Social Laws for Multi-Agent Systems: Logic and Games

Lecture 1: Specifying and verifying state-transition models

Thomas Ågotnes[1]

[1]Department of Information Science and Media Studies
University of Bergen, Norway

NII Tokyo 13 December 2011

Contents

Lecturer

- Professor Thomas Ågotnes
- University of Bergen, Norway
  - 14500 students, 3200 faculty and staff
- Visiting NII 3 December 2011 – 4 February 2012
- Contact: thomas.agotnes@infomedia.uib.no

About the lecture series

Goals:

- To motivate and introduce the idea of *social laws* , or *normative systems*, as a coordination mechanism for multi-agent systems
- To motivate the idea of using *logic* in this context, and in particular *temporal logics* and *cooperation logics* in the formal specification and verifaction of social laws

A social law is a *restriction on the behaviour* of individual agents, which ensures some desired global properties of the behaviour of the system

About the lecture series

1. Specifying and veryfying state-transition models for multi-agent systems
2. Social laws for coordination
3. Dealing with non-compliance
4. Coordinating self-interested agents
5. Social laws design as an optiomisation problem, and as an optimisation problem
6. Reasoning about social laws
7. Strategic reasoning under imperfect information

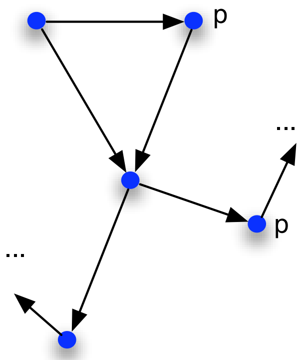
Contents

Multi-agent systems

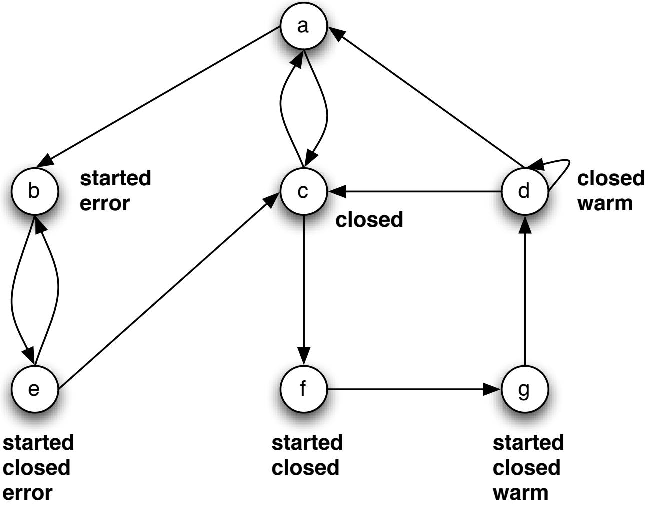
- Consists of several *autonomous agents* that *interact*
- An agent is an entity that *perceives the enviroment*, and *acts*
- An *agent* can be an (artificially) *intelligent agent program*
- But can also be a simple component of some system, like a thermostate

Formal models of multi-agent systems

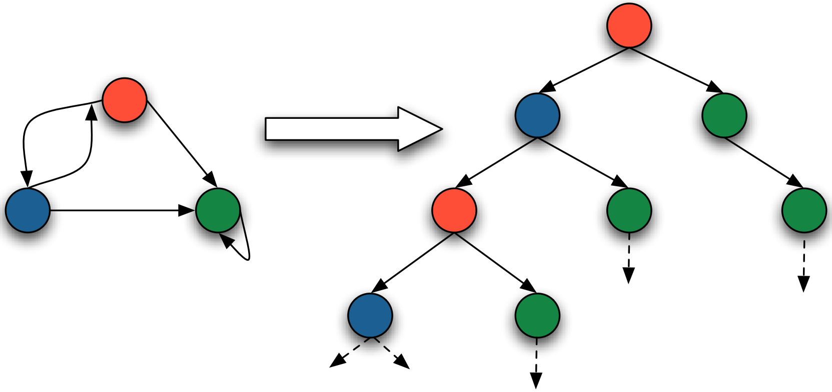
- State-transition models
- Quite common abstraction
  - Ex.: UML



Example: microwave oven

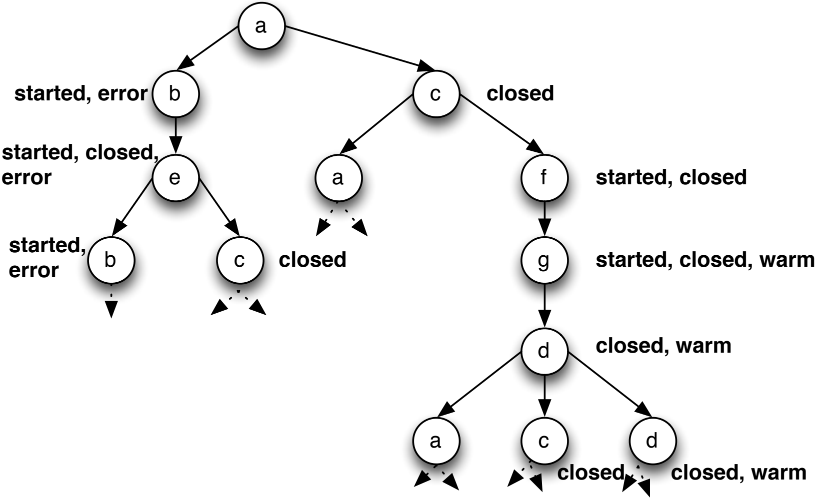


Computations



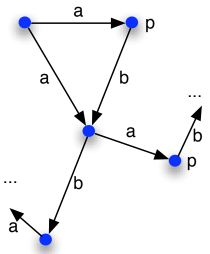
Every path in the tree obtained from a given *initial state* represents a *possible computation*

Computations: example



Adding agents

- If the system is a *multi-agent system*, the states are *global states*
- We label the transitions with the *name of the agent* that causes the transition by executing some *action*
- This assumes *asynchronous action*



Formally

An agent-labelled Kripke structure (over Φ) is a 6-tuple:

$$K = \langle S, S^0, R, Ag, \alpha, V \rangle, \text{ where}$$

- S is a finite, non-empty set of states,
- $S^0 \subseteq S$ ($S^0 \neq \emptyset$) is the set of initial states;
- $R \subseteq S \times S$ is a total (each state has a successor) binary transition relation on S;
- $Ag = \{1, \ldots, n\}$ is the set of agents;
- $\alpha : R \to Ag$ labels each transition in R with an agent
- $V : S \to 2^\Phi$ labels states with a set of propositional atoms

Note: simplifying assumption: single agent execute single action in each state (*interleaved concurrency*)

Contents

Introduction

- Consider this statement:

  The microwave can only start without error if it is closed

  Is it true in our microwave model?
- This statement is true in *some* models (including ours), but false in others
- We want to be able to check whether such properties hold or not *automatically*
- Thus we need *a precise way of writing down statements*
- For that we can use modal logic, and in particular *Computation Tree Logic (CTL)*

