Module Checking

An overview by
Maël Hörz

Paper by

Orna Kupfermann, Moshe Y. Vardi, Pierre Wolper

Talk-Outline

- Closed Systems vs. Open Systems
- Transition Systems (Programs/Reactive Programs)
- Temporal Logic
- Module Checking
- Complexity of Module Checking

Introduction

- System Design
  - Closed systems vs. Open systems

Closed system

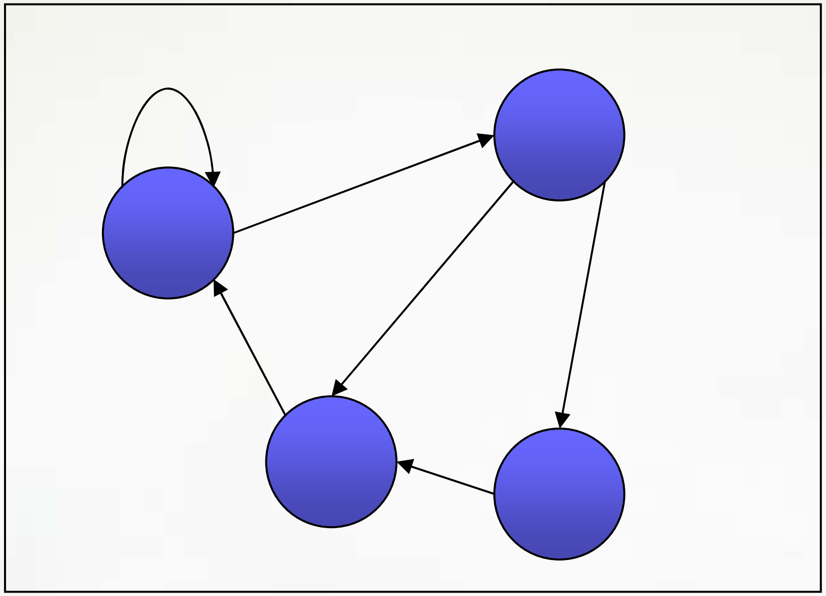
- Behavior
  - Completely determined by the state of the system
- One kind of non-determinism
  - Only internal

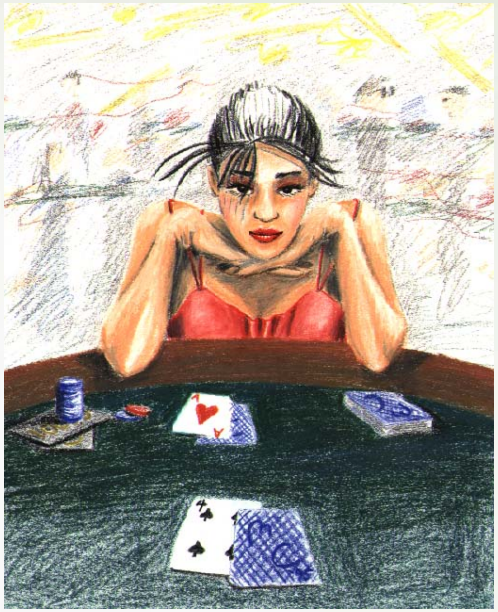
Open system

- Behavior
  - Depends on the interaction with its environment
- Two kinds of non-determinism
  - External (uncontrollable)
  - Internal (controllable)

Closed system – One player

- System / Internal player (= only player)

