

On Module Checking and Strategies

Aniello Murano

Università di Napoli “Federico II”

*A long-standing ongoing project with Moshe Vardi,
Orna Kupferman, Laura Bozzelli, Benjamin Aminof, and Wojtek Jamroga*

Actions@KR'18, Tempe, 29 October 2018

Model Checking

□ Let S be a finite-state system and P its desired behavior

◆ S	→	labelled state-transition graph M
◆ P	→	a temporal logic formula ψ

□ We check whether S has the required behavior P by checking whether

$$M \models \psi$$

Classes of Models

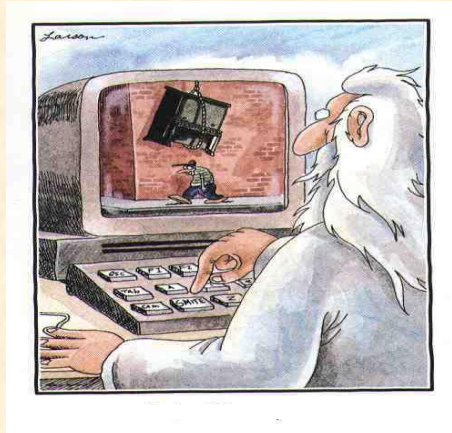
❑ Closed Systems

- Behavior is fully characterized by system state

❑ Open Systems

- Behavior depends on the interaction with the environment

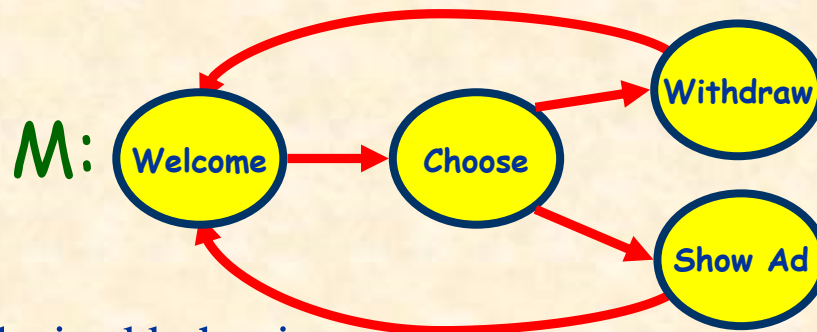
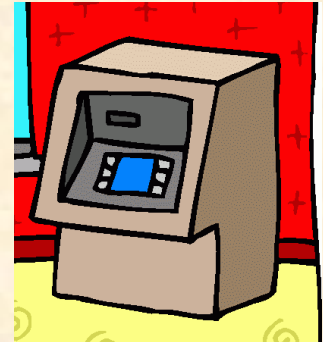
It must be
"reactive"



- Open System Model: ~~Labelled State-Transition Graph~~
- A solution for Open Finite-State Systems: Module Checking [Kupferman, Vardi, Wolper 1996-2001]

Model checking

- ❑ Consider an ATM machine that
 1. Displays a welcome screen
 2. Makes an internal nondeterministic choice
 3. **Withdraws money or shows an advertisement (Ad)**
- ❑ The machine is a closed system !
- ❑ M is a labeled-state transition graph modeling the machine

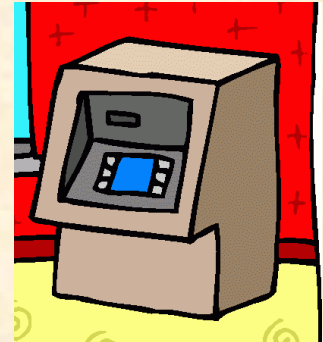


- ❑ A desired behavior:
“It is always possible to show an ad”

$$\varphi = \forall G \exists F \text{ Show Ad}$$

Model checking

- ❑ Consider an ATM machine that
 1. Displays a welcome screen
 2. Makes an internal nondeterministic choice
 3. **Withdraws money or shows an advertisement (Ad)**
- ❑ The machine is a closed system !
- ❑ M is a labeled-state transition graph modeling the machine
- ❑ T is an infinite tree obtained by unwinding M



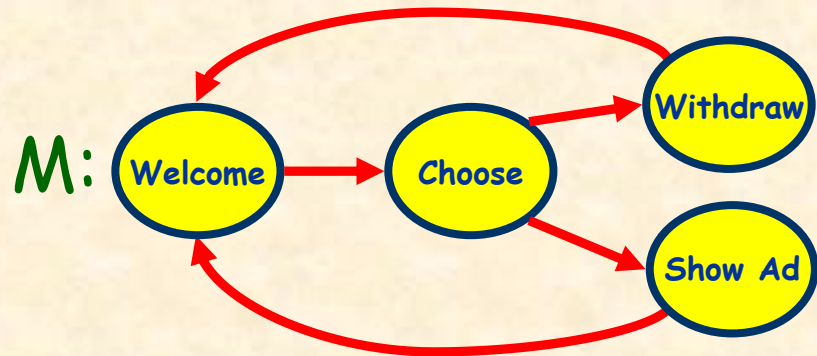
- ❑ A desired behavior:
“It is always possible to show an ad”

$$\boxed{\varphi = \forall G \exists F \text{ Show Ad}} \quad \longrightarrow \quad \boxed{M \models \varphi \text{ iff } T \models \varphi}$$

