Lecture 3
Module I: Model Checking
Topic: Property specification in Temporal Logic CTL*

J.Vain

10.02.2022

Brushup: Model Checking

$$M \models P \ ?$$

Given:

- M – model
- P – property to be checked on the model M
- $\models$ – satisfiability relation („*M satisfies P*“)

Goal: Check if *M satisfies P*

If $M \models P$, it is said in logic that M is a model of formula P

Our model is Kripke Structure (KS)

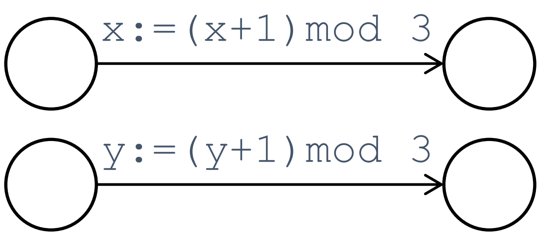
- Formally:

  KS is tuple (S, S_0, L, R) over a set of atomic propositions (AP) where
  - S set of symbolic states (a symbolic state encodes a set of explicit states)
  - S_0 is an initial state
  - L is a labeling function: $S \rightarrow 2^{AP}$
  - R is the transition relation: $R \subseteq S \times S$
- KS is a state-transition system that captures:
  - what is true in a state (labeling of the states with APs)
  - what can be viewed as an atomic move (denoted as state transition)
  - the succession of states (paths on the model graph)
- KS is a static representation that can be unfolded to a *tree of execution traces* on which temporal properties are verified.

Representing transition as formula

- In Kripke structure, transition $(s, s') \in R$ corresponds to one step of program execution.

- Suppose a program has two steps
  - x := (x+1) mod 3;
  - y := (y+1) mod 3.



- Then

  $R = \{R_1, R_2\}$
  - $R_1 : (x' = (x+1)\ mod\ 3) \wedge (y' = y)$
  - $R_2 : (y' = (y+1)\ mod\ 3) \wedge (x' = x)$

Consecutive States

- State space S:

We can restrict our attention to pairs of consecutive states $s = (x, y)$ and $s' = (x', y')$ in the state space $\{0, 1, 2\} \times \{0, 1, 2\}$, i.e.

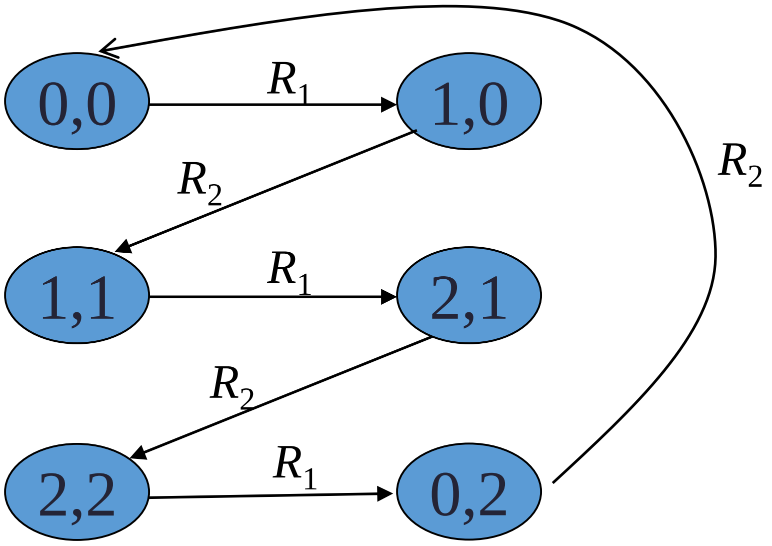
$$(s, s') \in \{0, 1, 2\} \times \{0, 1, 2\}$$

- Question: Can we construct a logic formula that describes the relation between <u>any</u> two consecutive states s and s'?

- Assume each pair of consecutive states is an instance of R, e.g. in set notation we have $R = \{R_1, R_2\}$ and in logic notation $R \equiv (R_1 \vee R_2)$

Set of transitions is represented by $R_1 \vee R_2$

By connecting pairs of consecutive states we get execution paths of KS



Representing transitions (revisited II)

- In Kripke structure, a transition $(s, s') \in R$ corresponds to one step of program execution.