CTL: language

The language of CTL (CTL formulas) is defined as follows:

- Propositional atoms such as *p* or *started* are formulas
- Formulas can be combined using propositional connectives such as $\wedge$ (and), $\vee$ (or), $\neg$ (not), $\rightarrow$ (implication), etc.
- We can construct new formulas by putting *temporal connectives* in front of an existing formula. If φ and ψ are formulas, then the following as also formulas:

$E\bigcirc\varphi$	on some path, φ is true next
$E(\varphi\mathcal{U}\psi)$	on some path, φ until ψ
$E\Diamond\varphi$	on some path, eventually φ
$E\Box\varphi$	on some path, always φ
$A\bigcirc\varphi$	on all paths, φ is true next
$A(\varphi\mathcal{U}\psi)$	on all paths, φ until ψ
$A\Diamond\varphi$	on all paths, eventually φ
$A\Box\varphi$	on all paths, always φ

CTL: language

The language of CTL (CTL formulas) is defined as follows:

- Propositional atoms such as *p* or *started* are formulas
- Formulas can be combined using propositional connectives such as $\wedge$ (and), $\vee$ (or), $\neg$ (not), $\rightarrow$ (implication), etc.
- We can construct new formulas by putting *temporal connectives* in front of an existing formula. If φ and ψ are formulas, then the following as also formulas:

$\mathsf{E}\bigcirc\varphi$	on some path, φ is true next
$\mathsf{E}(\varphi\,\mathcal{U}\,\psi)$	on some path, φ until ψ
$\mathsf{E}\Diamond\varphi$	on some path, eventually φ
$\mathsf{E}\Box\varphi$	on some path, always φ
$\mathsf{A}\bigcirc\varphi$	on all paths, φ is true next
$\mathsf{A}(\varphi\,\mathcal{U}\,\psi)$	on all paths, φ until ψ
$\mathsf{A}\Diamond\varphi$	on all paths, eventually φ
$\mathsf{A}\Box\varphi$	on all paths, always φ

Examples

- $\mathsf{E}\bigcirc \mathit{error}$ (it is possible that there is an error in the next state)
- $\neg\mathsf{A}\bigcirc \mathit{error}$ (it is not necessary that there is an error in the next state)
- $\mathsf{E}\Diamond \mathit{warm}$ (it is possible that the oven will eventually be warm)
- $\mathsf{A}\Box\neg(\mathit{warm} \wedge \mathit{error})$ (it is necessary that the microwave can never be both warm and have an error)
- $\mathsf{A}\Diamond \mathit{closed}$ (it is necessary that the microwave will eventually be closed)
- $\mathsf{A}(\neg \mathit{warm} \,\mathcal{U}\, \mathit{closed})$ (it is necessary that the microwave is cold until it is closed)
- $\mathsf{A}\Box(\neg \mathit{closed} \rightarrow \neg\mathsf{E}\bigcirc(\mathit{start} \wedge \neg \mathit{error}))$ (the microwave can only start without error if it is closed)

Examples

- $\mathsf{E}\bigcirc$ *error* (it is possible that there is an error in the next state)
- $\neg\mathsf{A}\bigcirc$ *error* (it is not necessary that there is an error in the next state)
- $\mathsf{E}\Diamond$ *warm* (it is possible that the oven will eventually be warm)
- $\mathsf{A}\Box\neg(warm \wedge error)$ (it is necessary that the microwave can never be both warm and have an error)
- $\mathsf{A}\Diamond$ *closed* (it is necessary that the microwave will eventually be closed)
- $\mathsf{A}(\neg warm\,\mathcal{U}\,closed)$ (it is necessary that the microwave is cold until it is closed)
- $\mathsf{A}\Box(\neg closed \rightarrow \neg\mathsf{E}\bigcirc(start \wedge \neg error))$ (the microwave can only start without error if it is closed)

Examples

- $\mathsf{E}\bigcirc$ *error* (it is possible that there is an error in the next state)
- $\neg\mathsf{A}\bigcirc$ *error* (it is not necessary that there is an error in the next state)
- $\mathsf{E}\Diamond$*warm* (it is possible that the oven will eventually be warm)
- $\mathsf{A}\,\Box\neg(\mathit{warm} \wedge \mathit{error})$ (it is necessary that the microwave can never be both warm and have an error)
- $\mathsf{A}\Diamond$*closed* (it is necessary that the microwave will eventually be closed)
- $\mathsf{A}(\neg\mathit{warm}\,\mathcal{U}\,\mathit{closed})$ (it is necessary that the microwave is cold until it is closed)
- $\mathsf{A}\,\Box(\neg\mathit{closed} \rightarrow \neg\mathsf{E}\bigcirc(\mathit{start} \wedge \neg\mathit{error}))$ (the microwave can only start without error if it is closed)

Examples

- $E\bigcirc$ *error* (it is possible that there is an error in the next state)
- $\neg A\bigcirc$ *error* (it is not necessary that there is an error in the next state)
- $E\Diamond$ *warm* (it is possible that the oven will eventually be warm)
- $A\Box\neg(warm \wedge error)$ (it is necessary that the microwave can never be both warm and have an error)
- $A\Diamond$ *closed* (it is necessary that the microwave will eventually be closed)
- $A(\neg warm\,\mathcal{U}\,closed)$ (it is necessary that the microwave is cold until it is closed)
- $A\Box(\neg closed \rightarrow \neg E\bigcirc(start \wedge \neg error))$ (the microwave can only start without error if it is closed)

Examples

- $\mathsf{E}\bigcirc$ *error* (it is possible that there is an error in the next state)
- $\neg\mathsf{A}\bigcirc$ *error* (it is not necessary that there is an error in the next state)
- $\mathsf{E}\Diamond$ *warm* (it is possible that the oven will eventually be warm)
- $\mathsf{A}\Box\neg(\mathit{warm} \wedge \mathit{error})$ (it is necessary that the microwave can never be both warm and have an error)
- $\mathsf{A}\Diamond$ *closed* (it is necessary that the microwave will eventually be closed)
- $\mathsf{A}(\neg \mathit{warm}\,\mathcal{U}\,\mathit{closed})$ (it is necessary that the microwave is cold until it is closed)
- $\mathsf{A}\Box(\neg \mathit{closed} \rightarrow \neg\mathsf{E}\bigcirc(\mathit{start} \wedge \neg \mathit{error}))$ (the microwave can only start without error if it is closed)

