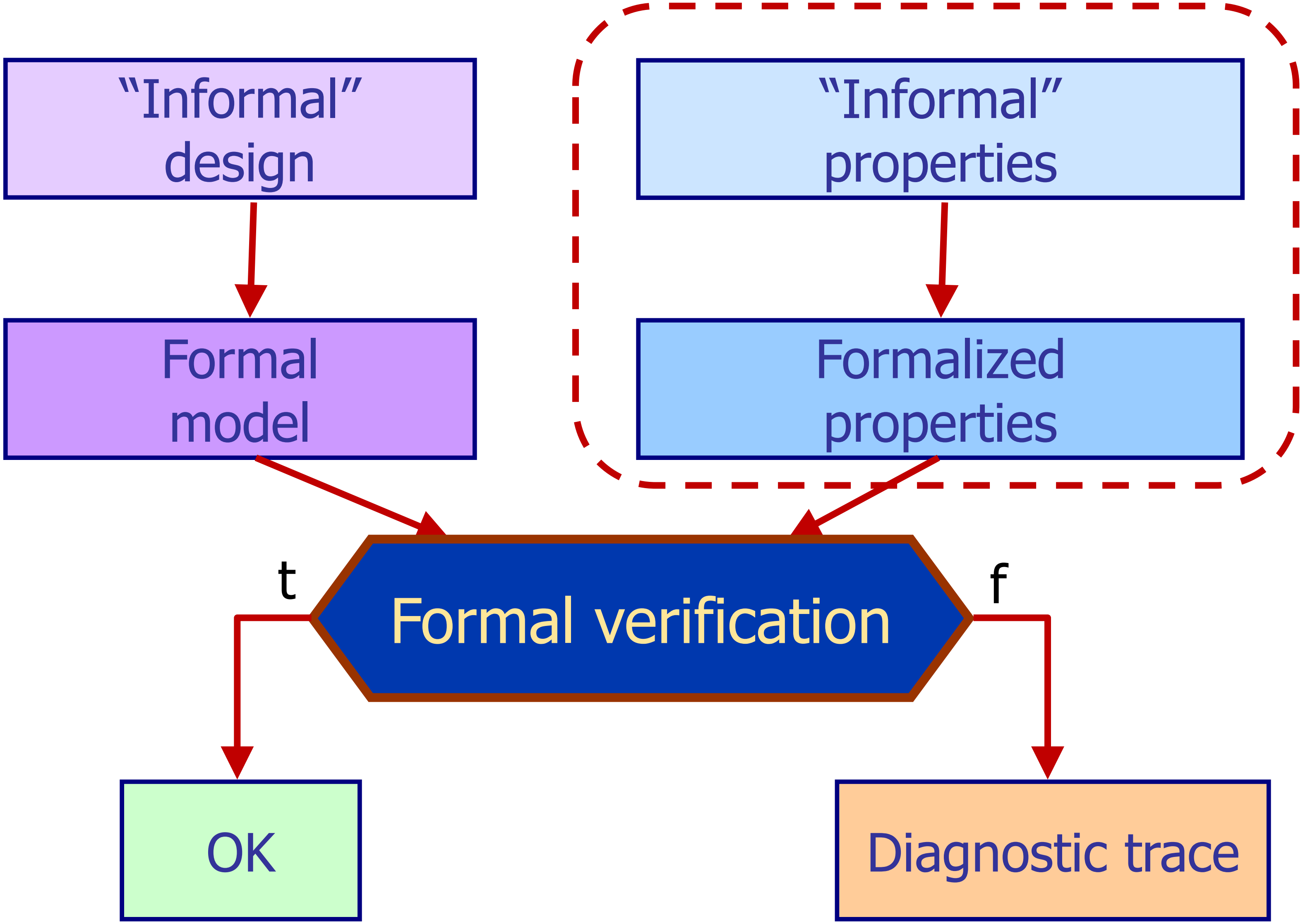
Formalizing and checking properties: Temporal logic LTL

Istvan Majzik
majzik@mit.bme.hu

Budapest University of Technology and Economics
Dept. of Measurement and Information Systems

Recap: Goals of formal verification



Overview

- Temporal operators of LTL
- Formal syntax and semantics of LTL
  - Extending LTL to LTS
- Examples
- Verification of LTL properties
  - The model checking problem
  - LTL model checking: Automata based approach

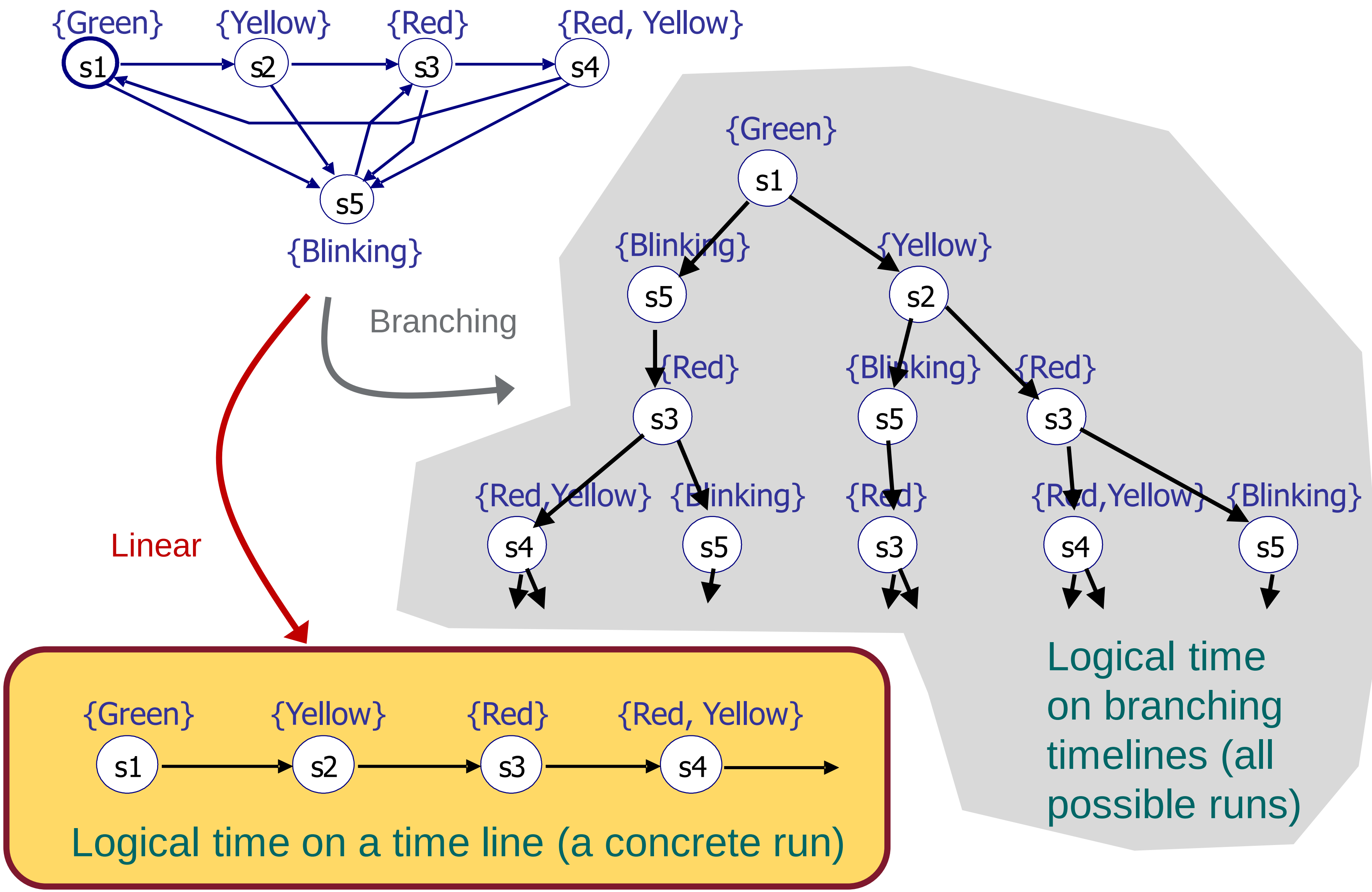
Linear Temporal Logic (LTL)

