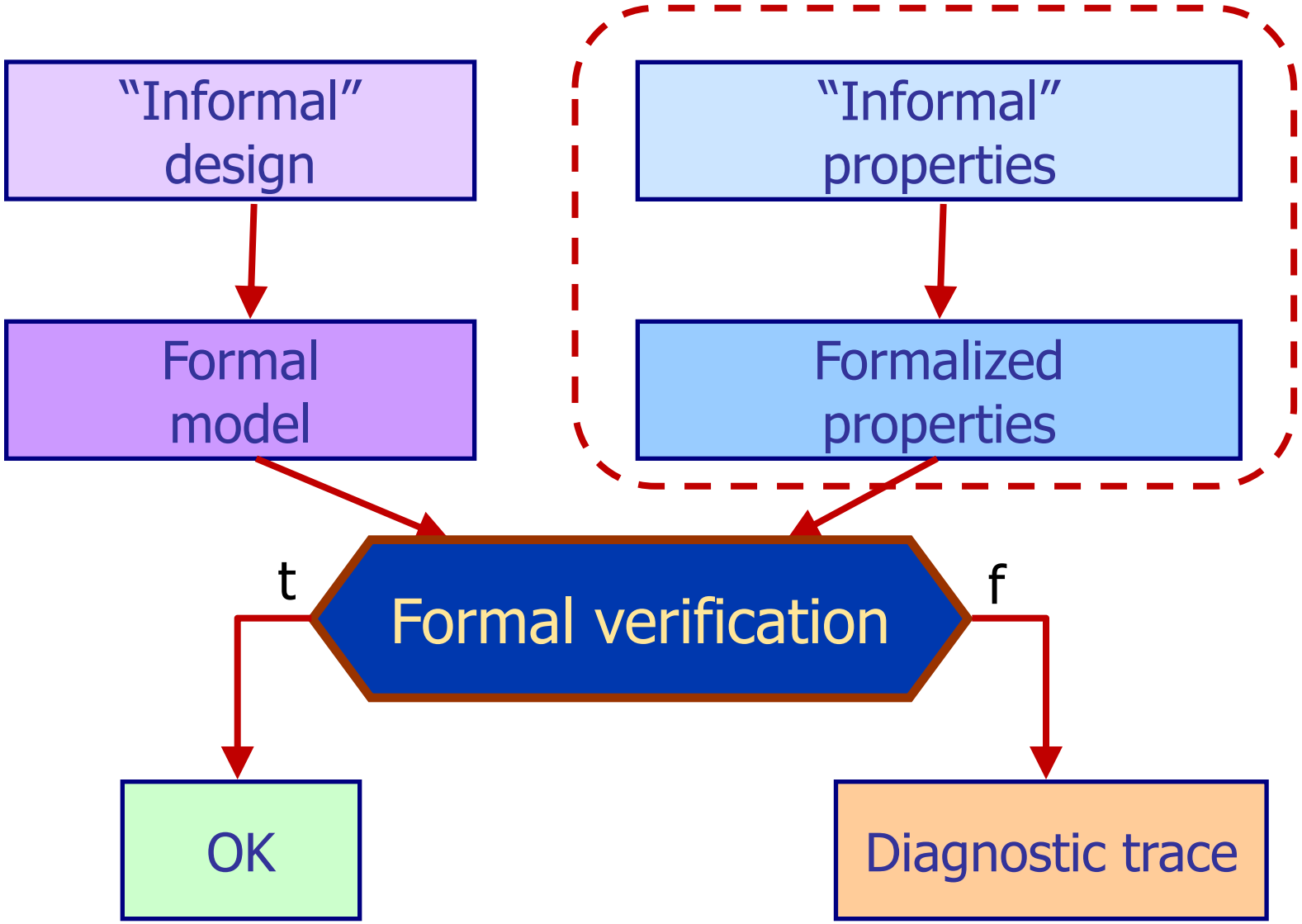
Formalizing and checking properties: Temporal logics HML and LTL

Istvan Majzik
majzik@mit.bme.hu

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

Formal verification: Goals



Overview

- Formalization of requirements
- Temporal logics
- HML: Hennessy-Milner logic
  - Temporal operators
  - Model checking: Tableau method
- LTL: Linear Temporal Logic
  - Temporal operators
  - Syntax and semantics
  - Model checking: Automata based approach

Formalization of requirements

Frequent patterns of requirements

Handling textual requirements

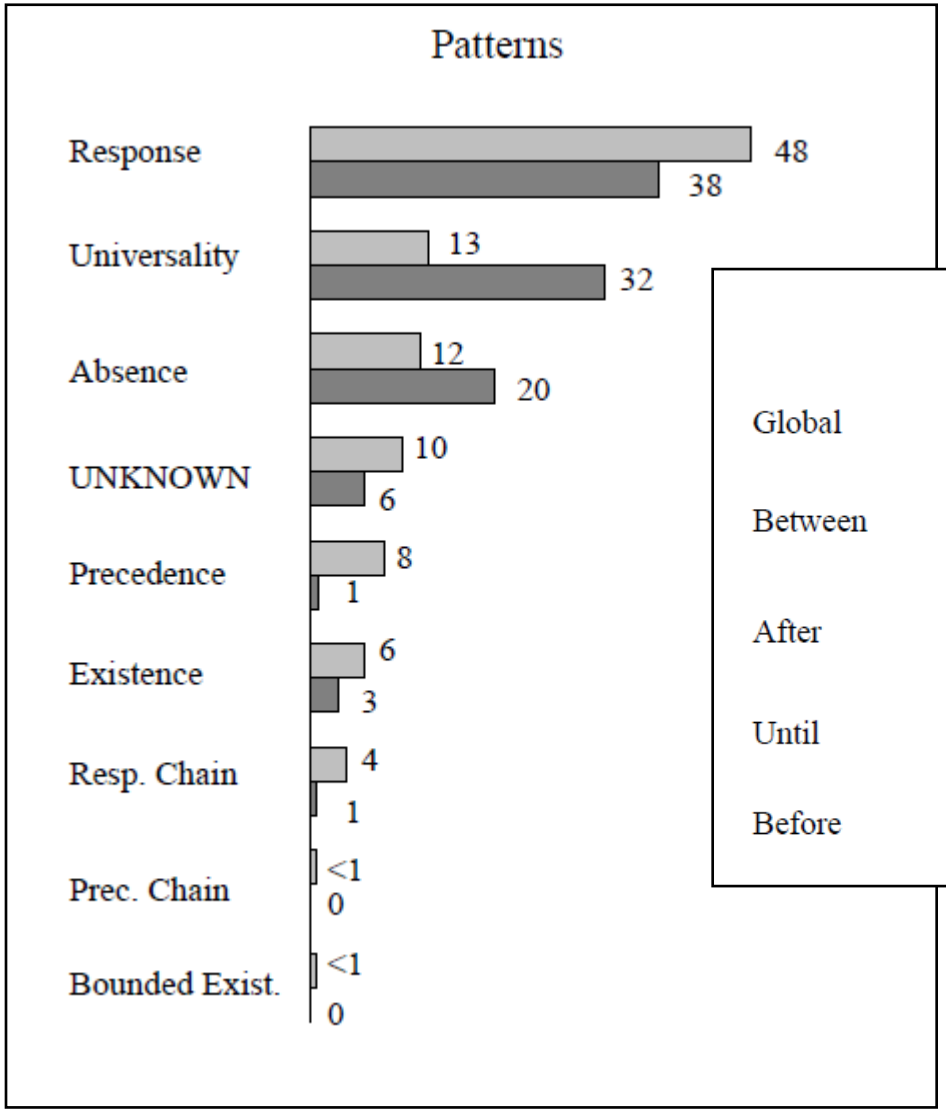
- Specifying a requirement in natural language:

> If `alarm` is on and `alert` occurs, the output of `safety` should be true as long as `alarm` is on.

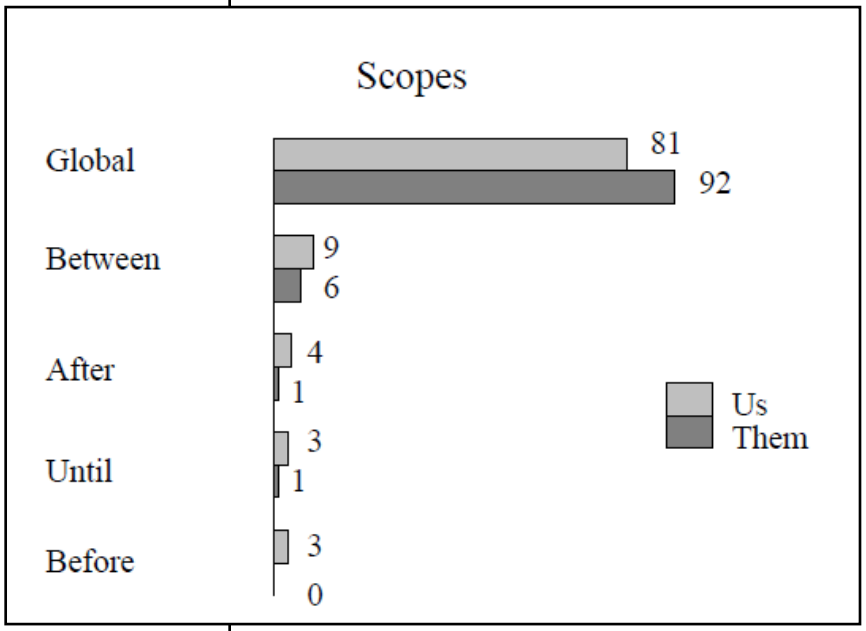
> If the `switch` is turned to AUTO, and the `light intensity` is LOW then the `headlights` should stay or turn immediately ON, afterwards the `headlights` should continue to stay ON in AUTO as long as the `light intensity` is not HIGH.

  - Is the textual description easy to understand and unambiguous?
  - Structure is not clear (conditions, sequence, …)

Requirements in critical systems: result of a survey



A significant proportion of requirements match to certain patterns



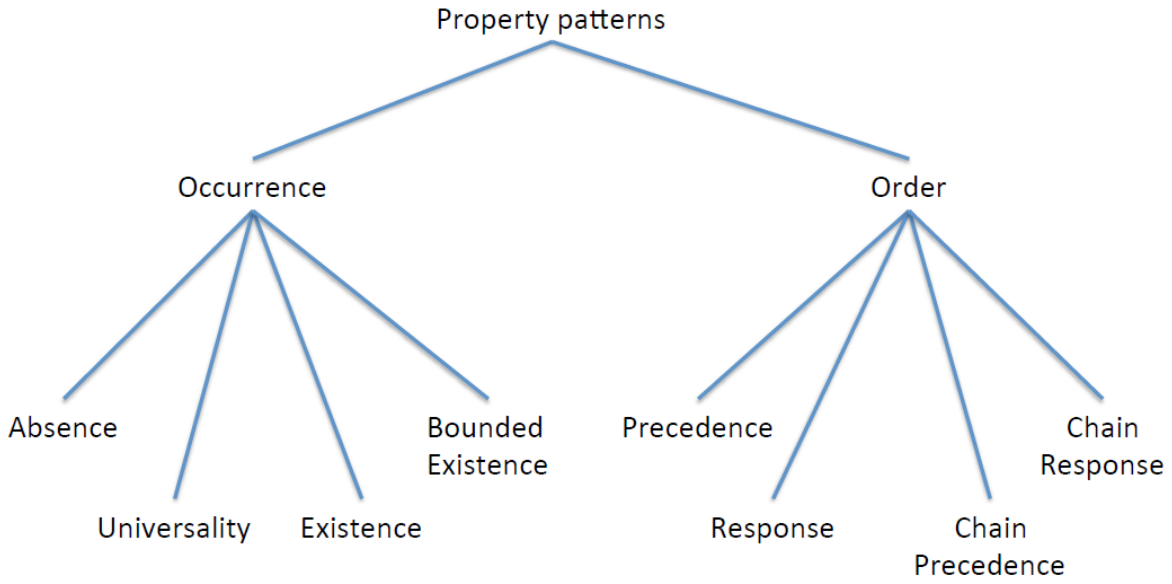
Figures: The distribution of matched patterns for requirements from two development teams

