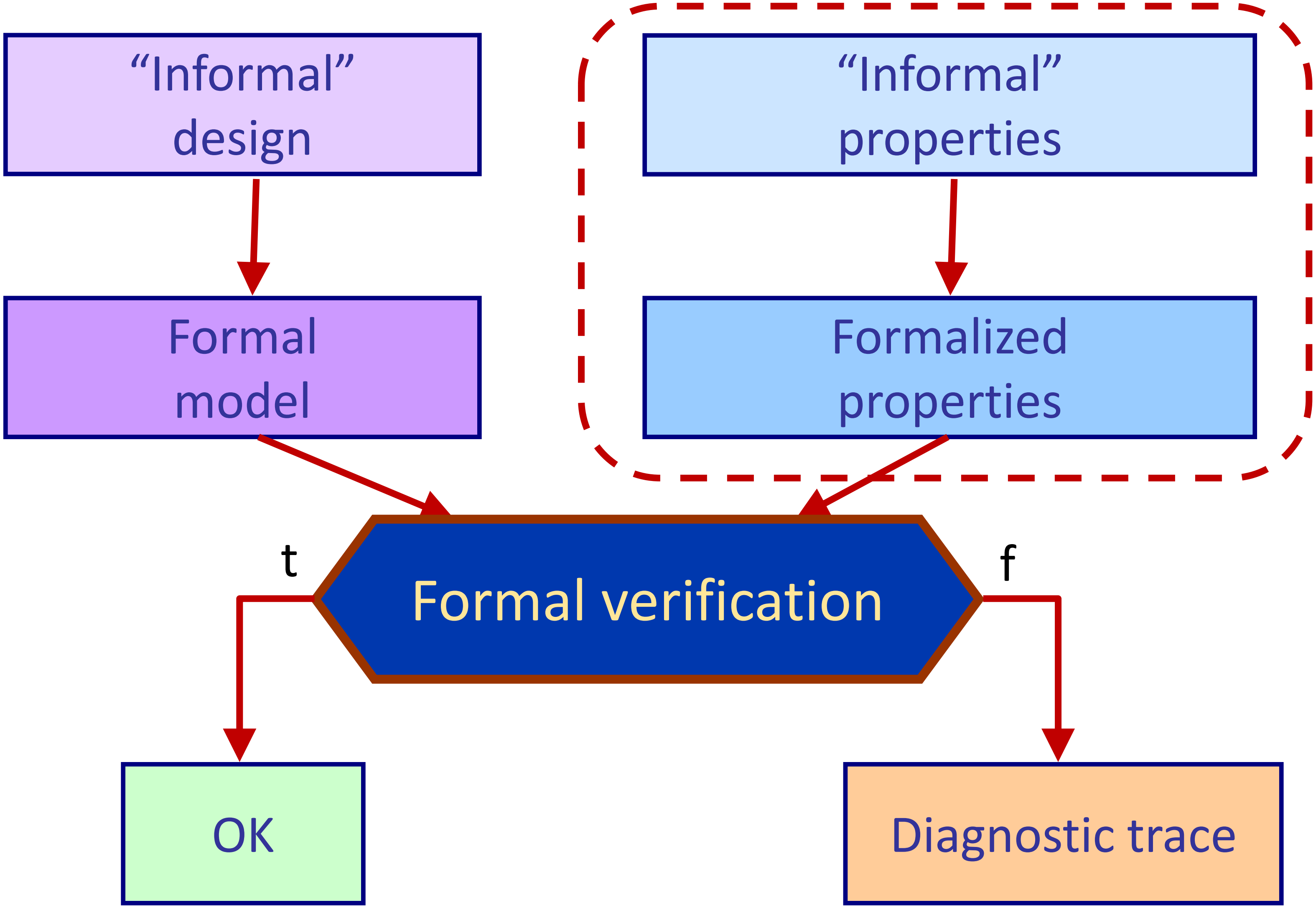
Formalizing and checking properties: Temporal logics CTL and CTL*

