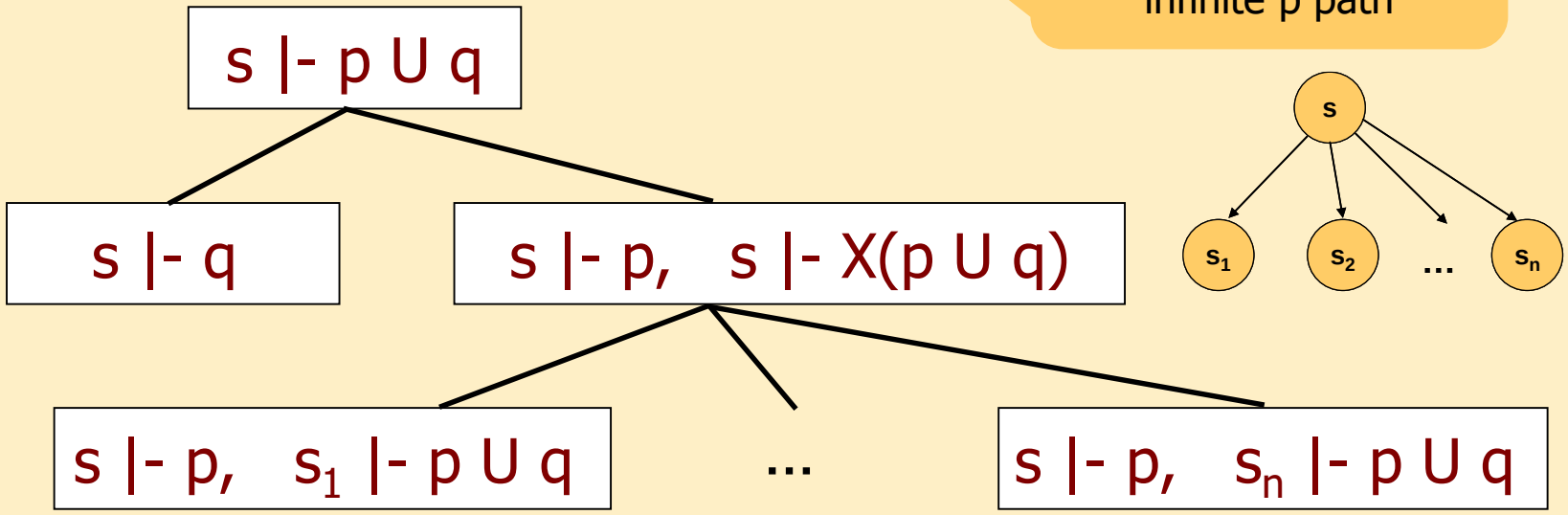
For model:



Decomposition for operator U

- We use: $p \; U \; q = q \vee (p \wedge X \; (p \; U \; q))$

Needs extra attetnion:
finite path,
infinite p path

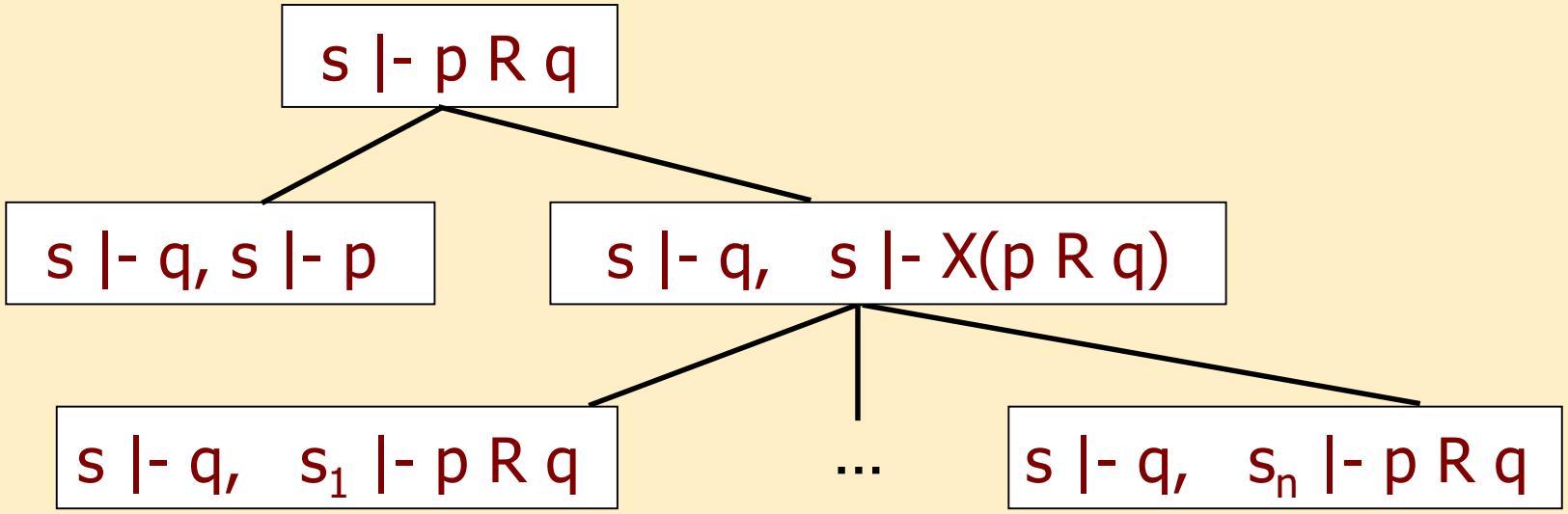


- When can we terminate?
  - Contradiction:
    - Atomic propositions contradict each other
    - Operator X – the path terminates without encountering q
    - Cycle of p states without encountering q
  - Successful branches:
    - Atomic propositions can be satisfied
    - Cycle without contradiction