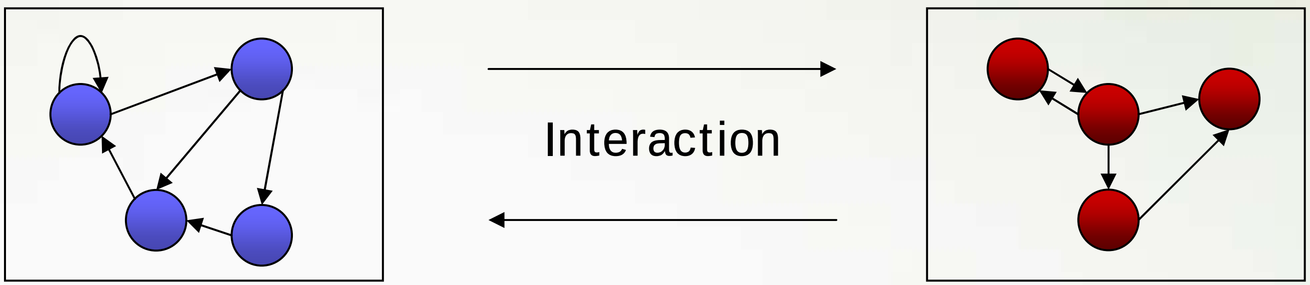




Open system - Two players

- System/ internal player
- controllable

- Environment / external player
- uncontrollable



versus

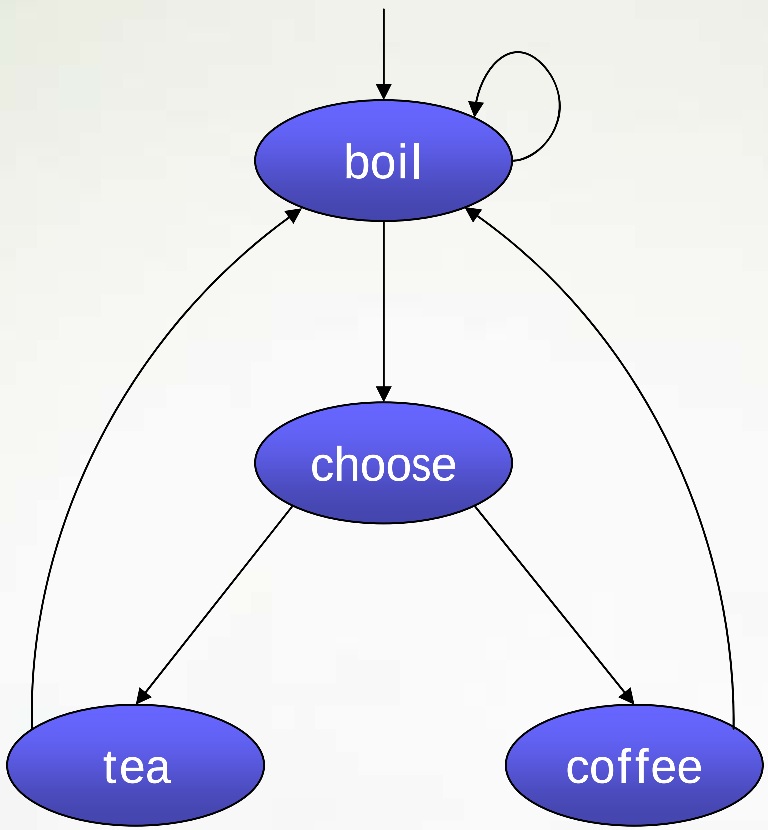
No gambling

- We are only interested in safe systems

Introduction

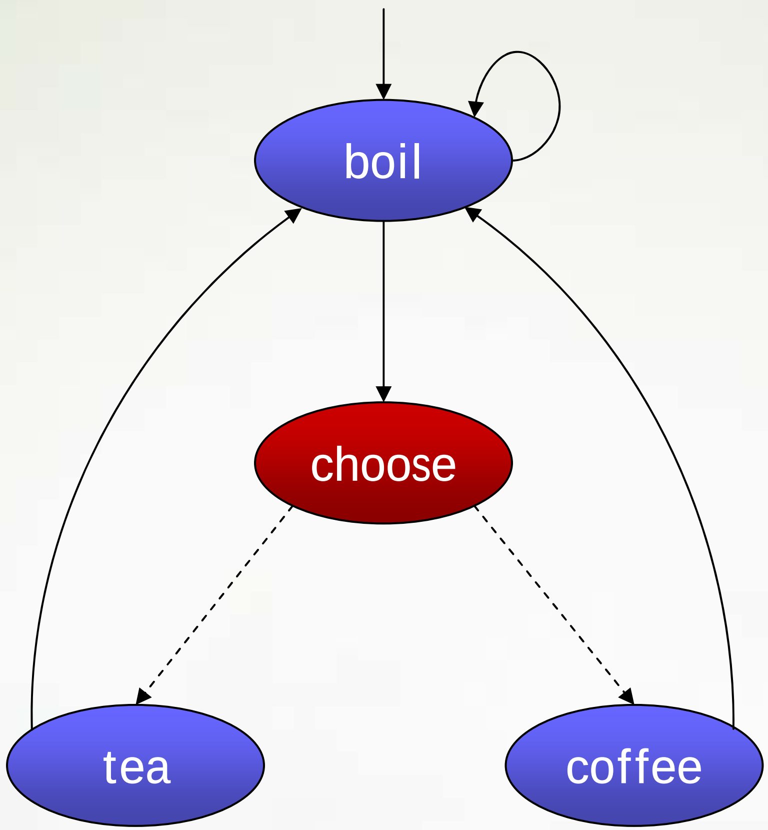
- Environment
  - Everything not under control of system itself
- Reactive system
  - Open system
  - Does not terminate
  - Interacts with an environment
- External vs. internal
  - Environment makes external choices
  - System makes internal choices