Temporal operators

Syntax and semantics

Examples

Illustration of linear and branching timelines



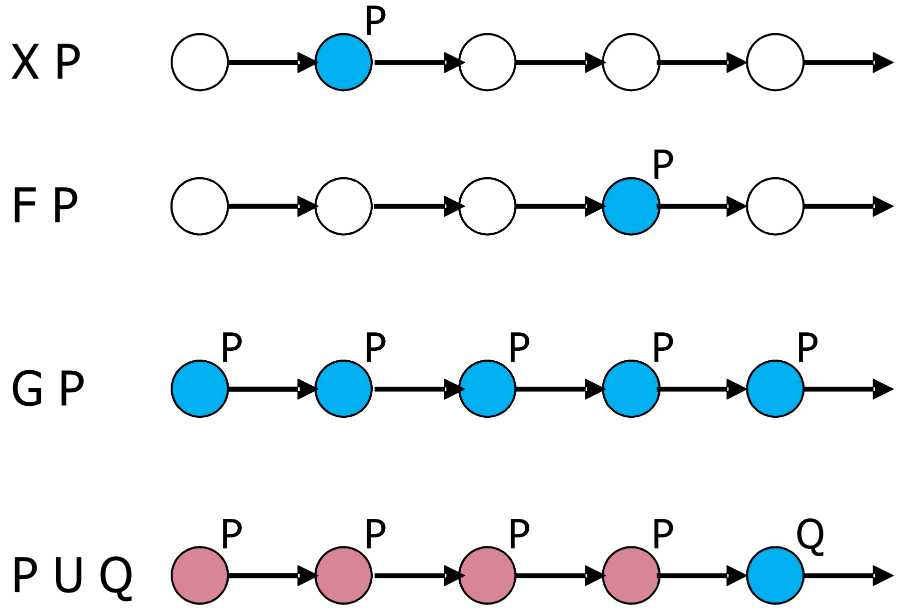
Linear temporal logic – Formulas

Construction of formulas: p, q, r, ...

- Atomic propositions (elements of AP): P, Q, ...
- Boolean operators: ∧, ∨, ¬, ⇒

  ∧: conjunction, ∨: disjunction, ¬: negation , ⇒: implication

- Temporal operators: X, F, G, U informally:
  - X p: "neXt p"
    p holds in the next state
  - F p: "Future p"
    p holds eventually
    on the path
  - G p: "Globally p"
    p holds in all states
    on the path
  - p U q: "p Until q"
    p holds at least until q,
    which holds on the path



LTL examples

- $p \Rightarrow \mathbf{F}q$

  If p holds (in the initial state), then eventually q holds.
  - Example: Start $\Rightarrow$ **F** End
- $\mathbf{G}(p \Rightarrow \mathbf{F}q)$

  For all states, if p holds, then eventually q holds.
  - E.g. : **G** (Request $\Rightarrow$ **F** Reply); for all requests, a reply eventually arrives
- $p\ \mathbf{U}\ (q \vee r)$

  Starting from the initial state, p holds until q or r eventually holds.
  - Example: Requested **U** (Accept $\vee$ Refuse)
    A continuous request either gets accepted or refused
- **GF** p

  Globally along the path (in any state), eventually p holds
  - There is no state after which p does not hold eventually
  - Example: **GF** Start; the Start state is reached from all states
- **FG** p

  Eventually, p continuously holds
  - Example: **FG** Normal
    (After an initial transient) the system keeps operating normally

LTL syntax

Syntax: What are the well-formed formulas (wff)?

The set of well-formed formulas in LTL are given by three syntax rules:

Let $P \in AP$ and p and q be wffs. Then

- **L1**: P is a wff
- **L2**: $p \land q$ and $\neg p$ are wffs
- **L3**: **X** q and p **U** q are wffs

Precedence rules:

$$\mathbf{X}, \mathbf{U} > \neg > \land > \lor > \Rightarrow > \equiv$$

Derived operators

- true holds for all states
  false holds in no state
- $p \lor q$ means $\neg(\neg p \land \neg q)$
  $p \Rightarrow q$ means $\neg p \lor q$
  $p \equiv q$ means $p \Rightarrow q \land q \Rightarrow p$
- **F**p means true **U** p
  Gp means $\neg$**F**($\neg$p)
- "Before" operator:
  p **WB** q = $\neg((\neg p)$ **U** q) (weak before)
  p **B** q = $\neg((\neg p)$ **U** q) $\land$ **F** q (strong before)

Informally:
It is not true that p does not occur until q

Included: q shall occur

LTL semantics – Notation

Rationale of having formal semantics:

- When does a given formula hold for a given model?
  - The semantics of LTL defines when a wff holds over a path
- Allows deciding "tricky" questions:
  - Does F p hold if p holds in the first state of a path?
  - Does p U q hold if q holds in the first state of a path (without p)?

Notation:

- $M = (S, R, L)$ Kripke structure
- $\pi = (s_0, s_1, s_2, \ldots)$ a path of M
  where $s_0 \in I$ and $\forall\, i \geq 0 : (s_i, s_{i+1}) \in R$

  $\pi^i = (s_i, s_{i+1}, s_{i+2}, \ldots)$ the suffix of π from index i
- $M,\pi \models p$ denotes:
  in Kripke structure M, along path π, property p holds

LTL semantics

Defined recursively w.r.t. syntax rules:

- **L1**: $M,\pi \models P$ iff $P \in L(s_0)$
- **L2**: $M,\pi \models p \wedge q$ iff $M,\pi \models p$ and $M,\pi \models q$

  $M,\pi \models \neg q$ iff not $M,\pi \models q$.
- **L3**: $M,\pi \models \mathbf{X}\, p$ iff $\pi^1 \models p$

  $M,\pi \models (p\ \mathbf{U}\ q)$ iff

  $\pi^j \models q$ for some $j \geq 0$ and

  $\pi^k \models p$ for all $0 \leq k < j$

Formalizing requirements: Example

Consider an air conditioner with the following operating modes:

AP={Off, On, Error, MildCooling, StrongCooling, Heating, Ventilating}

- At a time, more than one modes may be active
  - E.g. {On, Ventilating}
- When formalizing requirements, we may not yet know the state space (all potential behaviors)
  - We use only the labels belonging to operating modes

Formalizing requirements: Example

Air conditioner with the following operating modes:

AP = {Off, On, Error, MildCooling, StrongCooling, Heating, Ventilating}

- The air conditioner can (and will) be turned on

  $\mathbf{F}\ On$

- At some point, the air conditioner always breaks down

  $\mathbf{GF}\ Error$

- If the air conditioner breaks down, it eventually gets repaired

  $\mathbf{G}\ (Error \Rightarrow \mathbf{F}\ \neg Error)$

- A broken air conditioner does not heat:

  $\mathbf{G}\ \neg(Error \wedge Heating)$

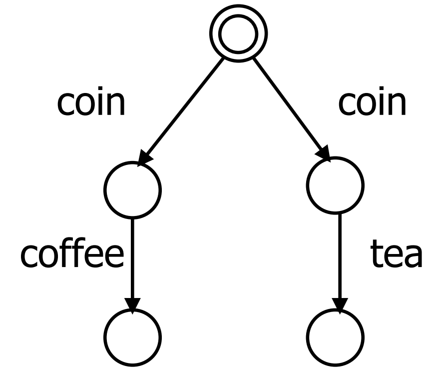
- After finishing the heating, the air conditioner must ventilate:

  $\mathbf{G}\ ((Heating \wedge \mathbf{X}\ \neg Heating) \Rightarrow \mathbf{X}\ Ventilating)$

- After ventilation the air conditioner must not cool strongly until it performs some mild cooling:

  $\mathbf{G}\ ((Ventilating \wedge \mathbf{X}\ \neg Ventilating) \Rightarrow \mathbf{X}(\neg StrongCooling\ \mathbf{U}\ MildCooling))$

Extending LTL for LTS



LTL: Transitions are labeled by actions

A path in LTS is an alternating sequence of states and actions:

- $\pi = (s_0, a_1, s_1, a_2, s_2, a_3, \ldots)$

Extending the syntax:

- **L1***: If $a \in Act$ then (a) is a wff.

The corresponding case in semantics:

- **L1***: $M,\pi \models (a)$ iff. $a_1=a$
  where a_1 is the first action in π.