A special operator: R

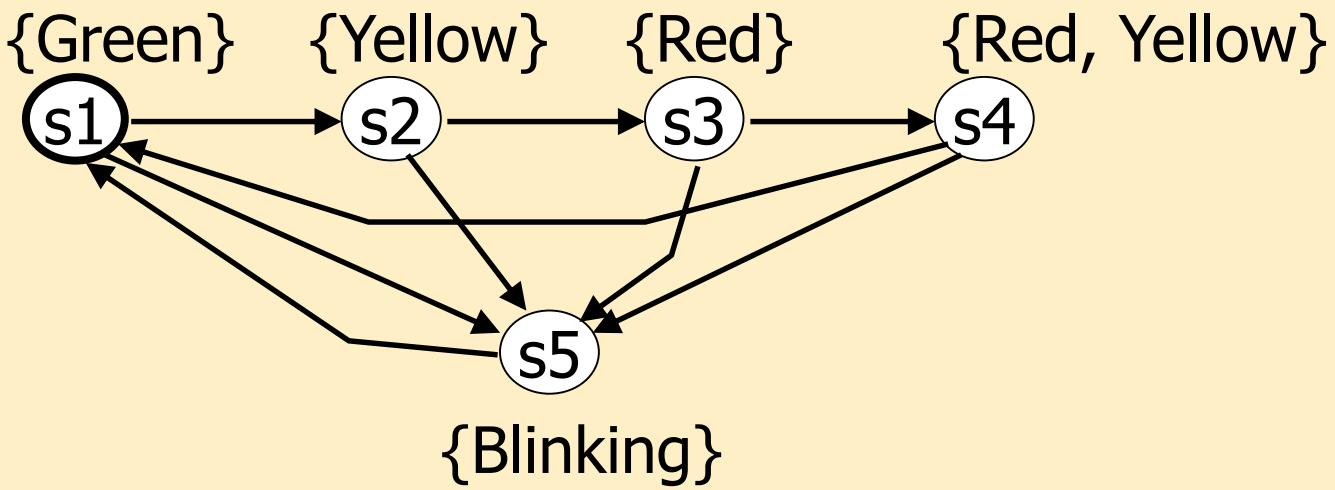
- NNF for operator U:
  $\neg(p\ U\ q) = ?$
  - We introduce the dual of operator U: R (Release)

    $$\neg(p\ U\ q) = (\neg p)\ R\ (\neg q)$$
  - We use: $p\ R\ q = q \wedge (p \vee X\ (p\ R\ q))$
- The tableau for operator R:



An example

- Traffic light (KS)
- Is it true that if initially Green holds, then eventually Red will hold?
  - The formula to check: Green $\Rightarrow$ F Red



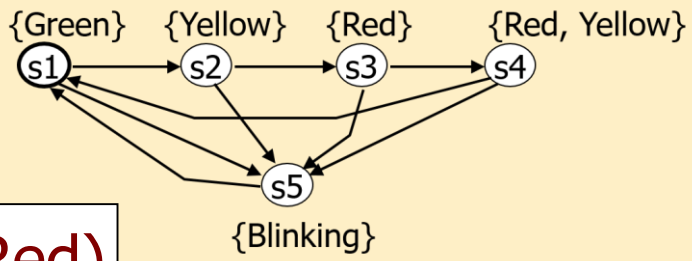
- Based on the model, can we construct a counterexample?

The tableau for the property

- Negation of the formula: $s_1 \vdash \neg(\text{Green} \Rightarrow \text{F Red})$
- NNF (based on $P \Rightarrow Q = \neg P \vee Q$):

$\neg(\text{Green} \Rightarrow \text{F Red}) = \text{Green} \wedge \neg\text{F Red} = \text{Green} \wedge \text{G}(\neg\text{Red})$

- Tableau construction:



S1 is labeled Green

s1 |- Green ∧ G(¬Red)

s1 |- Green, s1 |- G(¬Red)

Simplification: s1 |- Green removed

s1 |- G(¬Red)

The tableau for the property (cont.)

s1 is not labeled Red

s1 is followed by s2 and s5

s2 is not labeled Red

s2 is followed by s3 and s5

Contradicting branch, s3 is labeled Red

s1 |- G(¬Red)

s1 |- ¬Red, XG(¬Red)

s1 |- XG(¬Red)

s2 |- G(¬Red)

s2 |- ¬Red, XG(¬Red)

s2 |- XG(¬Red)

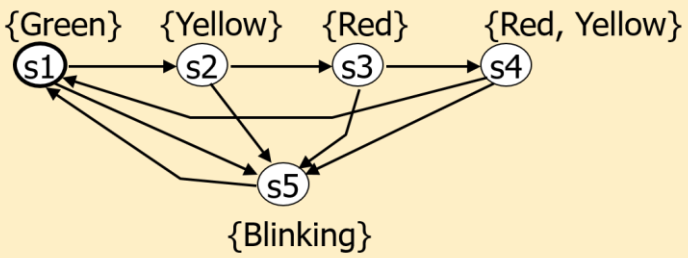
s3 |- G(¬Red)

s3 |- ¬Red, XG(¬Red)

s5 |- G(¬Red)

s5 |- ¬Red, XG(¬Red)

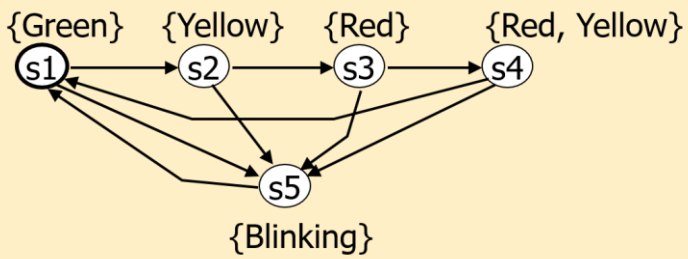
s5 |- XG(¬Red)



