TU Clausthal
Clausthal University of Technology

UNIVERSITY OF LIVERPOOL

Chapter 14: Specification and Verification of Multi-Agent Systems

Jürgen Dix and Michael Fisher

Multi-Agent Systems, edited by Gerhard Weiss
MIT Press, May 2012

Time

Duration: Six lectures

Course type

Level: advanced
Prerequisites:

Course website

http://mitpress.mit.edu/multiagentsystems

Course Overview

The course can be divided into 6 lectures à 60 minutes:

Lec. 1: Agent Specification
Lec. 2: From Specifications to Implementations
Lec. 3: Formal Verification
Lec. 4: Deductive Verification
Lec. 5: Algorithmic Verification of Models
Lec. 6: Algorithmic Verification of Agents

Reading Material I

- Baier, C. and Katoen, J.-P. (2008).
  Principles of Model Checking.
  The MIT Press.
- Bulling, N., Dix, J., and Jamroga, W. (2010).
  Model checking logics of strategic ability: Complexity.
  In Dastani, M., Hindriks, K. V., and Meyer, J.-J. C., editors,
  Specification and Verification of Multi-Agent Systems. Springer.
- Clarke, E., Grumberg, O., and Peled, D. (1999).
  Model Checking.
  MIT Press.

Reading Material II

- Jürgen Dix and Michael Fisher (2012).
  Chapter 14: Specification and Verification of Multi-agent Systems.
  In G. Weiss (Ed.), Multiagent Systems, MIT Press.

- Fisher, M. (2011).
  An Introduction to Practical Formal Methods Using Temporal Logic.
  Wiley.

Outline

1. Introduction

Why do we need verification methods?

AT&T Telephone Network Outage (1990)

- Problem in New York City: 9 hour outage of large parts of US telephone network.
- Costs: several 100 million $.
- Source: wrong interpretation of a break statement in C.

"...Virtually the entire AT&T network of 4ESS toll tandems switches went in and out of service over and over again on Jan. 15, 1990 .... A software bug was found." [Wikipedia]

The following eight slides are partly based on the book ‘Principles of Model Checking” by Christel Baier and Joost-Pieter Katoen.

Pentium FDIV BUG (1994)

(FDIV: Floating point division unit)

- Incorrect results.
- Costs: 500 million $ and image loss.
- Source:

“. . . Certain floating point division operations performed with these processors would produce incorrect results.” [Wikipedia]

Ariane 5 Desaster (1996)

- Crash of Ariane 5-missle.
- Costs: > 500 million $.
- Source:

"…a data conversion from a 64-bit floating point to 16-bit signed integer value caused a hardware exception…" [Wikipedia]

Ariane 5 Desaster (1996)

- Crash of Ariane 5-missle.
- Costs: > 500 million $.
- Source:

  "... a data conversion from a 64-bit floating point to 16-bit signed integer value caused a hardware exception..." [Wikipedia]

What are the lessons learned?

$\leadsto$ Verification may pay off!

In such cases the extra costs and efforts put into proper verification techniques may be cheaper as the results of an error.

- Software becomes larger.
- Use in safety-critical systems, important domains.
- Increasing need for reliable software.

- Errors can be costly and fatal (Ariane-5 launch, stock market systems,...).
- Mass production of products (errors are expensive, computer chips,...).

- Testing and reviewing ($\rightsquigarrow$ non-formal methods)

- Testing and reviewing ($\rightsquigarrow$ non-formal methods)
- Deductive methods (Hoare Calculus), code integration ($\rightsquigarrow$ undecidable, expertise during programming necessary)

- Testing and reviewing ($\rightsquigarrow$ non-formal methods)
- Deductive methods (Hoare Calculus), code integration ($\rightsquigarrow$ undecidable, expertise during programming necessary)
- Model checking ($\rightsquigarrow$ how is the correct model obtained?)

Model Checking Technique

Errors are expensive: Ariane 5 missile crash,. . .

Model checking provides means to detect such erros!

Problem **(e.g. mobile phone)** **+** **(Safety) Property** **(e.g. deadlock free)**

Model Checking Technique

Errors are expensive: Ariane 5 missile crash,. . .

Model checking provides means to detect such erros!

Problem
(e.g. mobile phone)
+
(Safety) Property
(e.g. deadlock free)

Model Checking Technique

Errors are expensive: Ariane 5 missile crash,. . .

Model checking provides means to detect such erros!

Problem
(e.g. mobile phone)
+
(Safety) Property
(e.g. deadlock free)

Model Checking Technique

Errors are expensive: Ariane 5 missile crash,. . .

Model checking provides means to detect such erros!

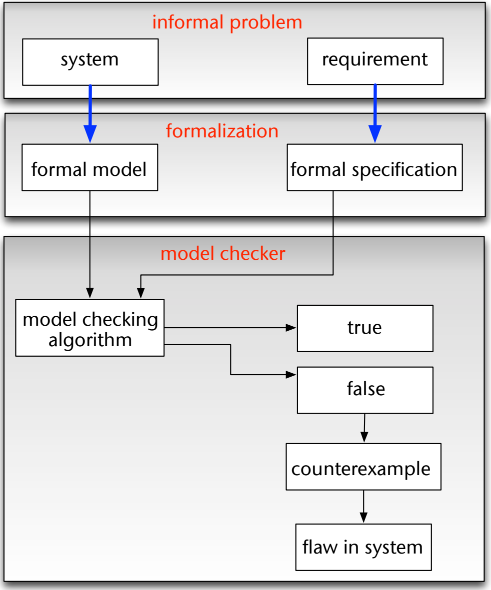
Problem **(e.g. mobile phone)** **+** **(Safety) Property** **(e.g. deadlock free)**

Model Checking Technique

Errors are expensive: Ariane 5 missile crash,...