Example



- Closed system
  - Repeatedly boils water
  - Makes internal non-det. choice
  - Serves either tea or coffee
  - One player:
    machine

Example



- Open system
  - Repeatedly boils water
  - Asks environment to choose between coffee and tea
  - Serves a drink according to the external choice
  - Two players:
    machine vs. user
  - Controllability of non-det. depends on player

Model vs. Module Checking

- Verification of closed systems
  - Model checking:
    - machine has to satisfy given requirements
- Verification of **open** systems:
  - **Module checking**
    - **For all env.**: machine has to satisfy given requirements
    - “Model checking of open systems”

Model/Module Checking

- General idea of model/module checking
  - Express design as a formal model M
    - Finite state transition system
  - Specify required behavior with a logic formula ψ
  - Check that M satisfies ψ

Transition Systems

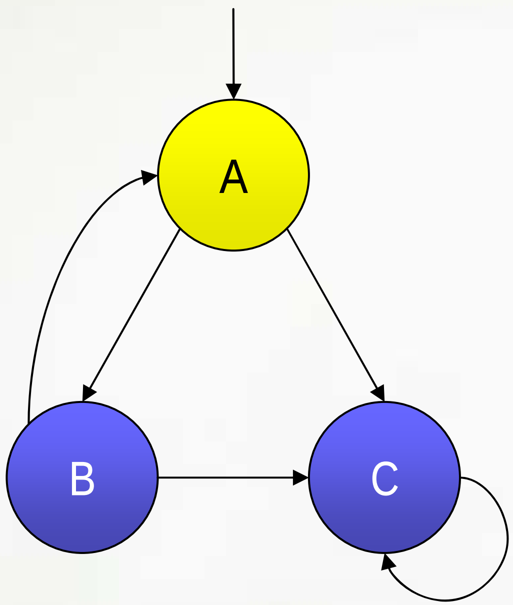
- Program $P = \left(AP, W, R, w_0, L \right)$
  - *AP* : set of atomic propositions
  - *W* : set of states
  - $R \subseteq W \times W$: transition relation (must be total)
  - w_o : an initial state
  - $L : W \to 2^{AP}$: maps each state to a set of atomic propositions true in this state
- Atomic propositions
  - The state of a variable, e.g. x = 5 and y = 7 always holds in state s, then L(s) = {x = 5, y = 7}
  - In state “tea” it holds that tea is served