Cycles without contradiction
s1, s5, …
s1, s2, s5, …

The results of model checking

- The results of tableau for the negated formula:
  - A contradicting branch (here the property holds)
  - Two cycles without contradiction: counterexamples
- Conclusions:
  - There are executions where the negated property holds:
    Cycle 1: s1, s2, s5, …
    Cycle 2: s1, s5, …



- The original formula Green $\Rightarrow$ F Red thus fails
  - Counterexamples can be shown

Semantic tableau (summary)

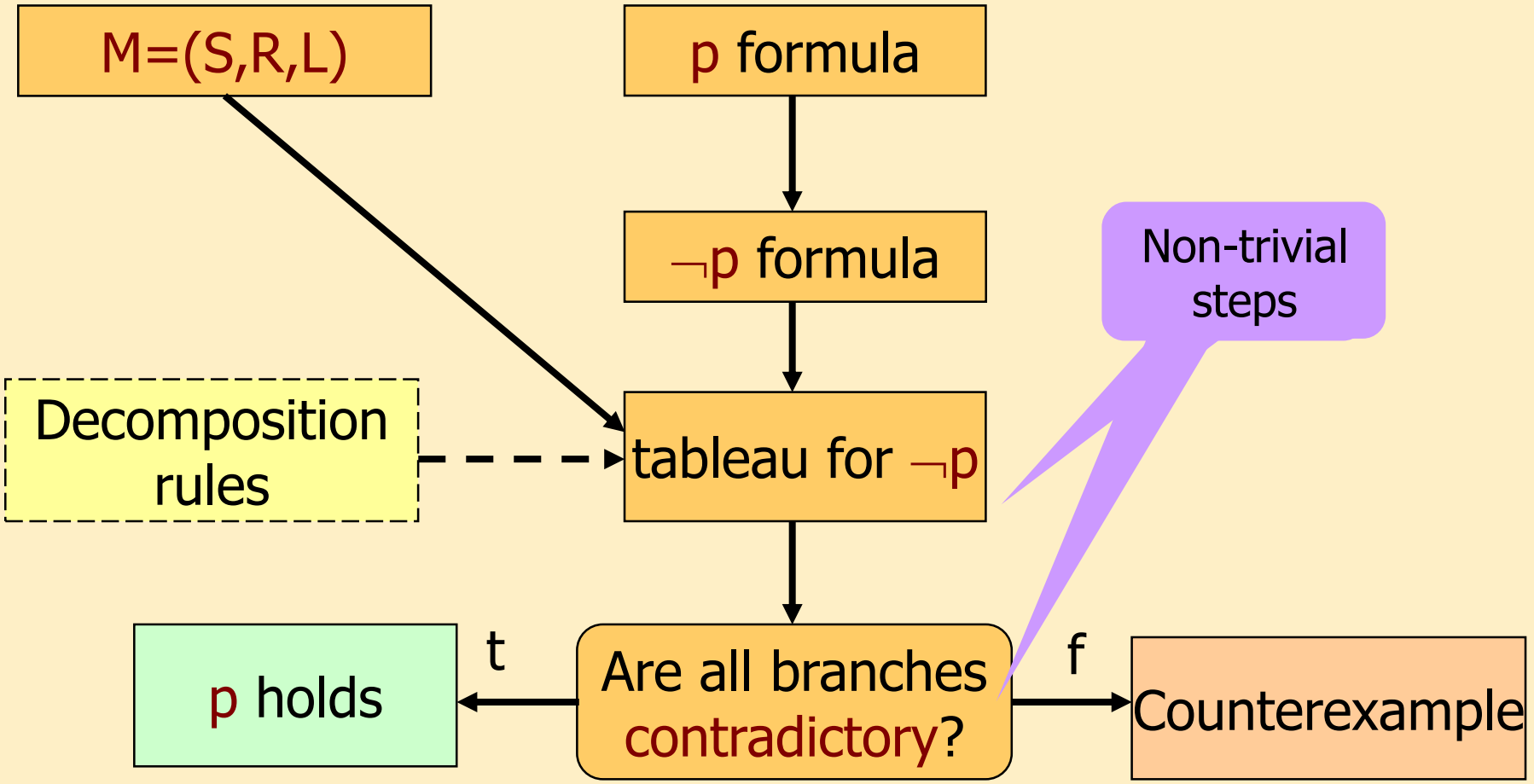
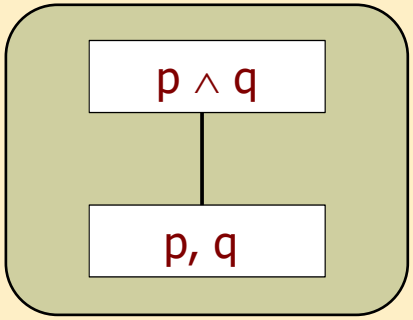
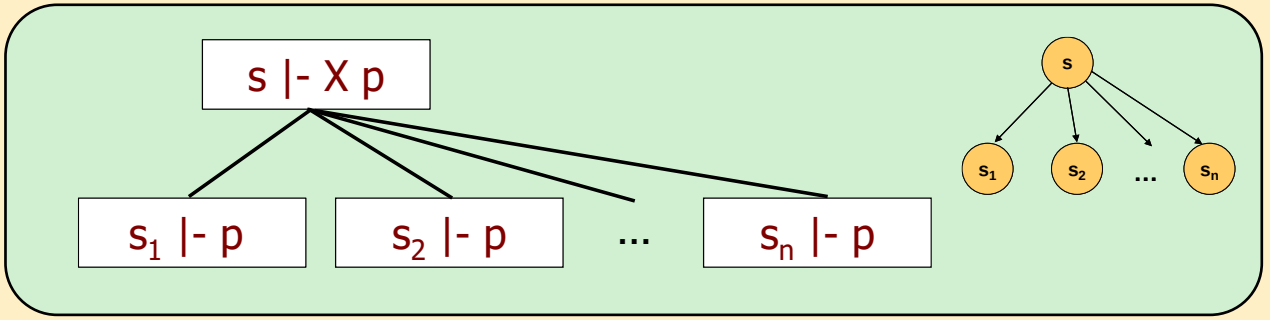
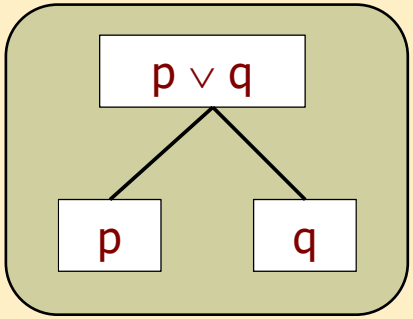
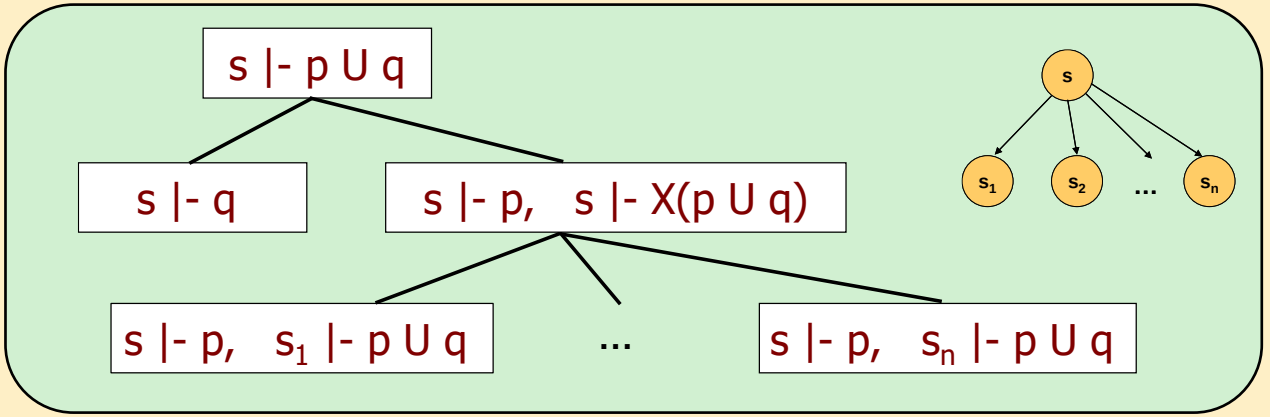