Model checking provides means to detect such erros!

Problem **(e.g. mobile phone)** **+** **(Safety) Property** **(e.g. deadlock free)**


informal problem
system
requirement
formalization
formal model
formal specification
model checker
model checking algorithm
true
false
counterexample
flaw in system

- Model checking refers to the problem to determine whether a given formula φ is satisfied in a state q of model $\mathcal{M}$.

- Model checking refers to the problem to determine whether a given formula φ is satisfied in a state q of model $\mathcal{M}$.

- Local model checking is the decision problem that determines membership in the set

$$\mathsf{MC}(\mathcal{L}, \mathsf{Struc}, \models) := \{(\mathcal{M}, q, \varphi) \in \mathsf{Struc} \times \mathcal{L} \mid \mathcal{M}, q \models \varphi\},$$

where

  - $\mathcal{L}$ is a logical language,
  - Struc is a class of (pointed) models for $\mathcal{L}$ (i.e. a tuple consisting of a model and a state), and
  - $\models$ is a semantic satisfaction relation compatible with $\mathcal{L}$ and Struc.

- Global model checking: Determine all states in which φ is true.
- Here: The complexities of local and global model checking coincide.
- We are interested in the decidability and the computational complexity of determining whether an input instance $(\mathcal{M}, q, \varphi)$ belongs to $\mathsf{MC}(\dots)$.

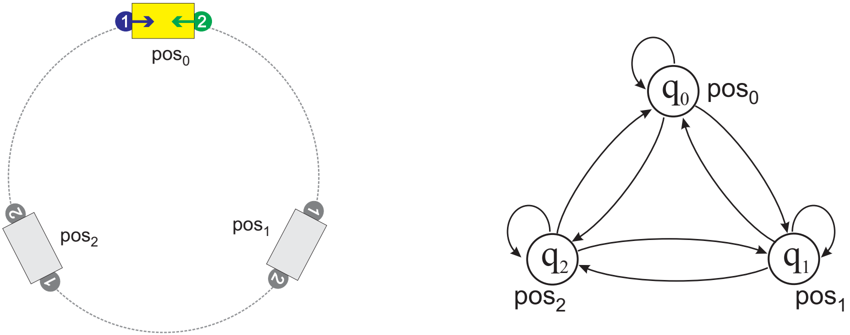


Figure 1 : Two robots and a carriage: a schematic view (left) and a transition system $\mathcal{M}_0$ that models the scenario (right).

Example 1.1 (Robots and Carriage)

Two robots push a carriage from opposite sides (Figure 1). As a result, the carriage can move clockwise or anticlockwise, or it can remain in the same place—depending on who pushes with more force (and, perhaps, who refrains from pushing). We identify 3 different positions of the carriage, and associate them with states q_0, q_1, and q_2. The arrows in transition system $\mathcal{M}_0$ indicate how the state of the system can change in a single step. We label the states with propositions $\mathsf{pos}_0, \mathsf{pos}_1, \mathsf{pos}_2$, to refer to the current position of the carriage.

Definition 1.2 (Kripke Model, Path)

A Kripke model (or unlabelled transition system) is given by $\mathcal{M} = \langle St, \mathcal{R}, \Pi, \pi \rangle$ where St is a nonempty set of states (or possible worlds), $\mathcal{R} \subseteq St \times St$ is a **serial** transition relation on states, Π is a set of atomic propositions, and $\pi : \Pi \to 2^{St}$ is a valuation of propositions. A path λ (or computation) in $\mathcal{M}$ is an infinite sequence of states that can result from subsequent transitions, and refers to a possible course of action. For $q \in St$ we use $\Lambda_{\mathcal{M}}(q)$ to denote the set of all paths of $\mathcal{M}$ starting in q and we define $\Lambda_{\mathcal{M}}$ as $\bigcup_{q \in St} \Lambda_{\mathcal{M}}(q)$. The subscript "$\mathcal{M}$" is often omitted when clear from the context.

1.1 Logics of Agency

Knowledge operators

Modal logics allow us to introduce operators of the form $K_{name}\phi$ meaning the the individual "name" **knows** that ϕ is true. Here are some examples:

$K_{Jürgen}raining$: **Jürgen knows it is raining**

$K_{Jürgen}K_{Jürgen}raining$: **Jürgen knows that he knows it is raining**

$K_{Jürgen}\neg K_{Jürgen}warm$: **Jürgen knows that he doesn't know it is warm**.

$K_{Jürgen}K_{Michael}warm$: **Jürgen knows that Michael knows it is warm**-

Schemata

We can also consider schemata of the form

$$K_{\textit{Jürgen}}\phi \rightarrow K_{\textit{Michael}}\phi$$

for all formulae ϕ. This means that **whatever Jürgen knows, Michael knows and so** ***Michael knows at least as much as Jürgen***.

Temporal operators

Often, temporal dependencies are important and needed in the language besides the knowledge operators.