Transition Systems

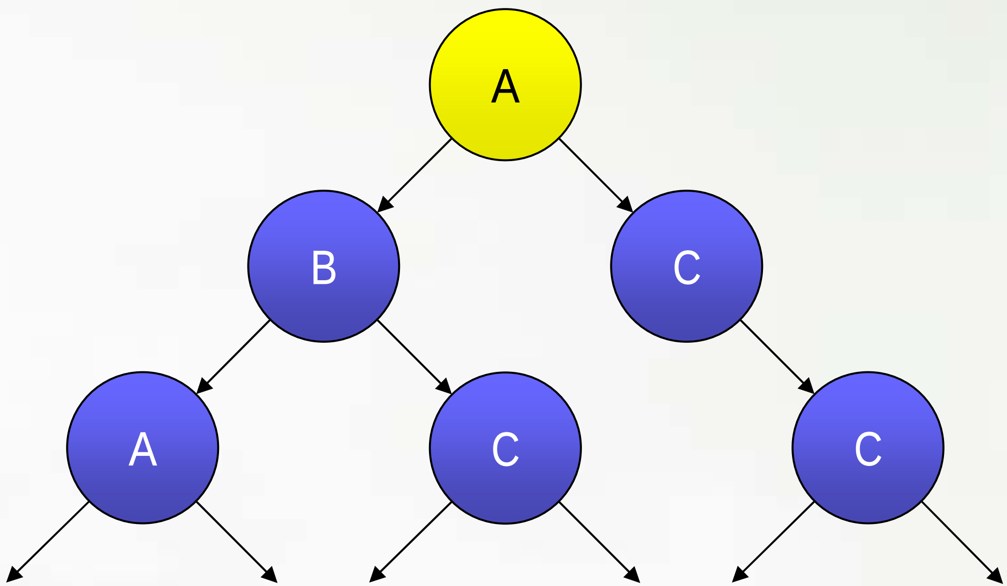
- Module $M = \left(AP, W_s, W_e, R, w_0, L\right)$
  - *AP* : set of atomic propositions
  - W_s, W_e : set of system/environment states
  - $W = W_s \cup W_e$
  - $R \subseteq W \times W$: transition relation (must be total)
  - w_o : an initial state
  - $L : W \rightarrow 2^{AP}$: maps each state to a set of atomic propositions true in this state

TS and its unwinding

- Transition System



- Unwinding of TS (Infinite execution tree)
  - Every path represents a possible execution



Temporal Logics

- Reactive systems do not terminate
  - ⇒ We get infinite execution trees
  - ⇒ No end state where we can check requirements
  - ⇒ Want to check properties **while** execution
  - ⇒ Need to express temporal aspects in requirements

LTL – Linear Temporal Logic

- Syntax

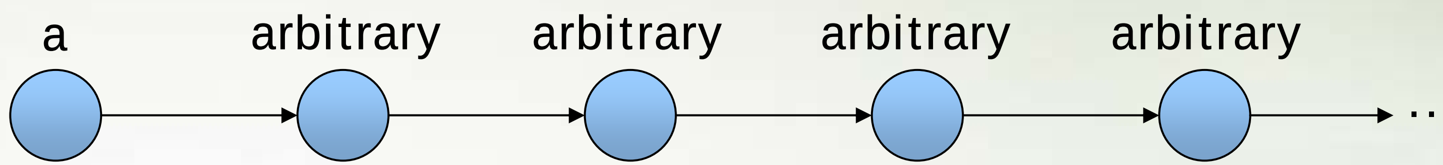
  $$\varphi ::= true \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid X\varphi \mid F\varphi \mid G\varphi \mid \varphi_1 \cup \varphi_2$$

  where $a \in AP$

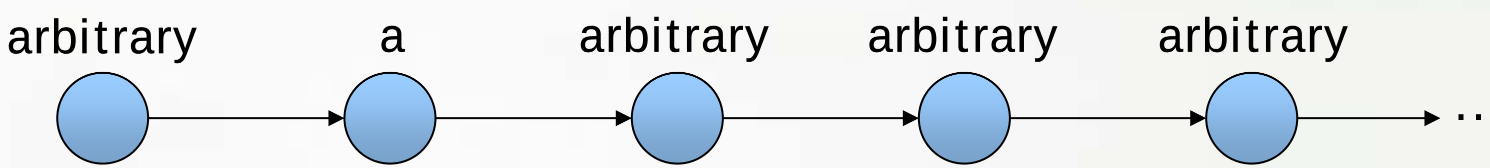
- Formula composed of
  - Atomic propositions
  - Boolean connectors
  - Temporal operators
- Using $\wedge$ and $\neg$ the remaining connectives $\vee$, $\oplus$, $\rightarrow$, $\leftrightarrow$, *nor*, *nand* can be derived
- Similarily, *F* and *G* could be derived from the *U* operator