Examples

- $\mathsf{E}\bigcirc \mathit{error}$ (it is possible that there is an error in the next state)
- $\neg\mathsf{A}\bigcirc \mathit{error}$ (it is not necessary that there is an error in the next state)
- $\mathsf{E}\Diamond \mathit{warm}$ (it is possible that the oven will eventually be warm)
- $\mathsf{A}\Box\neg(\mathit{warm} \wedge \mathit{error})$ (it is necessary that the microwave can never be both warm and have an error)
- $\mathsf{A}\Diamond \mathit{closed}$ (it is necessary that the microwave will eventually be closed)
- $\mathsf{A}(\neg \mathit{warm}\,\mathcal{U}\,\mathit{closed})$ (it is necessary that the microwave is cold until it is closed)
- $\mathsf{A}\Box(\neg \mathit{closed} \rightarrow \neg\mathsf{E}\bigcirc(\mathit{start} \wedge \neg \mathit{error}))$ (the microwave can only start without error if it is closed)

Examples

- $E\bigcirc \mathit{error}$ (it is possible that there is an error in the next state)
- $\neg A\bigcirc \mathit{error}$ (it is not necessary that there is an error in the next state)
- $E\Diamond \mathit{warm}$ (it is possible that the oven will eventually be warm)
- $A\Box\neg(\mathit{warm} \wedge \mathit{error})$ (it is necessary that the microwave can never be both warm and have an error)
- $A\Diamond \mathit{closed}$ (it is necessary that the microwave will eventually be closed)
- $A(\neg \mathit{warm}\,\mathcal{U}\,\mathit{closed})$ (it is necessary that the microwave is cold until it is closed)
- $A\Box(\neg \mathit{closed} \rightarrow \neg E\bigcirc(\mathit{start} \wedge \neg \mathit{error}))$ (the microwave can only start without error if it is closed)

Examples

- $A\Box(A\Diamond enabled)$ (the process is infinitely often enabled)
- $A\Box(A\Diamond deadlock)$ (the process will eventually be in a permanent deadlock)
- $A\Box(E\Diamond restart)$ (it is always possible to get to the restart-state)
- $A\Box\neg(c_1 \wedge c_2)$ (safety)
- $A\Box(t_1 \rightarrow \Box\Diamond c_1)$ (liveness)

Examples

- $A\Box(A\Diamond \mathit{enabled})$ (the process is infinitely often enabled)
- $A\Box(A\Diamond \mathit{deadlock})$ (the process will eventually be in a permanent deadlock)
- $A\Box(E\Diamond \mathit{restart})$ (it is always possible to get to the restart-state)
- $A\Box\neg(c_1 \wedge c_2)$ (safety)
- $A\Box(t_1 \rightarrow \Box\Diamond c_1)$ (liveness)

Examples

- $A\Box(A\Diamond \mathit{enabled})$ (the process is infinitely often enabled)
- $A\Box(A\Diamond \mathit{deadlock})$ (the process will eventually be in a permanent deadlock)
- $A\Box(E\Diamond \mathit{restart})$ (it is always possible to get to the restart-state)
- $A\Box\neg(c_1 \wedge c_2)$ (safety)
- $A\Box(t_1 \rightarrow \Box\Diamond c_1)$ (liveness)

Examples

- $A\Box(A\Diamond \textit{enabled})$ (the process is infinitely often enabled)
- $A\Box(A\Diamond \textit{deadlock})$ (the process will eventually be in a permanent deadlock)
- $A\Box(E\Diamond \textit{restart})$ (it is always possible to get to the restart-state)
- $A\Box\neg(c_1 \wedge c_2)$ (safety)
- $A\Box(t_1 \to \Box\Diamond c_1)$ (liveness)

Examples

- $A\Box(A\Diamond \mathit{enabled})$ (the process is infinitely often enabled)
- $A\Box(A\Diamond \mathit{deadlock})$ (the process will eventually be in a permanent deadlock)
- $A\Box(E\Diamond \mathit{restart})$ (it is always possible to get to the restart-state)
- $A\Box\neg(c_1 \wedge c_2)$ (safety)
- $A\Box(t_1 \rightarrow \Box\Diamond c_1)$ (liveness)

Formal interpretation

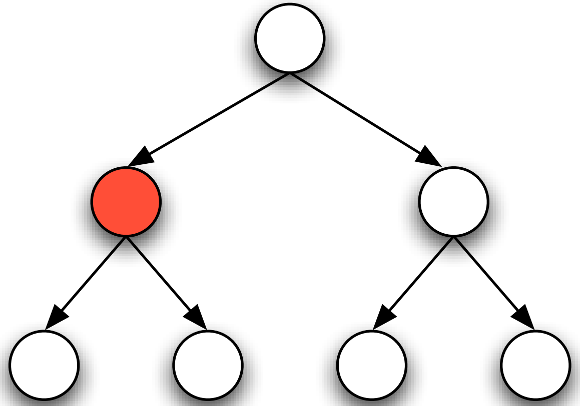
Given a Kripke model K, a state s in K and a CTL formula φ,

$$K, s \models \varphi$$

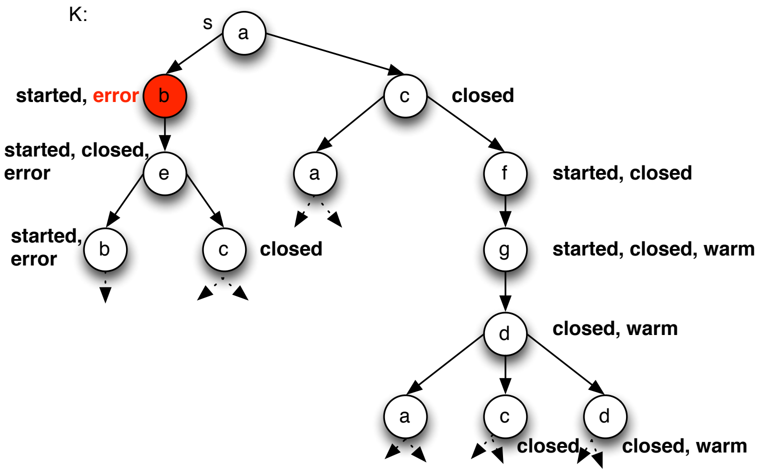
means that φ is true (or satisfied) in state s of K.