$\bigcirc K_{\textit{Jürgen}}\, warm$: **in the next moment, Jürgen will know it is warm**

$K_{\textit{Michael}} \Diamond raining$: **Michael knows it will eventually be raining**

Structure of Agent theories

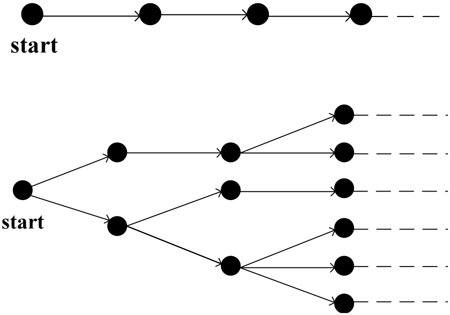
This leads us to a very common structure for agent theories, and so for agent specification languages, comprising

1. a logical dimension describing the underlying dynamic/temporal nature of the agents, for example *dynamic logic* or *temporal logic*,
2. a logical dimension describing the information the agent has, for example a **logic of belief** or **logic of knowledge** (as above), and
3. a logical dimension describing the motivations and agent has, for example a logic of **goals**, **desires**, **wishes**, or **intentions**.

1.2 Temporal Logics

Reasoning about Time

- The **accessibility relation** represents time.
- Time: **linear** vs. **branching**.
- Reasoning about a particular computation of a system.
- Models: paths (e.g. obtained from Kripke structures)



Temporal logic was originally developed in order to represent tense in natural language.

Temporal logic was originally developed in order to represent tense in natural language.

Within Computer Science, it has achieved a significant role in the **formal specification and verification of concurrent and distributed systems**.

Temporal logic was originally developed in order to represent tense in natural language.

Within Computer Science, it has achieved a significant role in the **formal specification and verification of concurrent and distributed systems**.

Much of this popularity has been achieved because a number of useful concepts can be formally, and concisely, specified using temporal logics, e.g.

Temporal logic was originally developed in order to represent tense in natural language.

Within Computer Science, it has achieved a significant role in the **formal specification and verification of concurrent and distributed systems**.

Much of this popularity has been achieved because a number of useful concepts can be formally, and concisely, specified using temporal logics, e.g.

- safety properties

Temporal logic was originally developed in order to represent tense in natural language.

Within Computer Science, it has achieved a significant role in the **formal specification and verification of concurrent and distributed systems**.

Much of this popularity has been achieved because a number of useful concepts can be formally, and concisely, specified using temporal logics, e.g.

- safety properties
- liveness properties

Temporal logic was originally developed in order to represent tense in natural language.

Within Computer Science, it has achieved a significant role in the **formal specification and verification of concurrent and distributed systems**.

Much of this popularity has been achieved because a number of useful concepts can be formally, and concisely, specified using temporal logics, e.g.

- safety properties
- liveness properties
- fairness properties

Typical temporal operators

$\mathbf{X}\varphi$	φ is true in the neXt moment in time
$\mathbf{G}\varphi$	φ is true Globally: in all future moments
$\mathbf{F}\varphi$	φ is true in Finally: eventually (in the future)
$\varphi\mathcal{U}\psi$	φ is true Until at least the moment when ψ becomes true (and this eventually happens)

Typical temporal operators

$\mathbf{X}\varphi$	φ is true in the neXt moment in time
$\mathbf{G}\varphi$	φ is true Globally: in all future moments
$\mathbf{F}\varphi$	φ is true in Finally: eventually (in the future)
$\varphi\mathcal{U}\psi$	φ is true Until at least the moment when ψ becomes true (and this eventually happens)

$$\mathbf{G}((\neg passport \lor \neg ticket) \rightarrow \mathbf{X}\neg board\_flight)$$

$$send(msg, rcvr) \rightarrow \mathbf{F}receive(msg, rcvr)$$

Safety Properties

"something bad will not happen"
"something good will always hold"

Safety Properties

"something bad will not happen"
"something good will always hold"

Typical examples:

Safety Properties

"something bad will not happen"
"something good will always hold"

Typical examples:

$\mathbf{G}\neg bankrupt$

Safety Properties

"something bad will not happen"
"something good will always hold"

Typical examples:

$\mathbf{G}\neg bankrupt$

$\mathbf{G} fuelOK$

Safety Properties

"something bad will not happen"
"something good will always hold"

Typical examples:

$\mathbf{G}\neg bankrupt$

$\mathbf{G}\,fuelOK$

and so on . . .

Safety Properties

“something bad will not happen”
“something good will always hold”

Typical examples:

$\mathbf{G}\neg bankrupt$

$\mathbf{G} fuelOK$

and so on ...

Usually: $\mathbf{G}\neg\ldots.$

Liveness Properties

"something good will happen"

Liveness Properties

"something good will happen"

Typical examples:

Liveness Properties

"something good will happen"

Typical examples:

$\mathbf{F} rich$

Liveness Properties

"something good will happen"

Typical examples:

$\mathbf{F} rich$

power_on $\rightarrow \mathbf{F} online$

Liveness Properties

"something good will happen"

Typical examples:

F$rich$

power_on $\rightarrow$ **F**$online$

and so on ...

Liveness Properties

"something good will happen"

Typical examples:

$\mathbf{F}rich$

power_on $\rightarrow \mathbf{F}online$

and so on ...