Istvan Majzik
majzik@mit.bme.hu

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

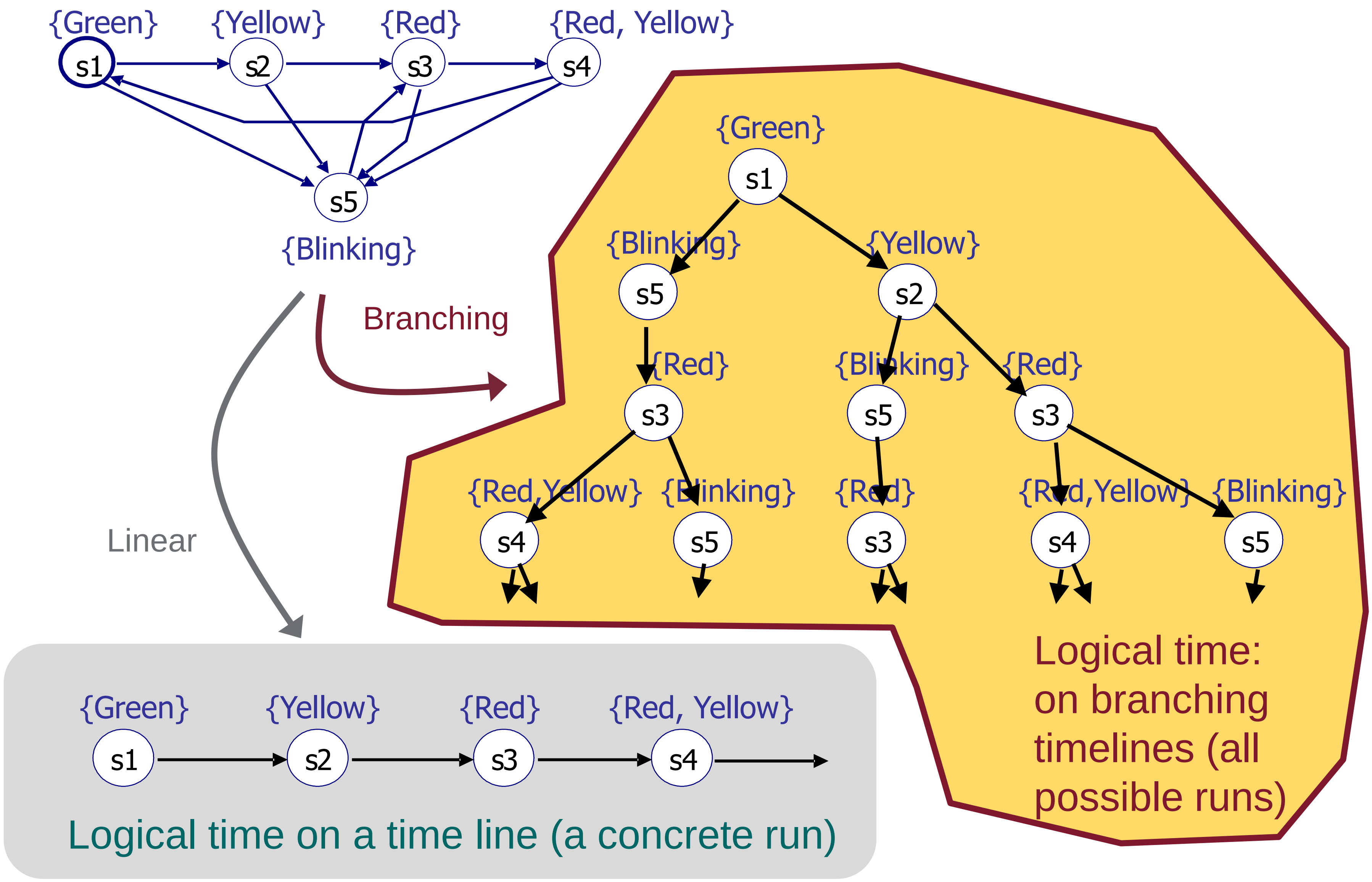
Formal verification: Goals



Overview

- Branching time temporal logics
- CTL*: Computational Tree Logic *
  - Operators
  - Syntax and semantics
- CTL: Computational Tree Logic
  - Operators
  - Syntax and semantics
  - Model checking CTL
- Outlook: Modal mu-calculus
  - Operators

Illustration of linear and branching timelines



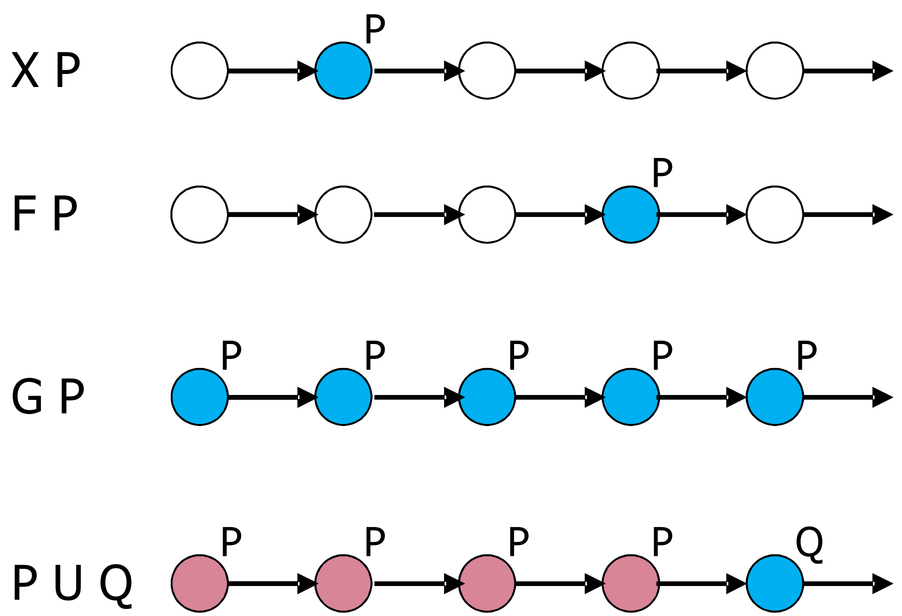
Recall: LTL operators on execution paths

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

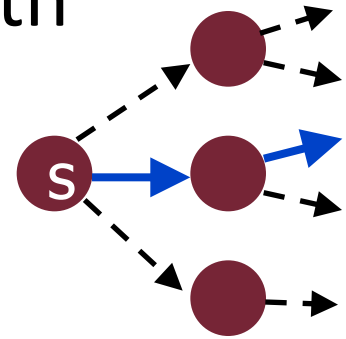
- Temporal operators: X, F, G, U informally:
  - X p: “neXt p”
    p holds in the next state
  - F p: “Future p”
    p holds eventually on the path
  - G p: “Globally p”
    p holds in all states on the path
  - p U q: “p Until q”
    p holds at least until q, which holds on the path



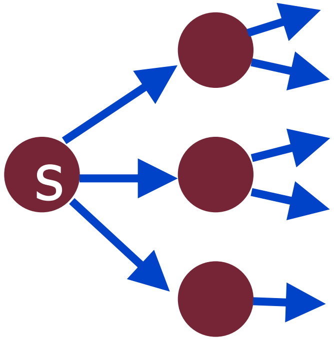
Branching: Path quantifiers

In a given state,
we formulate properties on the outgoing paths
from the state:

- E p (Exists p): there exists at least one path from the state for which p holds
  - Requirement on a single path
  - Existential operator



- A p (for All p): for all paths from the state p holds
  - Requirement on all possible paths
  - Universal operator



Branching time temporal logics

- CTL*: Computational Tree Logic *
  - An arbitrary combination of
    - path quantifiers (E, A),
    - and path-specific temporal operators (X, F, G, U)
  - E.g., EXXX p, A(X p $\vee$ F q)

- CTL: Computational Tree Logic
  - Specific CTL operators are formed:
    - Each temporal operator (X, F, G, U) is directly preceded by a path quantifier (E, A)
  - E.g. AX p, E(p U q)

CTL*: Computational Tree Logic *

Operators

Syntax and semantics

CTL* operators (informal)

- Path quantifiers (interpreted over states):
  - A: “for All futures”,
    for all possible paths from the current state
  - E: “Exists future”, “for some future”,
    for at least one path from the current state
- Path-specific operators (interpreted over paths):
  - X p: “neXt”, for the next state p holds
  - F p: “Future”, for a state along the path p holds
  - G p: “Globally”, for each state of the path p holds
  - p U q: “p Until q”, for a state of the path q will hold,
    and until then for all states p holds

CTL* formula examples

$$A(p \Rightarrow F\ q)$$

For all paths, we have that …

if initially p holds, …

then eventually …

q holds.

- $A(p \wedge G\ q)$

  For all possible paths: p holds (initially for the path) and q holds continuously for the path.