Requirements for action sequences

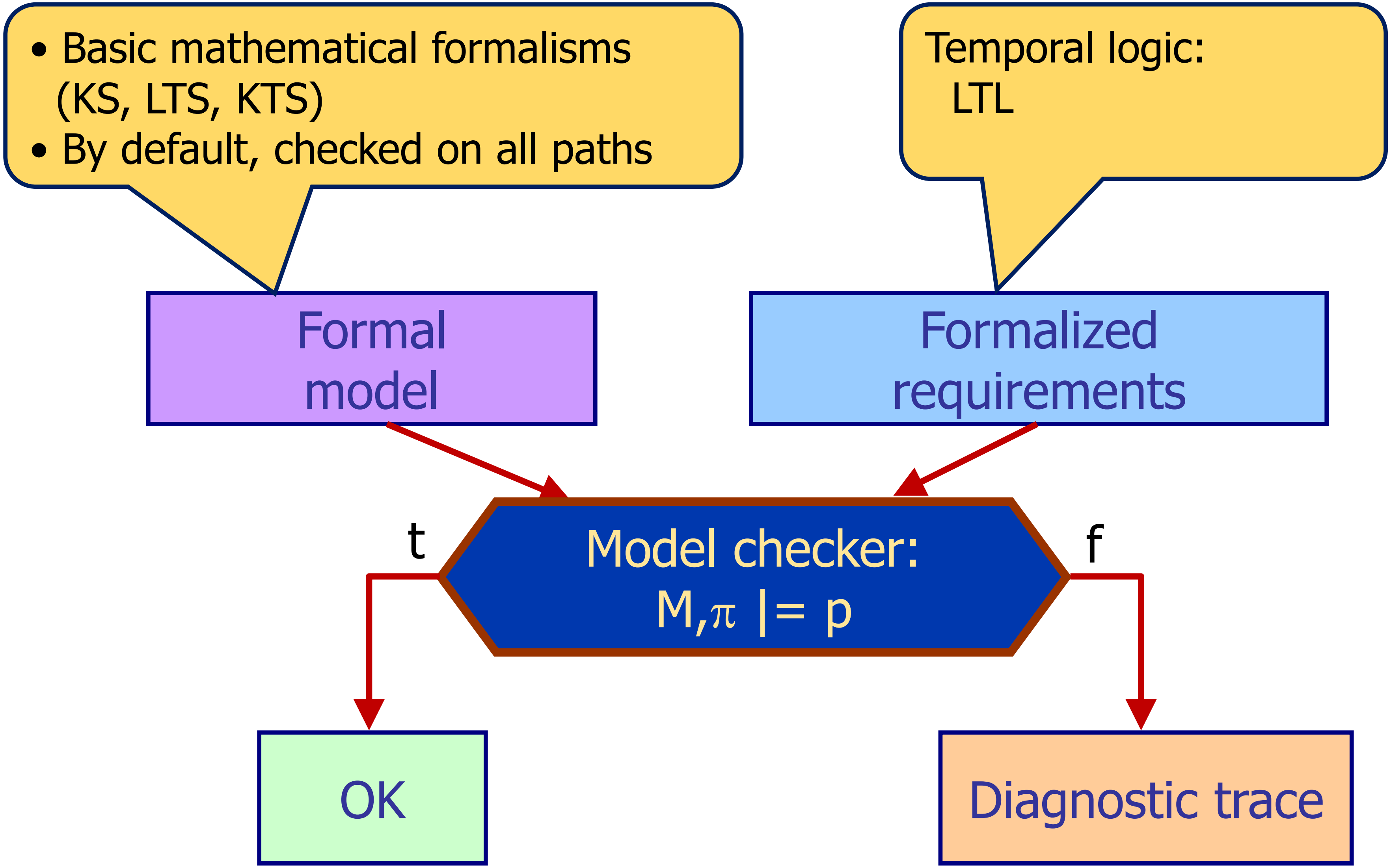
- Example: **G** ((coin) $\Rightarrow$ **X** ((coffee) $\vee$ (tee)))

Verification of LTL properties

The model checking problem

LTL model checking: The automata-based approach

Model based verification by model checking

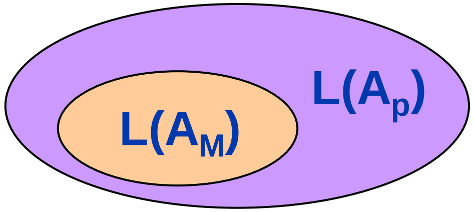


Automata based approach

- $A=(\Sigma, S, S_0, \rho, F)$ automaton on finite words
  - Here: Σ is formed as letters from the 2^{AP} alphabet
    - State labels $L(s)$ are considered as letters
    - E.g., {Red, Yellow} is a letter from the alphabet above
  - The path $\pi=(s_0, s_1, s_2, \ldots s_n)$ identifies a word as follows:
    $(L(s_0), L(s_1), L(s_2), \ldots L(s_n))$
- Two automata are needed:
  - Model automaton: Based on a model $M=(S,R,L)$ an automaton A_M is constructed that accepts and only accepts words that correspond to the paths of M
  - Property automaton: Based on the expression p an automaton A_p is constructed that accepts and only accepts words that correspond to paths on which p is true

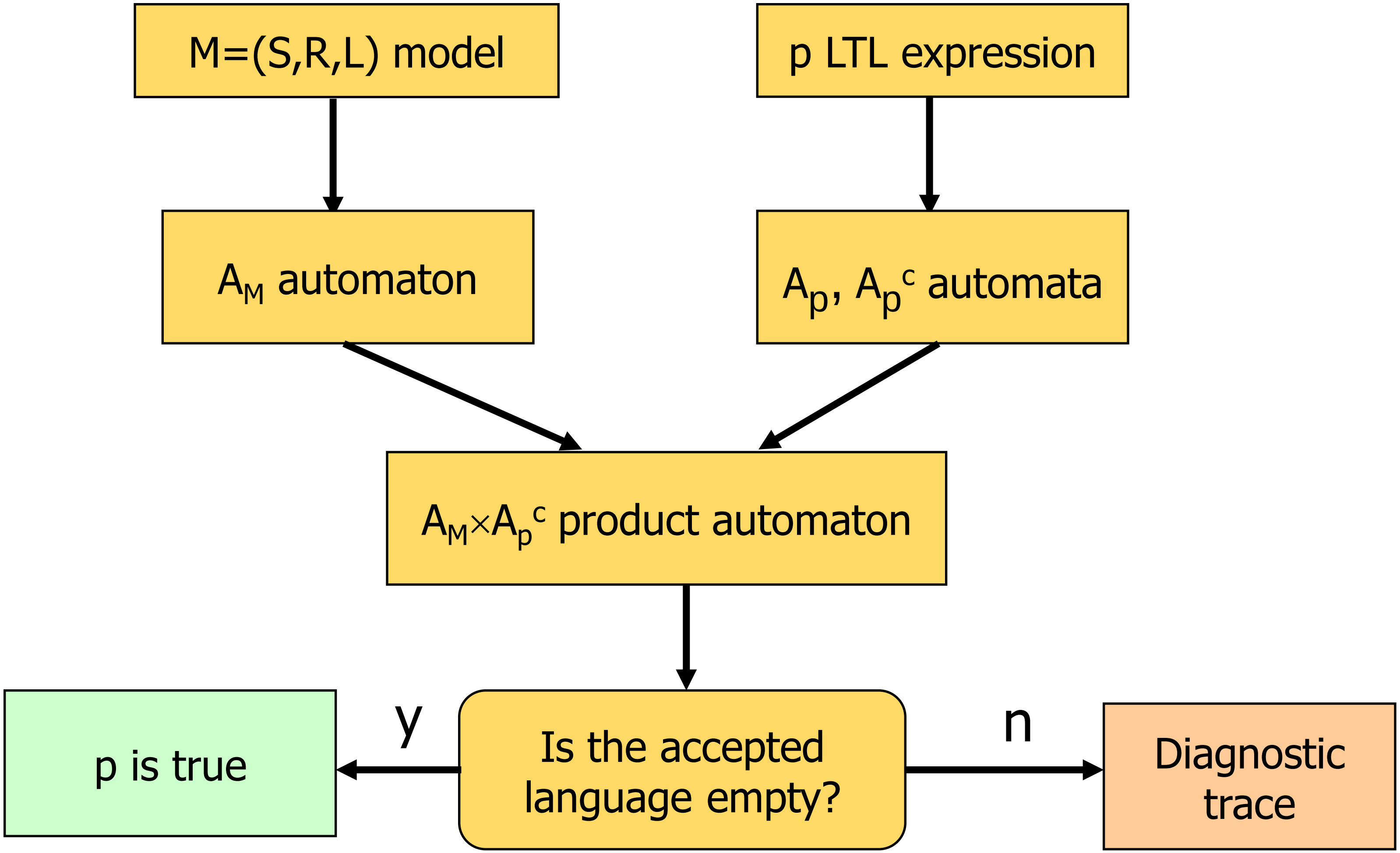
Model checking using the automata

- Model checking question: $L(A_M) \subseteq L(A_p)$?
  - I.e., is the language of the model automaton included in the language of the property automaton?
  - If yes, them $M,\pi \models p$ for all paths of M