Usually: $\mathbf{F}\ldots$

Fairness Properties

Combinations of safety and liveness possible:

Fairness Properties

Combinations of safety and liveness possible:

$$\mathbf{FG}\neg dead$$

$$\mathbf{G}(request\_taxi \rightarrow \mathbf{F}arrive\_taxi)$$

Fairness Properties

Combinations of safety and liveness possible:

$\mathbf{FG}\neg dead$

$\mathbf{G}(request\_taxi \rightarrow \mathbf{F}arrive\_taxi)$ $\rightsquigarrow$ fairness

Strong fairness

"If something is requested then it will be allocated":

$\mathbf{G}(attempt \rightarrow \mathbf{F}success)$,
$\mathbf{GF}attempt \rightarrow \mathbf{GF}success$.

- Scheduling processes, responding to messages, etc.
- No process is blocked forever, etc.

1.3 Sample Specification

Contract net protocol

Consider a simple **contract net** protocol between agents and begin with just the **seller** agent. A naive requirement for this seller might be that the seller will accept the first proposal it receives, e.g.

$$received(offer) \Rightarrow \bigcirc accept(offer).$$

Of course, it may well be that the $offer$ is not acceptable, so

$$(received(offer) \wedge acceptable(offer)) \Rightarrow \bigcirc accept(offer)$$

and, quite possibly, the acceptance will take some time:

$$(received(offer) \wedge acceptable(offer)) \Rightarrow \Diamond accept(offer).$$

Contract net protocol (cont.)

However, this is quite a strong requirement. More likely, we will require the agent accept **one** of the reasonable offers and so, using some additional first-order syntax,

$$
\begin{array}{c}
[\exists O_1 .\ received(O_1)\ \wedge\ acceptable(O_1)] \\
\Rightarrow \\
[\exists O_2 .\ received(O_2)\ \wedge\ acceptable(O_2)\ \wedge\ \Diamond accept(O_2)]\,.
\end{array}
$$

2. Agent Specification

2.1 LTL and variants

Linear-Time Temporal Logic

- Reasoning about a particular computation of a system.
- Time is linear: just one possible future moment!
- Models: paths (e.g. obtained from Kripke structures)

$$\lambda : \mathbb{N}_0 \rightarrow St.$$

Definition 2.1 (Language $\mathcal{L}_{LTL}$ [Pnueli, 1977])

The language $\mathcal{L}_{\mathsf{LTL}}(\mathcal{P}rop)$ is given by all formulae generated by the following grammar, where $\mathsf{p} \in \mathcal{P}rop$ is a proposition:

$$\varphi ::= \mathsf{p} \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \mathcal{U} \varphi \mid \mathbf{X}\varphi.$$

Definition 2.1 (Language $\mathcal{L}_{LTL}$ [Pnueli, 1977])

The language $\mathcal{L}_{\mathrm{LTL}}(\mathcal{P}rop)$ is given by all formulae generated by the following grammar, where $\mathrm{p} \in \mathcal{P}rop$ is a proposition:

$$\varphi ::= \mathrm{p} \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \mathcal{U} \varphi \mid \mathbf{X}\varphi.$$

The additional operators

- **F** (eventually in the future) and
- **G** (always from now on)

can be defined as macros :

Definition 2.1 (Language $\mathcal{L}_{LTL}$ [Pnueli, 1977])

The language $\mathcal{L}_{\mathsf{LTL}}(\mathcal{P}rop)$ is given by all formulae generated by the following grammar, where $\mathsf{p} \in \mathcal{P}rop$ is a proposition:

$$\varphi ::= \mathsf{p} \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi\,\mathcal{U}\,\varphi \mid \mathbf{X}\varphi.$$

The additional operators

- **F** (eventually in the future) and
- **G** (always from now on)

can be defined as macros :

$$\mathsf{F}\varphi \equiv \top\,\mathcal{U}\,\varphi \qquad \text{and}$$

Definition 2.1 (Language $\mathcal{L}_{LTL}$ [Pnueli, 1977])

The language $\mathcal{L}_{\mathsf{LTL}}(\mathcal{P}rop)$ is given by all formulae generated by the following grammar, where $\mathsf{p} \in \mathcal{P}rop$ is a proposition:

$$\varphi ::= \mathsf{p} \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi\,\mathcal{U}\,\varphi \mid \mathbf{X}\varphi.$$

The additional operators

- **F** (eventually in the future) and
- **G** (always from now on)

can be defined as macros :

$$\mathsf{F}\varphi \equiv \top\,\mathcal{U}\,\varphi \qquad \text{and} \qquad \mathsf{G}\varphi \equiv \neg\mathsf{F}\neg\varphi$$

The standard Boolean connectives $\top, \bot, \wedge, \rightarrow$, and $\leftrightarrow$ are defined in their usual way as **macros**.

Models of LTL

The semantics is given over **paths**, which are **infinite sequences of states** from St, and a standard **labelling function** $\pi : St \to 2^{\mathcal{P}rop}$ that determines which propositions are true at which states.

Definition 2.2 (Path $\lambda = q_1 q_2 q_3 \ldots$)

- A path λ over a set of states St is an infinite sequence from St^ω. We also identify it with a mapping $\mathbb{N}_0 \to St$.

Models of LTL

The semantics is given over **paths**, which are **infinite sequences of states** from St, and a standard **labelling function** $\pi : St \to 2^{\mathcal{P}rop}$ that determines which propositions are true at which states.