LTL- Intuitive Semantics (1/2)

atomic prop. a

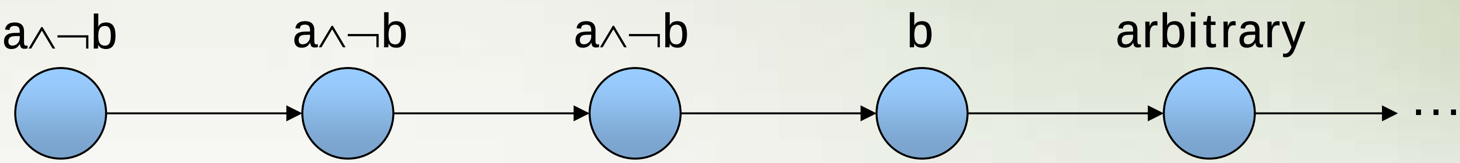


ne**x**t step Xa

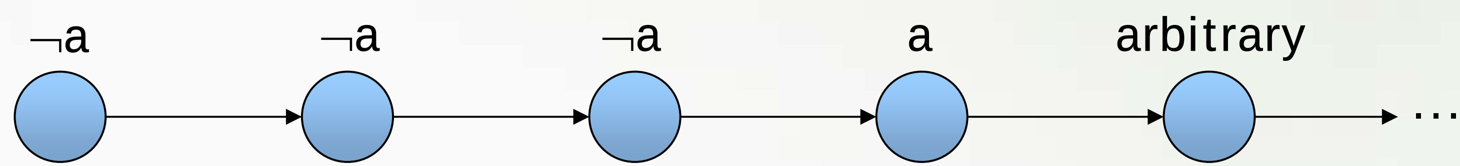


LTL- Intuitive Semantics (2/2)

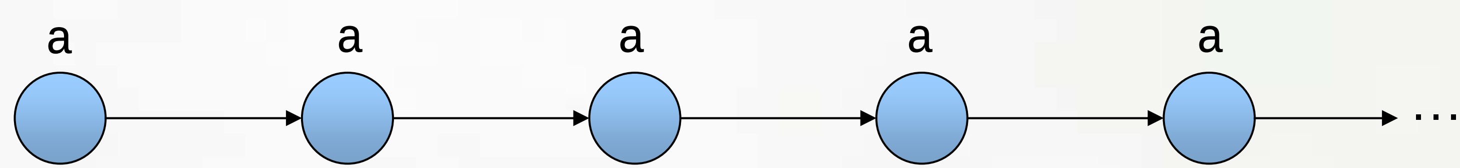
until *aUb*



eventually/ **f**inally *Fa*



always/ **g**lobally *Ga*



CTL – Computational Tree Logic

- State-formulas

  $$\Phi ::= true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \exists\varphi \mid \forall\varphi$$

  where $a \in AP$ and φ is a path-formula

  – State formulas express a property of a state

- Path-formulas

  $$\varphi ::= X\Phi \mid F\Phi \mid G\Phi \mid \Phi_1 \bigcup \Phi_2$$

  where Φ, Φ_1, Φ_2 are state-formulas

  – Path formulas express a property of a path

    - Path = infinite sequence of states

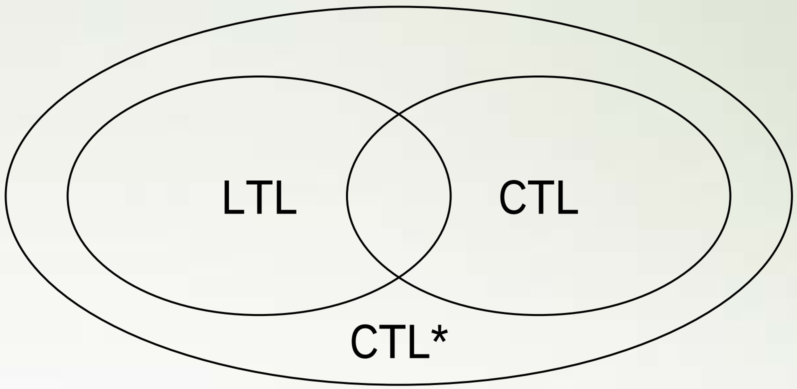
CTL – Semantics

- Temporal operators *X*,*F*,*G*,*U* have analog semantics to their LTL-semantics
- **New:** Path-quantifiers
  - Let φ be a path-formula
  - Universal path quantifier
    - ∀φ : φ has to hold on all paths
  - Existential path quantifier ∃φ
    - ∃φ : φ has to hold on at least one path
- CTL*
  - Like CTL, but temporal operators can be freely mixed