- $E(XXX\ p \vee F\ q)$

  There exists a path such that
  - p holds for its fourth state, or
  - eventually q holds

CTL* syntax

- State formulas: evaluated over states
  - **S1**: an atomic proposition P is a state formula
  - **S2**: for state formulas p and q,
    $\neg p$ and $p \wedge q$ are state formulas
  - **S3**: for a path formula p,
    E p and A p are state formulas
- Path formulas: evaluated over paths
  - **P1**: every state formula is a path formula
  - **P2**: for path formulas p and q,
    $\neg p$ and $p \wedge q$ are path formulas
  - **P3**: for path formulas p and q,
    X p and p U q are path formulas

Well-formed formulas in CTL*: state formulas

CTL* semantics: 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
- $M,\pi \models p$ (for a path formula p):
  in Kripke structure M, along path π, p holds
- $M,s \models p$ (for a state formula p):
  in Kripke structure M, in state s, p holds

CTL* semantics: State formulas

- **S1**:

  $M,s \models P$ iff $P \in L(s)$

- **S2**:

  $M,s \models \neg p$ iff not $M,s \models p$

  $M,s \models p \wedge q$ iff $M,s \models p$ and $M,s \models q$

- **S3**:

  $M,s \models E\ p$ (for path formula p)

  iff there exists a path $\pi=(s_0, s_1, s_2,\ldots)$ in M such that $s=s_0$ and $M,\pi \models p$

  $M,s \models A\ p$ (for a path formula p)

  iff for all paths $\pi=(s_0, s_1, s_2,\ldots)$ in M such that $s= s_0$ we have $M,\pi \models p$

CTL* semantics: Path formulas

- **P1**:

  $M,\pi \models p$ (for a state formula p) iff $M, s_0 \models p$

- **P2**:

  $M,\pi \models \neg p$ iff not $M,\pi \models p$

  $M,\pi \models p \wedge q$ iff $M,\pi \models p$ and $M,\pi \models q$

- **P3**:

  $M, \pi \models X\, p$ iff $M,\pi^1 \models p$

  $M, \pi \models p\ U\ q$ iff

  $\pi^j \models q$ for some $j \geq 0$ and
  $\pi^k \models p$ for all $0 \leq k < j$

Background: Computational complexity of evaluation

- Worst-case time complexity:
  at least $O\,(|S|^2 \times 2^{|p|})$
  - $|S|^2$ number of transitions in the model
    (Kripke structure) in the worst case
  - $|p|$ number of temporal operators in the formula
- The exponential complexity similar to LTL
  - Although temporal requirements tend to be short
- Goal: simplifying CTL*
  - Should remain usable in practice
  - Should reduce worst-case time complexity

CTL: Computational Tree Logic

Operators

Syntax and semantics

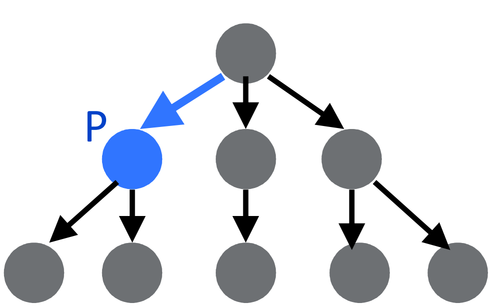
CTL operators (informal introduction)

Complex operators over states:

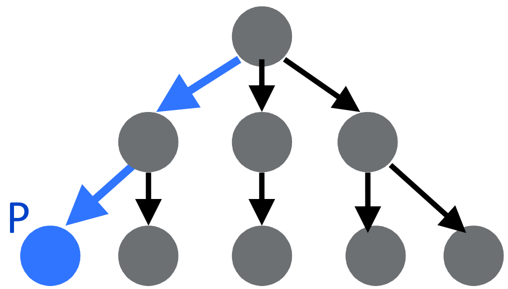
- EX p: there exists a path where p holds in the next state
- EF p: there exists a path where p holds in the future
- EG p: there exists a path where p holds globally
- E(p U q): there exists a path where p holds until q eventually holds

- 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

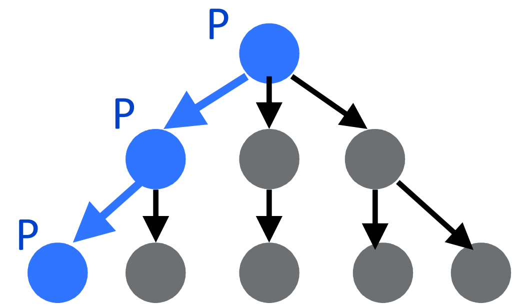
Illustration for CTL operators (examples)



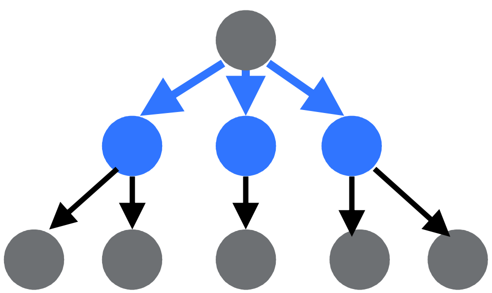
EX P



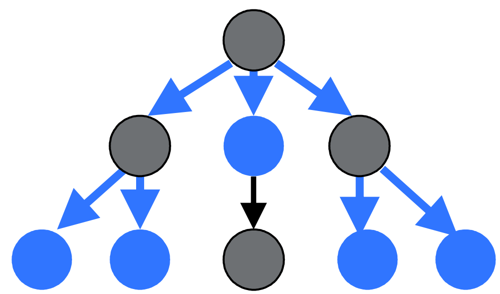
EF P



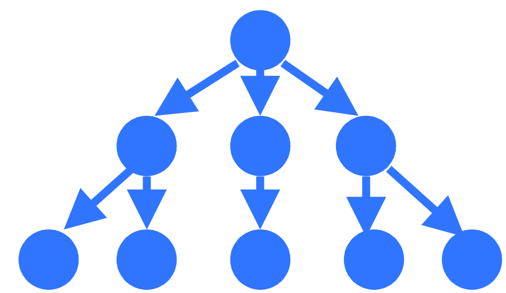
EG P



AX P



AF P



AG P

CTL formulas (examples)

- AG EF Reset

  Starting from any reachable state*,
  a state can eventually be reached where Reset holds
- AG AF Terminated

  Starting from any reachable state*,
  a state will eventually be reached where Terminated holds
- AG (Request $\Rightarrow$ AF Reply)

  Starting from any reachable state*,
  if we encounter a state where Request holds,
  then a state will eventually be reached where Reply holds.
- AF AG Normal

  Along all paths we will eventually reach a state
  from which Normal will always hold
- EF AG Stopped

  It is possible for the system to reach a state after which Stopped will hold in all states

* AG refers to states reachable from the initial state

Example: Formalizing requirements (1)

- Two processes in a system: P1 and P2
- The local properties of processes:
  - In critical section: C1, C2
  - Not in critical section: N1, N2
  - Waiting to enter critical section: W1, W2
- Atomic propositions:
  AP = {C1, C2, N1, N2, W1, W2}

Example: Formalizing requirements (2)

- There is at most one process in the critical section:

  $$AG\ (\neg(C1 \wedge C2))$$

- If a process is waiting to enter the critical section, then it will eventually enter the critical section:

  $$AG\ (W1 \Rightarrow AF(C1))$$
  $$AG\ (W2 \Rightarrow AF(C2))$$

- Processes enter the critical section in alternating order; one exits, then the other enters:

  $$AG(C1 \Rightarrow A(C1\ U\ (\neg C1 \wedge A((\neg C1)\ U\ C2))))$$
  $$AG(C2 \Rightarrow A(C2\ U\ (\neg C2 \wedge A((\neg C2)\ U\ C1))))$$