Definition 2.2 (Path $\lambda = q_1 q_2 q_3 \ldots$)

- A **path** λ over a set of states St is an infinite sequence from St^ω. We also identify it with a mapping $\mathbb{N}_0 \to St$.
- $\lambda[i]$ **denotes the ith position** on path λ (starting from $i = 0$) and
- $\lambda[i, \infty]$ **denotes the subpath of λ starting from i** ($\lambda[i, \infty] = \lambda[i]\lambda[i+1]\ldots$).

$$\lambda = q_1 q_2 q_3 \ldots \in St^\omega$$

Definition 2.3 (Semantics of LTL)

Let λ be a path and π be a labelling function over St. The semantics of **LTL**, $\models^{\mathsf{LTL}}$, is defined as follows:

- $\lambda, \pi \models^{\mathsf{LTL}}$ p iff

$$\lambda = q_1 q_2 q_3 \ldots \in St^\omega$$

Definition 2.3 (Semantics of LTL)

Let λ be a path and π be a labelling function over St. The semantics of **LTL**, $\models^{\mathsf{LTL}}$, is defined as follows:

- $\lambda, \pi \models^{\mathsf{LTL}}$ p iff p $\in \pi(\lambda[0])$ and p $\in \mathcal{P}rop$;
- $\lambda, \pi \models^{\mathsf{LTL}} \neg\varphi$ iff

$$\lambda = q_1 q_2 q_3 \ldots \in St^\omega$$

Definition 2.3 (Semantics of LTL)

Let λ be a path and π be a labelling function over St. The semantics of **LTL**, $\models^{\text{LTL}}$, is defined as follows:

- $\lambda, \pi \models^{\text{LTL}}$ p iff p $\in \pi(\lambda[0])$ and p $\in \mathcal{P}rop$;
- $\lambda, \pi \models^{\text{LTL}} \neg\varphi$ iff **not** $\lambda, \pi \models^{\text{LTL}} \varphi$ (we will also write $\lambda, \pi \not\models^{\text{LTL}} \varphi$);
- $\lambda, \pi \models^{\text{LTL}} \varphi \vee \psi$ iff

$$\lambda = q_1 q_2 q_3 \ldots \in St^\omega$$

Definition 2.3 (Semantics of LTL)

Let λ be a path and π be a labelling function over St. The semantics of **LTL**, $\models^{\mathsf{LTL}}$, is defined as follows:

- $\lambda, \pi \models^{\mathsf{LTL}} \mathsf{p}$ iff $\mathsf{p} \in \pi(\lambda[0])$ and $\mathsf{p} \in \mathcal{P}rop$;
- $\lambda, \pi \models^{\mathsf{LTL}} \neg\varphi$ iff **not** $\lambda, \pi \models^{\mathsf{LTL}} \varphi$ (we will also write $\lambda, \pi \not\models^{\mathsf{LTL}} \varphi$);
- $\lambda, \pi \models^{\mathsf{LTL}} \varphi \vee \psi$ iff $\lambda, \pi \models^{\mathsf{LTL}} \varphi$ **or** $\lambda, \pi \models^{\mathsf{LTL}} \psi$;

$$\lambda = q_1 q_2 q_3 \ldots \in St^{\omega}$$

Definition 2.3 (Semantics of LTL)

Let λ be a path and π be a labelling function over St. The semantics of **LTL**, $\models^{\mathsf{LTL}}$, is defined as follows:

- $\lambda, \pi \models^{\mathsf{LTL}} \mathsf{p}$ iff $\mathsf{p} \in \pi(\lambda[0])$ and $\mathsf{p} \in \mathcal{P}rop$;
- $\lambda, \pi \models^{\mathsf{LTL}} \neg\varphi$ iff **not** $\lambda, \pi \models^{\mathsf{LTL}} \varphi$ (we will also write $\lambda, \pi \not\models^{\mathsf{LTL}} \varphi$);
- $\lambda, \pi \models^{\mathsf{LTL}} \varphi \vee \psi$ iff $\lambda, \pi \models^{\mathsf{LTL}} \varphi$ **or** $\lambda, \pi \models^{\mathsf{LTL}} \psi$;
- $\lambda, \pi \models^{\mathsf{LTL}} \mathbf{X}\varphi$ iff

$$\lambda = q_1 q_2 q_3 \ldots \in St^\omega$$

Definition 2.3 (Semantics of LTL)

Let λ be a path and π be a labelling function over St. The semantics of **LTL**, $\models^{\mathsf{LTL}}$, is defined as follows:

- $\lambda, \pi \models^{\mathsf{LTL}} \mathsf{p}$ iff $\mathsf{p} \in \pi(\lambda[0])$ and $\mathsf{p} \in \mathcal{P}rop$;
- $\lambda, \pi \models^{\mathsf{LTL}} \neg\varphi$ iff **not** $\lambda, \pi \models^{\mathsf{LTL}} \varphi$ (we will also write $\lambda, \pi \not\models^{\mathsf{LTL}} \varphi$);
- $\lambda, \pi \models^{\mathsf{LTL}} \varphi \vee \psi$ iff $\lambda, \pi \models^{\mathsf{LTL}} \varphi$ **or** $\lambda, \pi \models^{\mathsf{LTL}} \psi$;
- $\lambda, \pi \models^{\mathsf{LTL}} \mathbf{X}\varphi$ iff $\lambda[1, \infty], \pi \models^{\mathsf{LTL}} \varphi$; and
- $\lambda, \pi \models^{\mathsf{LTL}} \varphi \mathcal{U} \psi$ iff

$$\lambda = q_1 q_2 q_3 \ldots \in St^\omega$$

Definition 2.3 (Semantics of LTL)