Comparison of LTL, CTL, CTL*

- CTL vs. CTL*
  - CTL: one path operator followed by a state operator
  - CTL*: temporal operators can be freely mixed
- LTL expressed in CTL*
  - LTL-formula checked on all paths of TS
    $\Rightarrow$ implicit $\forall$-quantification
  - LTL formula φ expressed as CTL* formula $\forall\varphi$

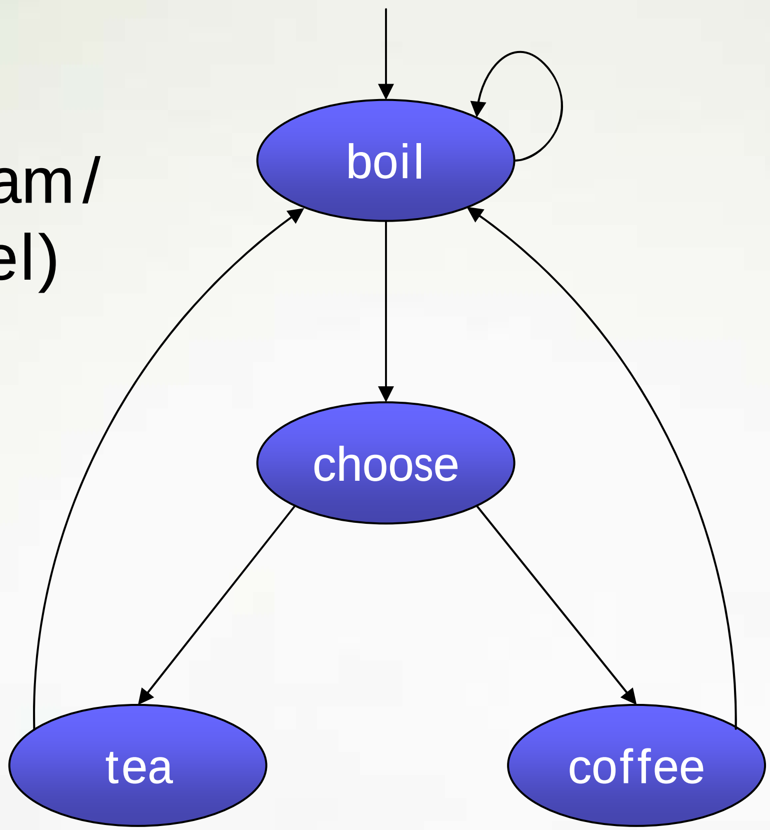
Expressiveness of LTL, CTL, CTL*



- LTL $\not\subset$ CTL
  - FGa $\in$ LTL but FGa $\notin$ CTL (idea: ∀F∀Ga $\neq$ ∀FGa)
- CTL $\not\subset$ LTL
  - ∀G∃F $\in$ CTL but ∀G∃F $\notin$ LTL (idea: no ∃ in LTL)
- LTL $\subset$ CTL*
  - Add ∀ in front of LTL formula
- CTL $\subset$ CTL*
  - CTL* is an extension of CTL