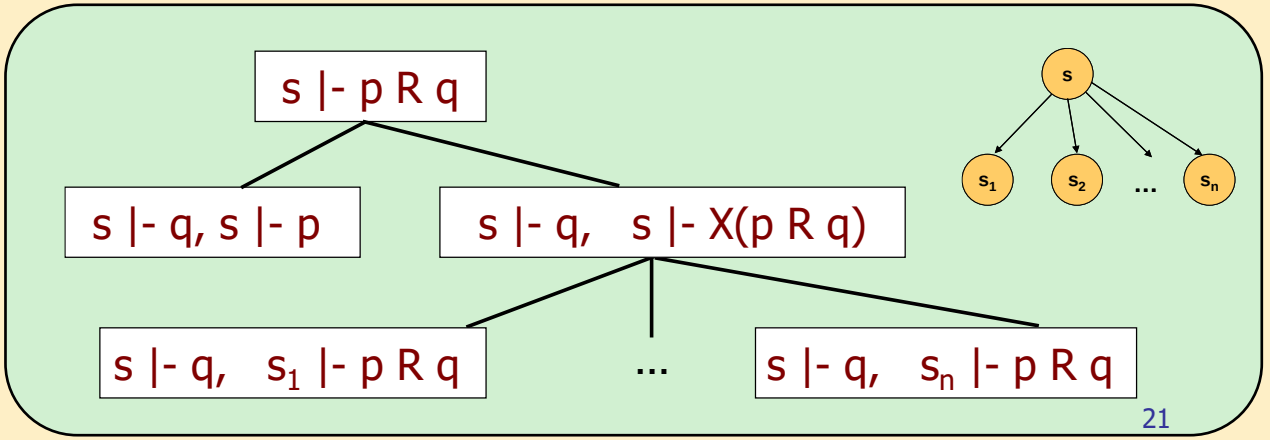
Tableau construction rules (summary)