Let λ be a path and π be a labelling function over St. The semantics of **LTL**, $\models^{\textsf{LTL}}$, is defined as follows:

- $\lambda, \pi \models^{\textsf{LTL}}$ p iff $p \in \pi(\lambda[0])$ and p $\in \mathcal{P}rop$;
- $\lambda, \pi \models^{\textsf{LTL}} \neg\varphi$ iff **not** $\lambda, \pi \models^{\textsf{LTL}} \varphi$ (we will also write $\lambda, \pi \not\models^{\textsf{LTL}} \varphi$);
- $\lambda, \pi \models^{\textsf{LTL}} \varphi \vee \psi$ iff $\lambda, \pi \models^{\textsf{LTL}} \varphi$ **or** $\lambda, \pi \models^{\textsf{LTL}} \psi$;
- $\lambda, \pi \models^{\textsf{LTL}} \mathbf{X}\varphi$ iff $\lambda[1, \infty], \pi \models^{\textsf{LTL}} \varphi$; and
- $\lambda, \pi \models^{\textsf{LTL}} \varphi \mathcal{U} \psi$ iff there is an $i \in \mathbb{N}_0$ such that $\lambda[i, \infty], \pi \models \psi$ and

$$\lambda = q_1 q_2 q_3 \ldots \in St^\omega$$

Definition 2.3 (Semantics of LTL)

Let λ be a path and π be a labelling function over St. The semantics of **LTL**, $\models^{\mathsf{LTL}}$, is defined as follows:

- $\lambda, \pi \models^{\mathsf{LTL}} \mathsf{p}$ iff $\mathsf{p} \in \pi(\lambda[0])$ and $\mathsf{p} \in \mathcal{P}rop$;
- $\lambda, \pi \models^{\mathsf{LTL}} \neg\varphi$ iff **not** $\lambda, \pi \models^{\mathsf{LTL}} \varphi$ (we will also write $\lambda, \pi \not\models^{\mathsf{LTL}} \varphi$);
- $\lambda, \pi \models^{\mathsf{LTL}} \varphi \vee \psi$ iff $\lambda, \pi \models^{\mathsf{LTL}} \varphi$ **or** $\lambda, \pi \models^{\mathsf{LTL}} \psi$;
- $\lambda, \pi \models^{\mathsf{LTL}} \mathbf{X}\varphi$ iff $\lambda[1, \infty], \pi \models^{\mathsf{LTL}} \varphi$; and
- $\lambda, \pi \models^{\mathsf{LTL}} \varphi\,\mathcal{U}\,\psi$ iff there is an $i \in \mathbb{N}_0$ such that $\lambda[i, \infty], \pi \models \psi$ and $\lambda[j, \infty], \pi \models^{\mathsf{LTL}} \varphi$ for all $0 \leq j < i$.

Other temporal operators

$\lambda, \pi \models \mathbf{F}\varphi$ iff

Other temporal operators

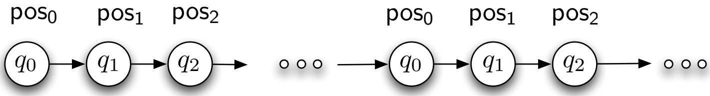
$\lambda, \pi \models \mathbf{F}\varphi$ iff $\lambda[i, \infty], \pi \models \varphi$ for some $i \in \mathbb{N}_0$;
$\lambda, \pi \models \mathbf{G}\varphi$ iff

Other temporal operators

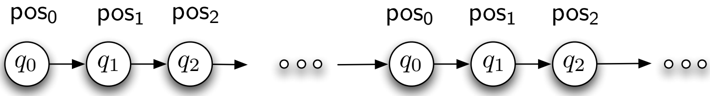
$\lambda, \pi \models \mathbf{F}\varphi$ iff $\lambda[i, \infty], \pi \models \varphi$ for some $i \in \mathbb{N}_0$;
$\lambda, \pi \models \mathbf{G}\varphi$ iff $\lambda[i, \infty], \pi \models \varphi$ for all $i \in \mathbb{N}_0$;

Exercise

Prove that the semantics does indeed match the definitions $\mathbf{F}\varphi \equiv \top \mathcal{U} \varphi$ and $\mathbf{G}\varphi \equiv \neg\mathbf{F}\neg\varphi$.