http://patterns.projects.cis.ksu.edu/documentation/patterns.shtml

Groups of patterns

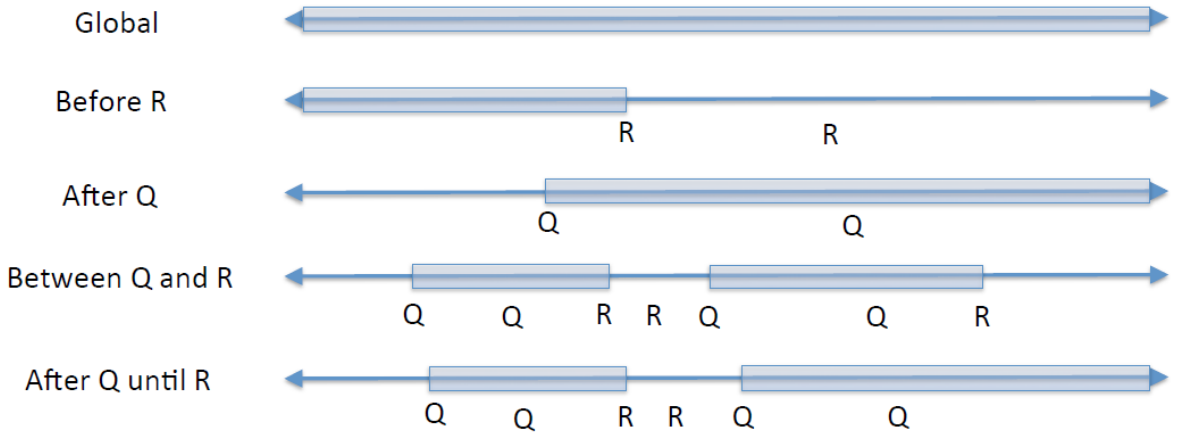
Pattern:

order or occurrence



Scope:

relative to further events



Patterns: Explanations

Occurrence:

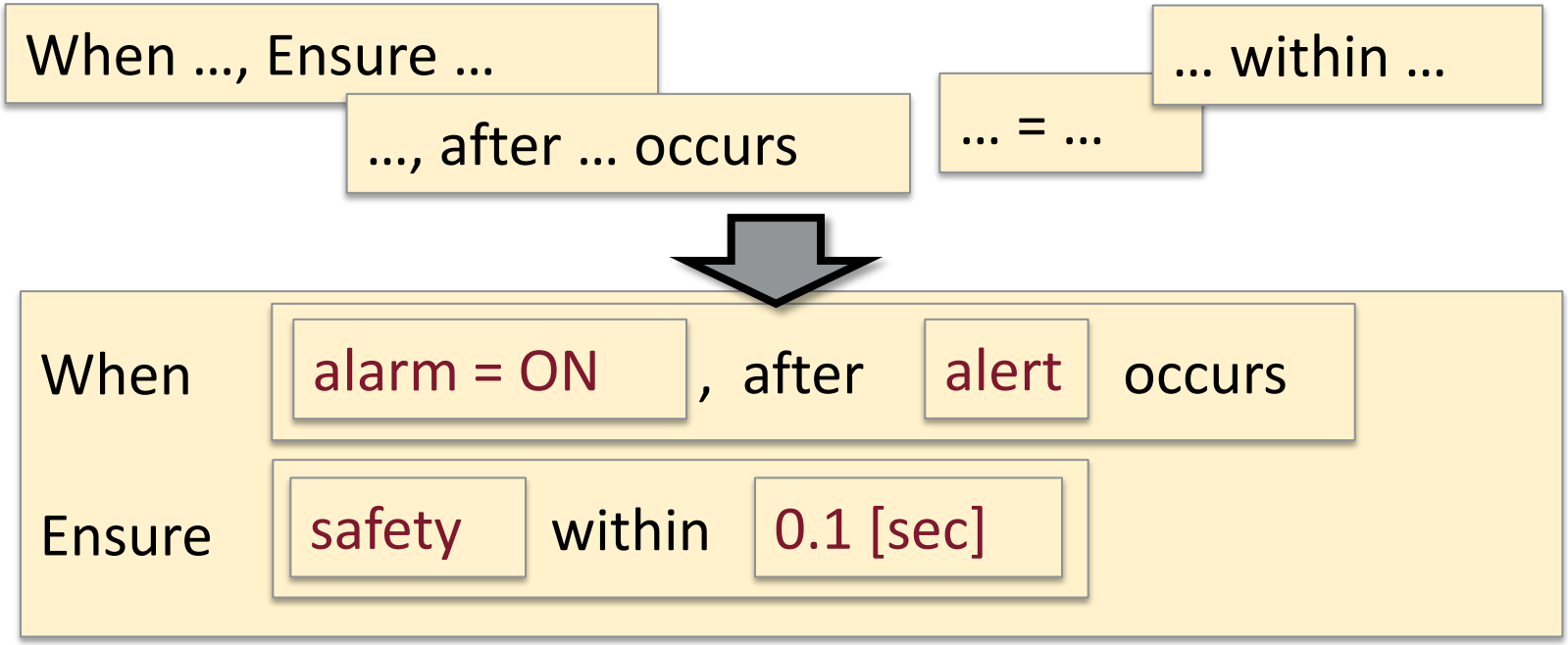
- Absence: the referenced state/event never occurs
- Universality: the referenced state/event is always present
- Existence: the referenced state/event eventually occurs
- Bounded existence: the referenced state/event occurs at least k times

Order:

- Precedence: the referenced state/event precedes another state/event
- Response: the referenced state/event is followed by another state/event
- Chain precedence: generalization of Precedence to sequences
- Chain response: generalization of Response to sequences

Composition of patterns

- Patterns can be composed
  - Boolean operators (and, or, implication)
  - Embedding patterns in other patterns
- Example: Patterns in textual form



Outcome

- The majority of properties match certain patterns
  - If … then …, Never …, After …, Before …
  - Occurrence/order of states/events
  - More complex requirements composed from simpler ones
- These properties can be formalized if the basic patterns can be captured using a formal language
  - Absence, universality, existence, precedence, response
  - Temporal scope (globally, after, before)
- Formalization of requirements helps
  - Verification of design: exhaustive analysis of executions
  - Test evaluation, runtime monitoring: components can be automatically generated
- Applied formalism: Temporal logic

Preview: Formalization of properties in LTL

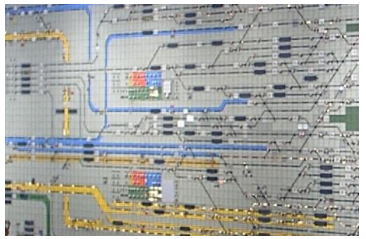
Universality within scope	**Property in LTL**	**Meaning of LTL expressions**
Event P occurs in each step of the execution globally.	G P	Globally P
Event P occurs in each step of the execution before event Q.	F Q $\rightarrow$ (P U Q)	(Eventually Q) implies (P Until Q)
Event P occurs in each step of the execution after event Q.	G (Q $\rightarrow$ G P)	Globally (Q implies Globally P)
Event P occurs in each step of the execution between events Q and R.	G ((Q $\wedge \neg$R $\wedge$ F R) $\rightarrow$ (P U R))	Globally ((Q and not R and Eventually R) implies (P Until R))

Existence within scope	**Property in LTL**
Event P occurs in the execution globally.	F P
Event P occurs in the execution before event Q.	$\neg$ Q W (P $\wedge \neg$ Q)
Event P occurs in the execution after event Q.	G ($\neg$Q) $\vee$ F (Q $\wedge$ F P))
Event P occurs in the execution between events Q and R.	G (((Q $\wedge \neg$R) $\wedge$ (F R)) $\rightarrow$ ($\neg$R W (P $\wedge \neg$R)))

Temporal requirements and temporal logics

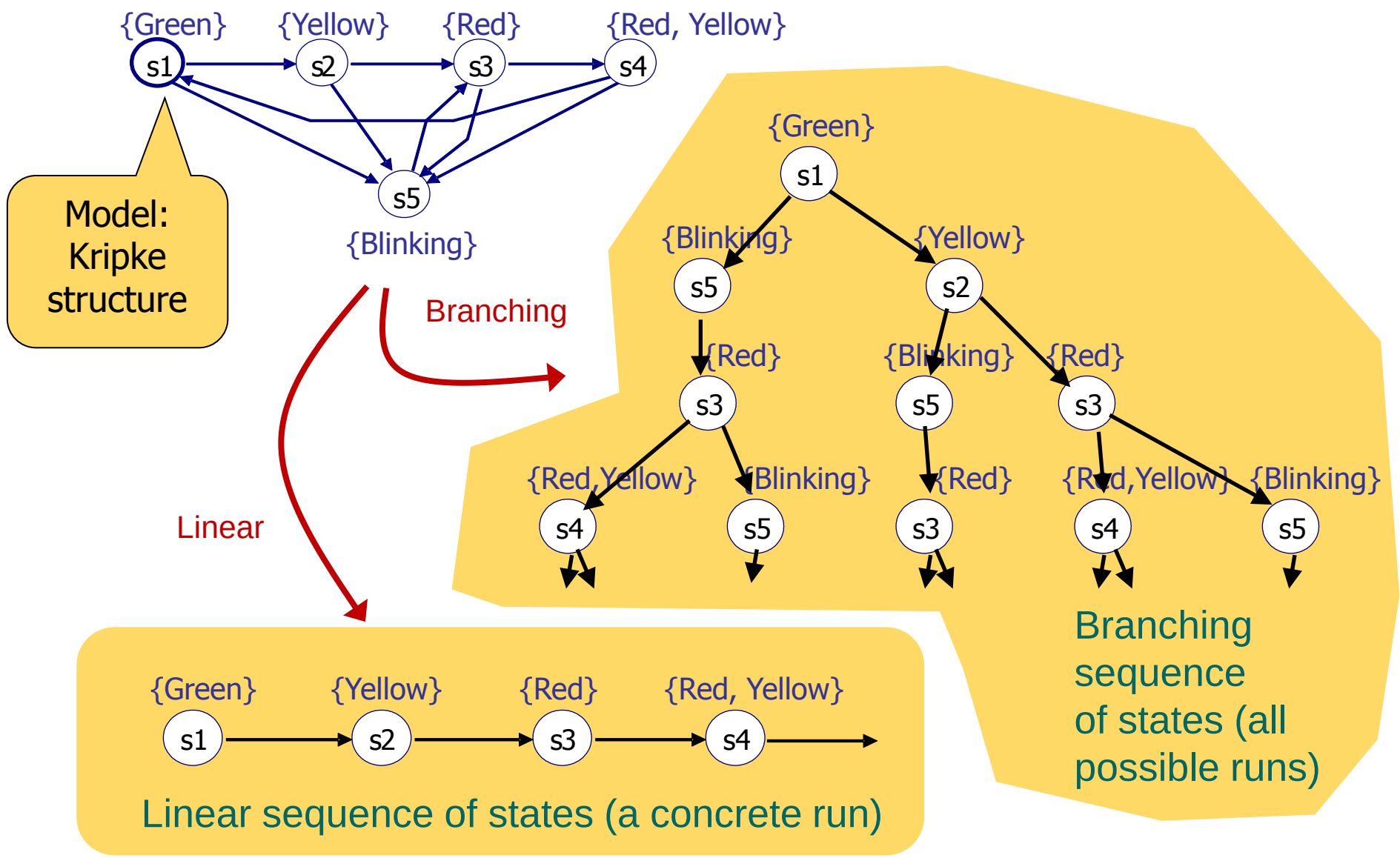
Classification of temporal logics

Formalization of reachability properties



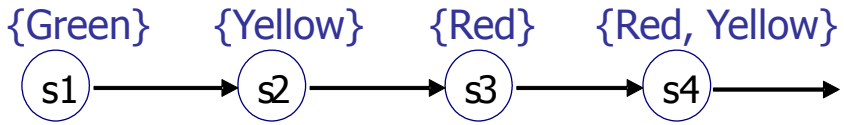
- Goal: Formalizing state reachability properties
  - Important in state based, event driven systems
  - Occurrence: based on local properties of states
    - Name, valuation of variables, mode of operation, …
  - Ordering: logical time
    - "Current" point in time: active state
    - "Subsequent" points in time: next state(s)
  - Typical reachability properties
    - Safety properties: Absence of "bad" state (universal)
    - Liveness properties: Eventual occurrence of "good" state (existential property)
- Language used for formalization: Temporal logics
  - Formal system for evaluating changes in logical time
  - Temporal operators: "always", "eventually", "before", "while", …