E◯

E◯ = for some path, in the next state



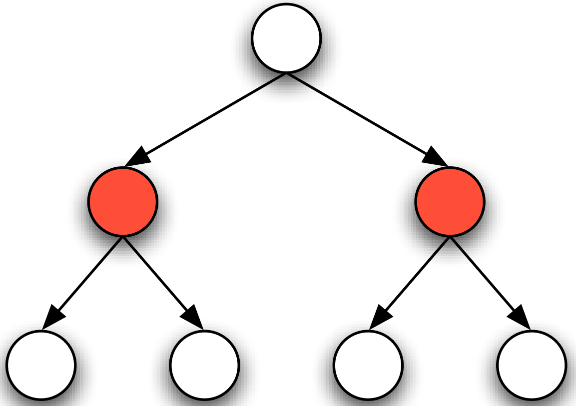
Example: E$\bigcirc$



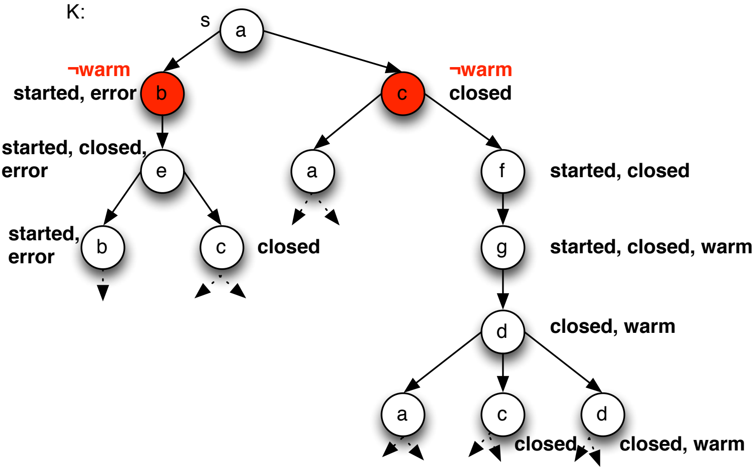
$$K, s \models \mathrm{E}\bigcirc \mathit{error}$$

A○

A○ = for all paths, in the next state



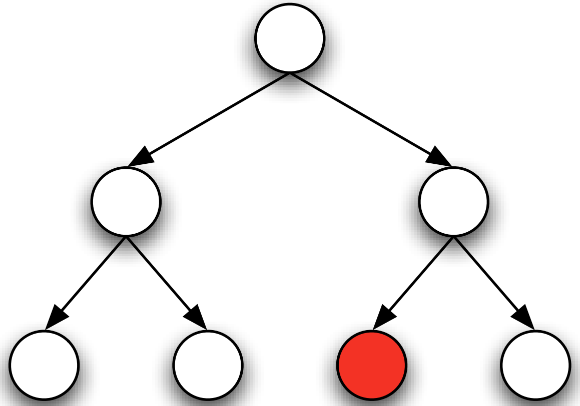
Example: A◯



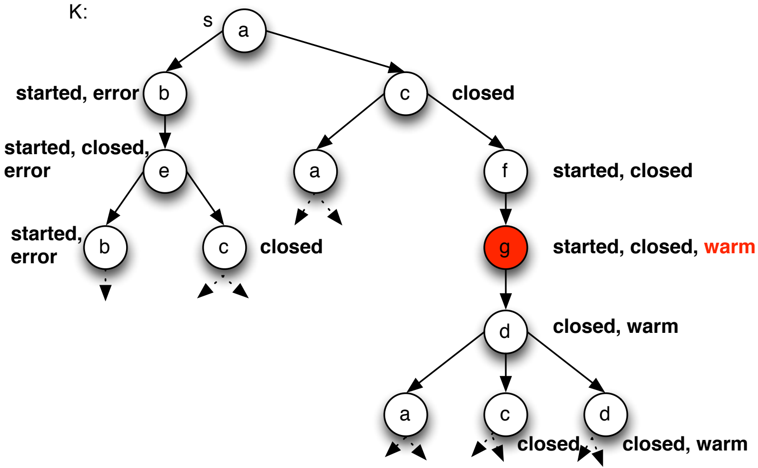
$K, s \models A\bigcirc\neg warm$

E◇

E◇ = for some path, in some future state



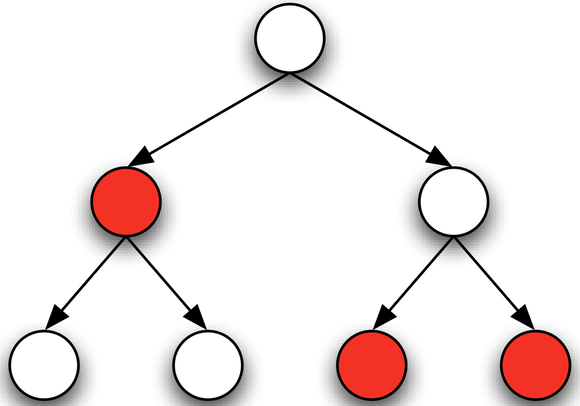
Example: E♢



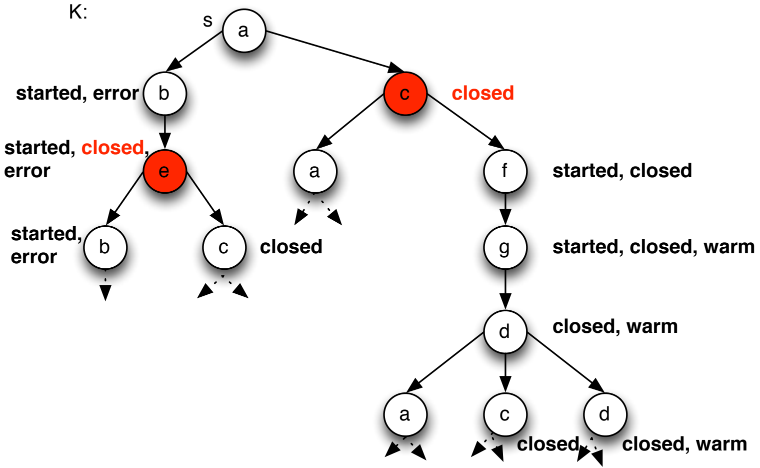
$$K, s \models \mathsf{E}\Diamond warm$$