P2 in critical section

P2 not in critical section

P1 enters the critical section

CTL syntax

State formulas: The same as in CTL*

- **S1**: an atomic proposition P is a state formula
- **S2**: for state formulas p and q,
  $\neg p$ and $p \wedge q$ are state formulas
- **S3**: for a path formula p,
  $E\ p$ and $A\ p$ are state formulas

Path formulas: Only a single rule

- **P0**: for state formulas p and q,
  $X\ p$ and $p\ U\ q$ are path formulas

- Path formulas cannot be directly nested (only state formulas in P0)
- Path formulas are only used in rule S3:
  Path formulas $X\ p$ and $p\ U\ q$ can only be under E and A

Derived operators and example formulas

- Derived operators of CTL
  - EF p means E (true U p)
  - AF p means A (true U p)
  - EG p means $\neg$AF ($\neg$p)
  - AG p means $\neg$EF ($\neg$p)

- CTL* but not CTL
  - E(X Red $\vee$ F Yellow)

    Boolean operator between path formulas
  - A(X G (Red $\wedge$ Yellow)), and E(XXX Red)

    Nested path formulas

CTL formal semantics

- State formulas:
  - Rules **S1**, **S2**, **S3** (see CTL*) remain unchanged
- Path formulas:
  - Rules **P1**, **P2**, **P3** are replaced by a new rule **P0**:

P0: Only state formulas can be nested

- $M,\pi \models X\ p$ where p is a state formula iff
  $M,s_1 \models p$
- $M,\pi \models p\ U\ q$ where p,q are state formulas iff
  $M,s_j \models q$ for some $j \geq 0$ and
  $M,s_k \models p$ for all $0 \leq k < j$

Here we have state formulas according to syntax rule **P0**

Background: Computational complexity of evaluation

- Worst case time complexity: $O(|S|^2 \times |p|)$
  - $|S|^2$ number of transitions in the model (Kripke structure) in the worst case
  - $|p|$ number of temporal operators in the formula
- Complexity is lower than in case of CTL*
  - No $2^{|p|}$ factor
  - Expressive enough for many practical requirements
    - Safety requirements: AG
    - Liveness requirements: EF, AF
- What is the cost?
  - CTL* is more expressive than CTL

Expressive power

- A temporal logic is more expressive than another temporal logic iff
  - it is able to formalize all properties that the other logic can,
  - furthermore there is a property that can be expressed in the logic but not in the other logic
- Experience so far:
  - LTL can not consider branching (implicitly „for all paths”)
  - CTL is more restricted than CTL*, hence it is less expressive
  - CTL* also includes all properties expressible in LTL

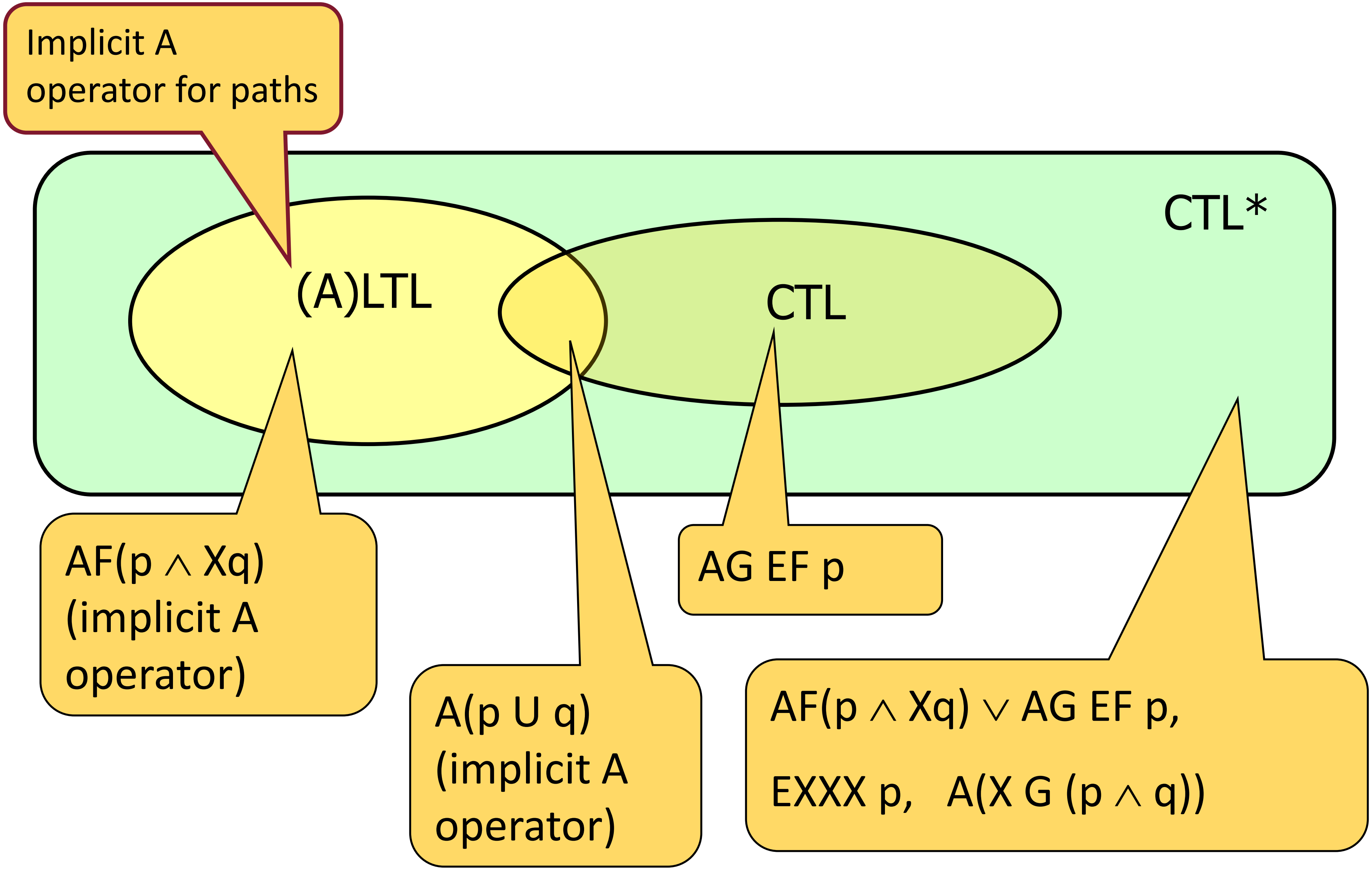
Expressive power – Formally

- The expressive power of TL2 is at least as big as the expressive power of TL1 iff
  for all Kripke structure M and for all its states s:

$$\forall p \in TL1:$$
$$\exists q \in TL2: \ (M, s \models p \quad \Leftrightarrow \quad M, s \models q)$$

- Iff this relation holds mutually then TL2 and TL1 have the same expressive power.