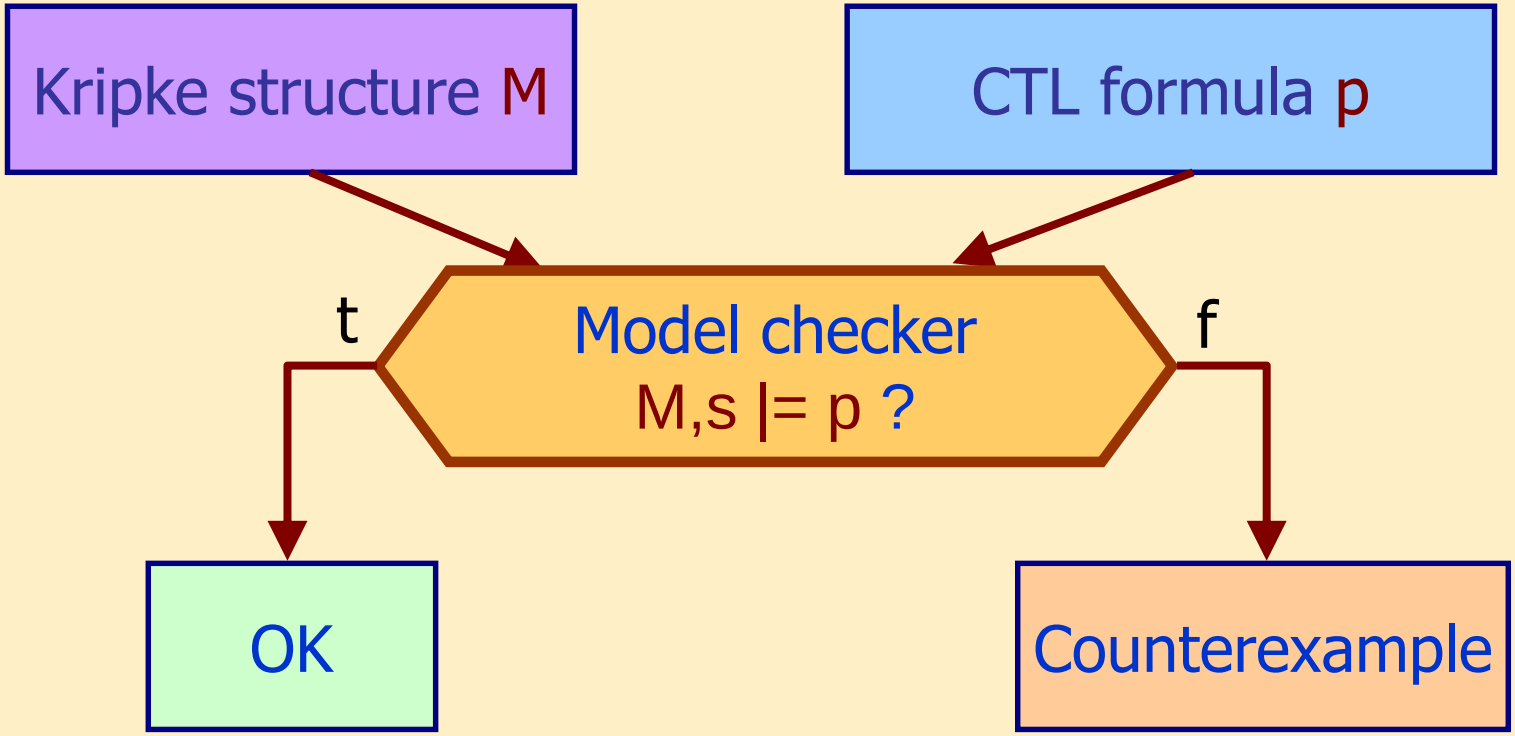






CTL Model Checking Based on Labeling

CTL model checking

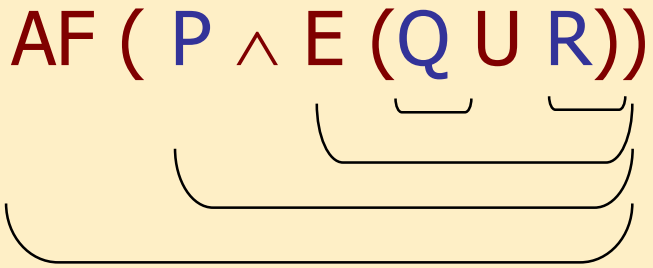


Idea: Labeling of states

- Global model checking:
  - Notation: Sat(p) denotes the set of states where CTL formula p holds
  - Labeling: we label these states by p
  - This way $s \in Sat(p)$ can be easily evaluated for a given state s (in particular for initial states): by checking whether it's labeled p
- The labeling, that is, Sat(p), is computed incrementally
  - We start from the labeling function L, and then expand it
  - The end of the iteration: fixed point reached

CTL model checking with state labeling

- Labeling of states: where the formula holds
- Labeling with complex formulas?
  - Decomposition of the formula based on its structure,
    and computing Sat() for subformulas (from the inside outwards):



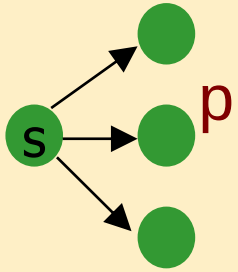
- Algorithm based on the decomposition of the formula:
  - Base case: KS is labeled by atomic propositions
  - Continuation: labeling with more complex formulas
  - Rules: if we have established labels p and q
    then we can establish where we have labels
    $\neg p$, $p \wedge q$, EX p, AX p, E(p U q), A(p U q)
    This way we progress outwards from the inside of a complex formula

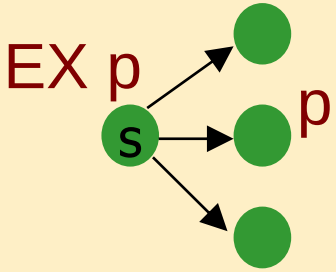
Rules: Atomic propositions and Boolean connectives

- P holds in a state s iff $P \in L(s)$
  - Here, P is already a label of s
- $\neg P$ holds in a state s iff $P \notin L(s)$
  - These states can be labeled $\neg P$
- $p \wedge q$ holds in a state s where p and q holds
  - A state can be labeled $p \wedge q$ iff it is already labeled p and q
- Temporal operators: EX, AX, E(U), A(U)
  - More complex labeling rules

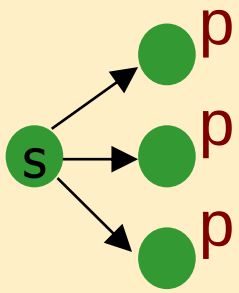
Rules: AX, EX

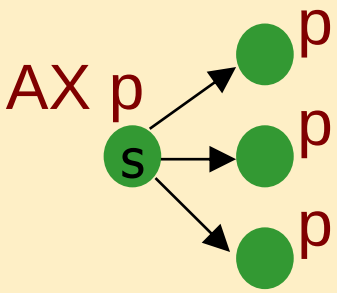
- EX p holds in a state s iff it has a successor where p holds
  - A state can be labeled EX p iff it has a successor labeled p





- AX p holds in a state s iff for all its successors p holds
  - A state can be labeled AX p iff all its successors are labeled p