Checking reachability properties

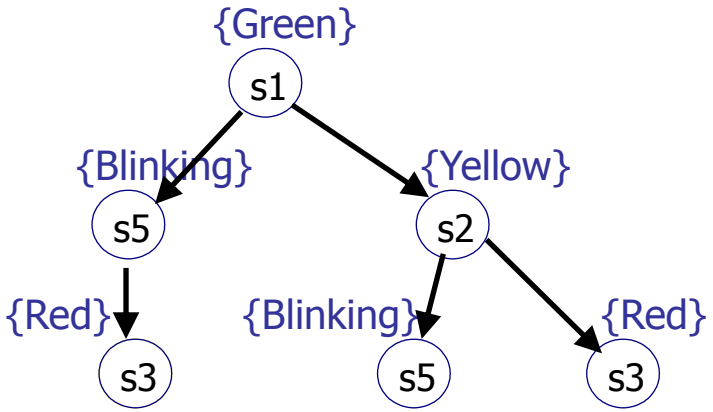


Classification of temporal logics

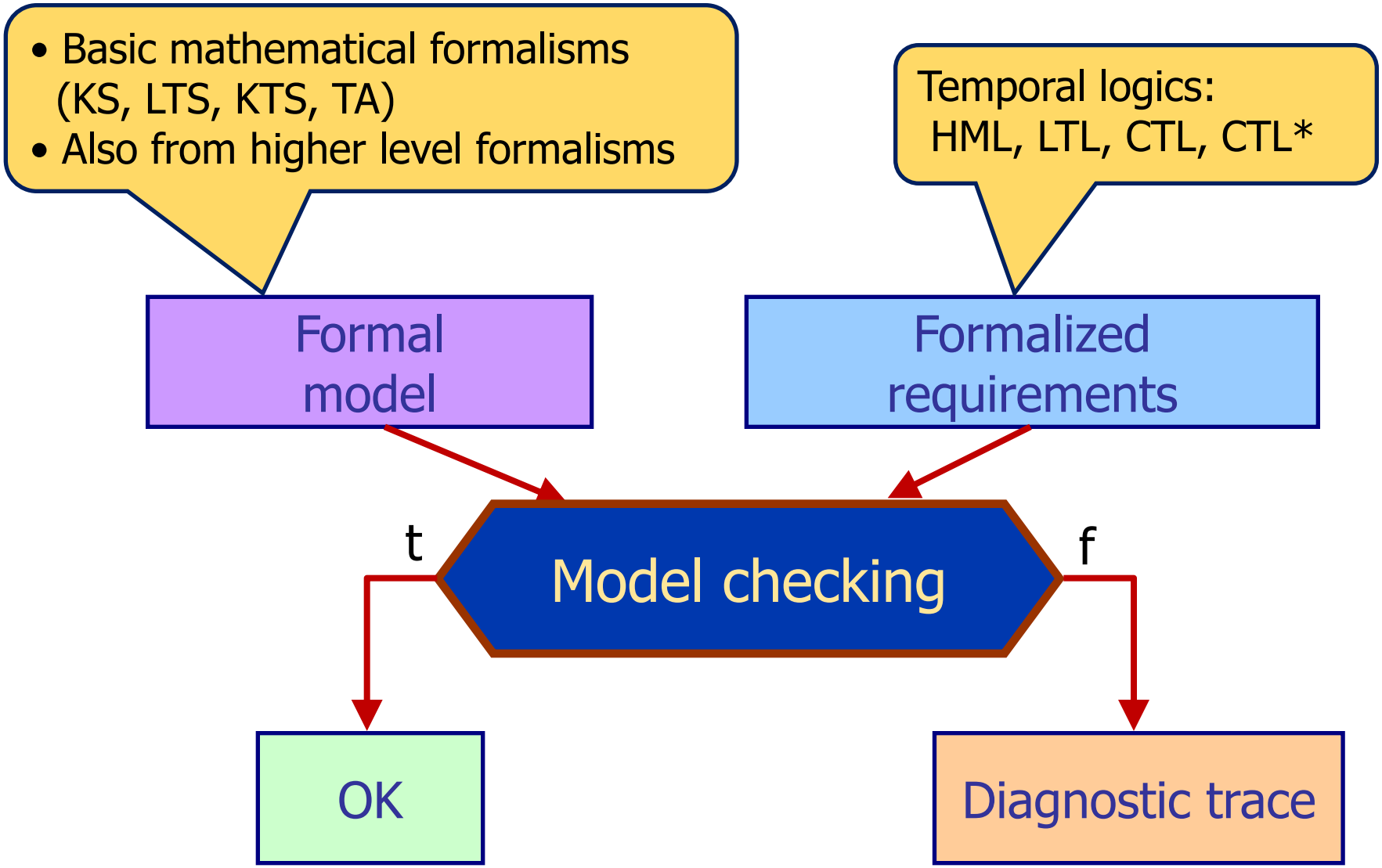
- Linear:
  - We consider individual executions of the system
  - Each state has exactly one subsequent state
  - Logical time along a linear timeline (trace)



- Branching:
  - We consider trees of executions of the system
  - Each state possibly has many subsequent state
  - Logical time along a branching timeline (so-called computation tree)



Model based verification by model checking



Hennessy-Milner Logic

Temporal operators

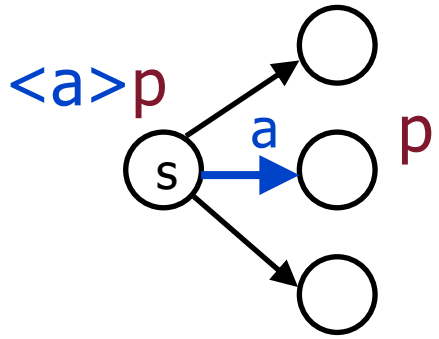
Checking HML formula

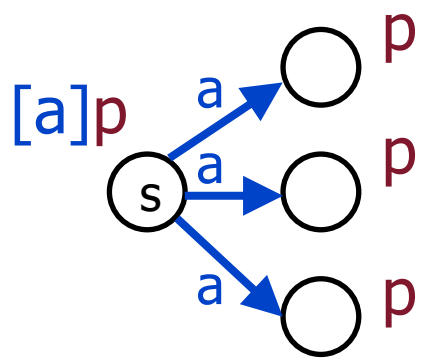
The Hennessy-Milner logic

- Simple logic interpreted on LTS T=(S, Act, $\rightarrow$)
- Finite action sequences (scenarios, traces) can be captured by HML
- Syntax: Rules for composing well-formed HML formula (p and q are well-formed formula, a is an action):

  HML ::= true | false | p$\wedge$q | p$\vee$q | [a]p | <a>p

- Informal semantics of temporal operators:





Semantics of the Hennessy-Milner logic

Model of HML: LTS $T=(S, Act, \rightarrow)$

Semantic rules: Specify when p is true (satisfied) on state s of LTS T

Notation: $T,s \models p$

- $T,s \models true$, $T,s \not\models false$
- $T,s \models p \wedge q$ if and only if (iff) $T,s \models p$ and $T,s \models q$
  $T,s \models p \vee q$ iff $T,s \models p$ or $T,s \models q$
- $T,s \models [a]p$ iff $\forall s'$ where $s \rightarrow^a s'$: $T,s' \models p$
- $T,s \models \langle a \rangle p$ iff $\exists s'$: $s \rightarrow^a s'$ and $T,s' \models p$

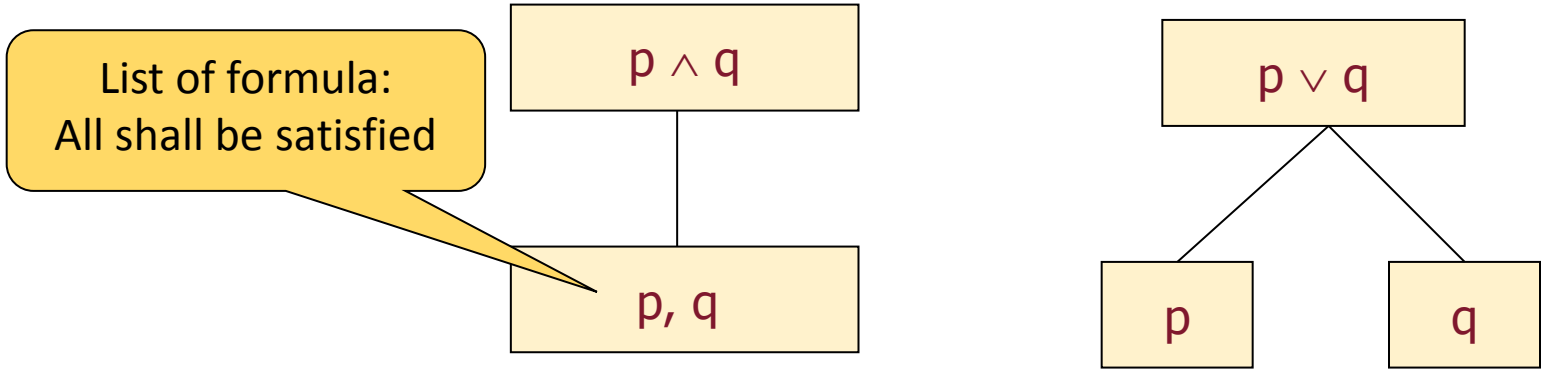
HML examples:

- $\langle a \rangle true$: satisfied if there exists an outgoing transition labeled with a
- $[a]false$: satisfied if there is no outgoing transition labeled with a
- $\langle a \rangle\langle b \rangle\langle c \rangle true$: satisfied if there exists an action sequence a,b,c

Model checking for HML expressions: Tableau method

Introduction: Tableau for Boolean logic

- Goal: Determine the satisfiability of a logic formula
  - Decomposing the original formula in subformulas in a tree structure
  - In each node: a subformula of the original formula to be satisfied
  - Branches: determined by construction rules
- Construction rules of the tableau for Boolean logic