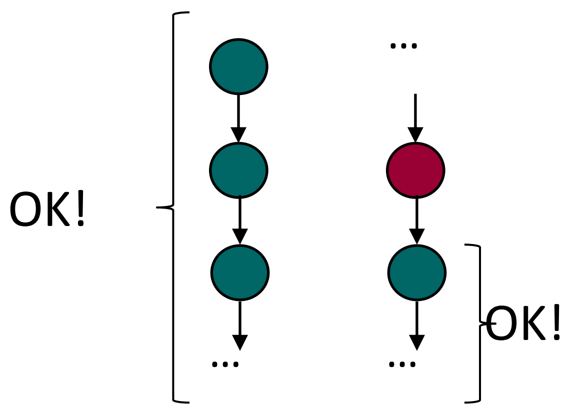
Expressive power of LTL, CTL, CTL*

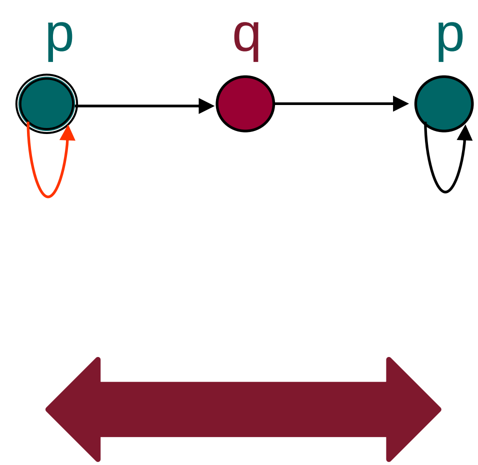


Expressive power of CTL and (A)LTL (in more detail)

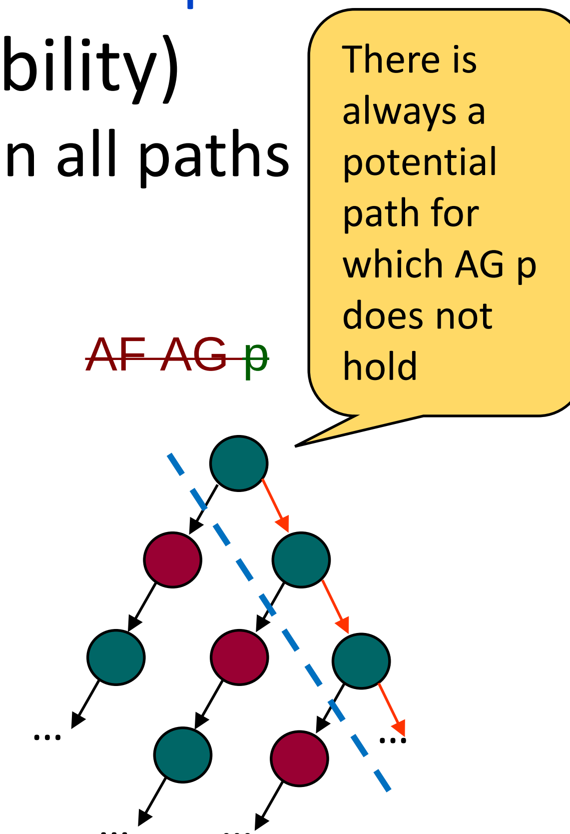
- Cannot be expressed in (A)LTL: AG EF p
  - In LTL there are no “possibilities”
  - In case of GF p: a state in which p holds shall be always reachable, while AG EF p allows paths without p
- Cannot be expressed in CTL: FG p (stability)
  - AF EG p not good, since p will not hold on all paths
  - AF AG p is too strict:

FG p





~~AF AG~~ p



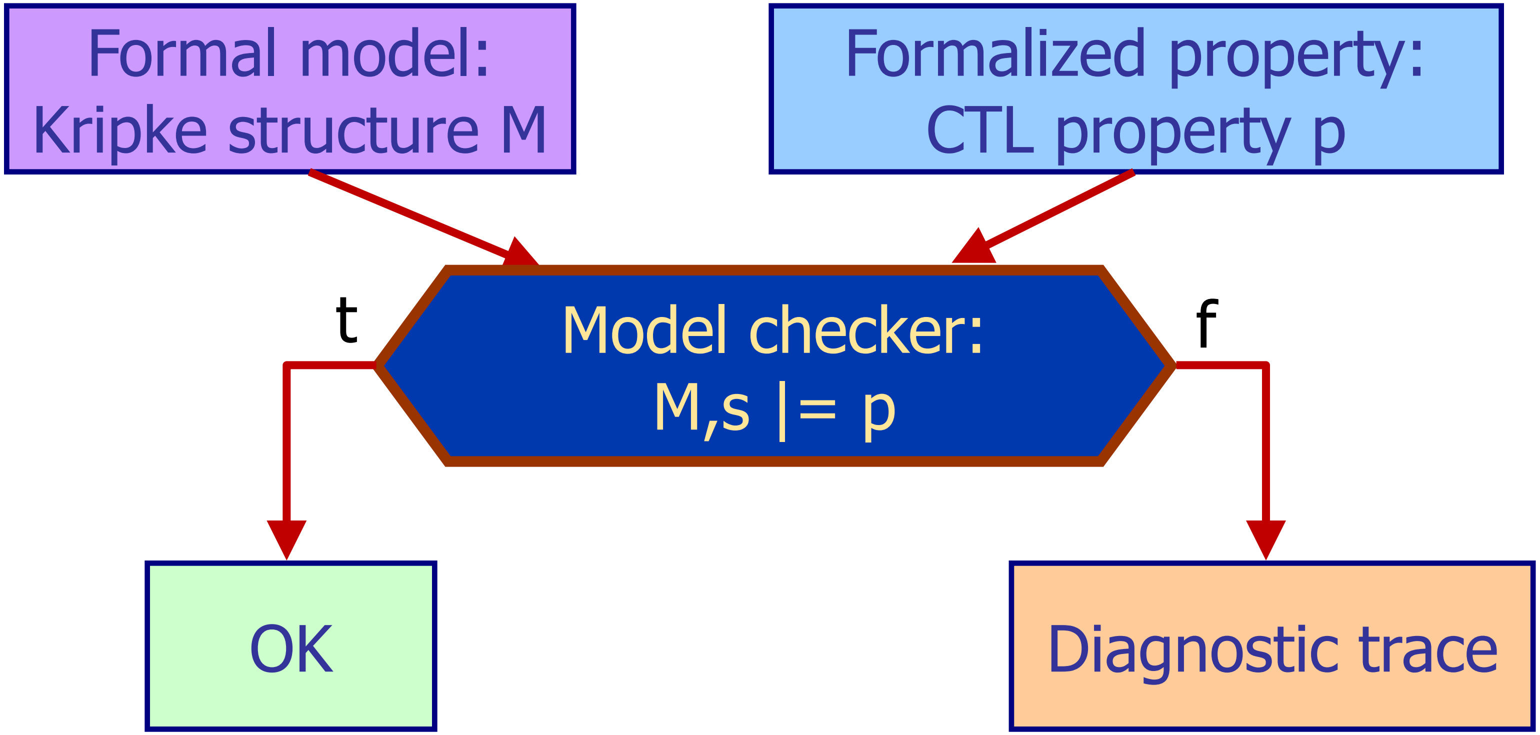
FairCTL: Specifying "fair" paths

- Properties shall be checked on "fair" paths only
  - Trivial counterexamples should be omitted:
    e.g., all messages are lost, the system is always reset etc.
- Fair paths are characterized by a q path formula in the form of:
  - GF r: The r state property occurs infinitely often (e.g., there is no starvation)
  - FG r: The r state property hold almost everywhere (e.g., stability is reached)
- Modified path quantifiers for fair paths:
  - A_q : for all "fair" paths
  - E_q : there exists a "fair" path
- Semantics of the modified path quantifiers:
  - $A_q F\ p$ means in CTL* $A(q \Rightarrow F\ p)$
  - $E_q G\ p$ means in CTL* $E(q \wedge G\ p)$
- Advantages of FairCTL:
  - Checking is restricted to "fair" paths
  - Complexity of checking FairCTL is less than the complexity of CTL*

CTL model checking

Semantics-based approach

Model based verification by model checking



Model checking approach

- Global model checking:
  - In case of CTL formula p: computing Sat(p), i.e., the set of states where p holds
  - This way $s \in Sat(p)$ can be checked for the initial state
- Sat(p) is computed in an “incremental” way, labeling the states with the sub-expressions of p
  - First step: States are already labeled with the atomic propositions of the formula
  - Next step: Labeling with sub-expressions of p that are composed by an operator from the existing labels
    - E.g., if states are labeled with p and q then p U q label is assigned
  - End of labeling: The original formula p is used as label

Labeling using sub-expressions

- Composition of a formula based on its syntactic structure (from inside out):

$$AF\ (\ P \wedge E\ (Q\ U\ R))$$

Q and R labels are included in the KS

E(_ U _) composition is applied, E(Q U R) label will appear

_ ∧ _ composition is applied, P∧E(Q U R) label will appear

AF _ composition is applied, AF(P∧E(Q U R)) label will appear

- Rules: Having labels p and q we establish where we have labels

  $\neg p$, $p \wedge q$, EX p, AX p, E(p U q), A(p U q)

- We progress “outwards” from the inside of a complex formula

Labeling rules: Based on the semantics (1)

- $\neg P$ holds in states s where $P \notin L(s)$
  - Rule: $\neg P$ label is applied on states s where there is no label P
- $p \wedge q$ holds in states s where both p and q are true
  - Rule: $p \wedge q$ label is applied on states s where both p and q labels are already present

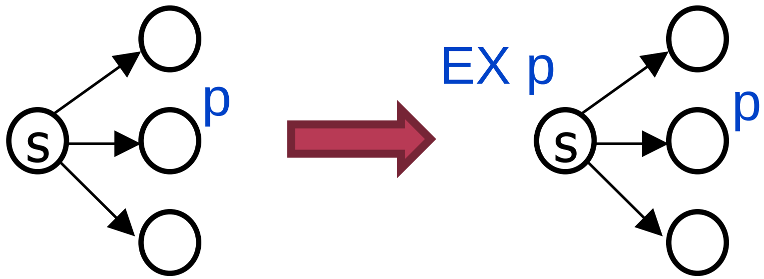
More complex rules for temporal operators

EX, AX refer to next states reachable from s

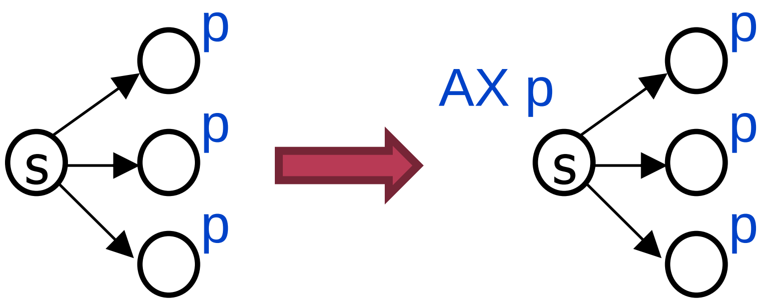
E(U), A(U) refer to paths reachable from s

Labeling rules: Based on the semantics (2)

- EX p holds in states s which have at least one next state in which p is true
  - Rule: State s is labeled with EX p, if it has at least one next state which is already labeled by p



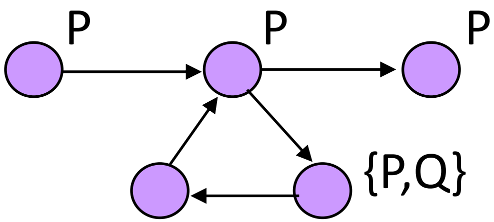
- AX p holds in states s if p is true in all next states of s
  - Rule: State s is labeled with AX p, if all of its next states are already labeled by p



Labeling rules: Based on the semantics (3)

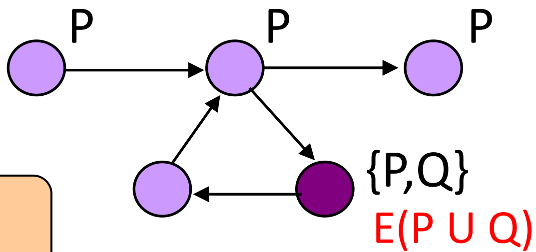
- Where does E(p U q) hold?
  - Decomposition: $E(p\ U\ q) = q \lor (p \land EX\ E(p\ U\ q))$
  - „Recursive” expression (in finite paths the last state needs specific care)
- Which states can be labeled with E(p U q)?
  - If state s is already labeled with q, or
  - if s is labeled with p, and there is at least one next state (cf. EX) that is already labeled with E(p U q)
- An iterative labeling algorithm is derived:
  - E(p U q) label is applied first on states that are already labeled with q
  - Then their predecessor states are checked:
    If label p is on a predecessor state then it is labeled with E(p U q)
  - … and so on until the set of labeled states increases
  - This way those paths are explored that lead to state with label q through states that are labeled with p

Example: Labeling with E(P U Q)

