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

J.Vain

22.02.2018

Model Checking

$$M \vDash P \ ?$$

Given

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

Check if *M satisfies P*

If $M \vDash P$ we say in logic that M is a model of formula P

Model: Kripke Structure (revisited I)

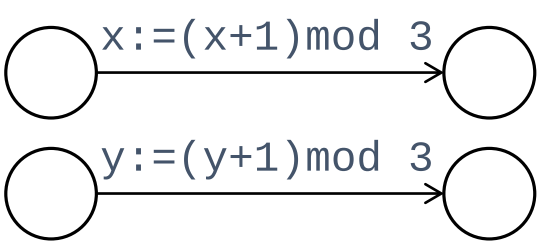
- KS is a state-transition system that captures
  - what is true in a state (denoted as labeling of the state)
  - what can be viewed as an atomic move (denoted as transition)
  - the succession of states   (paths on the model graph)

- KS is a static representation that can be unfolded to a *tree of execution traces* on which temporal properties are verified.

Representing transition as formuli

- 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:

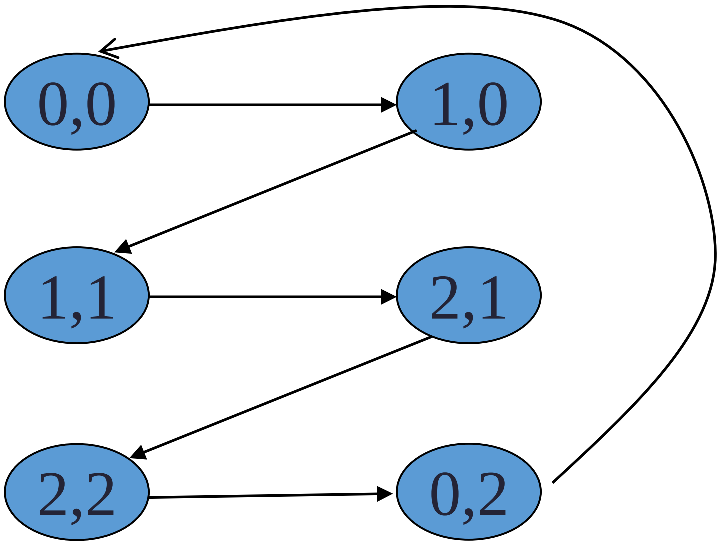
we can restrict our attention to pairs of consecutive states $s = (x, y)$ and $s' = (x', y')$ in the state space $\{0, 1, 2\} \times \{0, 1, 2\}$, i.e.

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

- 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 $R = \{R_1, R_2\}$ and in logic notation $R \Leftrightarrow (R_1 \text{ or } R_2)$

Consecutive states represented by $R_1 \vee R_2$



Representing transitions (revisited II)

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

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

- For the whole program we have

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

- (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*

- 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</u>!

Introducing program counter

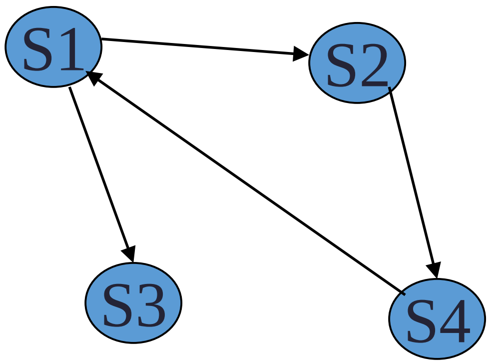
- In the computer, the order of execution is controlled by *program counters.*
- We introduce an auxilliary variable pc, and assume the program commands are labeled with `l`$_0$`,… ,l`$_n$.
- For instance
  - In the program:
    - `l`$_0$`: x := x+1;`
    - `l`$_1$`: y := x+1;`
    - `l`$_2$`: …`
  - In the 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 logic representation of program execution in our computation model *M*!