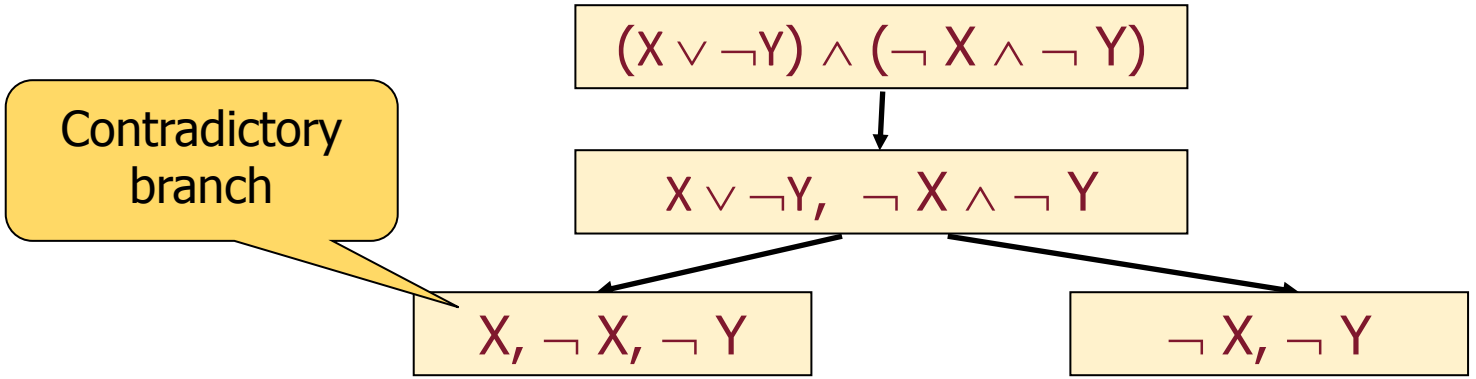
- Before decomposition, the formula shall be transformed to a negated normal form:
  - Negation ($\neg$) only before variables (literals) and not before a composite formula
  - De Morgan's laws can be used

Evaluation of the tableau

- Termination (closing) of a decomposition branch
  - Only a list of variables or negated variables is found in the current node
  - The satisfaction of the formula is given by assigning true or false (in case of negated variable) to variables
    - E.g., in case of $p, \neg q$ the assignment is: p is true, q is false
- After the termination of a decomposition branch:
  - "Contradictory" branch: A variable is found both with and without negation (i.e., there is no valid assignment)
    - E.g., contradiction in case of $p, \neg p, q$
  - "Successful" branch: There is no contradiction, the valid assignment satisfies the original formula
- The "successful" branches determine the satisfiability of the original formula

Example: Tableau for a Boolean logic formula

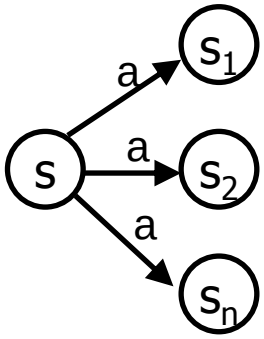
- Original formula: $\neg ((X \vee \neg Y) \rightarrow (X \vee Y))$
- Implication resolved: $\neg (\neg(X \vee \neg Y) \vee (X \vee Y))$
- Negated normal form: $(X \vee \neg Y) \wedge \neg(X \vee Y)$
  $(X \vee \neg Y) \wedge (\neg X \wedge \neg Y)$
- Construction of the tableau:



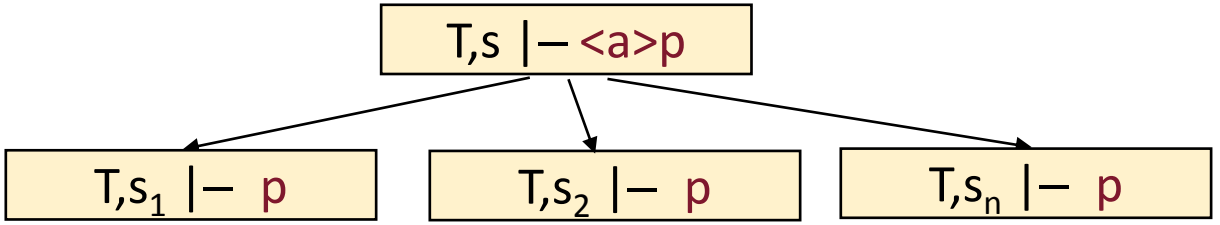
- One of the branches is contradictory, the other is not
  - The original formula can be satisfied

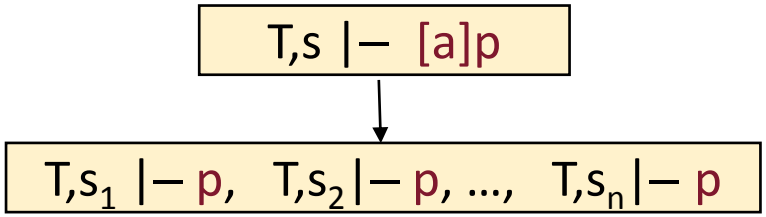
Checking HML by tableau construction

- Tableau construction rules in case of temporal operators:
  - Boolean operators: Branches as in case of Boolean tableau construction
  - Temporal operators: Shall be evaluated on the outgoing transitions of state s of the LTS T



where
$\{s_1,s_2,..,s_n\}=\{s' \mid s \rightarrow^a s'\}$





- The construction of the tableau is based on the model
- Successful branches:
  - s |- true is reached
  - s |- [a]p is reached where there is no outgoing transition labeled with a
- If the original formula is negated before the construction of the tableau:
  The successful branch is a counter-example

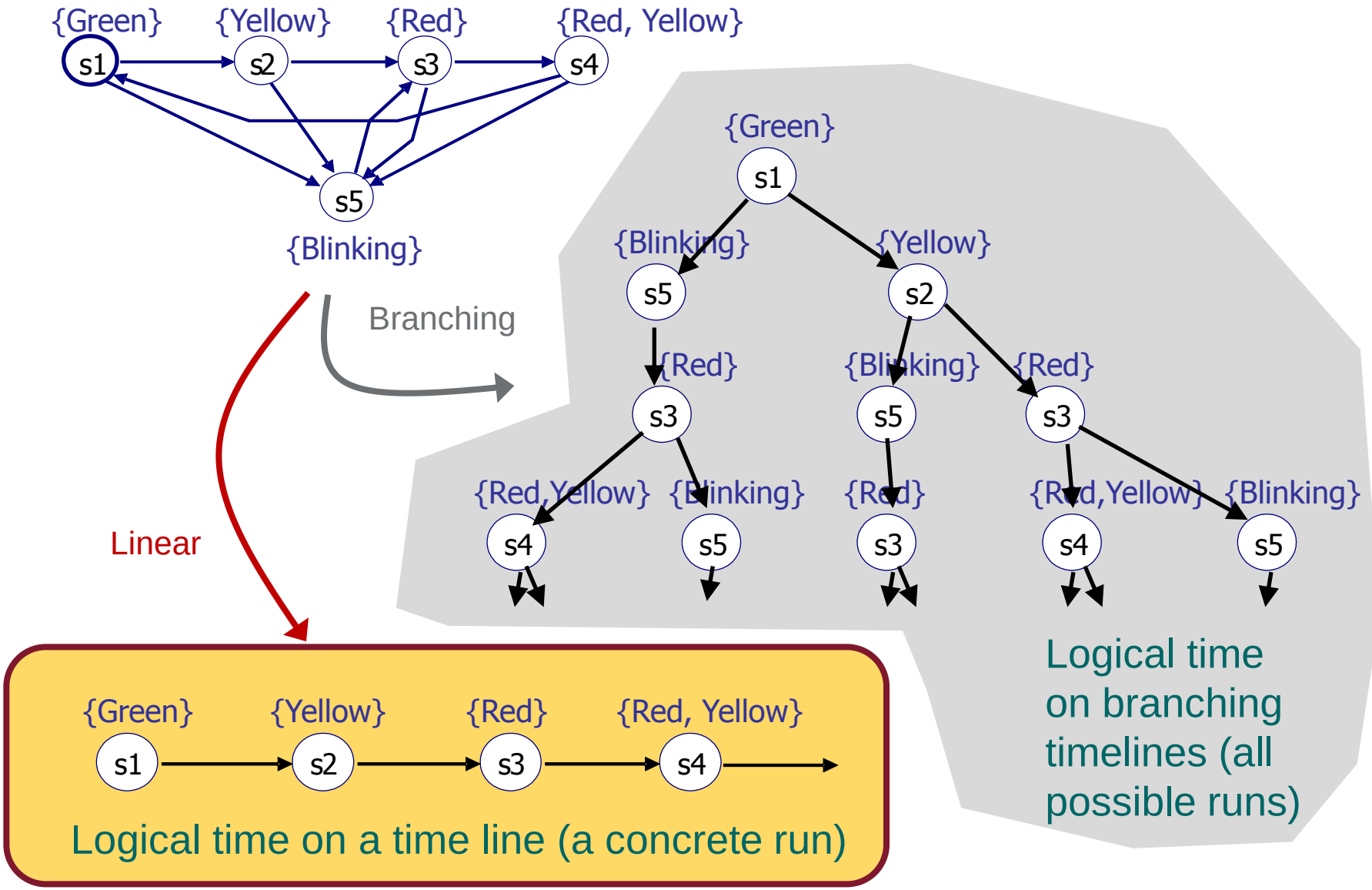
Linear Temporal Logic (LTL)

Temporal operators

Syntax and semantics

Examples

Illustration of linear and branching timelines

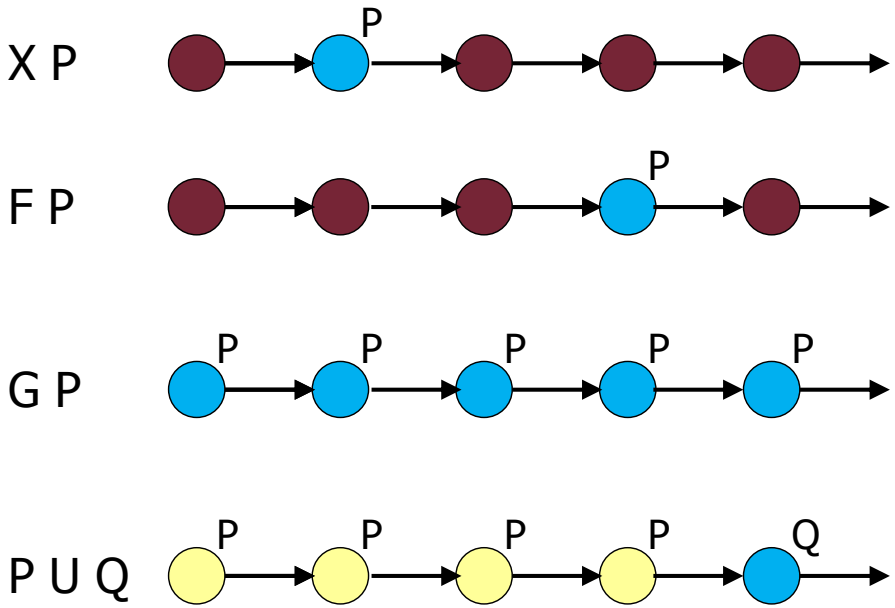


Linear temporal logic – Formulas

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

- Atomic propositions (elements of AP): P, Q, ...
- Boolean operators: $\wedge$, $\vee$, $\neg$, $\Rightarrow$

  $\wedge$: conjunction, $\vee$: disjunction, $\neg$: negation , $\Rightarrow$: 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 subsequent path
  - G p: "Globally p"
    p holds in all states
    on the subsequent path
  - p U q: "p Until q"
    p holds at least until q,
    which holds at the subsequent path



LTL examples

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

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

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

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

  Globally along the path (after any states), p will eventually hold
  - There is no state after which p does not hold eventually
  - Example: $\mathbf{GF}\ \text{Start}$; the Start state is reached from all states
- $\mathbf{FG}\ p$

  Eventually (after some states), p will continuously hold.
  - Example: $\mathbf{FG}\ \text{Normal}$
    After an initial transient the system operates 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**: $p \wedge q$ and $\neg p$ are wffs
- **L3**: p **U** q and **X** q are wffs

Precedence rules:

$$\mathbf{X}, \mathbf{U} > \neg > \wedge > \vee > \Rightarrow > \equiv$$

Derived operators

- true holds for all states
  false holds in no state

- $p \vee q$ means $\neg(\neg p \wedge \neg q)$
  $p \Rightarrow q$ means $\neg p \vee q$
  $p \equiv q$ means $p \Rightarrow q \wedge q \Rightarrow p$

- **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) $\wedge$ **F** q (strong before)

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

In any case, 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?