A◇

A◇ = for all paths, in *some future* state



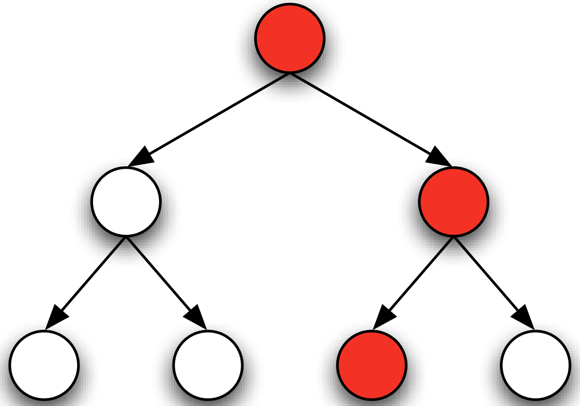
Example: A◇



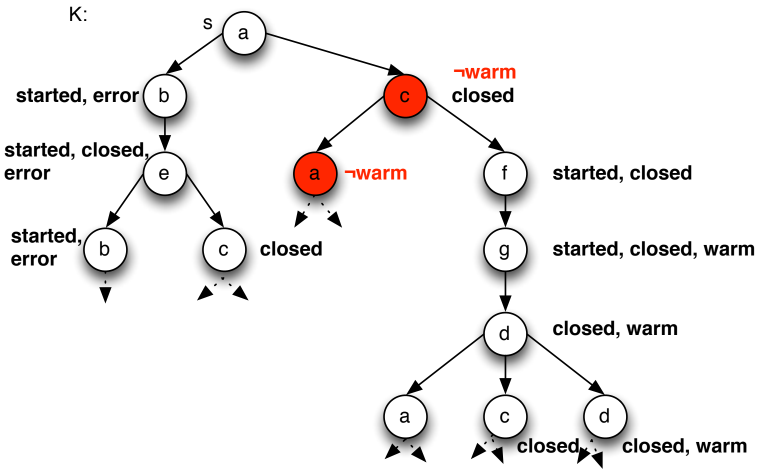
$K, s \models A\Diamond closed$

E $\Box$

E $\Box$ = for some path, in all future states



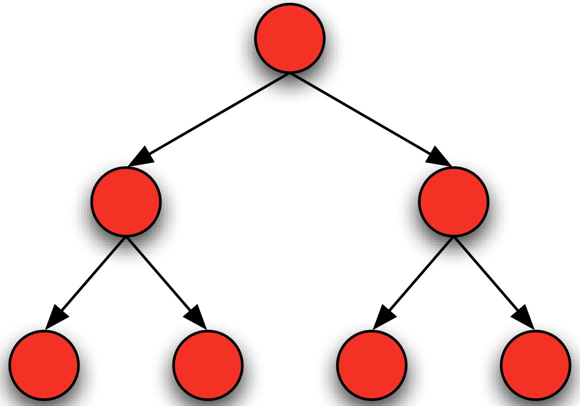
Example: E □



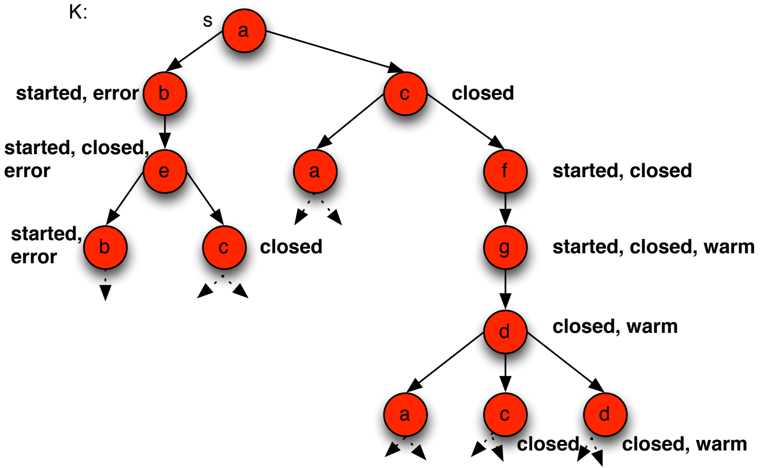
$K, s \models E\Box\neg warm$

A □

A □ = for all paths, in all future states



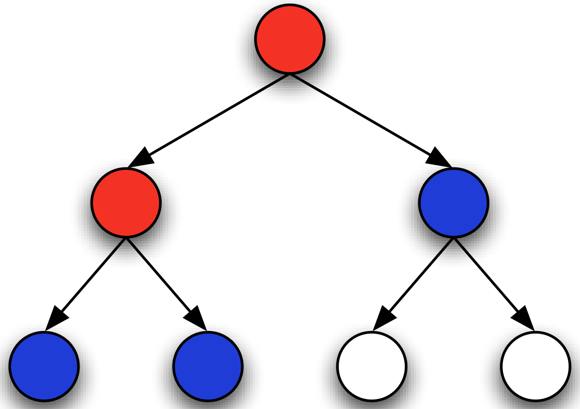
Example: A □



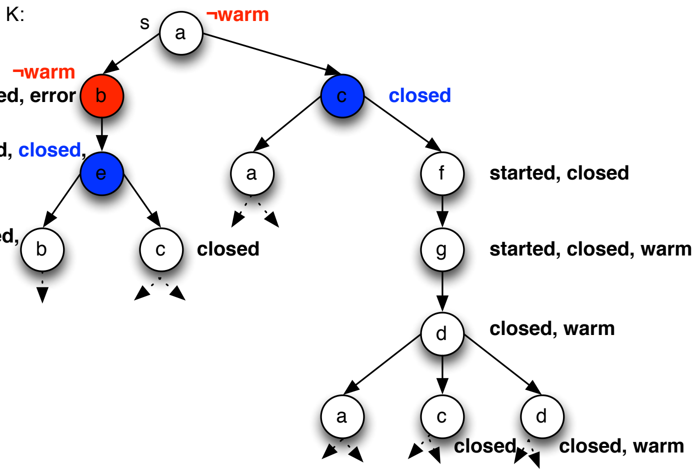
$K, s \models A\Box\neg(warm \wedge error)$

$\mathsf{A}\mathcal{U}$

$\mathsf{A}\mathcal{U}$ = for all paths, ψ becomes true in some future states and φ is true in all states before that

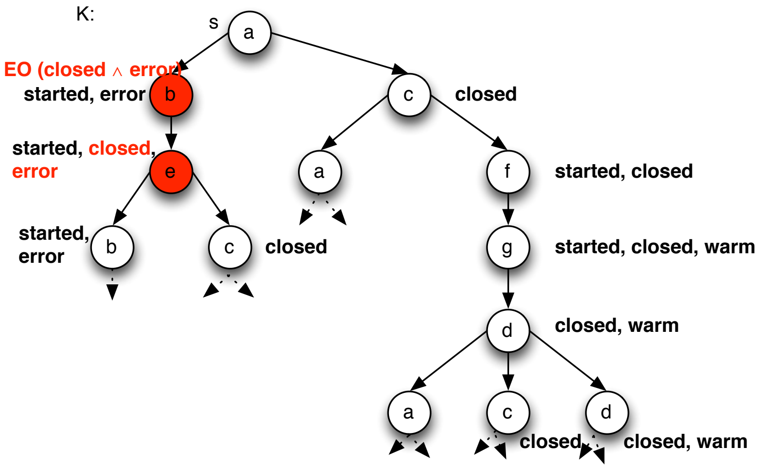


Example: $\square\ \mathcal{U}$



$$K, s \models A(\neg warm\ \mathcal{U}\ closed)$$

Example: nesting



$$K, s \models E\bigcirc E\bigcirc(closed \wedge error)$$

Notation

We use

$$K \models \varphi$$

to denote the fact that $K, s \models \varphi$ for all initial states $s \in S_0$.

Some equivalences

- $\varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi)$
- $\mathsf{A}\bigcirc\varphi \equiv \neg\mathsf{E}\bigcirc\neg\varphi$
- $\mathsf{E}\Diamond \equiv E(\top\, \mathcal{U}\, \varphi)$
- $\mathsf{A}\Box\varphi \equiv \neg\mathsf{E}\Diamond\neg\varphi$
- $\mathsf{A}(\varphi\mathcal{U}\psi) \equiv \neg(\mathsf{E}(\neg\psi\ \mathcal{U}\neg(\varphi\vee\psi)) \vee \mathsf{E}\Box\neg\psi)$

That means that we only need the operators $\{\top, \neg, \wedge, \mathsf{E}\Box, \mathsf{E}\mathcal{U}, \mathsf{E}\bigcirc\}$!