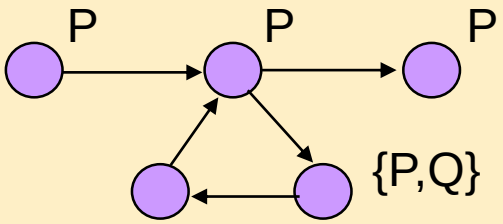




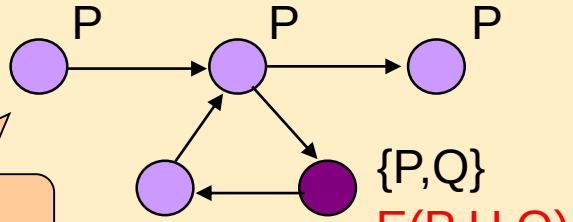
Rules: E(p U q)

- Where does E(p U q) hold?
  - We use: $E(p\ U\ q) = q \vee (p \wedge EX\ E(p\ U\ q))$
  - "Recursive" formula
- So when can a state s be labeled E(p U q)?
  - if s is labeled q, or
  - if s is labeled p and there is at least one succeeding state (EX) that is already labeled E(p U q)
- An iteration arises:
  - States labeled q are the states where label E(p U q) first appears
  - We consider the predecessors of these states:
    If it is labeled p, we can add label E(p U q)
  - This way we traverse those paths backwards that lead to states labeled q through states labeled p

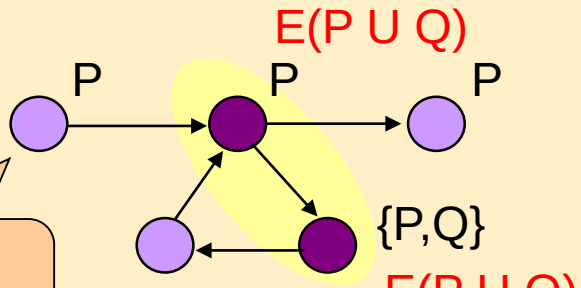
Labeling by E(P U Q)



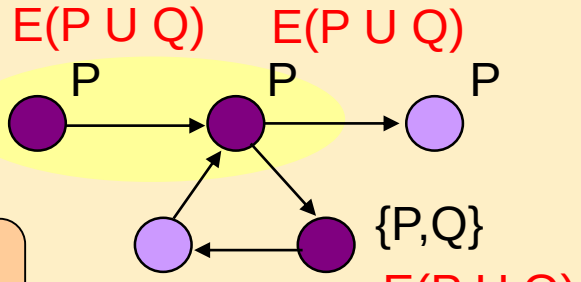
Kripke structure with initial labeling



First step: Q



Second step: "P∧ EX"



Third step: "P∧ EX"

- We iterate until a fixpoint is reached

Rules: A(p U q)

- Where does A(p U q) hold?
  - We use: $A(p\ U\ q) = q \vee (p \wedge AX\ A(p\ U\ q))$
  - "Recursive" formula
- So when can a state s be labeled A(p U q)?
  - if s is labeled q, or
  - if s is labeled p and all succeeding states (AX) are already labeled A(p U q)
- An iteration arises:
  - States labeled q are the states where label A(p U q) first appears
  - We consider the predecessors of these states:
    If it is labeled p, and all its successors are labeled A(p U q), we can add label A(p U q)

This way we covered all operators defined in the syntax.

An additional rule: AF p

- Where does AF p hold?
  - We use: AF p = p $\vee$ AX AF p
  - "Recursive" formula
- So when can a state s be labeled AF p?
  - if s is labeled p, or
  - all its successors (AX) are labeled AF p
- An iteration arises:
  - States labeled p are the states where label AF p first appears
  - We consider the predecessors of these states:
    If all its successors are AF p, we can add label AF p
  - This way we traverse those paths backwards that lead to a state labeled p

Iteration using set operations

- We expand the labeling using operations on sets
  - Initial set: states already labeled by subformulas
  - Expanding the labeling:
    - E(p U q): "At least one successor is labeled …"
    - A(p U q): "All successors are labeled …"
  - This way we can label preceding states
- How can we define the set of preceding states?
  - Based on set of already labeled states Z:

    $pre_E(Z) = \{s \in S \mid \text{there exists } s' \text{ such that } (s,s') \in R \text{ and } s' \in Z\}$

    $pre_A(Z) = \{s \in S \mid \text{for all } s' \text{ such that } (s,s') \in R \text{ we have } s' \in Z\}$

At least one successor is labeled

All successors are labeled

- Example: E(P U Q):
  - Initial set: $Z_0 = \{s \mid Q \in L(s)\}$
  - Expansion: $Z_{i+1} = Z_i \cup (pre_E(Z_i) \cap \{s \mid P \in L(s)\})$