Model Checking vs. Module Checking

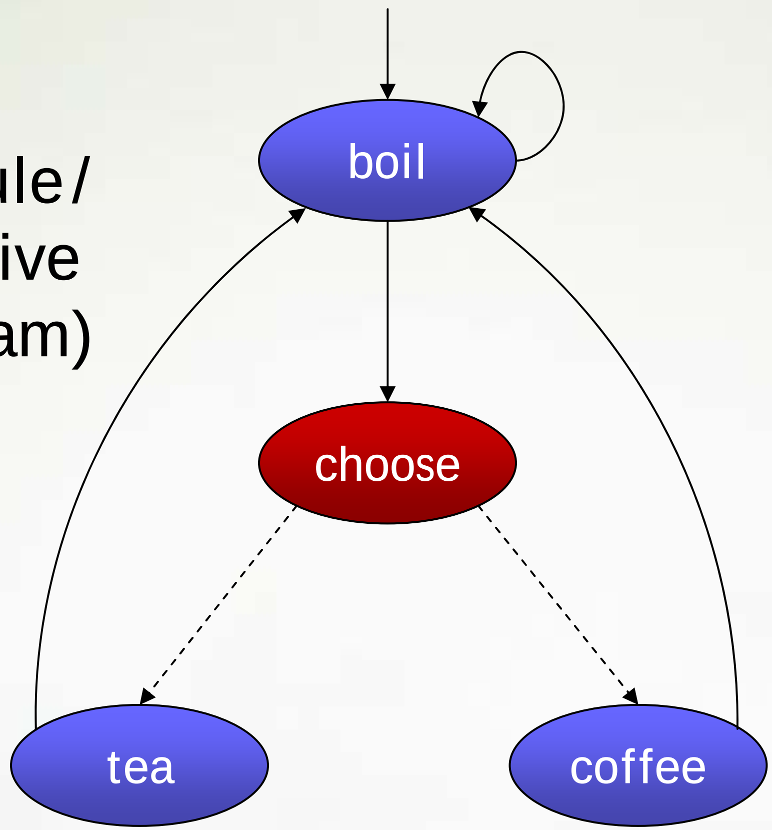
P: (program/ model)



- **Model checking**
- Is it always possible to eventually serve tea?
- Does P ⊨ $\forall G \exists F$ *tea* hold?
- Yes
  - System controls non-determinism of "choose"

Model Checking vs. Module Checking

M:
(module/
reactive
program)



- **Module checking**
- Is it always possible to eventually serve tea?
- Does M ⊨ $\forall G \exists F$ *tea* hold?
- No
  - Environment controls non-determinism of choose
  - If environment chooses always coffee ⇒ M can never serve tea

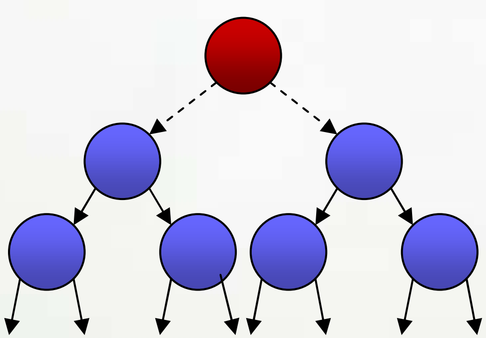
Model Checking vs. Module Checking

- Model checking tools will always answer yes
  - M is regarded as program (no environment)
  - ⇒ wrong answer for modules
  - Adapt model checking tools such that module checking works