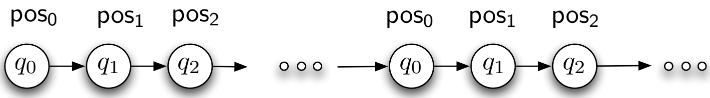


$$\lambda, \pi \models \mathbf{F}\mathsf{pos}_1$$



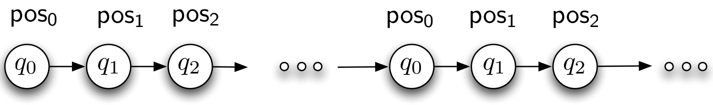
$$\lambda, \pi \models \mathbf{F}\mathsf{pos}_1$$

$$\lambda' = \lambda[1, \infty], \pi \models \mathsf{pos}_1$$

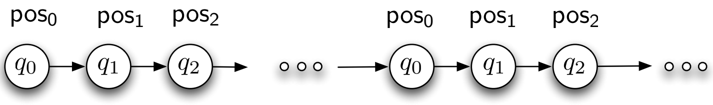


$$\lambda, \pi \models \mathbf{F}\mathsf{pos}_1$$

$$\lambda' = \lambda[1, \infty], \pi \models \mathsf{pos}_1$$
$$\mathsf{pos}_1 \in \pi(\lambda'[0])$$

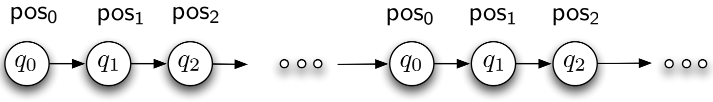


$\lambda, \pi \models \mathbf{GF}\mathsf{pos}_1$ iff



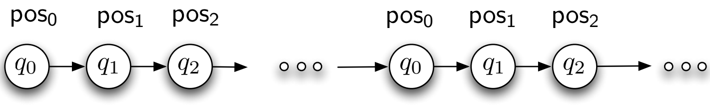
$\lambda, \pi \models$ **G**Fpos_1 iff

$\lambda[0, \infty], \pi \models$ **F**pos_1 and



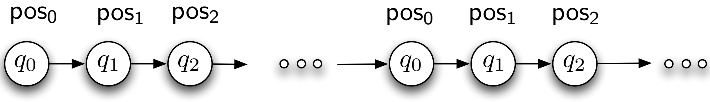
$\lambda, \pi \models \mathbf{GF}\mathsf{pos}_1$ iff

$\lambda[0,\infty], \pi \models \mathbf{F}\mathsf{pos}_1$ and



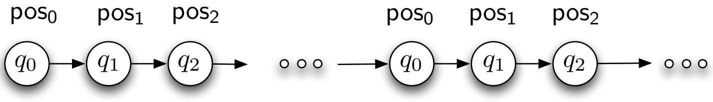
$\lambda, \pi \models \mathbf{GF}\mathsf{pos}_1$ iff

$\lambda[0,\infty], \pi \models \mathbf{F}\mathsf{pos}_1$ and
$\lambda[1,\infty], \pi \models \mathbf{F}\mathsf{pos}_1$ and



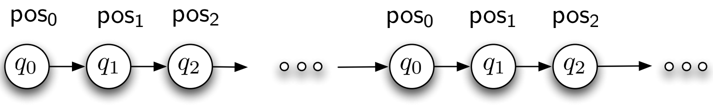
$$\lambda, \pi \models \mathbf{GF}\mathsf{pos}_1 \text{ iff}$$

$$\lambda[0,\infty], \pi \models \mathbf{F}\mathsf{pos}_1 \text{ and}$$
$$\lambda[1,\infty], \pi \models \mathbf{F}\mathsf{pos}_1 \text{ and}$$



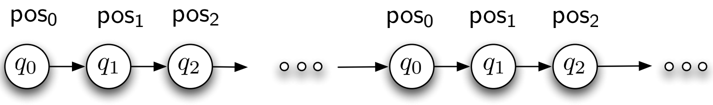
$\lambda, \pi \models \mathbf{GF}\mathsf{pos}_1$ iff

$\lambda[0, \infty], \pi \models \mathbf{F}\mathsf{pos}_1$ and
$\lambda[1, \infty], \pi \models \mathbf{F}\mathsf{pos}_1$ and
$\lambda[2, \infty], \pi \models \mathbf{F}\mathsf{pos}_1$ and



$\lambda, \pi \models$ **GF**pos_1 iff

$\lambda[0,\infty], \pi \models$ **F**pos_1 and
$\lambda[1,\infty], \pi \models$ **F**pos_1 and
$\lambda[2,\infty], \pi \models$ **F**pos_1 and



$$\lambda, \pi \models \mathbf{GF}\mathsf{pos}_1 \text{ iff}$$

$$\lambda[0,\infty], \pi \models \mathbf{F}\mathsf{pos}_1 \text{ and}$$
$$\lambda[1,\infty], \pi \models \mathbf{F}\mathsf{pos}_1 \text{ and}$$
$$\lambda[2,\infty], \pi \models \mathbf{F}\mathsf{pos}_1 \text{ and}$$
$$\ldots$$

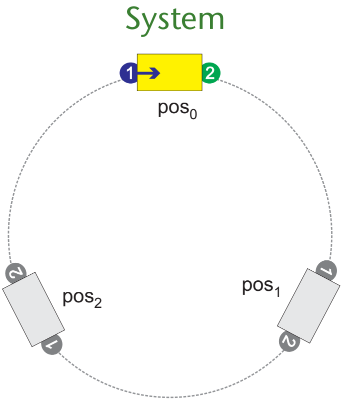
Representation of paths

- Paths are infinite entities.
- They are theoretical constructs.
- We need a finite representation!

Representation of paths

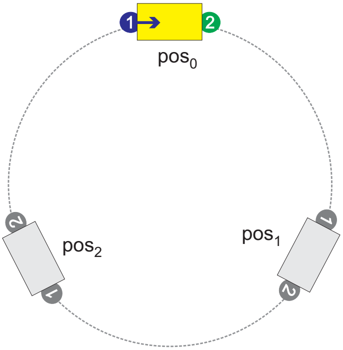
- Paths are infinite entities.
- They are theoretical constructs.
- We need a finite representation!
- Such a finite representation is given by a transition system or a pointed Kripke structure.