Temporal logic CTL*

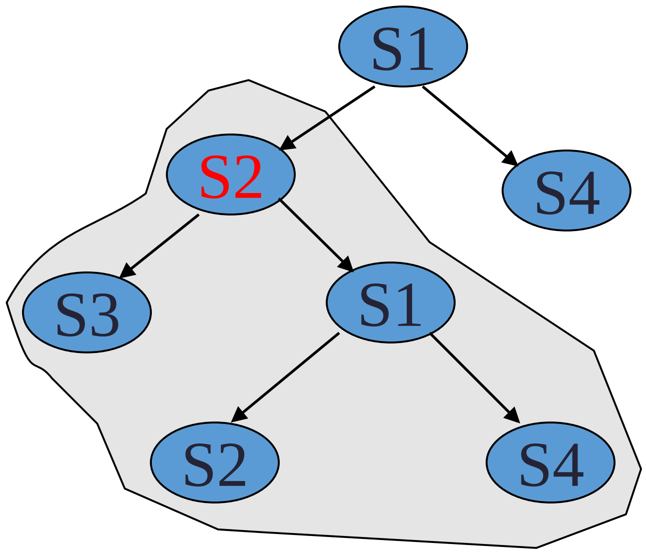
- Semantics

  KS and its logic representation are static models of program execution



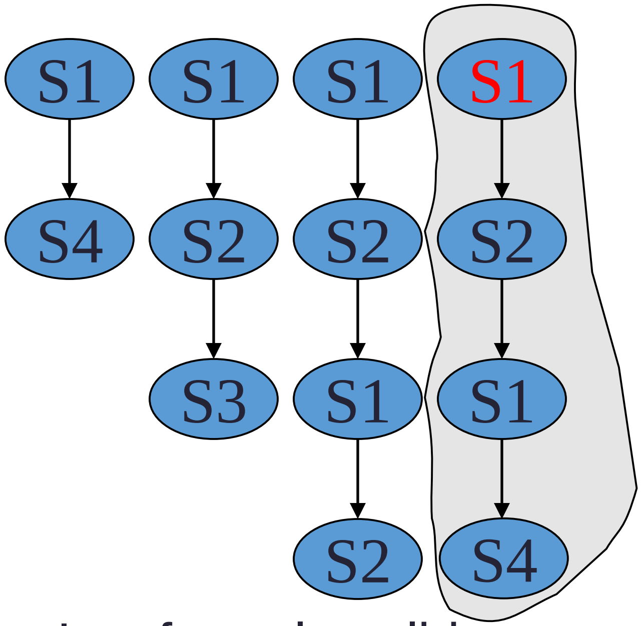
Dynamic model of program execution = unfolding of the static model

Branching time: tree structure



Is a formula valid at a given node, which represents a subtree?

Linear time: traces



Is a formula valid along a given path?

CTL* (Computation Tree Logic)

- Covers both branching time and linear time logics
- Basic Operators
  - X: neXt
  - F: Future (〈〉)
  - G: Global ([])
  - U: Until
  - R: Release

CTL*

- State formulas (are interpreted in states)
  - Express properties of states
  - Use path quantifiers:
    - **A** – for all paths,
    - **E** – for some paths
- 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:
  - If $p \in AP$, then p is a state formula
  - Examples: $x > 0$, $odd(y)$
- Propositional combinations of state formulas:
  - $\neg \varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi \ldots$
  - Examples:
    - $x > 0 \vee odd(y)$,
    - $req \Rightarrow (\mathrm{AF}\ ack)$
      - "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 a state formula from a path formula interpreted in the scope of A or E.