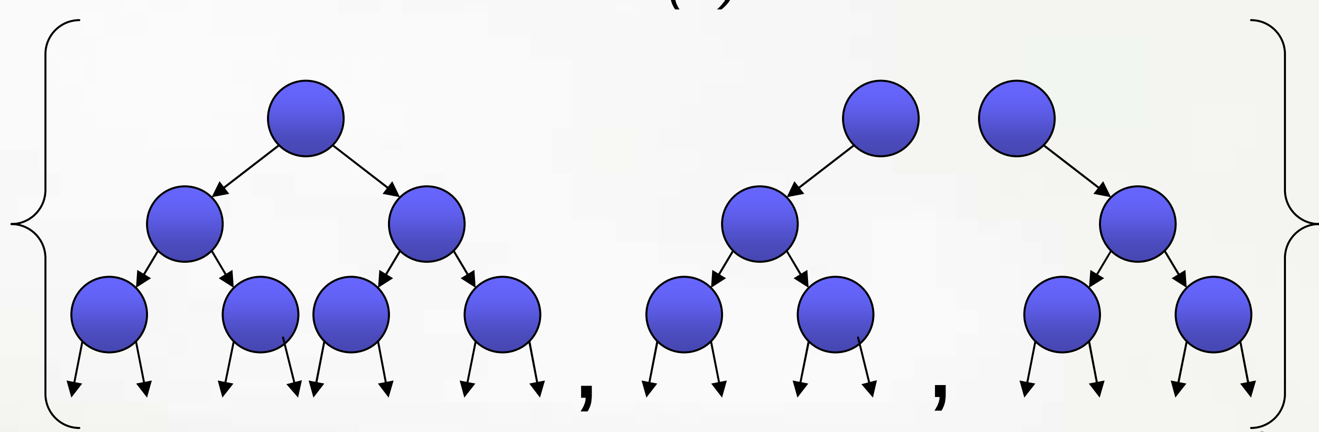
Module Checking

- Execution tree
  - Tree obtained from unrolling program/module
- In module checking **set of execution trees**, *exec(M)*
- *exec(M)*
  - Let *ET* be the execution tree
  - By pruning from *ET* sub-trees, which have as root node a successor of an environment node, we obtain *exec(M)*.

Execution tree of *M*:



exec(M):



Module Checking

- Intuitively
  - Each tree in *exec(M)* corresponds to a different behavior of the environment
- Module checking
  - Given module M, CTL* formula ψ
  - $M \models_r \psi$, if all trees in *exec(M)* satisfy ψ
  - I.e. apply model checking to all trees in *exec(M)*
- Module checking can be solved using non-det. tree automata

Complexity

- $\forall$CTL*, $\forall$CTL
  - CTL*/CTL restricted to $\forall$-quantification
- Module checking problem for $\forall$CTL*
  - Theorem
    - Coincides with model checking problem for $\forall$CTL*
  - Proof idea
    - All choices have to be considered since no $\exists$-quantification
    - No difference between system and environment non-determinism
  - Module/model checking of LTL, $\forall$CTL, $\forall$CTL* of same complexity
    - Since LTL, $\forall$CTL are subsets of $\forall$CTL*

Complexity

- Results
  - Module checking for ∀CTL is in linear time
  - Module checking problem for LTL, ∀CTL* is PSPACE-complete
  - Program complexity of module checking for LTL, ∀CTL, ∀CTL* is NLOGSPACE-complete

Complexity

- Module checking problem for CTL
  - EXPTIME-complete
- Module checking problem for CTL*
  - 2EXPTIME-complete
- Proof idea
  - Model checking problem for CTL is in linear-time
  - Module checking of model M runs model-checking on all trees in exec(M)
  - exec(M) is a subset of the power set of M, i.e. we get an exponential blow-up

Complexity

- Is it really that bad?
- Good news
  - Model checking tools can be easily adjusted for commonly-used fragment of CTL
  - Module checking problem for $\exists F\xi$ and $\forall G\exists F\xi$ in linear time
  - Program-complexity of module checking for $\exists F\xi$ and for $\forall G\exists F\xi$ is PTIME-complete

Complexity-Overview

	Model checking	Module checking	Program complexity of model checking	Program complexity of module checking
LTL	PSPACE	PSPACE	NLOGSPACE	NLOGSPACE
CTL	linear-time	EXPTIME	NLOGSPACE	PTIME
CTL*	PSPACE	2EXPTIME	NLOGSPACE	PTIME
∀CTL	linear-time	linear-time	NLOGSPACE	NLOGSPACE
∃CTL	linear-time	EXPTIME	NLOGSPACE	PTIME
∃Fξ ∀G∃Fξ	linear-time	linear-time	NLOGSPACE	PTIME

Summary

- Closed systems vs. Open system
  - Closed system
    - No interaction, total control
  - Open systems interact with an environment
    - “Game between system and environment player”
- Temporal logic formulas to specify requirements
  - Want to check properties **while** execution
- Module checking
  - $M \models_r \psi$
  - Module *M* has to satisfy ψ in all environments
- Module checking is much more complex than model checking
- Commonly used subset of CTL has linear complexity

References

- “Module Checking“
  by Orna Kupferman, Moshe Y. Vardi, Pierre Wolper
- “Principles of Model Checking”
  by Christel Baier and Joost-Pieter Katoen
- Pictures in introduction from various sources