Model checking an open system

- ❑ Consider the ATM machine as an open system:
 1. Displays a welcome screen
 2. Lets the environment choose to view an Ad or withdraw money
 3. Performs the requested operation and restarts from 1

Open
system



- ❑ The ATM can always eventually show an Ad iff

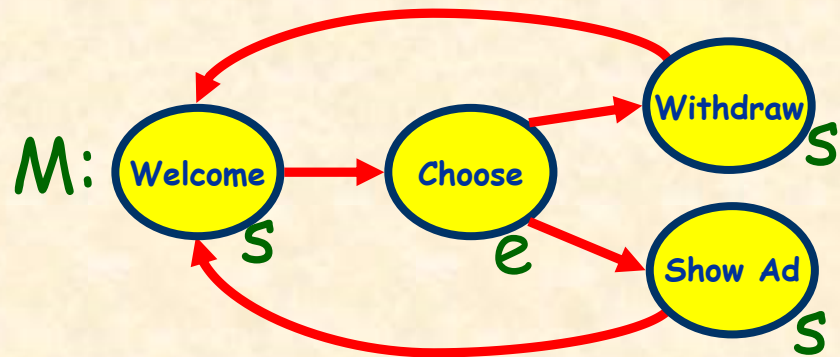
~~$T \models \forall G \exists F \text{ Show Ad}$~~

It may be impossible to show an ad!

Model checking an open system

- ❑ Consider the ATM machine as an open system:
 1. Displays a welcome screen
 2. Lets the environment choose to view an Ad or withdraw money
 3. Performs the requested operation and restarts from 1

Open
system



- ❑ To model the ATM we need a **Module**: a labeled transition graph with a partition into system and environment states
- ❑ Let T be the unwinding of M .
- ❑ Let $\text{Exec}(M)$ be the set of all trees obtained by pruning in T sub-trees rooted in successors of environment nodes (but one).
- ❑ M (reactively) satisfies φ iff φ holds in all trees of $\text{Exec}(M)$.

Module checking

$M \models_r \varphi$

Solving CTL/CTL* Module Checking

□ First, observe that

◆ $M \models_r \phi$ implies $M \models \phi$, while the convers may not be true.

◆ $M \not\models_r \phi$ iff there is a tree T in $\text{Exec}(M)$ such that $T \models \neg \phi$

□ An automata-theoretic solution:

1. Build a tree automaton $A_{\text{Exec}(M)}$ that accepts all trees in $\text{exec}(M)$
2. Build a tree automaton $A_{\neg\phi}$ that accepts all tree models of $\neg\phi$
3. Check whether $M \models_r \phi$ by checking $L(A_{\text{Exec}(M)}) \cap L(A_{\neg\phi}) = \emptyset$

Finite-state complexity results

Class	Model Checking (formula comp.)	Model Checking (system comp.)	Module Checking (formula complexity)	Module Checking (system complexity)
LTL	PSpace-Complete[4]	NLogSpace [4]	PSpace-Complete [5]	NLogSpace [5]
CTL	Linear Time [1]	NLogSpace[3]	ExpTime-Complete [5]	PTime [5]
CTL*	PSpace-Complete [2]	NLogSpace[3]	2ExpTime-Complete [5]	PTime [5]
1. [Clarke, Emerson, Sistla 1986] 2. [Emerson and Lei 1985] 3. [Kupferman, Vardi, Wolper 1994 & 2000]			4. [Sistla and Clarke 1985] 5. [Kupferman, Vardi, Wolper 1996 & 2001]	