Some equivalences

- $\varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi)$
- $\mathsf{A}\bigcirc\varphi \equiv \neg\mathsf{E}\bigcirc\neg\varphi$
- $\mathsf{E}\Diamond \equiv E(\top\ \mathcal{U}\ \varphi)$
- $\mathsf{A}\square\varphi \equiv \neg\mathsf{E}\Diamond\neg\varphi$
- $\mathsf{A}(\varphi\mathcal{U}\psi) \equiv \neg(\ \mathsf{E}(\neg\psi\ \mathcal{U}\neg(\varphi\vee\psi)) \vee \mathsf{E}\square\neg\psi\ )$

That means that we only need the operators $\{\top, \neg, \wedge, \mathsf{E}\square, \mathsf{E}\mathcal{U}, \mathsf{E}\bigcirc\}$!

Some equivalences

- $\varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi)$
- $\mathsf{A}\bigcirc\varphi \equiv \neg\mathsf{E}\bigcirc\neg\varphi$
- $\mathsf{E}\Diamond \equiv E(\top\ \mathcal{U}\ \varphi)$
- $\mathsf{A}\Box\varphi \equiv \neg\mathsf{E}\Diamond\neg\varphi$
- $\mathsf{A}(\varphi\mathcal{U}\psi) \equiv \neg(\mathsf{E}(\neg\psi\ \mathcal{U}\neg(\varphi\vee\psi)) \vee \mathsf{E}\Box\neg\psi)$

That means that we only need the operators $\{\top, \neg, \wedge, \mathsf{E}\Box, \mathsf{E}\mathcal{U}, \mathsf{E}\bigcirc\}$!

Some equivalences

- $\varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi)$
- $\mathsf{A}\bigcirc\varphi \equiv \neg\mathsf{E}\bigcirc\neg\varphi$
- $\mathsf{E}\Diamond \equiv E(\top\,\mathcal{U}\,\varphi)$
- $\mathsf{A}\Box\varphi \equiv \neg\mathsf{E}\Diamond\neg\varphi$
- $\mathsf{A}(\varphi\,\mathcal{U}\,\psi) \equiv \neg(\,\mathsf{E}(\neg\psi\,\mathcal{U}\,\neg(\varphi \vee \psi)) \vee \mathsf{E}\Box\neg\psi\,)$

That means that we only need the operators $\{\top, \neg, \wedge, \mathsf{E}\Box, \mathsf{E}\mathcal{U}, \mathsf{E}\bigcirc\}$!

Some equivalences

- $\varphi \lor \psi \equiv \neg(\neg\varphi \land \neg\psi)$
- $\mathsf{A}\bigcirc\varphi \equiv \neg\mathsf{E}\bigcirc\neg\varphi$
- $\mathsf{E}\Diamond \equiv E(\top\ \mathcal{U}\ \varphi)$
- $\mathsf{A}\Box\varphi \equiv \neg\mathsf{E}\Diamond\neg\varphi$
- $\mathsf{A}(\varphi\mathcal{U}\psi) \equiv \neg(\ \mathsf{E}(\neg\psi\ \mathcal{U}\ \neg(\varphi \lor \psi)) \lor \mathsf{E}\Box\neg\psi\ )$

That means that we only need the operators $\{\top, \neg, \land, \mathsf{E}\Box, \mathsf{E}\mathcal{U}, \mathsf{E}\bigcirc\}$!

Some equivalences

- $\varphi \vee \psi \equiv \neg(\neg\varphi \wedge \neg\psi)$
- $\mathsf{A}\bigcirc\varphi \equiv \neg\mathsf{E}\bigcirc\neg\varphi$
- $\mathsf{E}\Diamond \equiv E(\top\ \mathcal{U}\ \varphi)$
- $\mathsf{A}\Box\varphi \equiv \neg\mathsf{E}\Diamond\neg\varphi$
- $\mathsf{A}(\varphi\mathcal{U}\psi) \equiv \neg(\ \mathsf{E}(\neg\psi\ \mathcal{U}\neg(\varphi \vee \psi)) \vee \mathsf{E}\Box\neg\psi\ )$

That means that we only need the operators $\{\top, \neg, \wedge, \mathsf{E}\Box, \mathsf{E}\mathcal{U}, \mathsf{E}\bigcirc\}$!

Contents

ATL

- *Alternating-time Temporal Logic (ATL)* is an agentized extension of CTL introduced by Alur and colleagues (1997)
- The language of ATL is obtained by replacing A and E with $\langle\langle C \rangle\rangle$ where $C \subseteq Ag$ and Ag is the finite set of all agents in the system
- Intuitively,

$$\langle\langle C \rangle\rangle \Diamond \varphi$$

  means that
  - C can cooperate to ensure that φ becomes true sometime in the future no matter what the other agents do (and similarly for $\bigcirc$, $\Box$, $\mathcal{U}$)
  - C has a strategy to enforce that φ becomes true sometime in the future
  - Is used to reason about *game-like* distributed systems

Example

$$\langle\langle merkel, sarkozy \rangle\rangle \Diamond \neg crisis$$

Merkel and Sarkozy can cooperate to ensure that at some point in the future the crisis is over

Example

$$\langle\langle Ann \rangle\rangle \Box \langle\langle Bob \rangle\rangle \Diamond win$$

ATL and CTL

CTL is contained in ATL:

- $\mathsf{A} \equiv \langle\langle \emptyset \rangle\rangle$
- $\mathsf{E} \equiv \langle\langle Ag \rangle\rangle$

Definition (ATL models)

A concurrent game structure is a tuple $M = \langle Ag, S, \pi, Act, d, o\rangle$, where:

- Ag: a finite set of all agents
- S: a set of states
- π: a valuation of propositions
- Act: a finite set of (atomic) actions
- $d : Ag \times S \rightarrow \wp(Act)$ defines actions available to an agent in a state
- o: a deterministic transition function that assigns outcome states $q' = o(q, \alpha_1, \ldots, \alpha_k)$ to states and tuples of actions

A strategy for agent i is a function $f : S \rightarrow Act$ such that

$$f(s) \in d_(i, s)$$

Definition (ATL models)

A concurrent game structure is a tuple $M = \langle Ag, S, \pi, Act, d, o \rangle$, where:

- Ag: a finite set of all agents
- S: a set of states
- π: a valuation of propositions
- Act: a finite set of (atomic) actions
- $d : Ag \times S \rightarrow \wp(Act)$ defines actions available to an agent in a state
- o: a deterministic transition function that assigns outcome states $q' = o(q, \alpha_1, \ldots, \alpha_k)$ to states and tuples of actions

A strategy for agent i is a function $f : S \rightarrow Act$ such that

$$f(s) \in d_(i, s)$$

Definition (ATL models)