- For instance, if a program *P* has two commands
  - x := (x+1) mod 3;
  - y := (y+1) mod 3;

- then for the whole program we have transition relation *R*

  $R \equiv ((x' = x{+}1 \; mod \; 3) \wedge y' = y) \vee ((y' = y{+}1 \; mod \; 3) \wedge x'{=}x)$

- (s, s') that satisfies *R* means that from state *s* we can get to *s'* by some step of execution that satisfies *R.*

A ‘giant’ *R*

- Now we can compute *R* for the whole program
  - then we will know whether any of states is one-step reachable from some other

- Convenient, but globally we loose information:

  e.g., the order in which the statements are executed

- Comment:
  - without ordering, the disjuncts in *R* have <u>not clear precedence information</u>!

Introducing program counter

- In the computer, the order of executing commands is controlled by *program counters.*
- We introduce an auxilliary variable *pc* (for programm counter), and assume the commands in program are labeled with $l_0, \ldots, l_n$.
- For instance
  - In the program:
    - `l`$_0$`: x := x+1;`
    - `l`$_1$`: y := x+1;`
    - `l`$_2$`: …`
  - The effect of executing commands is represented in logic:
    - $R_1 : x' = x+1 \wedge y'=y \wedge pc = l_0 \wedge pc' = l_1$
    - $R_2 : y' = y+1 \wedge x'=x \wedge pc = l_1 \wedge pc' = l_2$

Now we have complete symbolic representation of program execution in our computation model *M*!

Brushup: Model Checking

$$M \vDash P \ ?$$

Given:

- *M* – model
- *P* – property to be checked on the model *M*
- $\vDash$ – satisfiability relation („*M satisfies P*“)

Goal: Check if *M satisfies P*

If $M \vDash P$ it is said in logic that *M* is a model of formula *P*

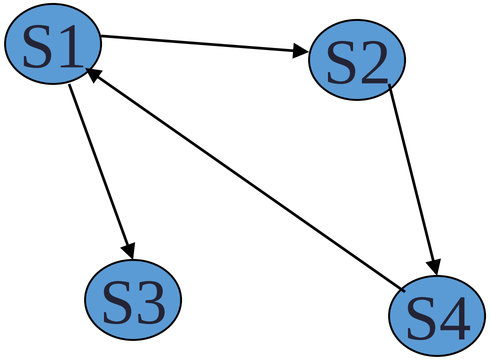
We have seen how M is constructed symbolically

How to express P in logic?

Temporal logic CTL*

- Let's start with semantics

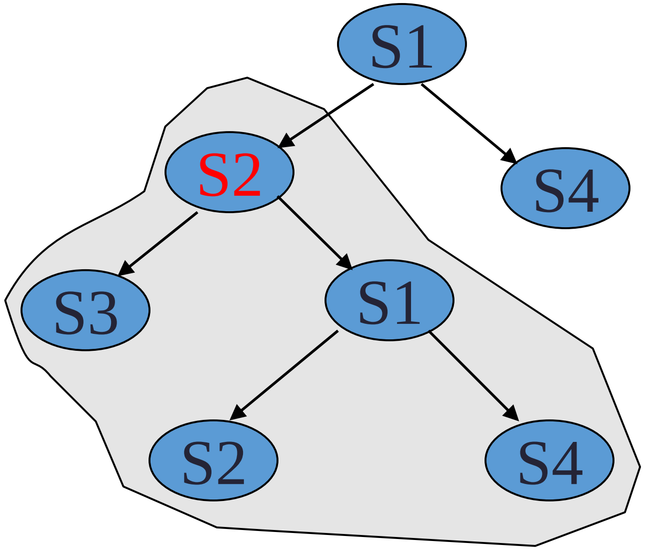
  KS and its logic representation provide us static model of program execution



Dynamic model of program execution is unfolding of the static model

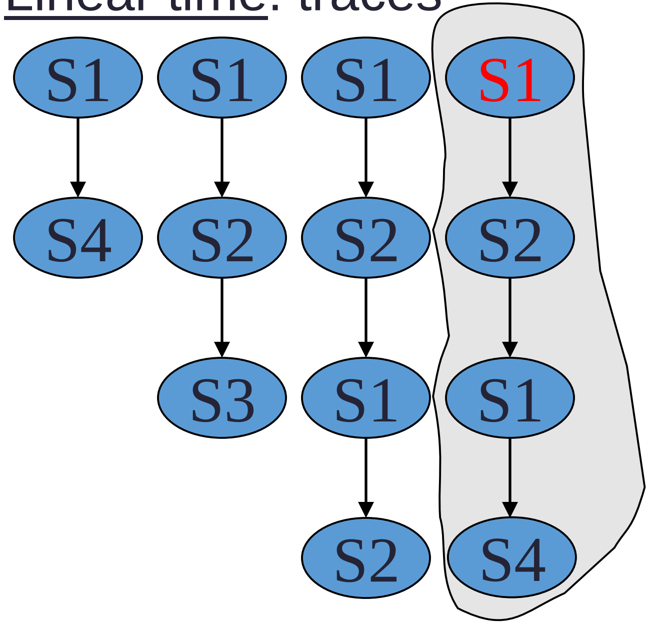
2 options of unfolding to define operational semantics:

Branching time: tree structure



Is a formula valid in given node (e.g. in S2), which is the root of a subtree?