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

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

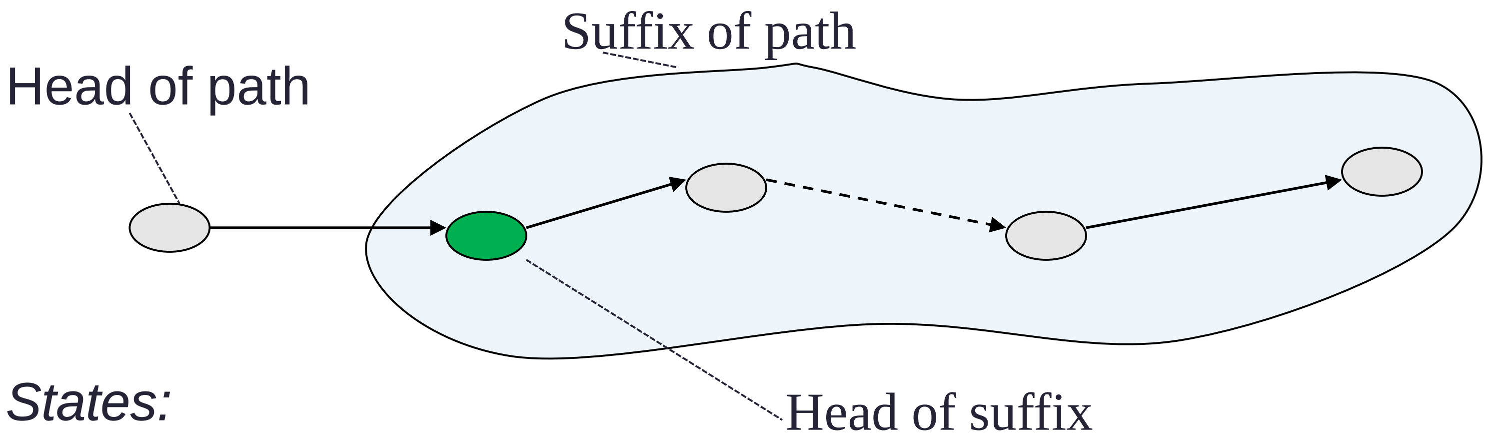
Forms of Path Formulas

- A state formula φ
  - φ is true in the first state of this path
- For path formulas φ and ψ, the path formulas are:
  - $\neg\varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$
  - $X\varphi$, $F\varphi$, $G\varphi$, $\varphi U \psi$, $\varphi R \psi$
    - $X - next$
    - $F - eventually$
    - $G - globally$
    - $U - until$
    - $R - releases$

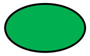
Path Formulas (I): *Next*-operator

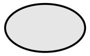
$X\ \varphi$, where φ is a path formula

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



States:

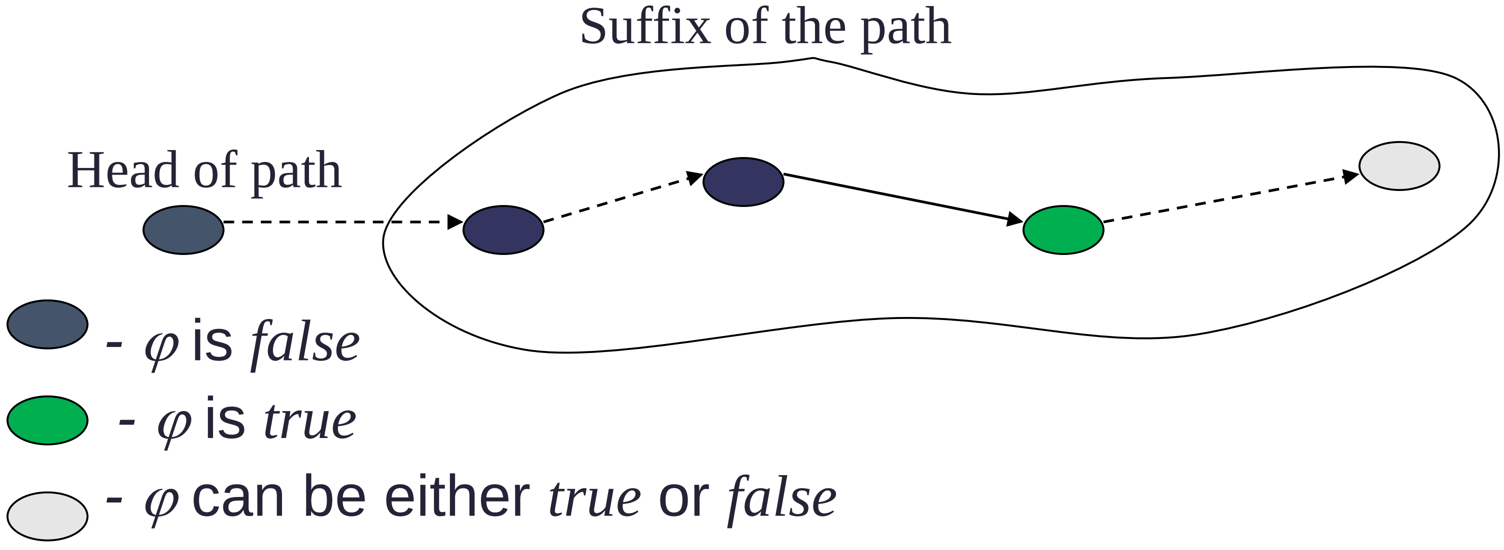
 - φ is true

 - φ can be either true or false in other states

Path Formulas II: *Eventually*-operator

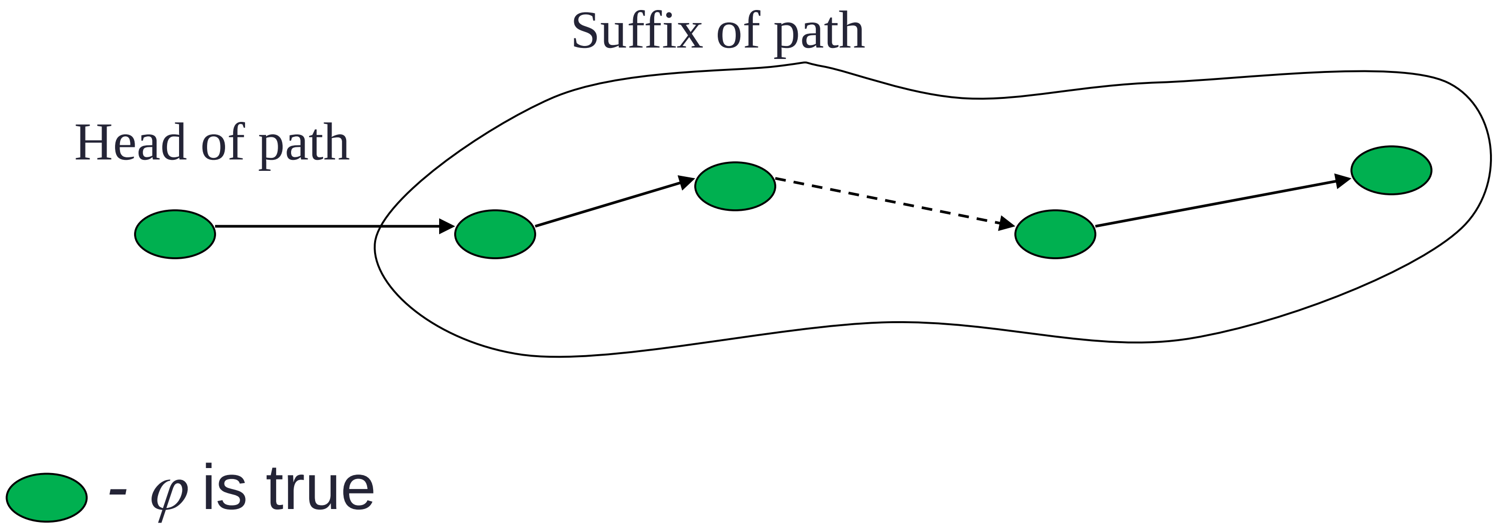
F φ:

φ is valid for this path



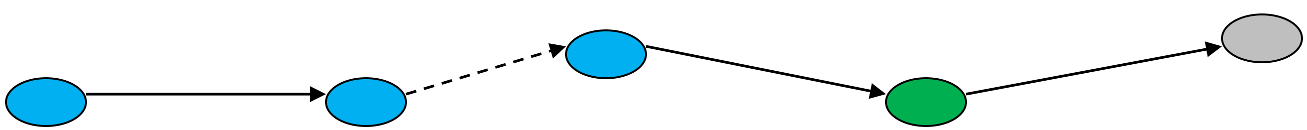
Path Formulas (III): *Globally*-operator

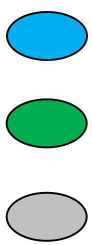
- G φ
  - φ is valid for head and every suffix of this path



Path Formulas IV: *Until*-operator

- $\varphi U \psi$
  - ψ is valid on a suffix of the path, before the first node of which φ is valid on every suffix thereon