A concurrent game structure is a tuple $M = \langle Ag, S, \pi, Act, d, o \rangle$, where:

- Ag: a finite set of all agents
- S: a set of states
- π: a valuation of propositions
- Act: a finite set of (atomic) actions
- $d : Ag \times S \to \wp(Act)$ defines actions available to an agent in a state
- o: a deterministic transition function that assigns outcome states $q' = o(q, \alpha_1, \ldots, \alpha_k)$ to states and tuples of actions

A strategy for agent i is a function $f : S \to Act$ such that

$$f(s) \in d_(i, s)$$

Definition (ATL models)

A concurrent game structure is a tuple $M = \langle Ag, S, \pi, Act, d, o \rangle$, where:

- Ag: a finite set of all agents
- S: a set of states
- π: a valuation of propositions
- Act: a finite set of (atomic) actions
- $d : Ag \times S \rightarrow \wp(Act)$ defines actions available to an agent in a state
- o: a deterministic transition function that assigns outcome states $q' = o(q, \alpha_1, \ldots, \alpha_k)$ to states and tuples of actions

A strategy for agent i is a function $f : S \rightarrow Act$ such that

$$f(s) \in d_(i, s)$$

Definition (ATL models)

A concurrent game structure is a tuple $M = \langle Ag, S, \pi, Act, d, o \rangle$, where:

- Ag: a finite set of all agents
- S: a set of states
- π: a valuation of propositions
- Act: a finite set of (atomic) actions
- $d : Ag \times S \to \wp(Act)$ defines actions available to an agent in a state
- o: a deterministic transition function that assigns outcome states $q' = o(q, \alpha_1, \ldots, \alpha_k)$ to states and tuples of actions

A strategy for agent i is a function $f : S \to Act$ such that

$$f(s) \in d_(i, s)$$

Definition (ATL models)

A concurrent game structure is a tuple $M = \langle Ag, S, \pi, Act, d, o \rangle$, where:

- Ag: a finite set of all agents
- S: a set of states
- π: a valuation of propositions
- Act: a finite set of (atomic) actions
- $d : Ag \times S \to \wp(Act)$ defines actions available to an agent in a state
- o: a deterministic transition function that assigns outcome states $q' = o(q, \alpha_1, \ldots, \alpha_k)$ to states and tuples of actions

A strategy for agent i is a function $f : S \to Act$ such that

$$f(s) \in d_(i, s)$$

Definition (ATL models)

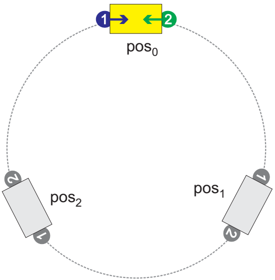
A concurrent game structure is a tuple $M = \langle Ag, S, \pi, Act, d, o \rangle$, where:

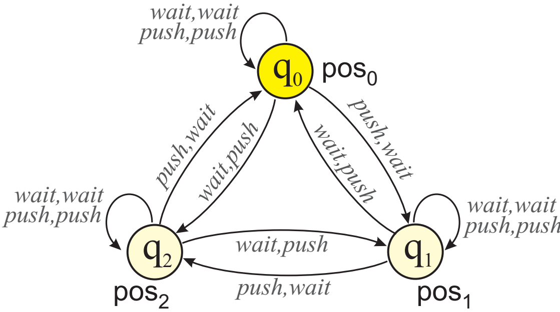
- Ag: a finite set of all agents
- S: a set of states
- π: a valuation of propositions
- Act: a finite set of (atomic) actions
- $d : Ag \times S \rightarrow \wp(Act)$ defines actions available to an agent in a state
- o: a deterministic transition function that assigns outcome states $q' = o(q, \alpha_1, \ldots, \alpha_k)$ to states and tuples of actions

A strategy for agent i is a function $f : S \rightarrow Act$ such that

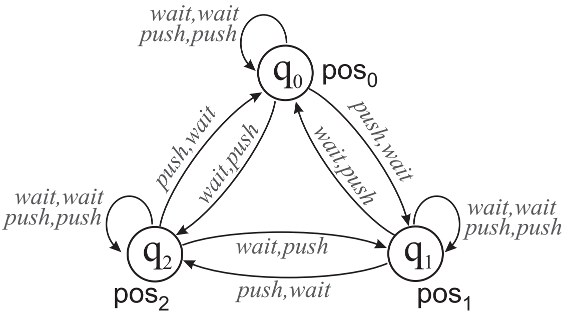
$$f(s) \in d_(i, s)$$

Example: Robots and Carriage



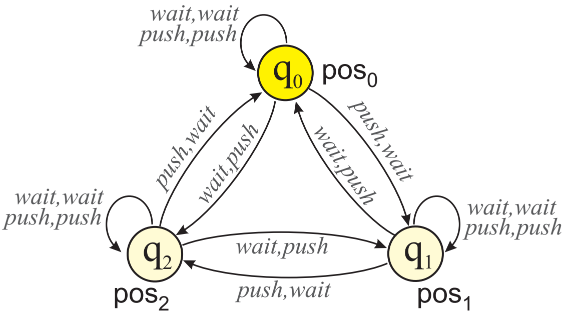


Example: Robots and Carriage



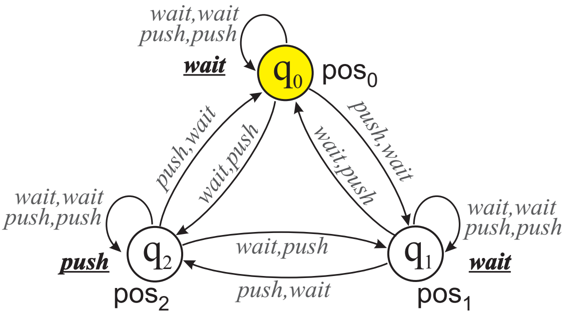
$$pos_0 \rightarrow \langle\langle 1 \rangle\rangle \Box \neg pos_1$$

Example: Robots and Carriage



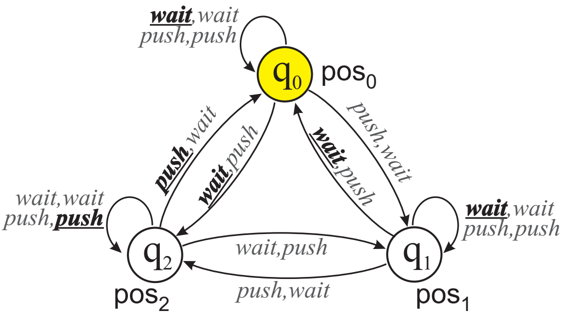
$$pos_0 \rightarrow \langle\langle 1 \rangle\rangle \Box \neg pos_1$$