- Verifying $L(A_M) \subseteq L(A_p)$ by alternative ways
  - Is the intersection of the following languages empty: $L(A_M) \cap L(A_p)^C$ where $L(A_p)^C$ is the complement language of $L(A_p)$
  - Is the language that is accepted by the $A_M \times A_p{}^c$ product automaton empty, where $A_p{}^c$ is the complement of A_p
    - In case of finite words (finite behavior): The language is empty if there is no reachable accepting state in $A_M \times A_p{}^c$
    - In case of infinite words (cyclic behavior): Büchi acceptance criteria can be used ($\rightarrow$ no cyclic behavior with accepting states)
    - $A_p{}^c$ construction (in fully defined and deterministic case): swapping accepting states with non-accepting states and vice versa

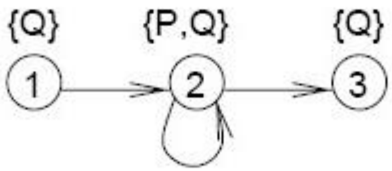
Overview of automata based model checking



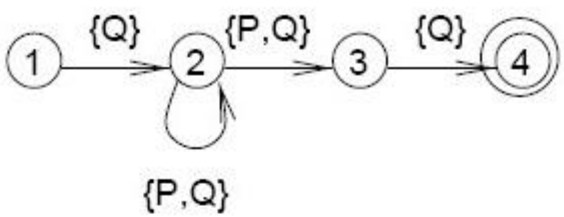
(In the following: Basic ideas will be discussed, not a complete algorithm.)