Linear time: traces



Is a formula valid along a given path starting from node S1?

CTL* (Computation Tree Logic)

- CTL* covers both branching time and linear time interpretations

- Syntax:
  - FOL

  +

  - Temporal Operators
    - X: neXt
    - F: Future (denoted as ⟨⟩ in Uppaal)
    - G: Global (denoted as [] in Uppaal)
    - U: Until
    - R: Release

CTL* state formulas and path formulas

- State formulas (are interpreted in states)
  - express properties of states
  - use path quantifiers:
    - **A** – for all paths (starting from a state),
    - **E** – for some paths (starting from a state)
- Path formulas (are interpreted on paths)
  - expess properties of paths
  - use state quantifiers:
    - **G** – for all states (of the path)
    - **F** – for some state (of the path)

State Formulas (1)

- Atomic propositions are state formulas:
  - If $p \in AP$, then p is a state formula
  - Examples: $x > 0$, $odd(y)$, …

- Propositional combinations of state formulas:
  - $\neg\, \varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$ …
  - Examples:
    - $x > 0 \vee odd(y)$,
    - $req \Rightarrow (\mathrm{AF}\ ack)$ where
      - "A" is a path quantifier
      - "F ack" is a path formula
      - "AF ack" is a state formula (interpreted in a state)

State Formulas (2)

- Quantifiers A and E make from a path formula a state formula that is interpreted in the scope of A and E.

- Eφ, where φ is a formula, which expresses property of a path
  - E means “there exists a path”
  - E φ - φ is *true* on some paths starting from this state on.

- A φ
  - A means “for all paths“
  - A φ - φ is *true* on all paths starting from this state.

Forms of Path Formulas

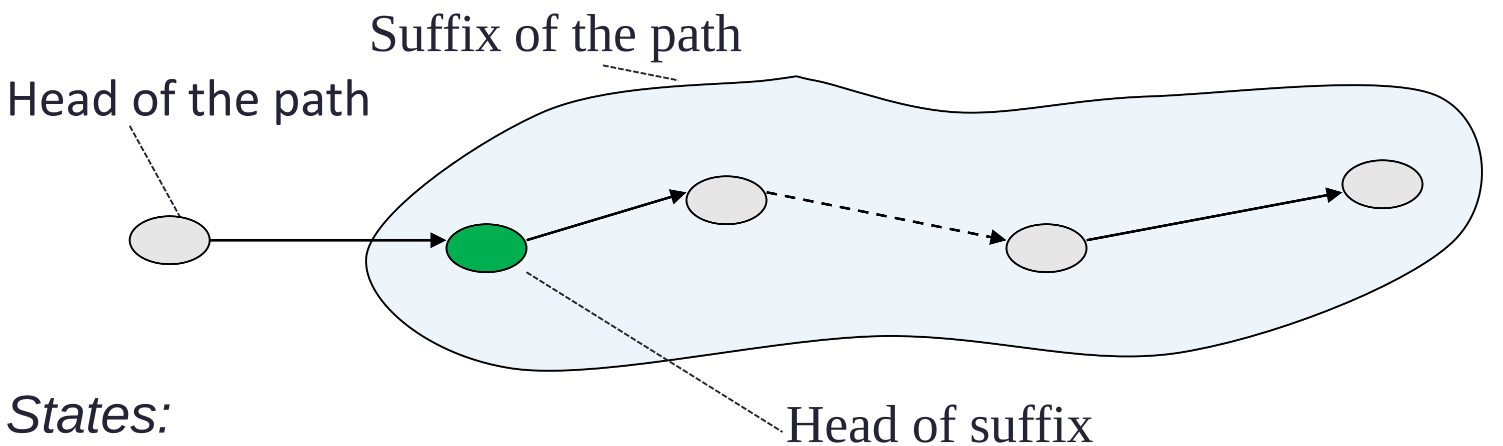
- A state formula φ
  - φ is true in the first state of the path that satisfies path formula prefixed by φ
- For path formulas φ and ψ, the path formulas are also:
  - $\neg\varphi, \quad \varphi \vee \psi, \quad \varphi \wedge \psi$
  - $X\varphi, \quad F\varphi, \quad G\varphi, \quad \varphi U \psi, \quad \varphi R \psi$