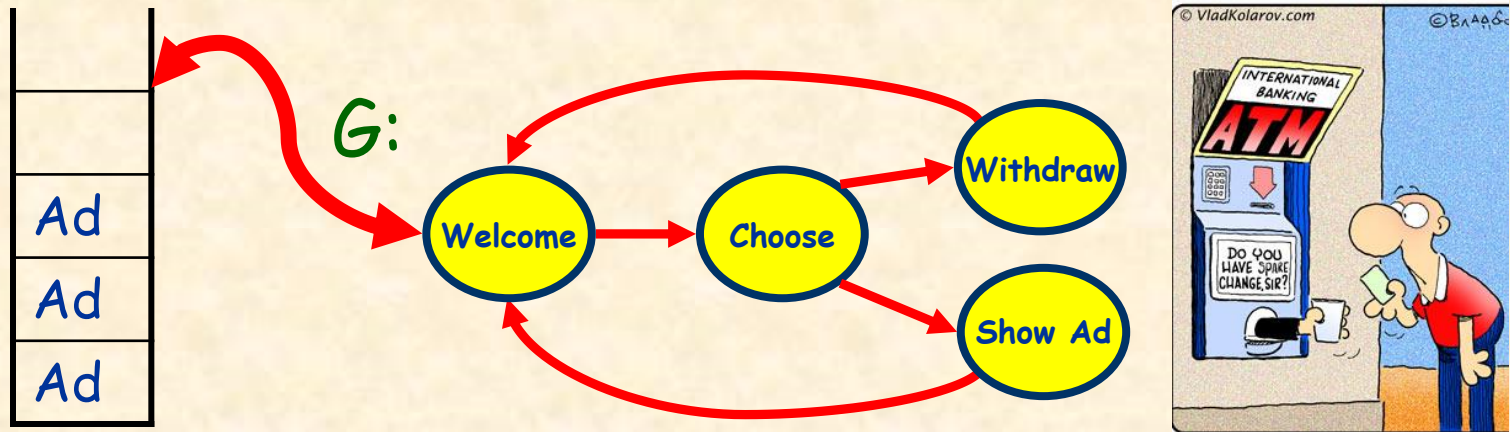
Module Checking Milestones

□ Timeline:

- ◆ 1996-2001: CTL/CTL* two-players turn-based finite-state perfect information.
- ◆ 1997: **mu-calculus** two-players **concurrent** finite-state **imperfect** information
- ◆ 2002-2005: Abstraction refinement and implementation.
- ◆ 2005-2010: two-players turn-based **infinite-state** perfect information
- ◆ 2007-2013: two-players **concurrent** infinite-state **imperfect** information
- ◆ And a number of other extensions in the last decade...

Pushdown Module Checking

- ❑ Consider an open ATM machine with the constraint
“it is not possible to make more withdraws than Ads viewed”
- ❑ We need a stack to count how many Ads remain to be shown



- ❑ A PD is a labeled transition graph augmented with a stack.
- ❑ (q, ξ) is a configuration if q is a node of G and ξ is a stack content
- ❑ An open PD (OPD) has environment and system configurations
- ❑ An OPD induces a Module M where nodes are Pushdown Configurations

PD Module Checking: decide whether $M \models_r \varphi$

- ❑ For example: $M \models_r \forall G \exists F \text{ Show Ad}$ but $M \not\models_r \forall G \exists F \text{ Withdraw}$

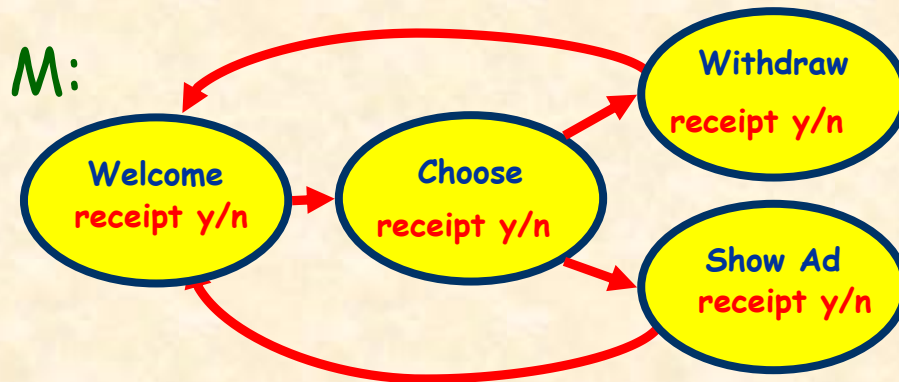
Pushdown Complexity Results

Class	System	PD Model Checking	PD Module Checking
LTL	finite-state	Pspace-Complete	PSpace-Complete
CTL	finite-state	Linear Time [1]	EXPTIME-Complete[3]
CTL*	finite-state	PSpace-Complete [2]	2EXPTIME-Complete[3]
LTL	Pushdown System	Exptime-Complete	Exptime-Complete
CTL	Pushdown System	EXPTIME-Complete[4]	2EXPTIME-Complete[5]
CTL*	Pushdown System	2EXPTIME-Complete[4]	3EXPTIME-Complete[5]
1. [Clarke, Emerson, Sistla 1986] 2. [Emerson and Lei 1985]		3. [Kupferman, Vardi, Wolper 2001] 4. [Walukiewicz 2000] 5. [Bozzelli, Murano, Peron, 2005-2010]	

Exptime-Complete w.r.t the system (fixed formula)

(PD) Module Checking with Imperfect Information

- ❑ The environment can have imperfect information (hidden information) regarding the (control) state and the stack content.