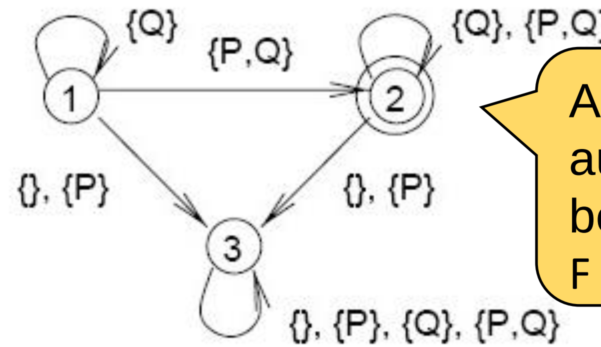
Example: Checking F P ∧ G Q

M model

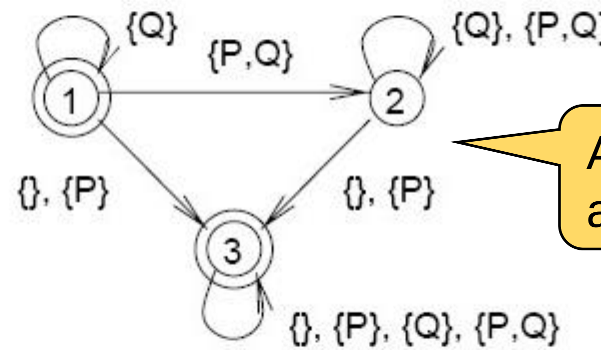


A_M automaton



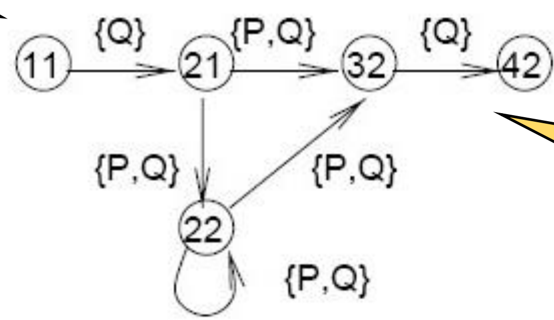


Assume: A_p automaton belonging to F P ∧ G Q



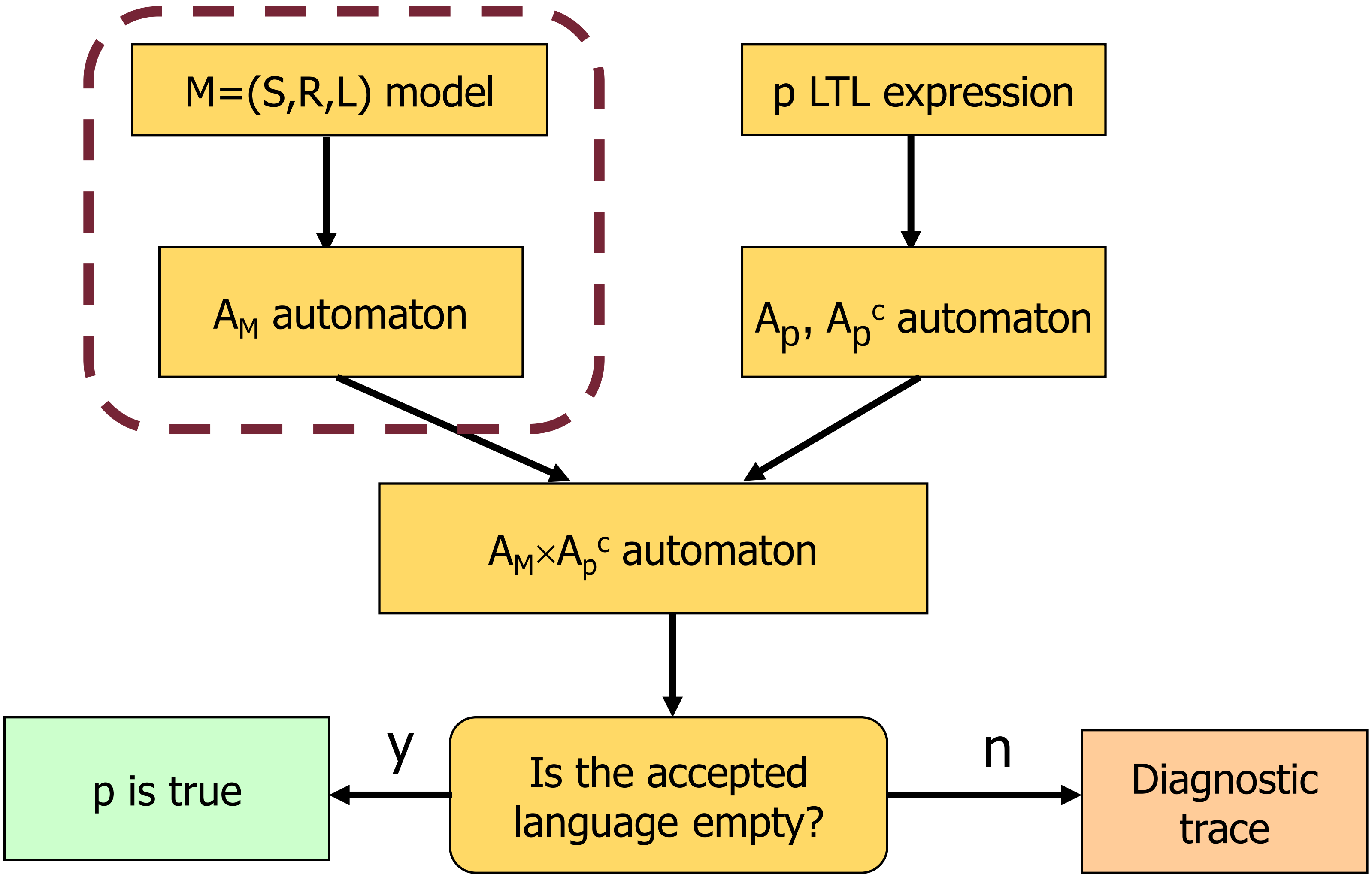
A_p^c automaton

Synchronous product of automata A_M and A_p^c



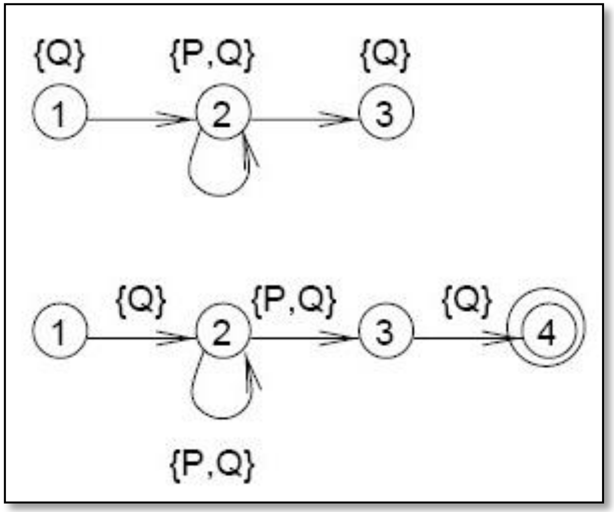
There is no accepting state: No counter-example for F P ∧ G Q

Overview of automata based model checking



Constructing A_M on the basis of M

- Labels are moved to outgoing transitions
- In case of finite behavior (finite words):
  - Accepting state s_f is added
  - Transitions are added from the end states (without outgoing transition) to the accepting state s_f



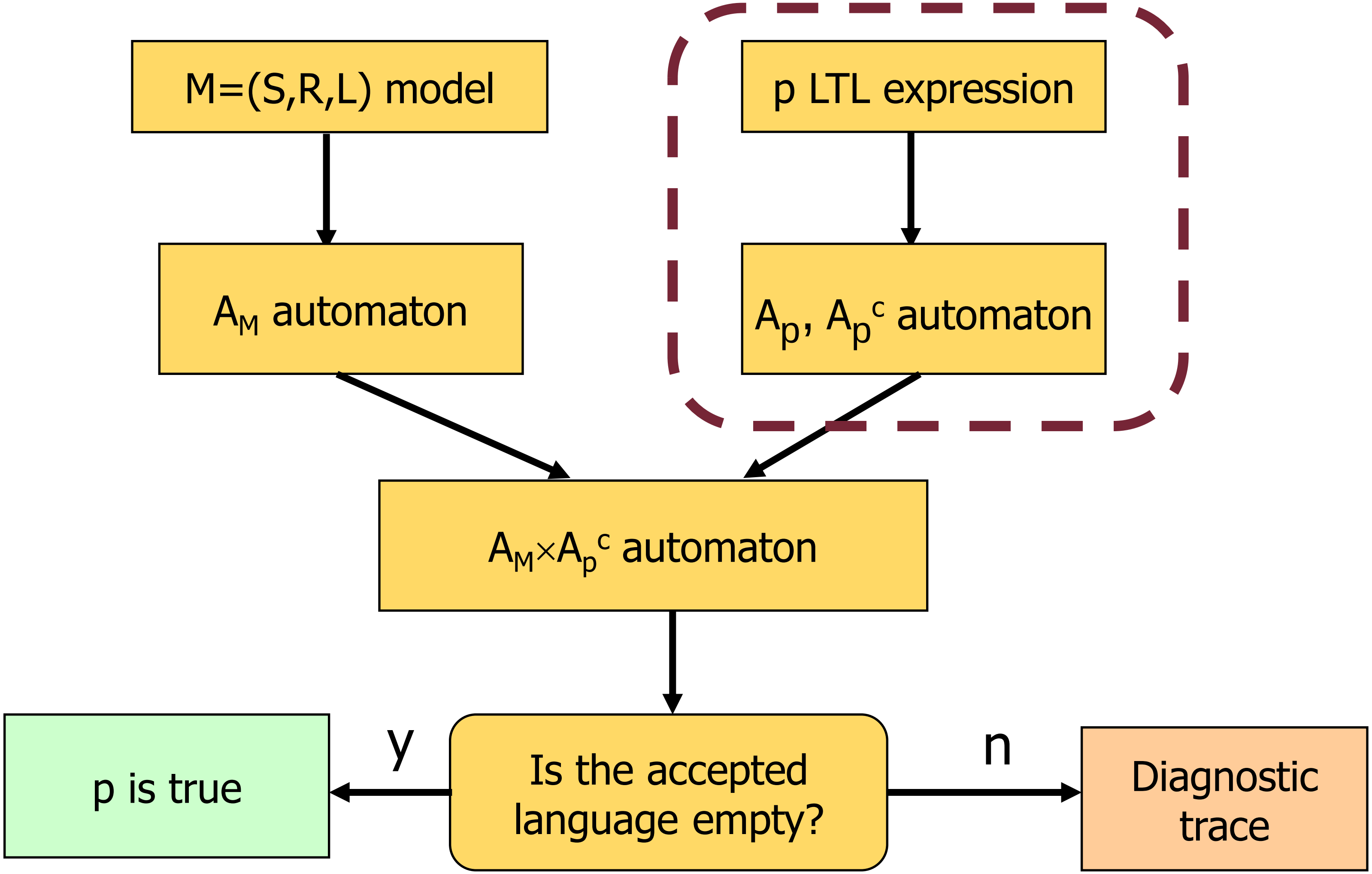
- Formally the automaton:

$$A_M=(2^{AP}, S\cup\{s_f\}, \{s_0\}, \rho, \{s_f\})$$

where the transitions relation is the following:

$$\rho=\{ (s,L(s),t) \mid (s,t)\in R \} \cup \{(s,L(s),s_f) \mid \text{no } t \text{, such that } (s,t)\in R \}$$

Overview of automata based model checking



Constructing A_p on the basis of p: The basic idea

- A_p automaton: Shall represent those paths on which p is true
- Basic idea: Decompose the expression similarly to the tableau method and this way identify the states and transitions of A_p
  - First decomposition: Identifies the initial state(s) of A_p
    - Labels of the state: Based on the atomic propositions (i.e., without temporal operators) resulting from the decomposition
    - Outgoing transitions to next states: Identified by the (sub)expressions with temporal operators, that have to be true from the next state
  - New decomposition for each formula belonging to a next state
- Initial step: Construct the negated normal form of the expression
  - For Boolean operators: de Morgan laws
  - For temporal operators:

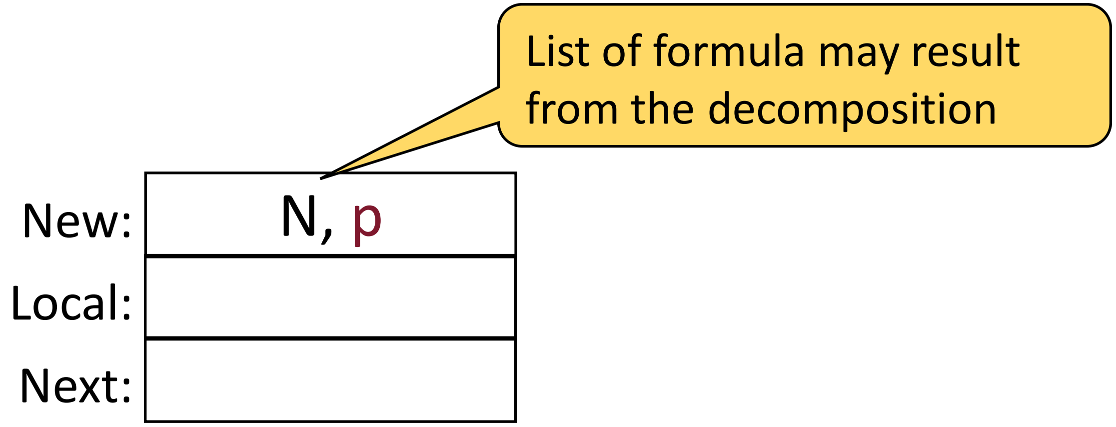
    $\neg(X\ p) = X\ (\neg p)$

    $\neg(p\ U\ q)$ can be handled by defining the R „release" operator:

    $\neg(p\ U\ q) = (\neg p)\ R\ (\neg q)$, from which $p\ R\ q = q \wedge (p \vee X\ (p\ R\ q))$

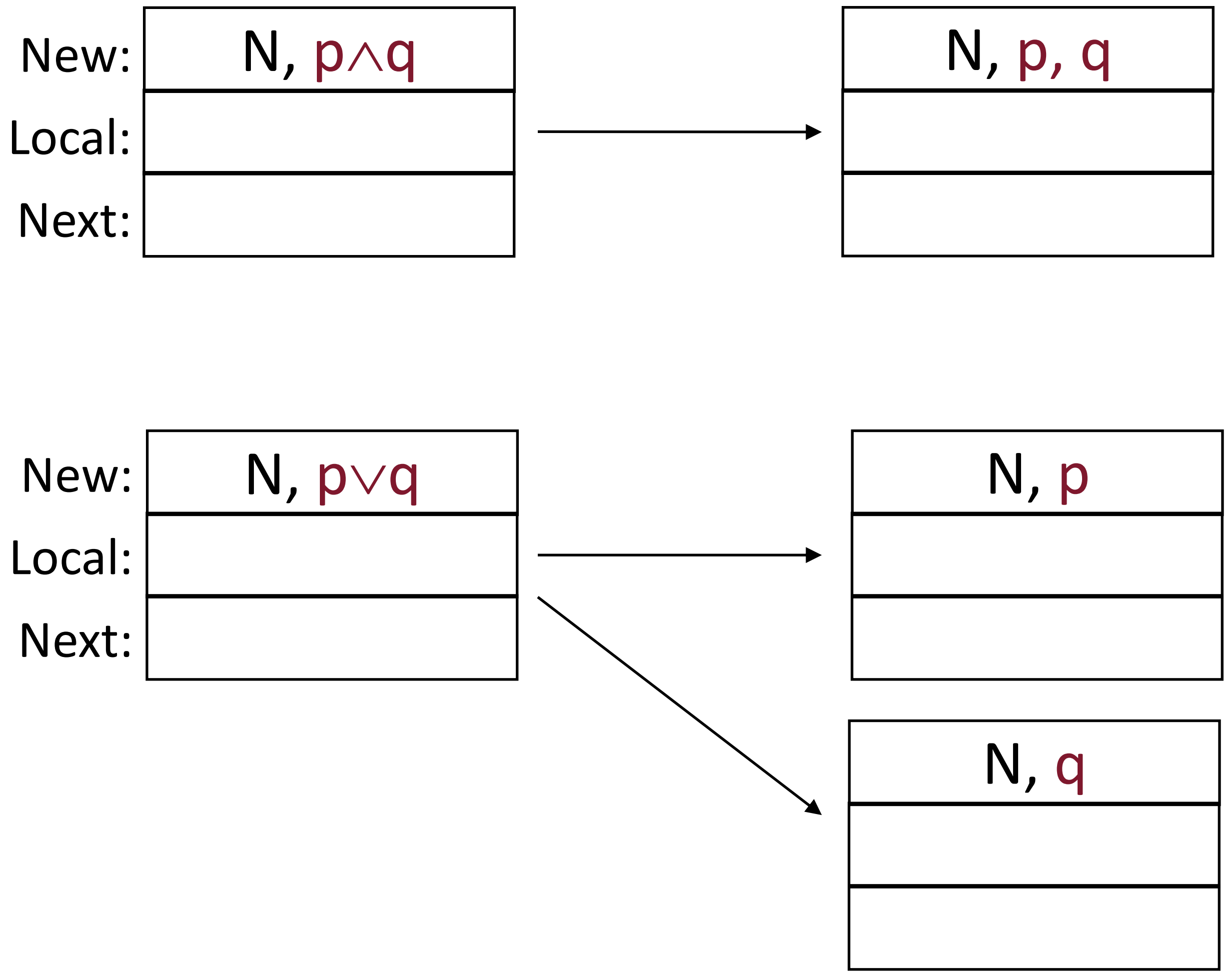
Constructing A_p on the basis of p: Data structure

- Data structure (a record) to represent the decomposition:
  - New: list of expressions to be decomposed
  - Local: atomic propositions related to the current state
  - Next: expression that has to be true from the next state



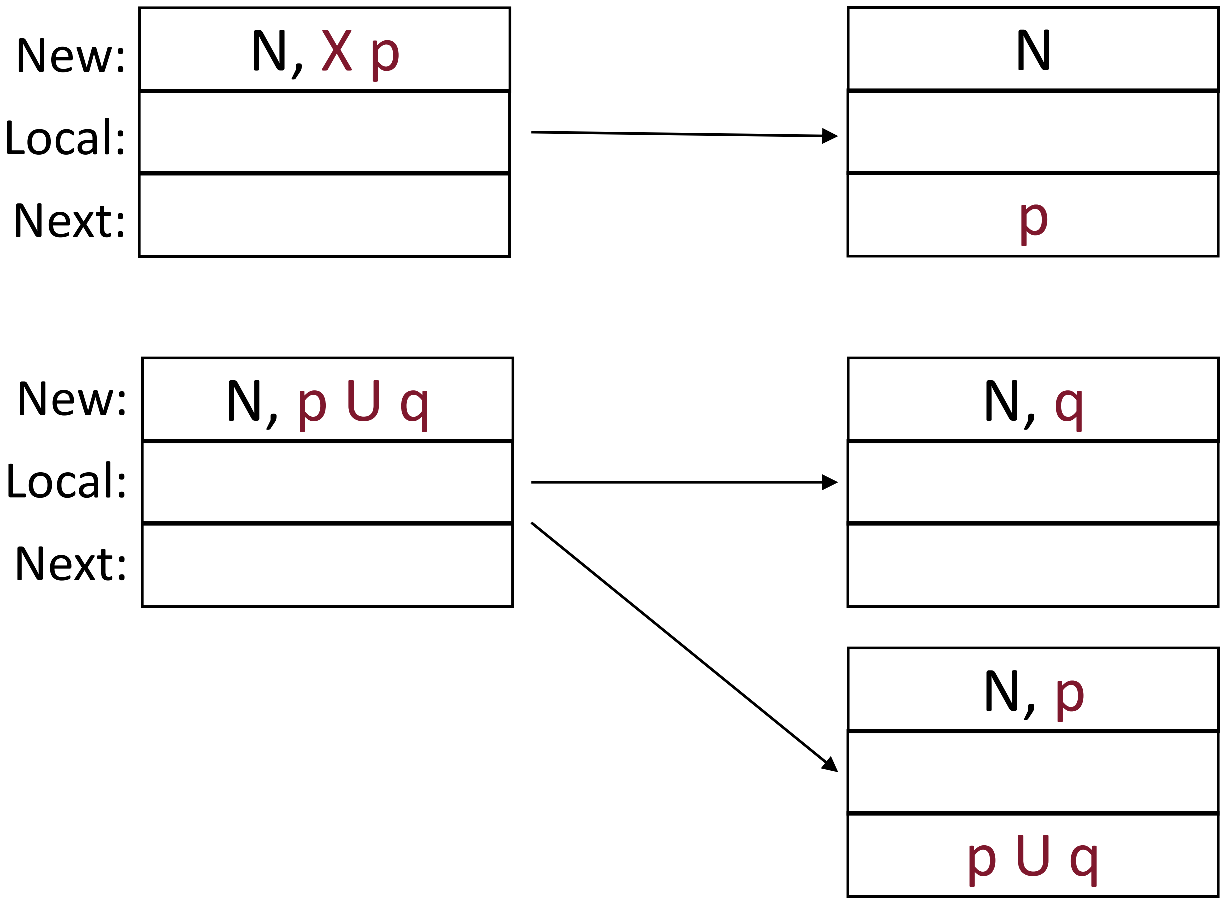
Constructing A_p on the basis of p: Decomposition rules

- Decomposition rules for ∧ and ∨:



Constructing A_p on the basis of p: Decomposition rules

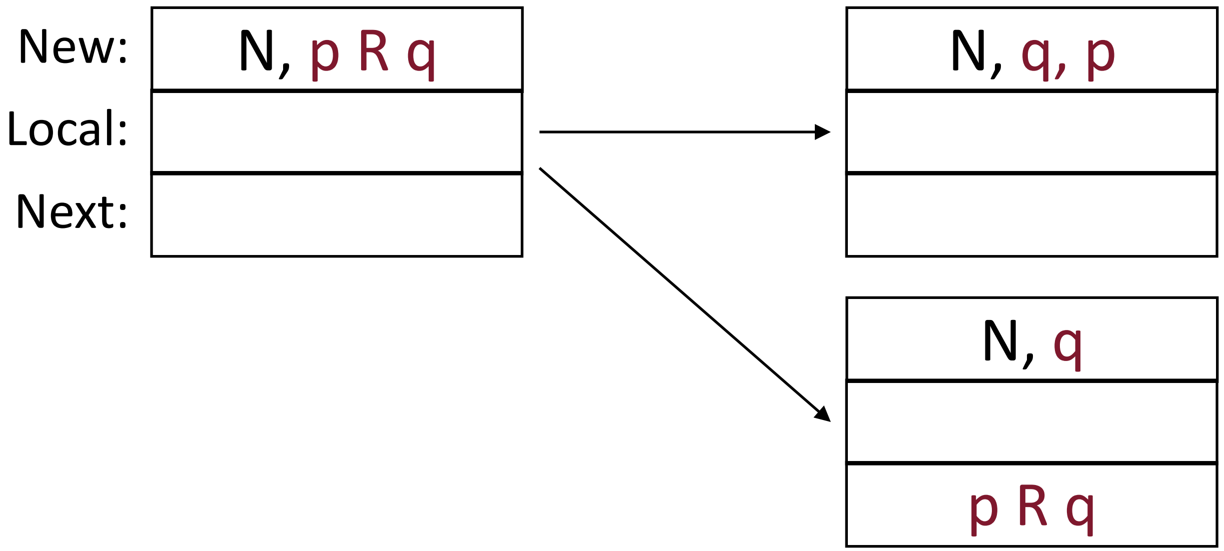
- Decomposition rules for X and U:



based on the rule $p \; U \; q = q \lor (p \land X(p \; U \; q))$

Constructing A_p on the basis of p: Decomposition rules

- Decomposition rule for R:

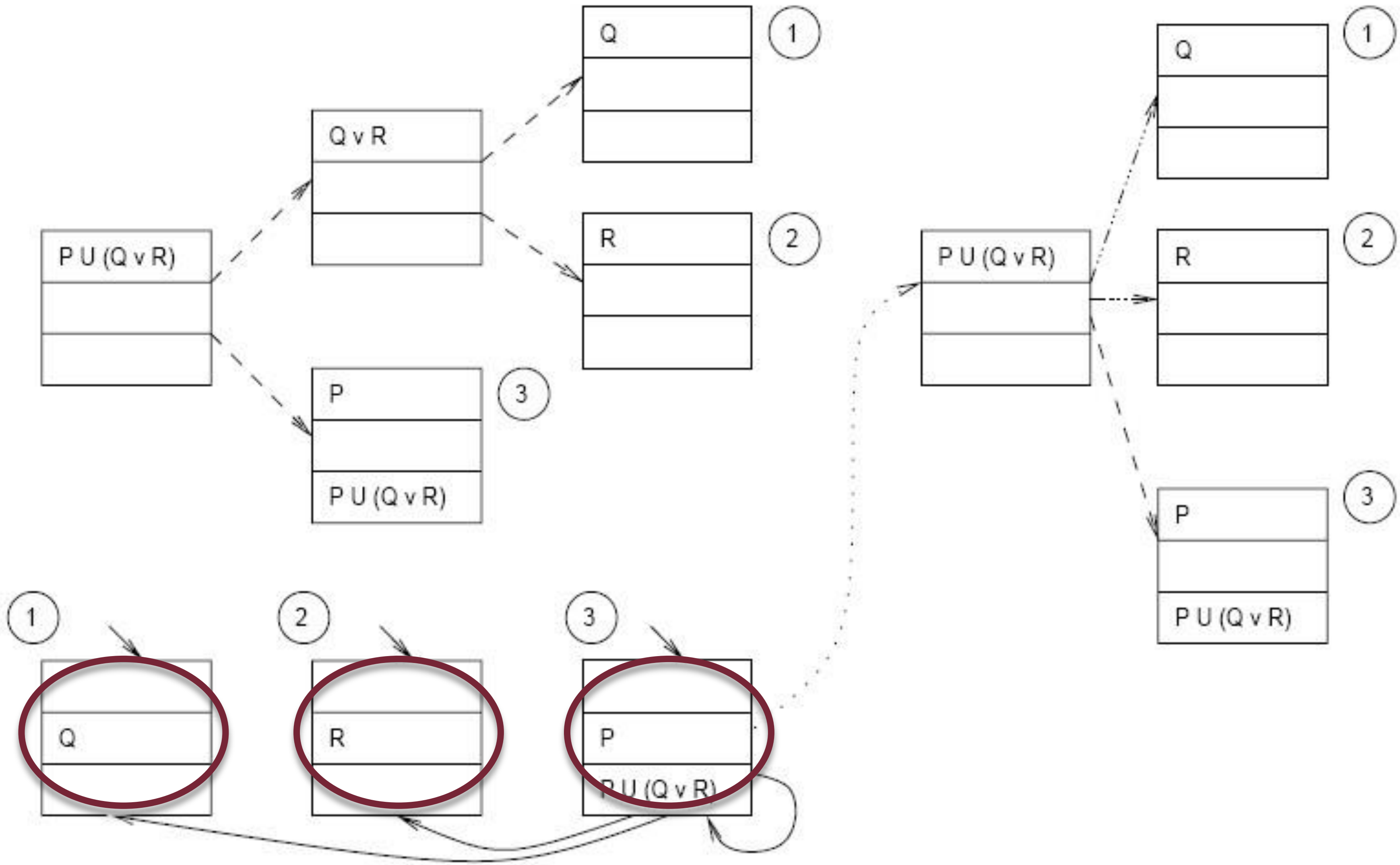


based on the rule $p\ R\ q = q \wedge (p \vee X(p\ R\ q))$

Constructing A_p on the basis of the decomposition (1)

- States: A state of the A_p automaton is identified from a node of the decomposition if:
  - There are only atomic propositions in the New field of the node;
    these are copied to the Local field and used to derive state labels,
  - and there is no state in A_p that was identified based on a node with the same Local and Next fields (otherwise the same state is identified again)
- Transitions: If a state s of the A_p automaton is identified then:
  - A new decomposition is started from the expression that is in the Next field of the node (copying it to the New field of a new node),
    since the Next field identifies property to be satisfied from the next state
  - Transitions of A_p are drawn from the state s to the states that result from the new decomposition
- Summary:
  - States of A_p are identified when the decomposition results in nodes with atomic propositions (there is no further operator to be decomposed)
  - Transitions from a state s are drawn to the states that result from the decomposition of the formula in the Next field of the node belonging to s

Example: P U (Q ∨ R)