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

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

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

$M,\pi \models \mathbf{X}\, p$ iff $\pi^1 \models p$

Formalizing requirements: Example

Consider an air conditioner with the following operating modes:

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

- Concurrently more than one modes may be active
  - E.g. {On, Ventilating}
- When formalizing requirements, we might 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:

  $\mathbf{F}\ \mathrm{On}$
- At some point, the air conditioner always breaks down

  $\mathbf{GF}\ \mathrm{Error}$
- If the air conditioner breaks down, it eventually gets repaired

  $\mathbf{G}\ (\mathrm{Error} \Rightarrow \mathbf{F}\ \neg \mathrm{Error})$
- A broken air conditioner does not heat:

  $\mathbf{G}\ \neg(\mathrm{Error} \wedge \mathrm{Heating})$
- After heating, the air conditioner must ventilate:

  $\mathbf{G}\ ((\mathrm{Heating} \wedge \mathbf{X}\ \neg \mathrm{Heating}) \Rightarrow \mathbf{X}\ \mathrm{Ventilate})$
- After ventilation the air conditioner must not cool strongly until it performs some mild cooling:

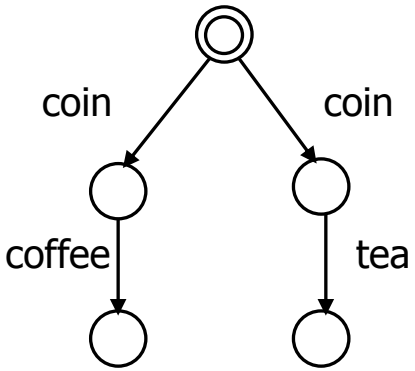
  $\mathbf{G}\ ((\mathrm{Ventilating} \wedge \mathbf{X}\ \neg \mathrm{Ventilating}) \Rightarrow \mathbf{X}(\neg \mathrm{StrongCooling}\ \mathbf{U}\ \mathrm{MildCooling}))$

Extending LTL for LTS

LTL: Transitions are labeled by actions

A path now 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.

This way we can describe requirements for action sequences

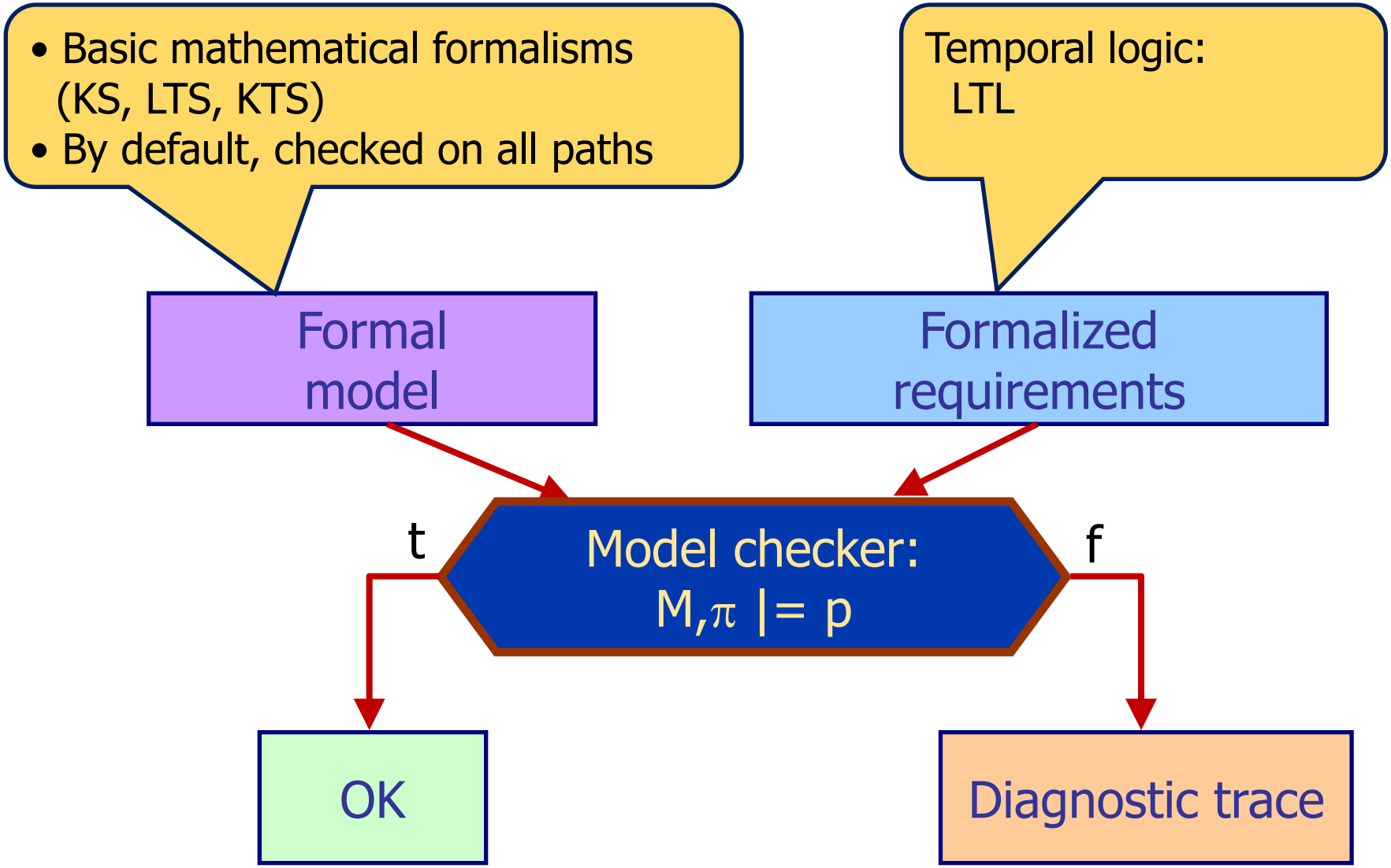
  - Example: $\mathbf{G}\ ((coin) \Rightarrow \mathbf{X}\ ((coffee) \vee (tee)))$

Checking LTL properties

Model checking problem

The automata-based approach

Model based verification by model checking

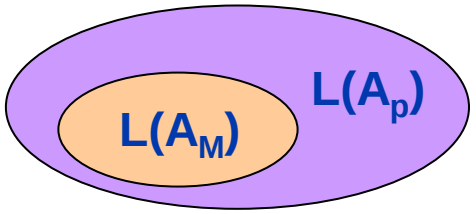


Automata based approach

- $A=(\Sigma, S, S_0, \rho, F)$ 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

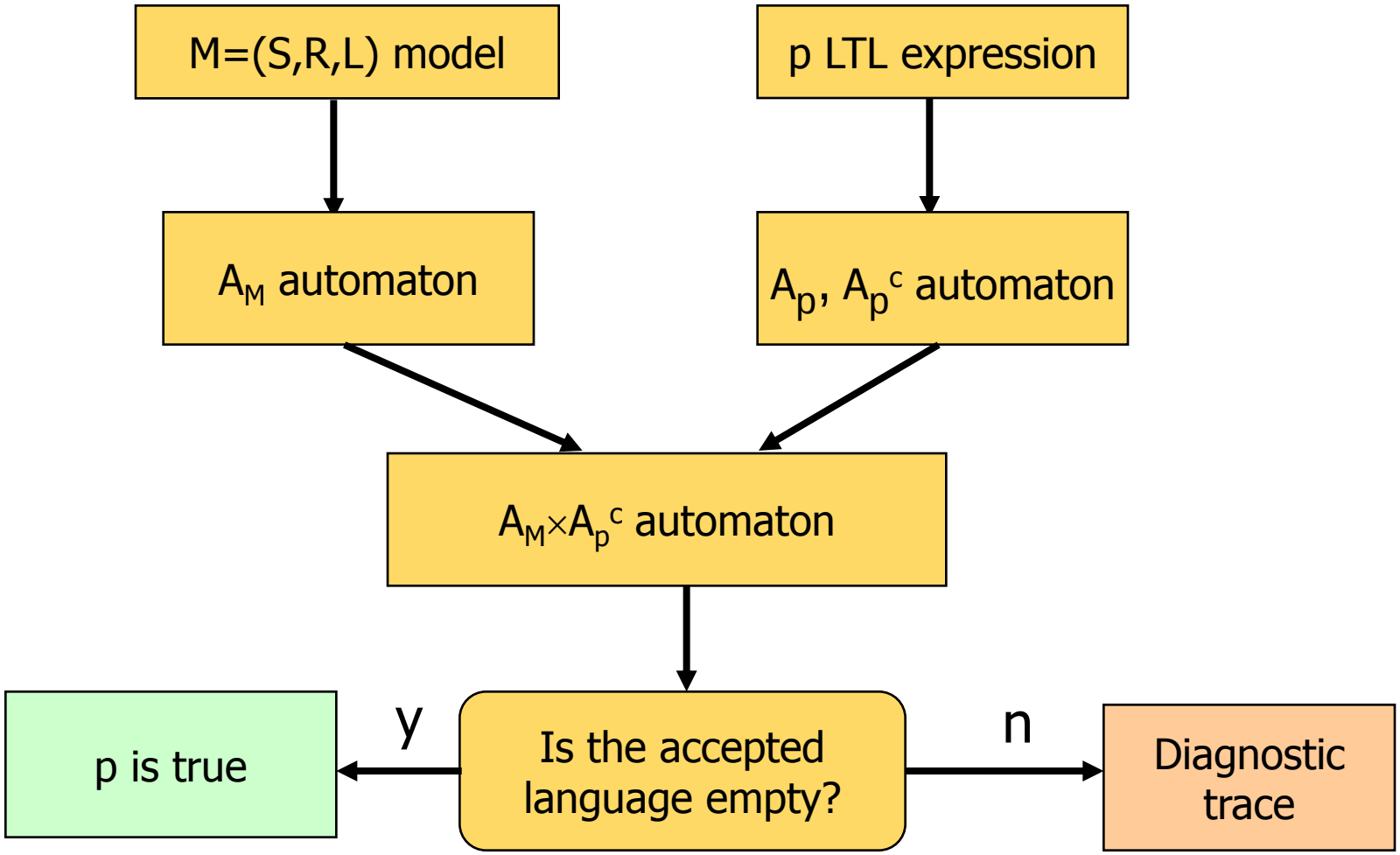
Model checking using the automata

- 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 (looking for cycles with accepting states)
    - A_p^c construction (in fully defined and deterministic case): swapping accepting states with non-accepting states and vice versa

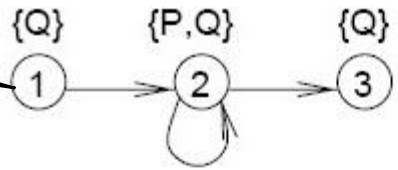
Overview of automata based model checking



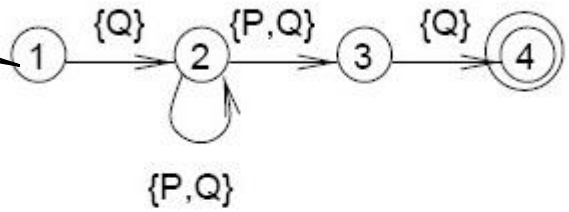
(In the following: Basic ideas will be discussed, not a complete algorithm.)