Labeled so far

Predecessors of already labeled states

labeled P

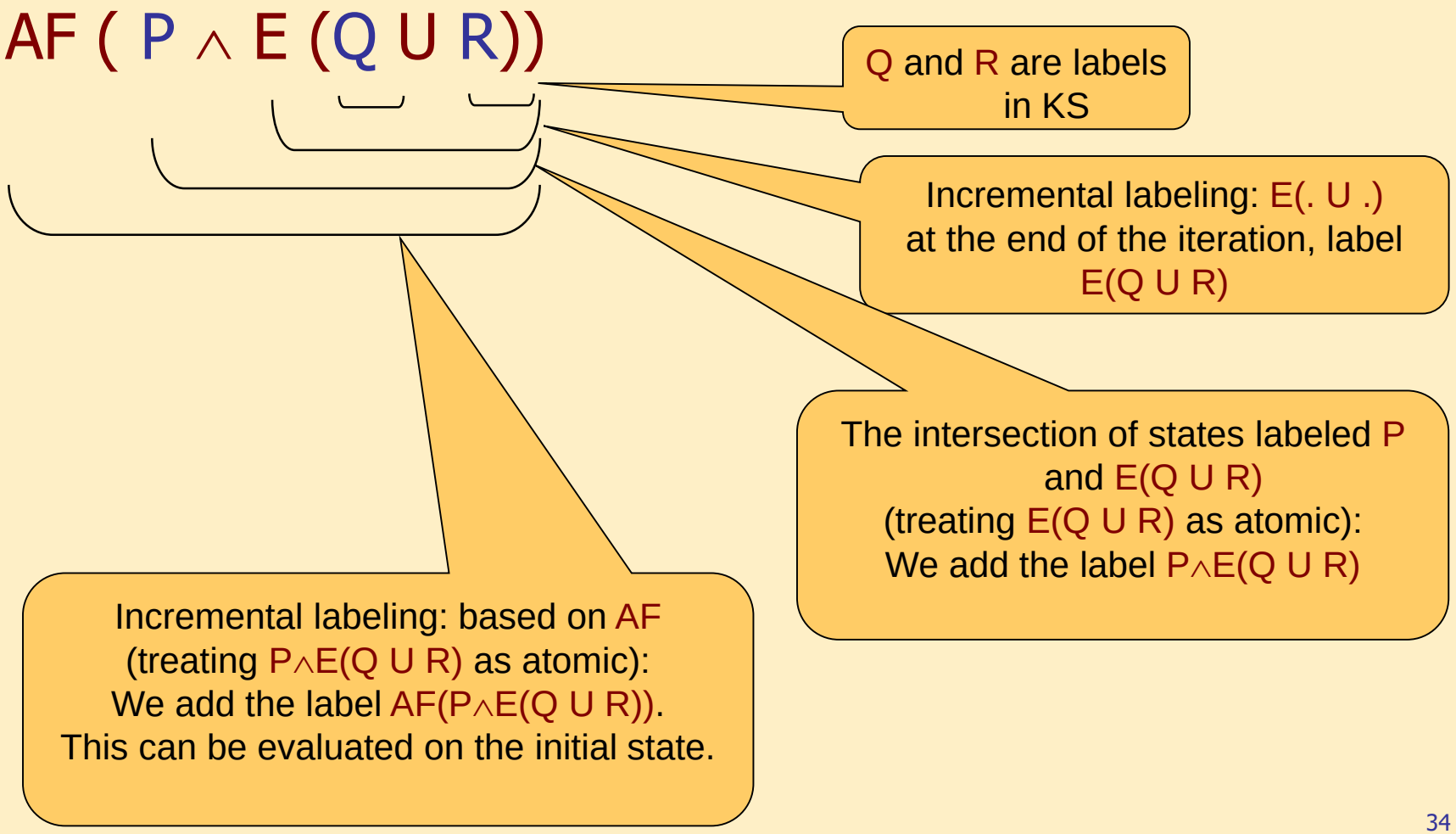
  - End of the iteration: if $Z_{i+1} = Z_i$ (fixedpoint)

CTL model checking – summary

- Global model checking:
  - Labeling of states by (sub)formulas that hold in the state
- Labeling by increasingly complex formulas
  - Starting from atomic formulas to more complex formulas, from the inside outwards
  - Using the labeling obtained in the previous iteration based on rules derived from operator semantics
- EX, AX: Examining and labeling predecessors
- E(p U q), A(p U q): Incremental labeling
  - Initial set:
    - State sets determined by the innermost formulas (p, q)
    - Iteration: based on semantics (applied to predecessors)
  - End of iteration: no more labels can be added to the labeling

Example

- Decomposition of formulas:



Exercise

- A traffic light has three aspects: red, yellow and green.
  - Initially all aspects are off.
  - After turning the light on, the red aspect is on.
  - From this, there are two ways to proceed: red-yellow (both are on), and green.
  - Red-yellow is followed by green, and green is followed by red again. From this, the behavior is the same as before.
- Check whether the following formula holds for the initial state of the model: E((¬red) U (EX green))

Summary

- LTL model checking
  - Tableau construction
    - Propositional logic: contradictory and successful branches
    - LTL: searching for a counterexample (witness for negated formula)
- CTL model checking
  - Iterative labeling
    - Incremental labeling with increasingly complex formulas (global model checking)
    - Set operations

How can these algorithms be implemented efficiently?

LTL model checking: Automata theoretic approach

(Supplementary)

Automata for finite words

- $A=(\Sigma, S, S_0, \rho, F)$ where
  - Σ is the alphabet, S are states, S_0 are initial states
  - ρ is the transition relation, $\rho: S \times \Sigma \rightarrow 2^S$
  - F is the set of accepting states
- A run of the automaton:
  - For a sequence of symbols from the alphabet – a word $w=(a_0, a_1, a_2, \ldots a_n)$ – a sequence of states $r=(s_0, s_1, s_2, \ldots s_n)$
  - r is an accepting run iff $s_n \in F$
  - Word w is accepted iff there exists an accepting run
- $L(A)=\{ w \in \Sigma^* \mid w \text{ is accepted} \}$ the language accepted by the automaton

Automata on infinite words

- Application: continuously operating systems
  - No final state – can not be checked for acceptance
- Büchi acceptance condition:
  - For a word $w=(a_0, a_1, a_2, \dots)$
    a sequence of states $r=(s_0, s_1, s_2, \dots)$
  - $\lim(r) = \{s \mid s$ occurs infinitely many times,
    that is, there is no j such that $\forall k>j: s \neq s_k\}$
  - A run is accepting iff $\lim(r) \cap F \neq 0$
  - A word w is accepted iff there exists an accepting run over it
    (an accepting state is encountered infinitely many times)
- $L(A)=\{w \in \Sigma^* \mid w$ accepted$\}$
  the language accepted by the automaton