- ❑ The environment does not see the full picture!
...but must act independently of the missing information...
- ❑ Not all the trees in EXEC(M) correspond to an actual environment .
- ❑ M reactively satisfies ϕ iff ϕ holds in all **consistent** (uniform) trees of Exec(M).
- ❑ Checking this consistency is the main difficulty here.
- ❑ [Aminof, Murano, Vardi] Using alternating state PD tree automata, we have proved decidability if the imperfect information resides only in the control states.

From Two Players to Multi Players

- ❑ In 1997, module checking “took” also another direction to deal with multi-player concurrent games

Alternating-Time Temporal Logic

Alternating-Time Temporal Logic

- ATL generalizes CTL: temporal operators are indexed by coalitions of agents.

$$\varphi := \text{true} \mid p \mid \varphi \wedge \varphi \mid \neg\varphi \mid \langle\langle A \rangle\rangle\psi \qquad \psi := X\varphi \mid \varphi U \varphi \mid \varphi R \varphi$$

- $\langle\langle A \rangle\rangle\psi$ means that the team of agents A has a (collective) strategy to enforce ψ .
- ATL formulas are generally interpreted over Concurrent Game Structures (CGS): a Kripke structure whose transitions are labeled with agents' decisions.
- ATL is a story of success with several applications in MAS!

A (refuted) common belief

- ❑ Since its definition, there has been a common belief:

ATL^(*) model checking subsumes CTL^(*) module checking!!!

- ❑ In Murano and Jamroga AAMAS 2014 it has been showed that it is not the case!
 - ◆ In module checking environment's strategies are nondeterministic and irrevocable.
 - ◆ In ATL^(*) agents can only use deterministic and revocable strategies.
 - ◆ ATL^(*) model checking does not have the distinguishing and expressive power of CTL^(*) module checking
 - ◆ To subsume CTL^(*) module checking we have introduced the logic MNIATL^(*)

.

ATL module checking

- ❑ In Murano and Jamroga - AAMAS 2015, finally a new framework that combines and extends the features of the two methodologies has been introduced:
 - ◆ The environment is a special agent acting as in classic module checking: it has nondeterministic irrevocable strategies, possibly acting under imperfect information
 - ◆ The other agents act as in classic ATL.



Conclusion

- ❑ Model checking has been conceived in the 1980s to check **closed systems**
 - ◆ Model behavior determined by internal states.
 - ◆ One source of nondeterminism: the unwinding returns an infinite computation tree
 - ◆ Model checking amounts checking whether this unique tree satisfies the specification

- ❑ Module checking is a powerful method proposed in 1990s for **open systems**:
 - ◆ Open systems adapt their behavior to the input received from the environment
 - ◆ Two sources of nondeterminism: an additional external one from the environment
 - ◆ All possible interactions system-environment induce an infinite set of trees ($\text{Exec}(M)$)
 - ◆ Module checking amounts checking whether all these trees satisfy the specification

- ❑ In the last 20 years, Module checking has been investigated in several settings:
 - ◆ Turn-based/concurrent, perfect/imperfect information, finite/infinite state, etc. 😊

- ❑ Little work has been done on the connection with other methodologies in open system verification and little investigation of its application in AI! 😊 😊

References

- ❑ *Kuperman, Vardi, Wolper. **Module Checking***. Information and Computation 2001. Vol 164(2): 322-344
- ❑ *Kuperman, Vardi. **Module Checking Revisited***. CAV 1997, LNCS 1254, pages 36-47
- ❑ *Bozzelli, Murano, Peron. **Pushdown Module Checking***. Formal Methods in System Design 2010. vol. 36 (1), 65-95
- ❑ *Ferrante, Murano, Parente. **Enriched μ -Calculi Module Checking***. LOGICAL METHODS IN COMPUTER SCIENCE 2008. Vol. 4 (3:1), 1-21
- ❑ *Aminof, Legay, Murano, Serre, Vardi. **Pushdown Module Checking with Imperfect Information***. Information and Computation 2013. Vol. 223, 1-17
- ❑ *Aminof, Murano, Vardi. **Pushdown Module Checking with Imperfect Information***. CONCUR 2007, 460-475
- ❑ *Murano, Parente, Napoli. **Program Complexity in Hierarchical Module Checking***. LPAR 2008, LNCS 4330, 318-332
- ❑ *Alur, Henzinger, Kupferman. **Alternating-Time Temporal Logic***. J. of ACM 2002. Vol 49(5): 672-713
- ❑ *Ågotnes, Goranko, Jamroga. **Alternating-Time Temporal Logics with Irrevocable Strategies***. TARK 2007, 15-24
- ❑ *Jamroga, Murano: **On module checking and strategies***. AAMAS 2014, pages 701-708
- ❑ *Jamroga, Murano: **Module Checking of Strategic Ability***. AAMAS 2015, pages 227-235