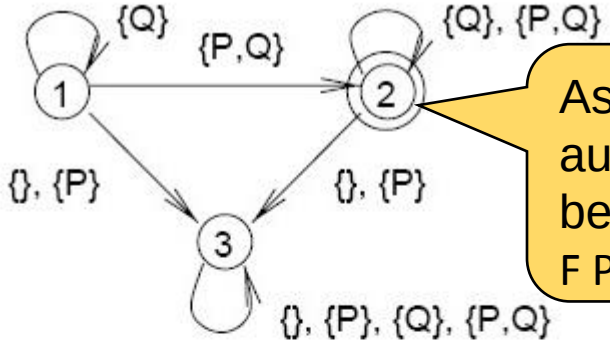
Example: Checking F P ∧ G Q

M model

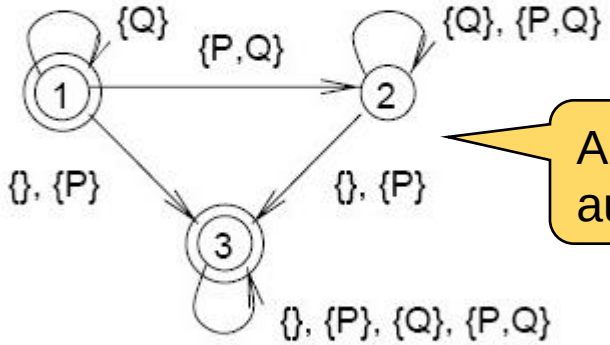


A_M automaton



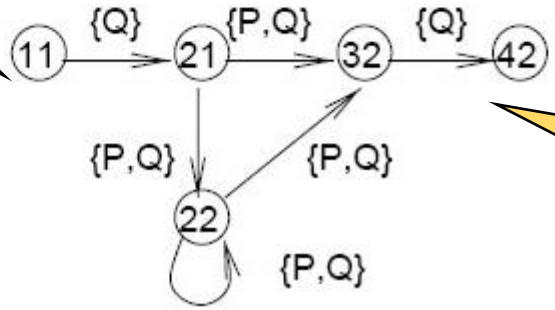


Assume: A_p automaton belonging to F P ∧ G Q



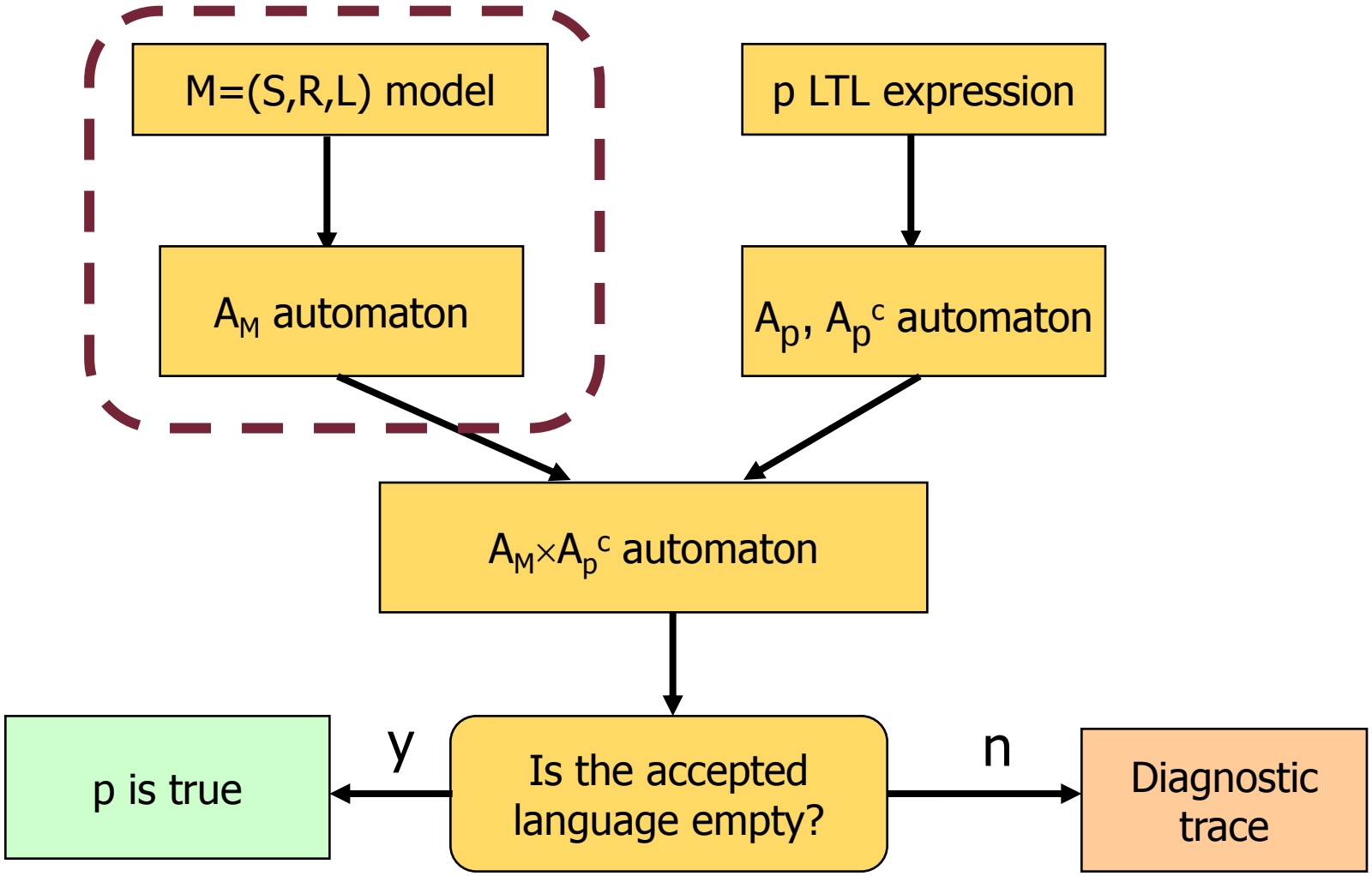
A_p^c automaton

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

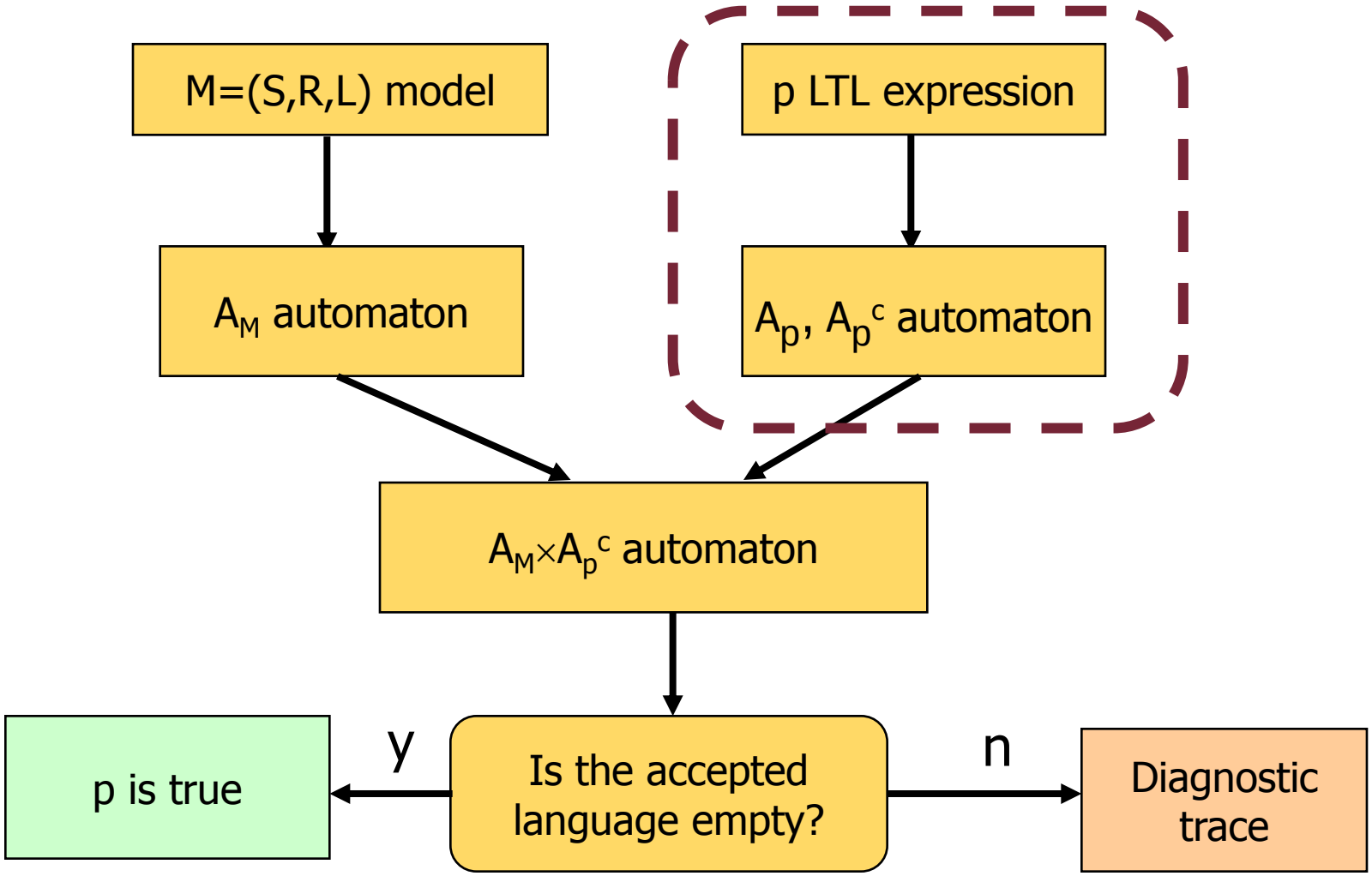
- Labels are moved to outgoing transitions
- In case of finite behavior (finite words):
  - Accepting state s_f is added
  - Transitions are added from the end states (without outgoing transition) to the accepting state s_f
- The automaton:

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

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



Basic idea: Constructing A_p on the basis of p (1)

- A_p automaton: Represents 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 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 next state
- Initial step: Construct the negated normal form of the expression
  - For Boolean operators: de Morgan laws
  - For temporal operators:

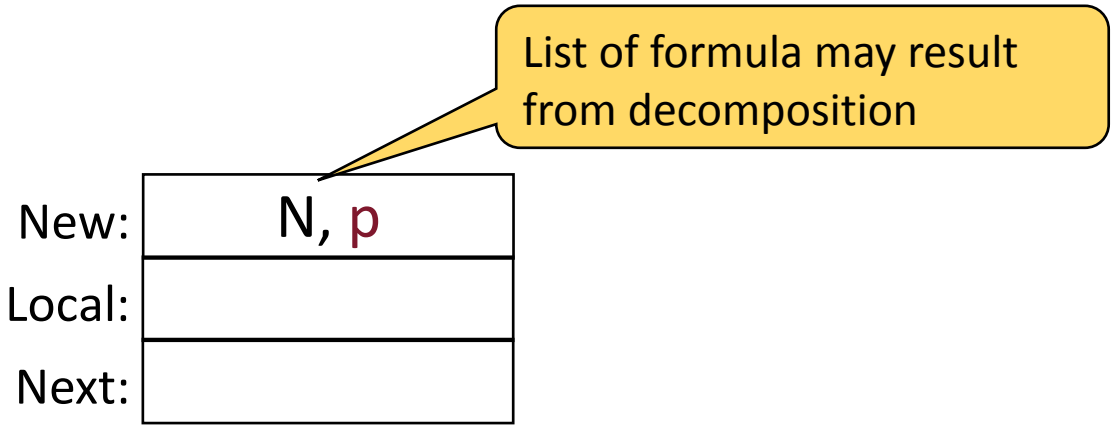
    $\neg(X\ p) = X\ (\neg p)$

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

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

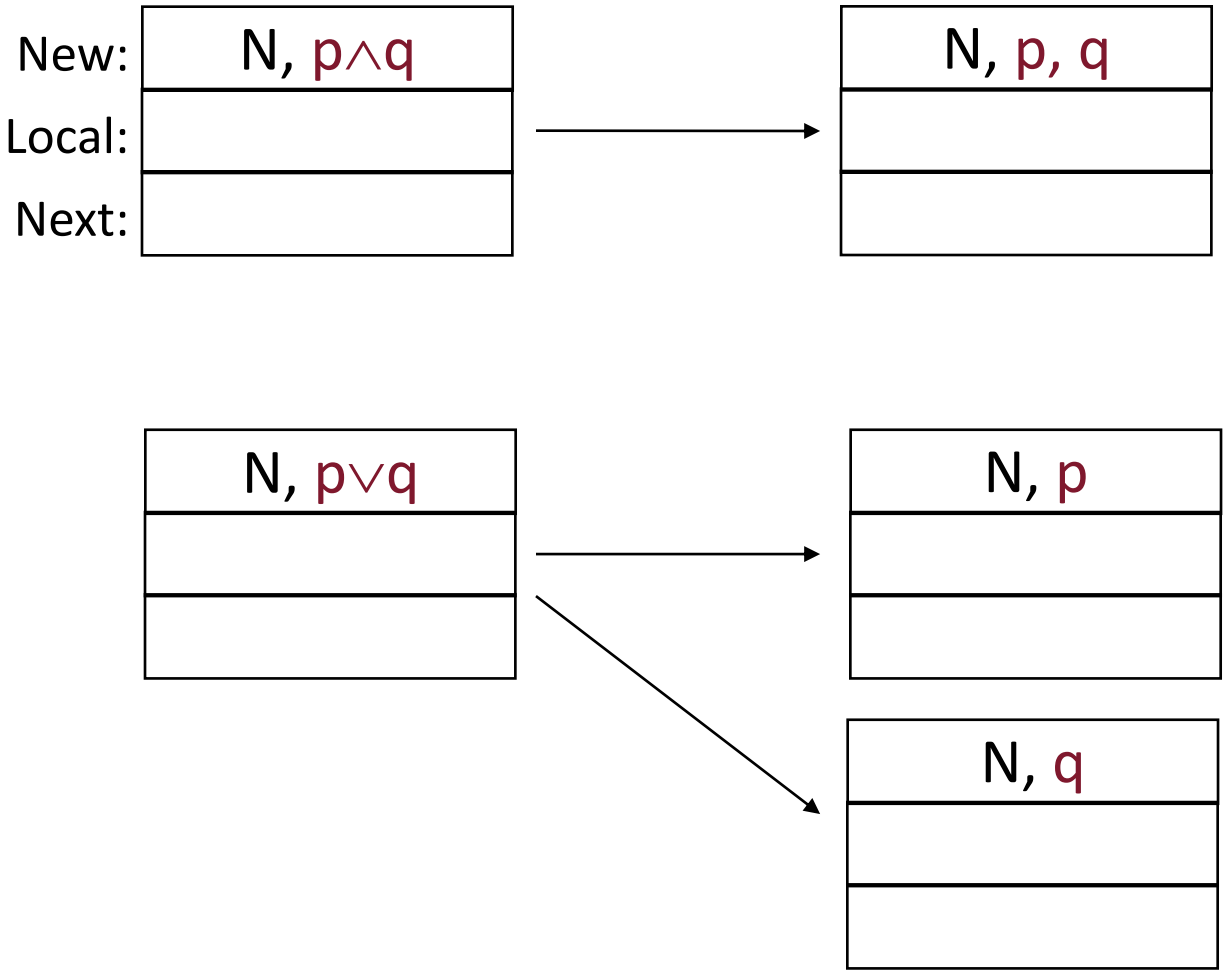
Constructing A_p on the basis of p (2)

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



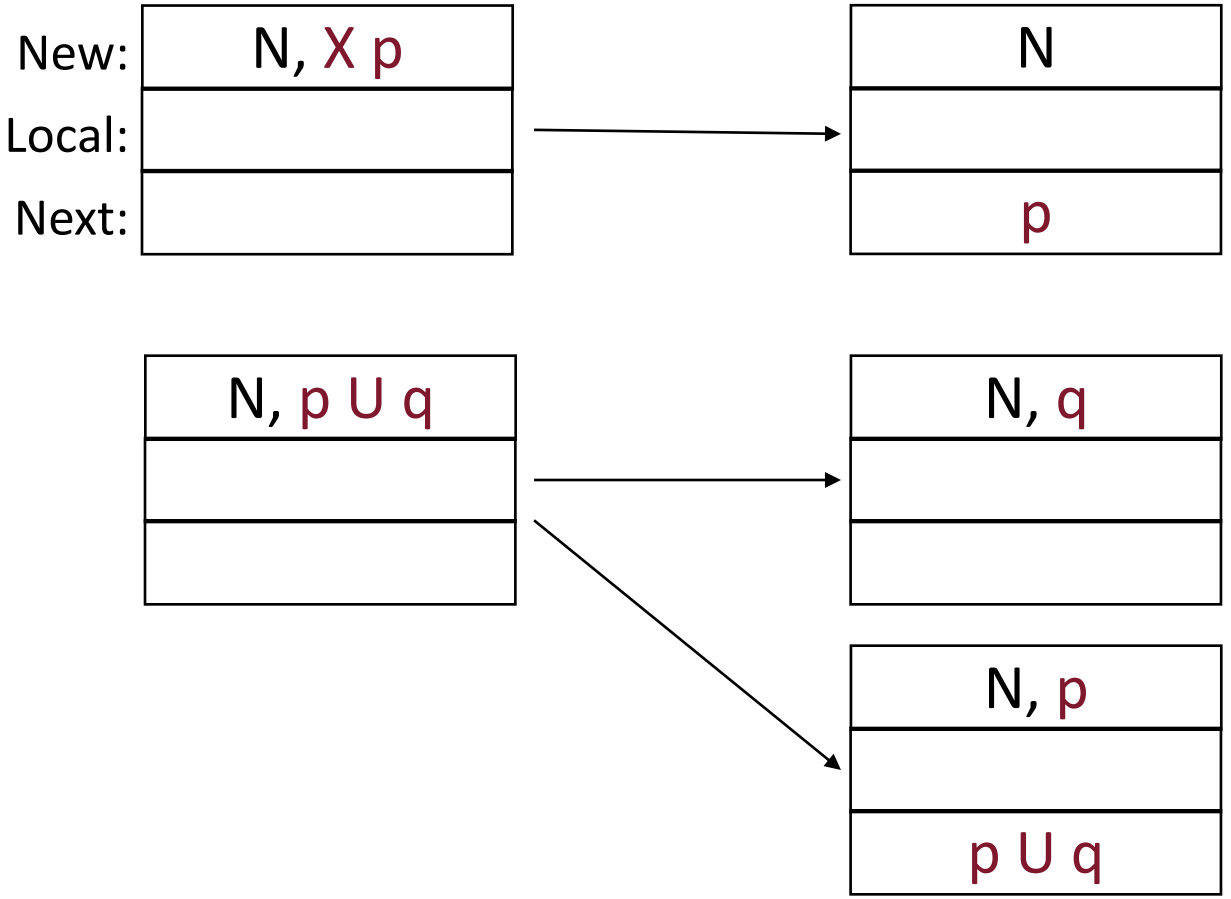
Constructing A_p on the basis of p (3)

- Decomposition rules for $\wedge$ and $\vee$:



Constructing A_p on the basis of p (4)

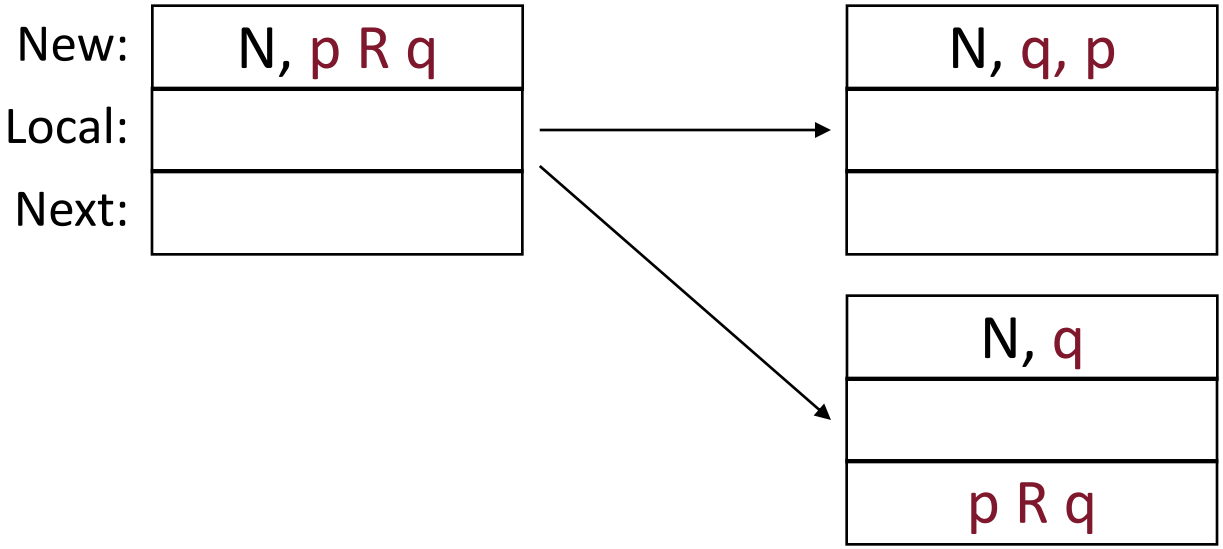
- Decomposition rules for X and U:



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

Constructing A_p on the basis of p (5)

- Decomposition rule for R:

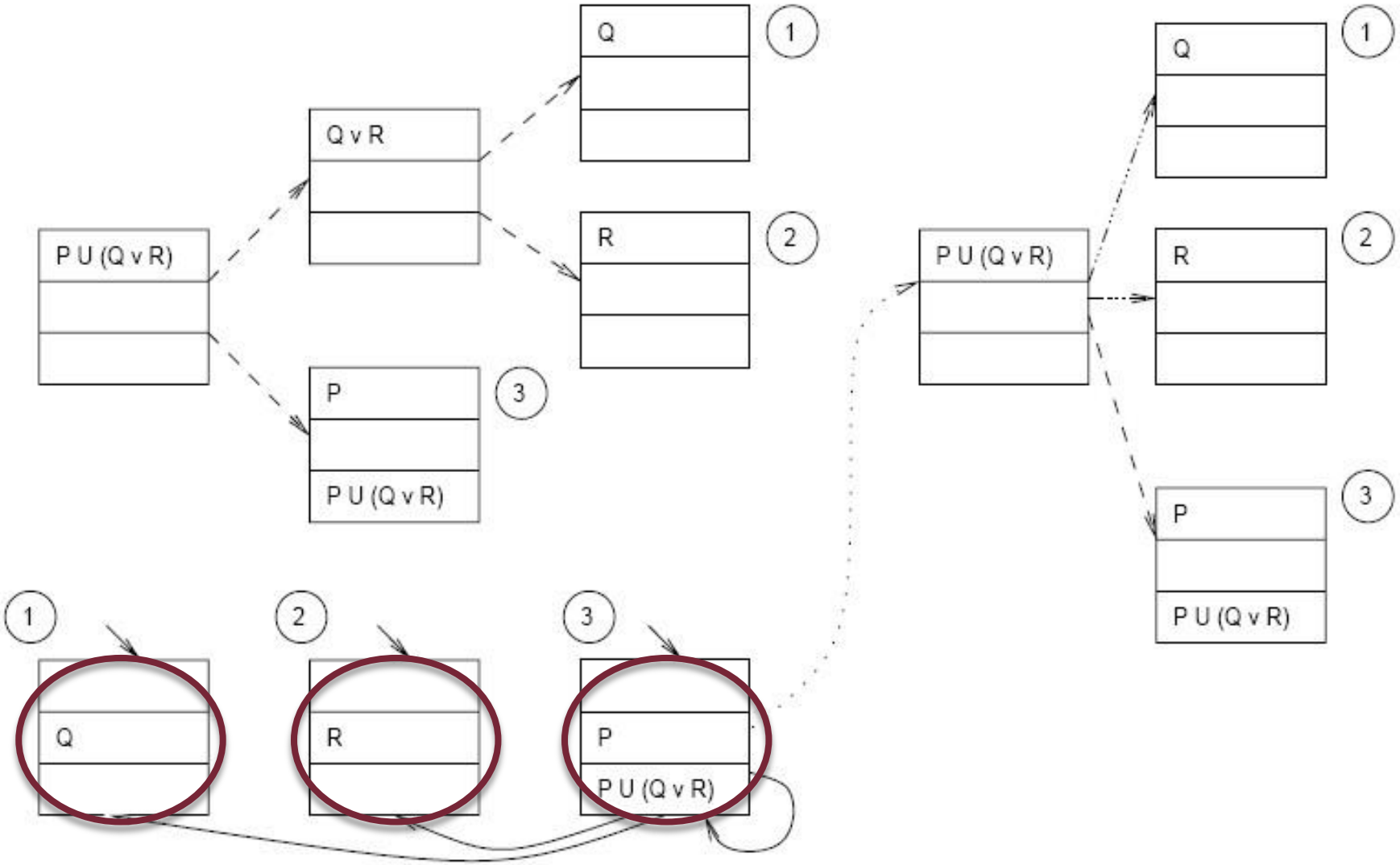


based on the rule $p\ R\ q = q \wedge (p \vee X(p\ R\ q))$

Constructing A_p on the basis of p (6)

- 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 become 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)
- 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 it was related to the path starting from the next state
  - Transitions of A_p are drawn from the state s to the states that result from the new decomposition
- Summary:
  - A_p states are identified when the decomposition results in atomic propositions (no further operator to be decomposed)
  - A_p transition is identified from a current node to the nodes that results from the decomposition of the formula in the Next field

Example: P U (Q ∨ R)

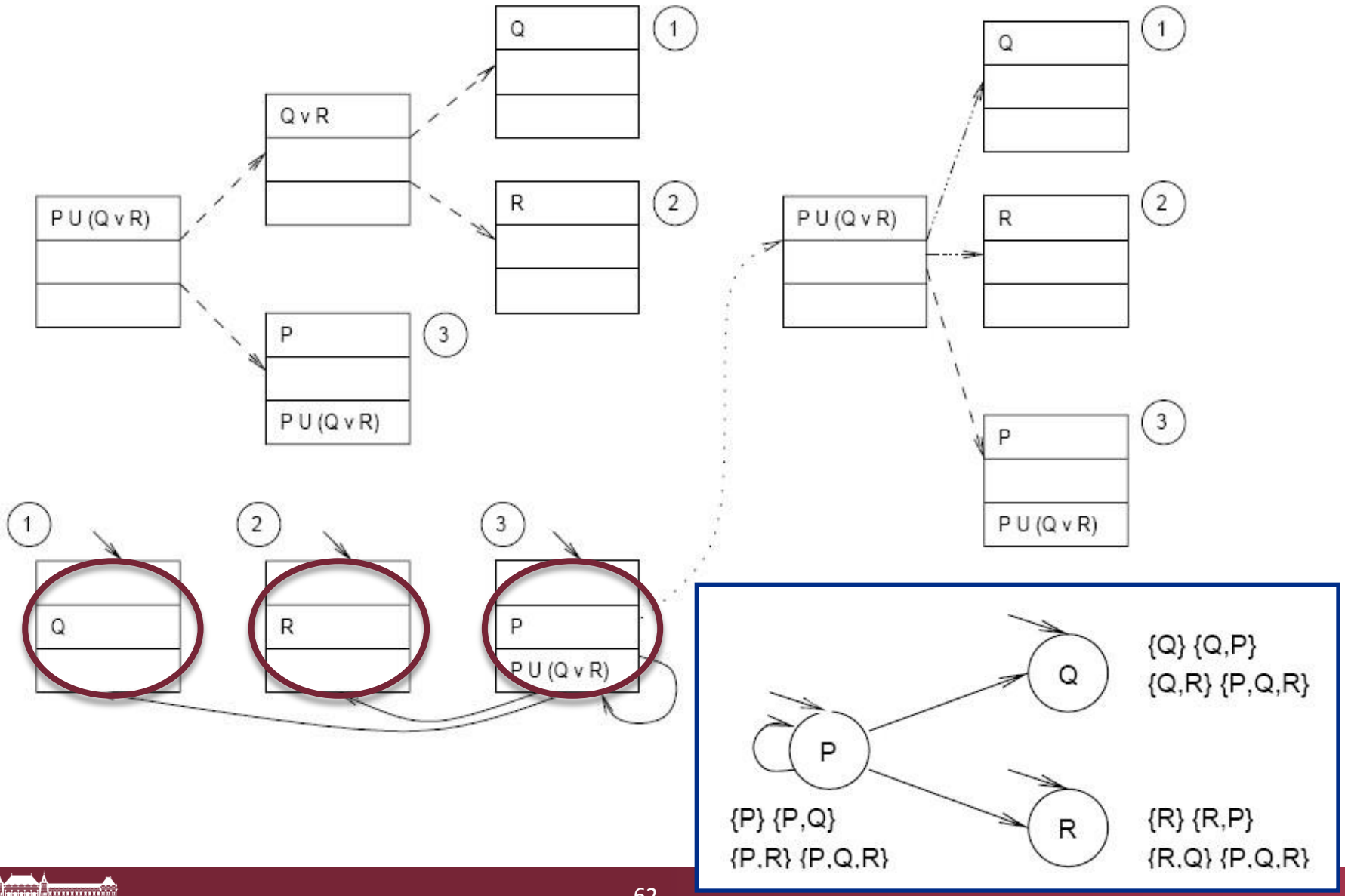


Constructing A_p on the basis of p (7)

- Further elements of A_p:
  - Initial state(s):
    - State(s) resulting from the first decomposition
  - Accepting states:
    - 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 listed 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}



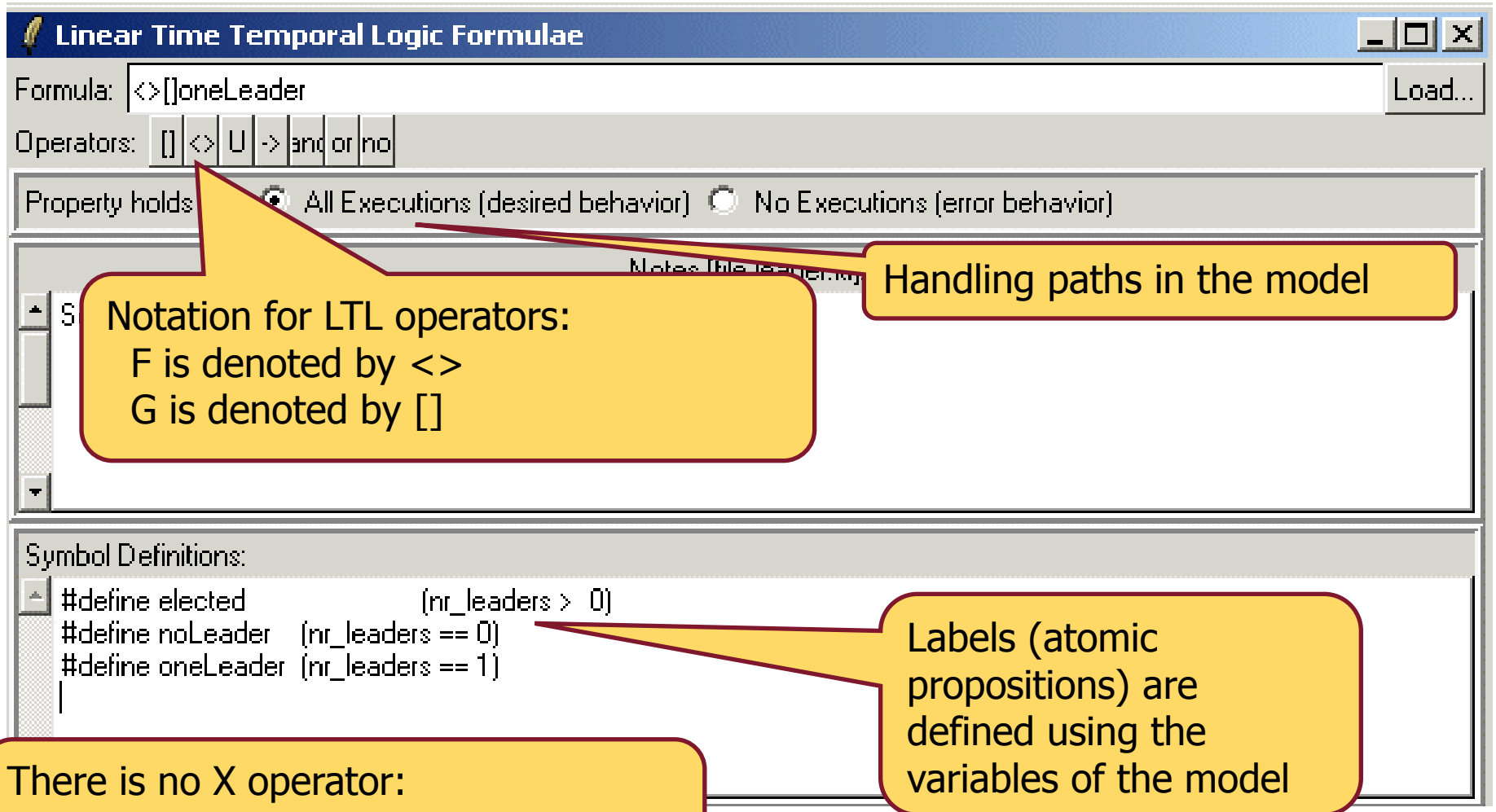
Complexity of PLTL model checking

- 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)
  - Complexity results from the size of the model automaton

The model checker SPIN



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

Handling paths in the model

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

- Formalization of requirements
- Temporal logics
- HML: Hennessy-Milner logic
  - Temporal operators
  - Model checking: Tableau method
- LTL: Linear Temporal Logic
  - Temporal operators
  - Syntax and semantics
  - Model checking: Automata based approach