    - X – *in the next state*
    - F – *eventually*
    - G – *globally*
    - U – *until*
    - R – *releases*

Path Formulas (I): *Next*-operaator X

$X\ \varphi$, where φ is a path formula, meaning

- φ is valid in the suffix of this path (path minus the first state)



States:

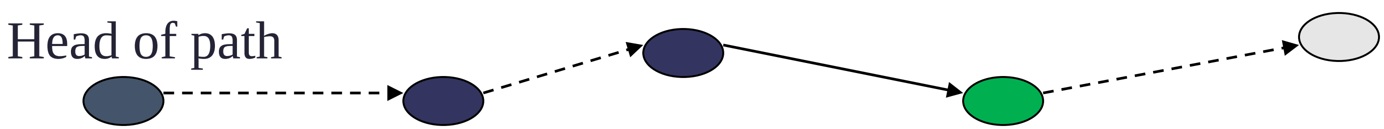
- φ is true

- φ can be either true or false in other states

Path Formulas II: *Eventually*-operator

$F\ \varphi$:

φ is valid in some state of this path



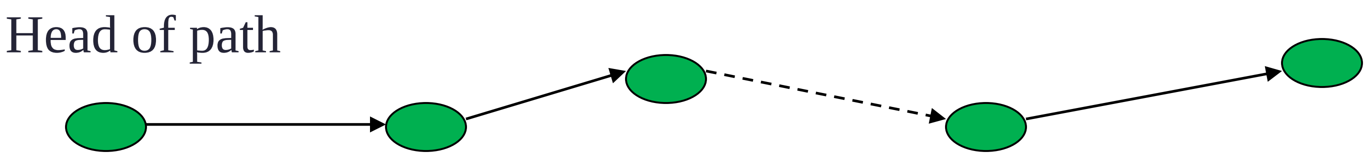
- φ is *false*

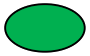
- φ is *true*

- after being true φ can be either *true* or *false*

Path Formulas (III): *Globally*-operator

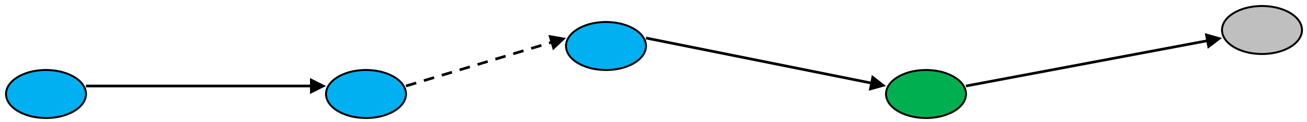
- *G* φ
  - φ is valid for head and every suffix of this path



 - *state where* φ is *true*

Path Formulas IV: *Until*-operaator (weak)

- $\varphi \, \mathrm{U} \, \psi$ is *true* on the path iff
  - *If* ψ is *true* in some state of the path
  - *then* in all states before this state φ must be *true*
- *Weak until* is *true* also on paths without states where ψ is true
- For *strong until* the occurence of state where ψ is true is required



- φ is *true*

-ψ is *true*

-φ and ψ are either *true* or *false*

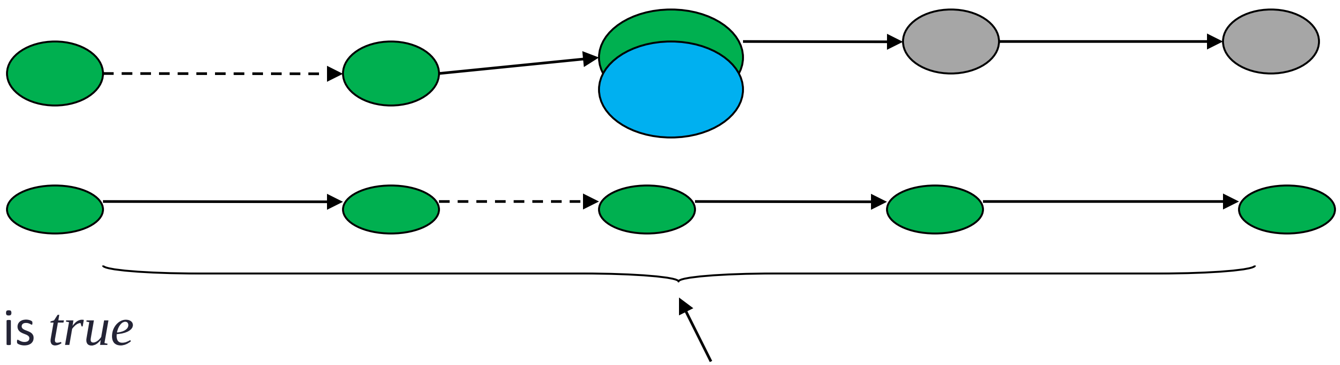
Path Formulas (V): *Release*-operator

φ R ψ

- ψ has to be *true* until and including the point where φ becomes *true*; if φ never becomes *true* then ψ must remain *true* forever