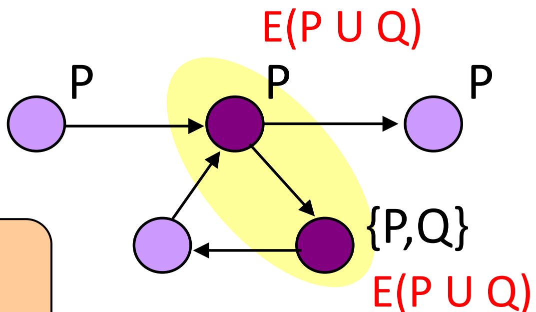


Kripke structure with initial labeling

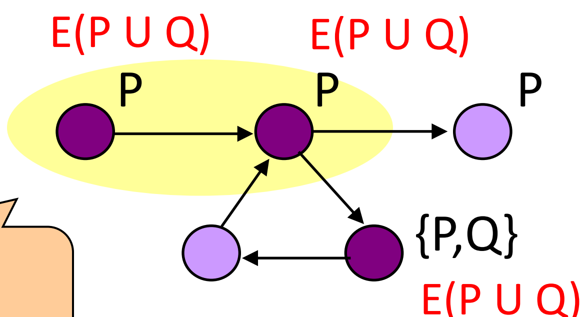
First step: Q



Second step:
$P \wedge EX$



Third steps:
$P \wedge EX$



- Exploiting:
  E(P U Q) =
  $Q \vee (P \wedge EX\ E(P\ U\ Q))$

- Iteration is finished when the set of labeled states does not change

Labeling rules: Based on the semantics (4)

- Where does A(p U q) hold?
  - Decomposition: $A(p\ U\ q) = q \vee (p \wedge AX\ A(p\ U\ q))$
  - „Recursive" expression (on infinite paths)
- Which states can be labeled with A(p U q)?
  - If state s is already labeled with q, or
  - if s is labeled with p, and all its next states are already labeled with A(p U q)
- An iterative labeling algorithm is derived:
  - A(p U q) label is applied first on states that are labeled with q
  - Then their predecessor states shall be checked:
    If label p is on a predecessor state and all its next states are already labeled with A(p U q) then it is labeled with A(p U q)
  - … and so on until the set of labeled states can be increased

This way all operators included in the formal syntax are covered.

Describing the labeling with set operations

- We need sets of states that have proper successor states
  - E(p U q): "At least one successor state is labeled …"
  - A(p U q): "All successor states are labeled …"
- Notation: If the set of states labeled with p is Z then
  - $pre_E(Z) = \{s \in S \mid$ there exists s', such that $(s,s') \in R$ and $s' \in Z\}$
    i.e., at least one successor is in Z (already labeled)
  - $pre_A(Z) = \{s \in S \mid$ for all s' where $(s,s') \in R$: $s' \in Z\}$
    i.e., all successors are in Z (already labeled)
- Example: Iterative labeling with E(P U Q)
  - Initial set: $X_0 = \{s \mid Q \in L(s)\}$
  - Next iteration: $X_{i+1} = X_i \cup (pre_E(X_i) \cap \{s \mid P \in L(s)\})$

    States labeled so far, plus …

    … their predecessor states that …

    … are labeled with P
  - End of iteration: If $X_{i+1} = X_i$, the set is not increased

CTL model checking: Summary

- Global model checking:
  - States are labeled with (sub)expressions that hold in that state
  - More and more complex (sub)expressions are used as labels until the original property formula is used as label
- Labeling with a (sub)expression:
  - Based on the existing labels (assigned in previous steps) applying labeling rules determined by the semantics of the operators
  - In case of EX, AX: Checking and labeling predecessor states
  - In case of E(p U q), A(p U q): Iterative labeling on paths
    - Initial set: Labeled on the basis of the q expressions
    - Iteration: Labeling p predecessor states on the basis of the semantics
    - End of iteration: The set of labeled states is constant
- Mathematical basis for model checking: Fixed-point iterations

Supplementary material: Fixed-point iterations and mu-calculus

Recap: Describing the labeling with set operations

- We need sets of states that have proper successor states
  - E(p U q): “At least one successor state is labeled …”
  - A(p U q): “All successor states are labeled …”
- Notation: If the set of states labeled with p is Z then
  - $pre_E(Z) = \{s \in S \mid$ there exists s’, such that $(s,s') \in R$ and $s' \in Z\}$
    i.e., at least one successor is in Z (already labeled)
  - $pre_A(Z) = \{s \in S \mid$ for all s’ where $(s,s') \in R$: $s' \in Z\}$
    i.e., all successors are in Z (already labeled)
- Example: Iterative labeling with E(P U Q)
  - Initial set: $X_0 = \{s \mid Q \in L(s)\}$
  - Next iteration: $X_{i+1} = X_i \cup (pre_E(X_i) \cap \{s \mid P \in L(s)\})$

States labeled so far, plus …

… their predecessor states that …

… are labeled with P

  - End of iteration: If $X_{i+1} = X_i$, the set is not increased

Background

- Iteration steps on sets can be given as a mapping (function) $\tau: 2^S \rightarrow 2^S$
  - Mapping from a set X_i to another set X_{i+1}: $X_{i+1} = \tau(X_i)$
  - The iteration ends when the set does not change:
    It is a fixed point in the application of the mapping, $X_{i+1} == X_i$
- Definitions:
  - Least fixed point: lfp $\tau(z)$ is the smallest $z \subseteq S$,
    for which fixed point is reached: $\tau(z)=z$
  - Greatest fixed point: gfp $\tau(z)$ is the biggest $z \subseteq S$,
    for which fixed point is reached: $\tau(z)=z$
- Theoretical background (theorems):
  - If S is finite then for monotonous τ there exist lfp τ and gfp τ
  - Computation of lfp: lfp $\tau(z) = \cup_i \tau^i(\emptyset)$ thus $\exists i_0$: lfp $\tau(z) = \tau^{i_0}(\emptyset)$
  - Computation of gfp: gfp $\tau(z) = \cap_i \tau^i(S)$ thus $\exists j_0$: gfp $\tau(z) = \tau^{j_0}(S)$