Example: Robots and Carriage



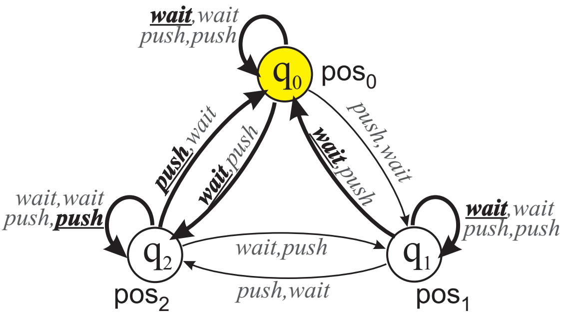
$$pos_0 \rightarrow \langle\langle 1 \rangle\rangle \Box \neg pos_1$$

Example: Robots and Carriage



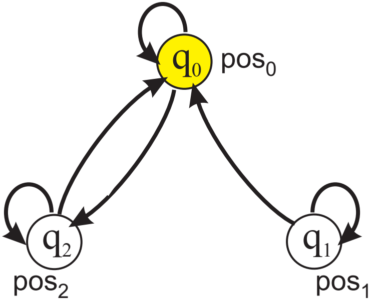
$$pos_0 \rightarrow \langle\langle 1 \rangle\rangle \Box \neg pos_1$$

Example: Robots and Carriage



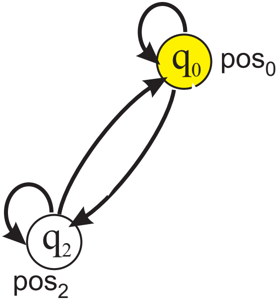
$pos_0 \rightarrow \langle\langle 1 \rangle\rangle \Box \neg pos_1$

Example: Robots and Carriage



$$pos_0 \rightarrow \langle\langle 1 \rangle\rangle \Box \neg pos_1$$

Example: Robots and Carriage



$$pos_0 \rightarrow \langle\langle 1 \rangle\rangle \Box \neg pos_1$$

Contents

Verification

- Verification means to make sure that the design of a (hardware or software) system is correct
- Ariane 5: the world's most expensive software bug (exploded 37 seconds after takeoff)

Formal specification and verification

- Traditional verification methods such as *simulation* or *testing* are not exhaustive, they don't explore all possible behaviours of the system
- Formal verification methods do, and they can therefore give a *guarantee* that the design does not have any errors
- We specify the properties we want to check that the system has as a formula in some formal logic
  - Example: $A\Box(t_1 \rightarrow A\Diamond c_1)$ (liveness)
- Two main techniques:
  - Proof-based: describe also the system using formal logic, and try to find a formal proof that the property follows
  - Model-based: describe the system using a mathematical structure, and use an algorithm to check whether the property holds

Reasoning: satisfiability

- The *satisfiability problem* is as follows: *Given a formula φ is there some interpretation that makes φ true?*
- How hard is the satisfiability problem?
  - For *Coalition Logic*: *PSPACE-complete* (Pauly, 2001).
  - For *CTL*: *EXPTIME-complete* – a lower bound for ATL.

Reasoning with ATL: Satisfiability

Theorem (van Drimmelen, 2003)

For any fixed set of agents Ag, satisfiability for ATL formulae over Ag is EXPTIME-complete.

But if the set of agents is not fixed, van Drimmelen's algorithm is *2EXPTIME*.

Theorem (Walther, Lutz, Wolter, Wooldridge, 2006)

The satisfiability problem for arbitrary ATL formulae is EXPTIME-complete (and hence no harder than CTL).

Reasoning with ATL: Satisfiability

Theorem (van Drimmelen, 2003)

For any fixed set of agents Ag, satisfiability for ATL formulae over Ag is EXPTIME-complete.

But if the set of agents is not fixed, van Drimmelen's algorithm is *2EXPTIME*.

Theorem (Walther, Lutz, Wolter, Wooldridge, 2006)

The satisfiability problem for arbitrary ATL formulae is EXPTIME-complete (and hence no harder than CTL).

Reasoning with ATL: Satisfiability

Theorem (van Drimmelen, 2003)

For any fixed set of agents Ag, satisfiability for ATL formulae over Ag is EXPTIME-complete.

But if the set of agents is not fixed, van Drimmelen's algorithm is *2EXPTIME*.

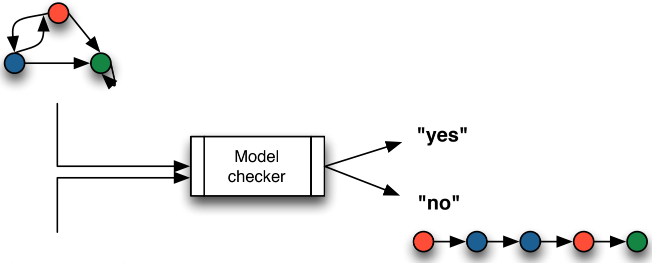
Theorem (Walther, Lutz, Wolter, Wooldridge, 2006)

The satisfiability problem for arbitrary ATL formulae is EXPTIME-complete (and hence no harder than CTL).

Model checking

- Successfull model-based verification technique
- Input: K, φ
- Output: yes if $K \models \varphi$
- Effective algorithms exist

Model checking



A□ (A◇ enabled)

Input: model and formula

Output: "no" and a counterexample (sequence of states where the property does not hold)

Model checking: complexity

Model checking CTL and ATL can be done in time polynomial in the size of the formula and the number of states in the model

... if the model is explicitly represented

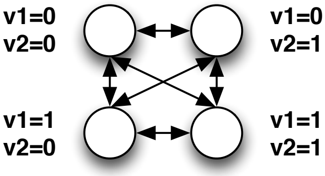
Model checking: complexity

Model checking CTL and ATL can be done in time polynomial in the size of the formula and the number of states in the model

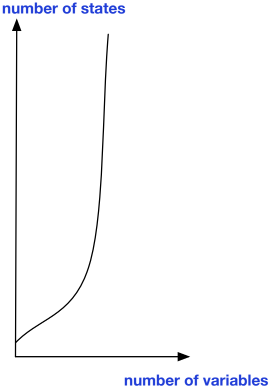
... if the model is explicitly represented

Explicit representation

- Explicit representation is *often not feasible*
- Number of states *increase exponentially* with the number of independent variables



2 independent variables



Model representation languages

- *Practical* model checkers use *high-level* model specification languages.
- *Reactive modules*: a rule-based language for model specification...

```
module toggle controls x
  init
  [] ⊤ -> x' := ⊤
  [] ⊤ -> x' := ⊥
  update
  [] x-> x' := ⊥
  [] (¬x)->x' := ⊤
```

Practical model checking

The complexity of model checking depends on the *specification language* and the *representation of the model*.

Theorem (van der Hoek, Lomuscio, Wooldridge, AAMAS06)

For Reactive Modules models, model checking is exactly as hard as theorem proving in the corresponding language:

ATL	*EXPTIME-complete*
Coalition Logic	*PSPACE-complete*
prop logic	*co-NP-complete*