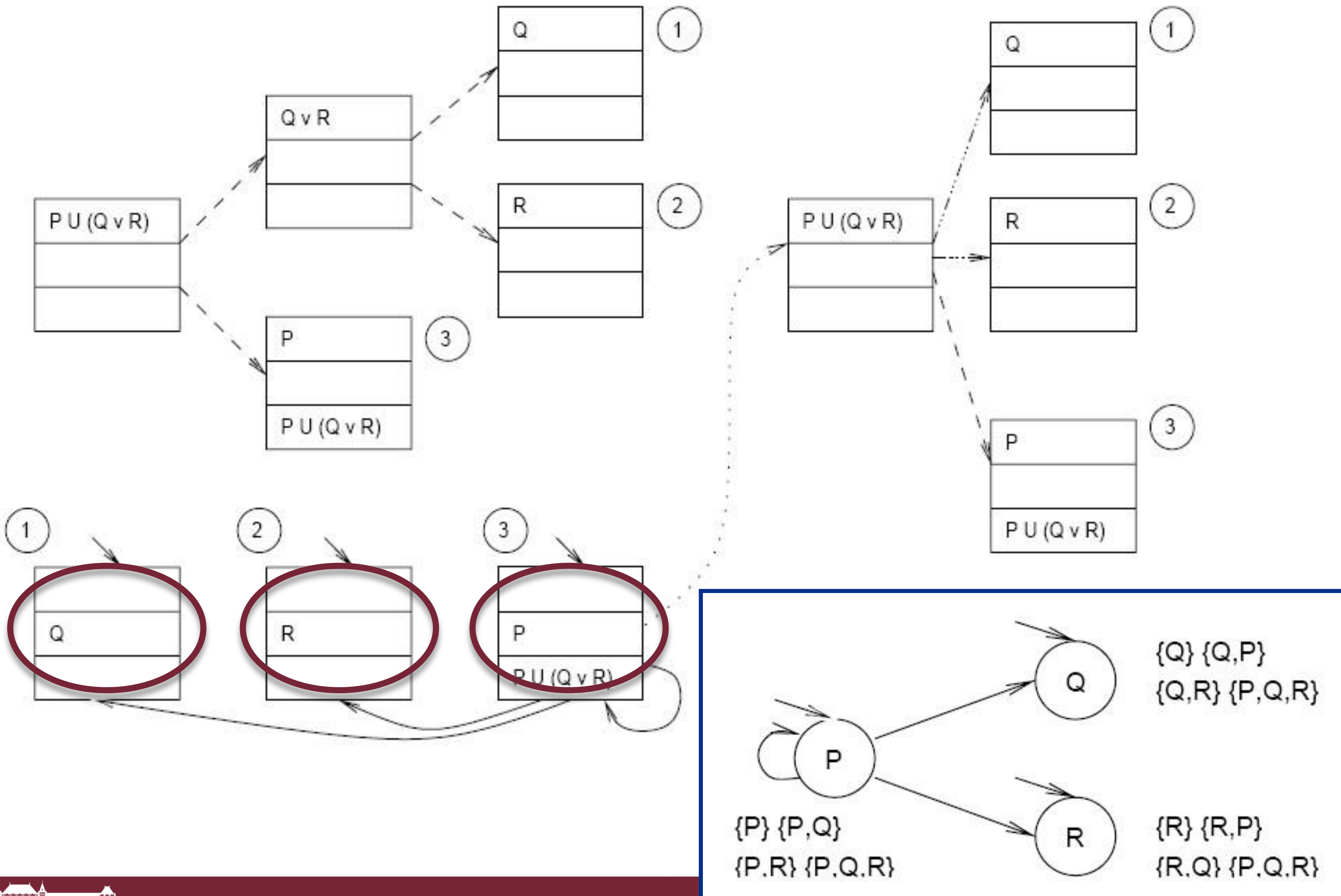


Constructing A_p on the basis of the decomposition (2)

- Further elements of A_p:
  - Initial state(s):
    - State(s) resulting from the first decomposition
  - Accepting states (in finite case):
    - When the Next field is empty (no formula refers to the next state)
  - Labeling of a state: All subsets of AP that are compatible with the atomic propositions found in the Local field of the node belonging to the state
    - Each atomic proposition is included that is non-negated in Local
    - There is no atomic proposition that is negated in Local

  Since each behavior is to be included in A_p that is allowed by the propositions in the Local field

Example: P U (Q ∨ R) with AP={P,Q,R}



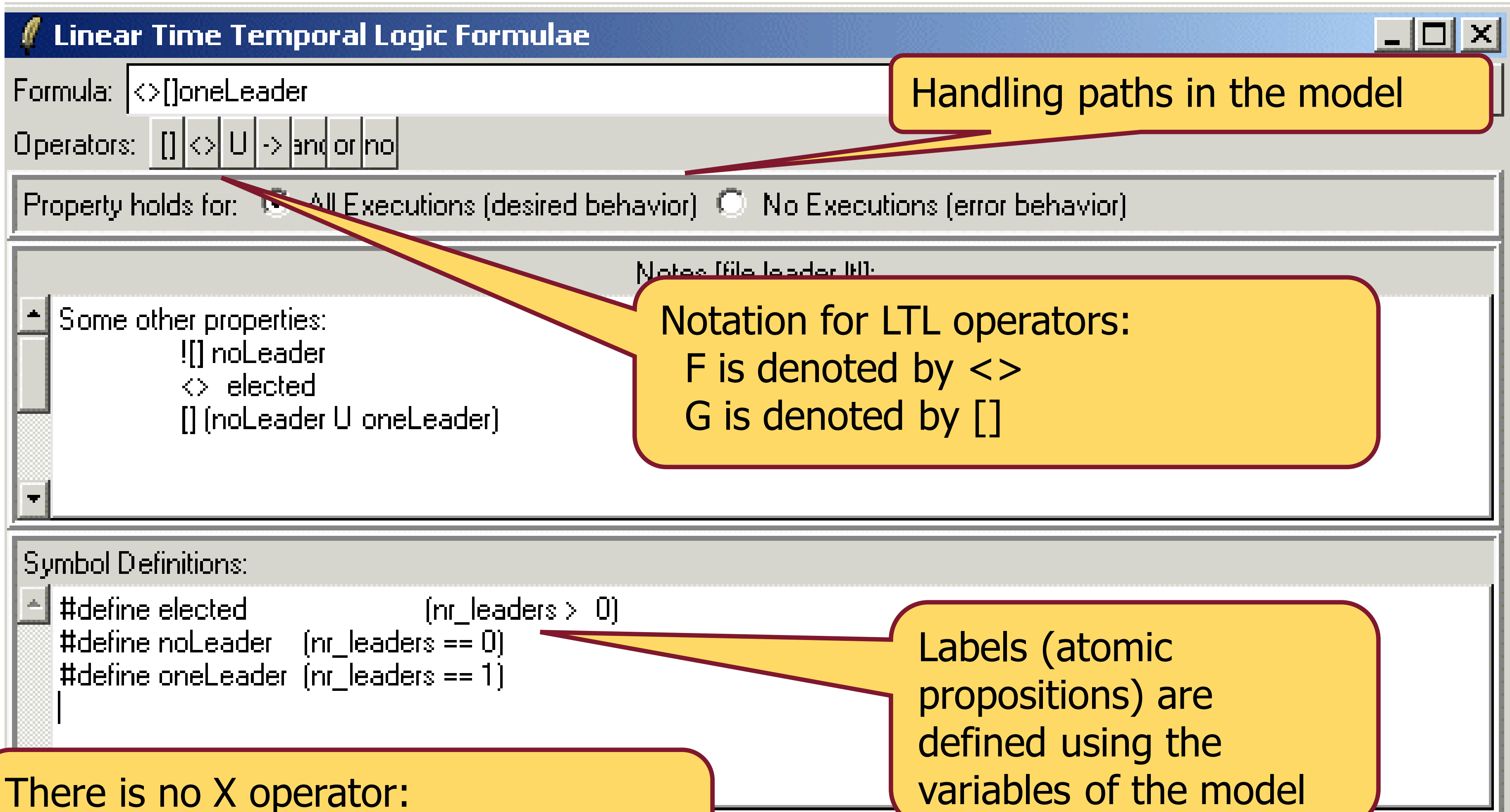
Complexity of PLTL model checking

- Worst-case time complexity of model checking the expression p on model M=(S,R,L):

$$O(|S|^2 \times 2^{|p|}), \text{ where}$$

  - |S| is the number of states
  - |p| is the number of operators in the LTL formula
  - $|S|^2$ is the number of transitions in the model automaton
    (maximum number of transitions; typically only linear with S)
  - $2^{|p|}$ is the number of transitions in the property automaton
    (maximum number of sub-expressions to be decomposed and resulting in new transitions)
  - $|S|^2 \times 2^{|p|}$ results from the state space of the product automaton
    (in which accepting states or cycles shall be found)
- The exponential complexity seems frightening, but
  - The LTL expressions are typically short (a few operators)
  - Time needs result mainly from the size of the model automaton

The model checker SPIN



Handling paths in the model

Notation for LTL operators:
F is denoted by <>
G is denoted by []

Labels (atomic propositions) are defined using the variables of the model

There is no X operator:
It is not supported by the states space reduction that is applied in SPIN

Summary

- Temporal operators of LTL
- Formal syntax and semantics of LTL
  - Extending LTL to LTS
- Examples
- Verification of LTL properties
  - The model checking problem
  - LTL model checking: Automata based approach