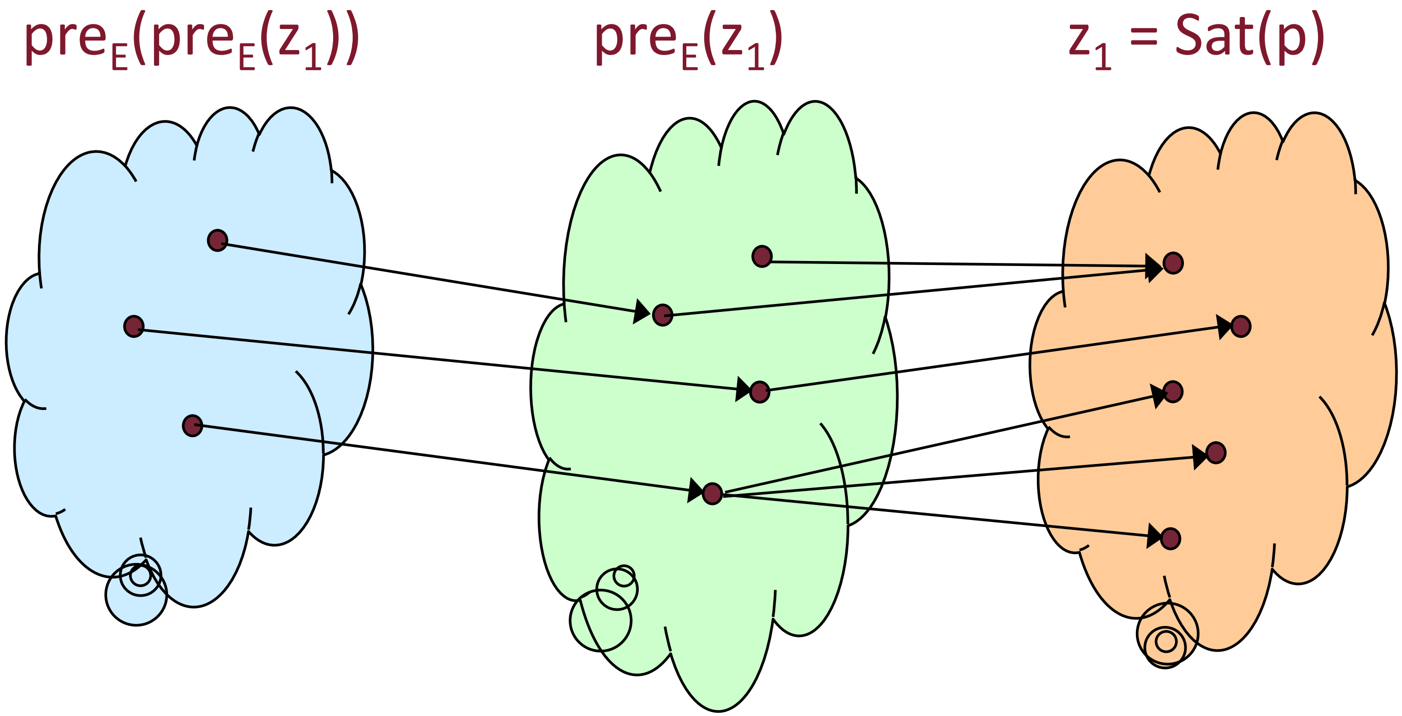
Mathematical theorems (1)

- Theorem: Sat(EF p)= lfp $\tau(z)$
  - where $\tau(z)$ = Sat(p) $\cup$ $pre_E(z)$ recap: EF(p)=p $\vee$ EX EF(p)
  - where $pre_E(z) = \{s \mid \exists t: (s,t) \in R \text{ és } t \in z\}$, as defined earlier

    i.e., the set of states from which there is transition to a state in z
- Applying the fixed point computation theorem: Union of sets
  - $z_0 = \emptyset$
  - $z_1 = \tau(z_0) = Sat(p) \cup pre_E(\emptyset) = Sat(p)$
  - $z_{i+1} = \tau(z_i) = Sat(p) \cup pre_E(z_i) = Sat(p) \cup \{s \mid \exists t: (s,t) \in R \text{ és } t \in z_i\}$
  - until $z_{i+1} == z_i$ and here z_i = lfp $\tau(z)$ = Sat(EF p)
- Here the fixed point computation:
  looking for paths backwards to initial states from states satisfying p
  - First step: $\emptyset$, from which Sat(p) is the first set
  - Then stepping backward on transitions according to $pre_E(z)$

Computation of the iteration

$\tau(z) = Sat(p) \cup pre_E(z)$



- Sat(p) is the result of the first iteration step
- Union with $pre_E(z)$ “steps” backwards on paths, looking for initial states for paths that lead to Sat(p)

Mathematical theorems (2)

- Theorem: $Sat(EG\ p) = gfp\ \tau(z)$
  - where $\tau(z) = Sat(p) \cap pre_E(z)$ recap: $EG(p) = p \wedge EX\ EG(p)$
  - where $pre_E(z) = \{s \mid \exists t: (s,t) \in R$ és $t \in z\}$ as defined earlier
- The iteration: Intersection of sets
  - $z_0 = S$
  - $z_1 = \tau(z_0) = \tau(S) = Sat(p) \cap pre_E(S)$
  - $z_{i+1} = \tau(z_i) = Sat(p) \cap \{s \mid \exists t: (s,t) \in R$ és $t \in z_i\}$
  - until $z_{i+1} == z_i$ and here $z_i = gfp\ \tau(z) = Sat(EG\ p)$
- Here the fixed point computation: looking for paths on which p is true, backwards to initial states from states satisfying p
  - First step: S
  - Then stepping backward on transitions according to $pre_E(z)$
- $Sat(E(p\ U\ q))$ computation is similar

Modal mu-calculus

- Syntax of mu-calculus on KTS:

$$p ::= P \mid Z \mid \neg p \mid p \wedge p \mid [a]p \mid \langle a \rangle p \mid \mu Z.p \mid \nu Z.p$$

- It contains directly the fixed point operators
  - $\nu Z.p$ is the greatest fixed point (where Z is a set variable, p is function of Z)
    - It is the biggest set $S^* \subseteq S$, that we get back when we compute $p(Z)$ with the interpretation that Z is S^*
  - $\mu Z.p$ is the least fixed point (where Z is a set variable, p is function of Z)
- Rule: Z shall occur in the scope of an even number of negations
  - This guarantees that functions (for iteration) will be monotonous, this way Sat(p) can be computed with iteration
- Expressive power is higher than CTL*
  - If a temporal logic is covered by the mu-calculus, then its model checking is possible by applying fixed-point iterations
- Worst case time complexity of checking: $O(|S|^2 \times |p|^a)$
  - Here a is the number of nested alternating (i.e., least / greatest) fixed point operations („alternation depth”)

CTL and the modal mu-calculus

- In case of CTL, the alternation depth of the corresponding mu-calculus formula is 1
  - E.g., $AG\ EF\ p = \nu Z.(\mu Y.(p \vee EX(Y)) \wedge AX(Z))$
  - There is no dependence between the nested fixed point operations: The "inner" fixed point formula does not depend on the variables of the "outer" fixed point formula
  - This way Sat(p) can be evaluated "from inside to outside", computation of the iterations belonging to the operators one by one
- In general case: There may be dependencies
  - E.g., $\nu Z.\mu Y.(\langle b\rangle Z \vee \langle a\rangle Y)$, means that there is a path consisting of a and b actions, where b occurs infinitely often
  - There is mutual dependency between the "inner" and "outer" fixed point formula
  - The iterations depend on each other, new inner iteration shall be computed in each step of the outer iteration

Summary

- Branching time temporal logics
- CTL*: Computational Tree Logic *
  - Operators
  - Syntax and semantics
- CTL: Computational Tree Logic
  - Operators
  - Syntax and semantics
  - Model checking
- Outlook: Modal mu-calculus
  - Fixed-point iterations
  - Mu-calculus operators