1)

2)



- φ is *true*
- ψ is *true*
- ψ can be either *true* or *false*

Formal semantics of CTL* (1)

- Formal semantics defines the validity of formulas in mathematically rigorous way.
- Notations

  $\vDash$ - satisfiability relation between formula and model:

  - $M, s \vDash \varphi$ iff φ holds in the state s of model M
  - $M, \pi \vDash \varphi$ iff φ holds along the path π in M
  - π^i : i-th suffix of π,
    - e.g. for path $\pi = s_0, s_1, s_2, \ldots,$ $\pi^1 = s_1, s_2, \ldots$

Semantics of CTL* (2)

- *Path formulas are interpreted on paths*:
  - $M, \pi \vDash \varphi$
  - $M, \pi \vDash X\ \varphi$
  - $M, \pi \vDash F\ \varphi$
  - $M, \pi \vDash \varphi\, U \psi$

Semantics of CTL* (3)

- State formulas are interpreted over a set of states (of a path)
  - $M, s \models p$
  - $M, s \models \neg \varphi$
  - $M, s \models \mathrm{E}\ \varphi$
  - $M, s \models \mathrm{A}\ \varphi$

CTL is special case of CTL*

- Quantifiers over paths
  - $\mathbf{A}\ \varphi$ – **A**ll: φ is true for all paths starting from the current state.
  - $\mathbf{E}\ \varphi$ – **E**xists: there exists at least one path starting from the current state where φ is true.

- In CTL, path formulas can occur only when paired with $\mathbf{A}$ or $\mathbf{E}$, *i.e.* one state operator followed by a path operator.

  if φ and ψ are state formulas, then
    - $X\ \varphi$, (next)
    - $F\ \varphi$, (eventually)
    - $G\ \varphi$, (globally)
    - $\varphi\, U \psi$, (until)
    - $\varphi\, R \psi$ (release)

  are path formulas

LTL is special case of CTL

- LTL contains only path formulas

Path formulas:

- If $p \in AP$, then p is a path formula
- If φ and ψ are path formulas, then
  - $\neg\varphi$
  - $\varphi \vee \psi$
  - $\varphi \wedge \psi$
  - $X\ \varphi$
  - $F\ \varphi$
  - $G\ \varphi$
  - $\varphi\, U\, \psi$
  - $\varphi\, R\, \psi$

  are also path formulas.

CTL vs. CTL*

- CTL*, CTL and LTL have *different expressive powers*:
- Example:
  - In CTL there is no formula equivalent to LTL formula $A(FG\ p)$.
  - In LTL there is no formula equivalent to CTL formula $AG(EF\ p)$.
  - $A(FG\ p) \vee AG(EF\ p)$ is a CTL* formula that cannot be expressed neither in CTL nor in LTL.

- We use in our course CTL!

Minimal set of CTL temporal operators

- CTL has some redundancy to make expressions more compact and better readable
- All CTL operators can be expressed using a minimal set of temporal operators {*EU, EF, EG*} and propositional connectives ¬, ∨
- Following equivalences are used for mapping temporal operators to minimal set of temporal operators {*EU, EF, EG*}:

strong until

  - $EF\ \varphi \equiv E\ [true\ U\ \ \varphi\ ]$ (because $F\ \varphi \equiv [true\ U\ \varphi\ ]$)
  - $AX\ \varphi \equiv \neg\ EX(\neg\ \varphi\ )$
  - $AG\ \varphi \equiv \neg\ EF(\neg\ \varphi\ ) \equiv \neg\ E\ [true\ U\ \neg\varphi\ ]$
  - $AF\ \varphi \equiv A\ [true\ U\ \varphi\ ] \equiv \neg\ EG\ \neg\ \varphi$
  - $A[\varphi\ U\ \psi] \equiv \neg(\ E[(\neg\ \psi)\ U\ \neg(\varphi \vee\ \psi)] \vee EG\ (\neg\psi)\ )$

Recap

- CTL* is general temporal logic that offers strong expressive power, more than CTL and LTL separately.

- CTL and LTL are practically useful, they are easier to interpret than CTL*
- CTL* helps to understand the relations between LTL and CTL.

- In the next lecture we will show how to check satisfiability of CTL formulas on Kripke structure.