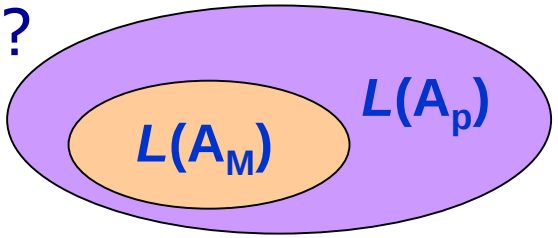
Automata theoretic approach

- For a state s of KS: L(s) is a symbol of alphabet 2^{AP}
  E.g. {Red, Yellow} is a symbol of the alphabet
- A path $\pi=(s_0, s_1, s_2, \ldots s_n)$ induces a word
  $(L(s_0), L(s_1), L(s_2), \ldots L(s_n))$

- We construct two automata:
  - Based on Kripke structure M=(S,R,L) an automaton A_M can be constructed that accepts exactly those words that correspond to paths of M.
  - Based on formula p an automaton A_p can be constructed that accepts exactly those words that characterize paths for which p holds
    Tableau construction rules can be used: what must hold in the current state, and what for the successor states

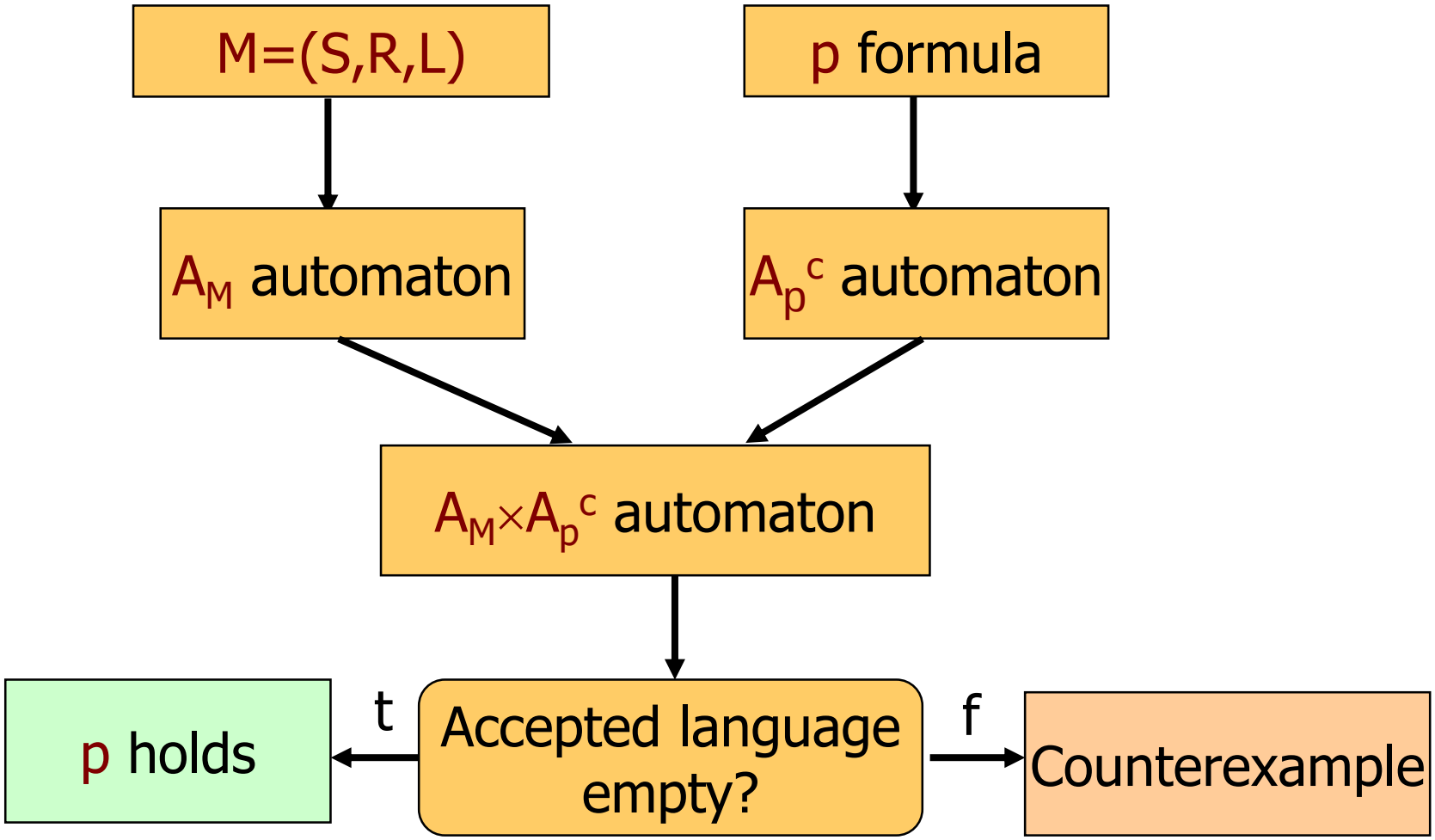
Model checking using automata

- Model checking problem: $L(A_M) \subseteq L(A_p)$, i.e., is the model's language part of the property's language?
  - If so then M |= p



- Reformulating the problem:
  - Checking emptiness of intersection of languages:
    $L(A_M) \cap L(A_p)^c = 0$, here $L(A_p)^c$ is the complement of the language
  - Is the language accepted by the synchronous product automaton $A_M \times A_p^c$, induced by the model automaton A_M and the complement automaton of the property A_p^c , empty?
    - If so then $M, \pi \models p$
    - The accepted language is empty iff there is no reachable accepting state
- Continuously operating systems
  - Automata on infinite words;
    Büchi acceptance condition: searching for loops

Automata theoretic model checking



"On-the-fly" model checking

- Idea:
  - During construction of automaton A_p the synchronous product can be constructed
- Construction of synchronous product automaton
  - Directed by the property to verify:
    as the states of the automaton A_p are established,
    the states of A_M has to be "looked up"
  - The generation of the full state space is not necessary
    - E.g. when deriving from a higher level formalism