- φ is true

-ψ is true

-φ and ψ are either true or false

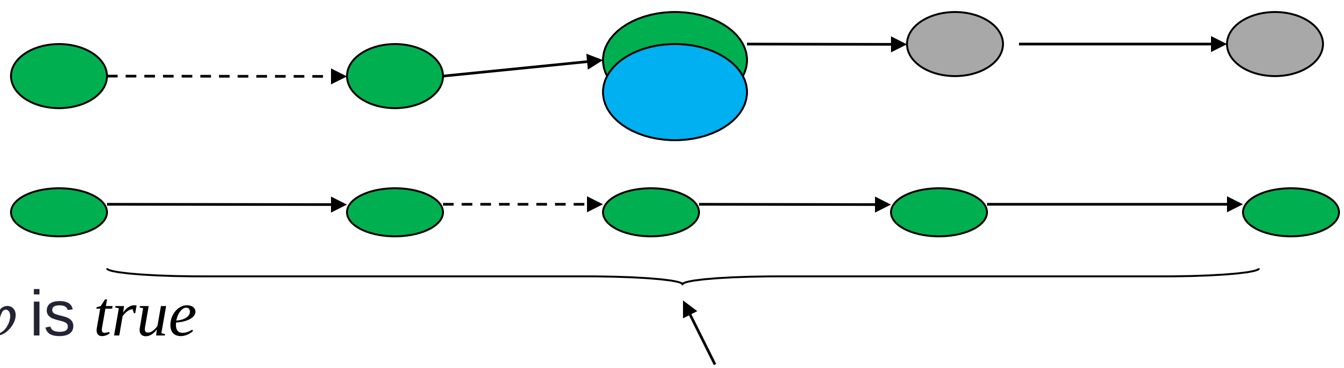
Path Formulas (V): *Release*-operator

φ R ψ

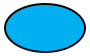
- ψ has to be *true* until and including the point where φ becomes *true*; if φ never becomes *true* then ψ must remain *true* forever

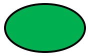
1)

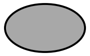
2)



φ never gets *true*

 - φ is *true*

 - ψ is *true*

 - ψ can be either *true* or *false*

Formal semantics of CTL* (1)

- Notations
  - $M, s \vDash \varphi$ iff φ holds in 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
    - $\pi = s_0, s_1, \ldots$, then $\pi^1 = s_1, \ldots$

Semantics of CTL* (2)

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

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

- Quantifiers over paths
  - $A\ \varphi$ – **A**ll: φ has to hold on all paths starting from the current state.
  - $E\ \varphi$ – **E**xists: there exists at least one path starting from the current state where φ holds.

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

  if φ and ψ are state formulas, then
    - $X\ \varphi$,
    - $F\ \varphi$,
    - $G\ \varphi$,
    - $\varphi\, U \psi$,
    - $\varphi\, R \psi$

  are path formulas

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 path formulas.

CTL vs. CTL*

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

Minimal set of CTL temporal operators

- Transformations used for mapping temporal operators to minimal set of temporal operators $\{EU, EF, EG\}$:
  - $EF\ \varphi == E\ [true\ U\ \ \varphi\ ]$ (because $F\ \varphi == [true\ U\ \varphi\ ]$)
  - $AX\ \varphi == \neg\ EX(\neg\ \varphi\ )$
  - $AG\ \varphi == \neg\ EF(\neg\ \varphi\ ) == \neg\ E\ [true\ U\ \neg\varphi\ ]$
  - $AF\ \varphi == A\ [true\ U\ \varphi\ ] == \neg\ EG\ \neg\ \varphi$
  - $A[\varphi\ U\psi] == \neg(\ E[(\neg\ \psi)\ U\ \neg(\varphi \vee\ \psi)] \vee EG\ (\neg\psi)\ )$

Summary

- CTL* is general temporal logic that offers strong expressive power, more than CTL and LTL separately.
- CTL and LTL are practically useful enough; CTL* helps to understand the relations between LTL and CTL.
- In the next lecture we will show how to check satisfiability of CTL formuli on Kripke structures.