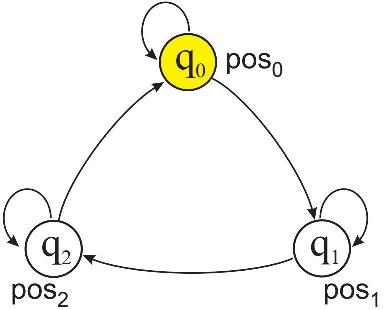
Computational vs. behavioral structure



Computational vs. behavioral structure

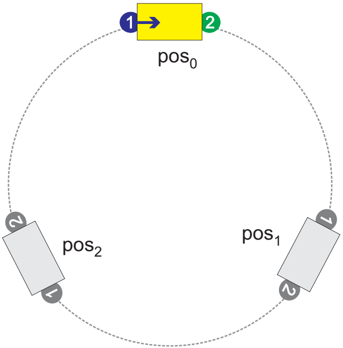
System



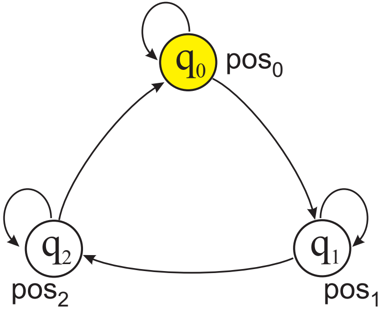


Computational vs. behavioral structure

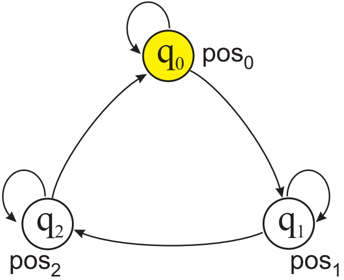
System



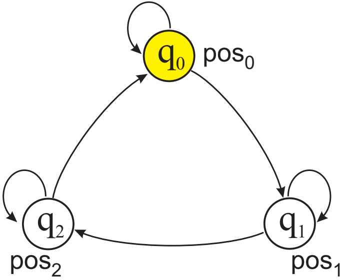
Computational str.



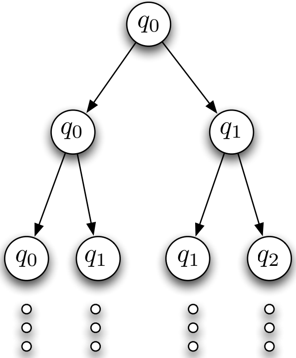
Computational str.



Computational str.



Behavioral str.



Important!

The behavioral structure is usually infinite! Here, it is an infinite tree. We say it is the q_0-unfolding of the model.

Some Exercises

Example 2.4

Formalise the following as **LTL** formulae:

1. r should never occur.
2. r should occur exactly once.
3. At least once r should directly be followed by s.
4. r is true at **exactly** all even states.
5. r is true at each even state (the odd states do not matter). Does $r \wedge \mathbf{G}(r \wedge \mathbf{XX}r)$ work?

Some Exercises

Example 2.4

Formalise the following as **LTL** formulae:

1. r should never occur.

$$\mathbf{G}\neg r$$

2. r should occur exactly once.

3. At least once r should directly be followed by s.

4. r is true at **exactly** all even states.

5. r is true at each even state (the odd states do not matter). Does $r \wedge \mathbf{G}(r \wedge \mathbf{XX}r)$ work?

Some Exercises

Example 2.4

Formalise the following as **LTL** formulae:

1. r should never occur.

$$\mathbf{G}\neg r$$

2. r should occur exactly once.

$$(\neg r)\,\mathcal{U}\,(r \wedge \mathbf{XG}\neg r)$$

3. At least once r should directly be followed by s.

4. r is true at **exactly** all even states.

5. r is true at each even state (the odd states do not matter). Does $r \wedge \mathbf{G}(r \wedge \mathbf{XX}r)$ work?

Some Exercises

Example 2.4

Formalise the following as **LTL** formulae:

1. r should never occur.

$$\mathbf{G}\neg r$$

2. r should occur exactly once.

$$(\neg r)\,\mathcal{U}\,(r \wedge \mathbf{XG}\neg r)$$

3. At least once r should directly be followed by s.

$$\mathbf{F}(r \wedge \mathbf{X}s)$$

4. r is true at **exactly** all even states.
5. r is true at each even state (the odd states do not matter). Does $r \wedge \mathbf{G}(r \wedge \mathbf{XX}r)$ work?

Some Exercises

Example 2.4

Formalise the following as **LTL** formulae:

1. r should never occur.

$$\mathbf{G}\neg r$$

2. r should occur exactly once.

$$(\neg r)\,\mathcal{U}\,(r \wedge \mathbf{XG}\neg r)$$

3. At least once r should directly be followed by s.

$$\mathbf{F}(r \wedge \mathbf{X}s)$$

4. r is true at **exactly** all even states. $r \wedge \mathbf{G}(r \leftrightarrow \neg\mathbf{X}r)$

5. r is true at each even state (the odd states do not matter). Does $r \wedge \mathbf{G}(r \wedge \mathbf{XX}r)$ work?

Some Exercises

Example 2.4

Formalise the following as **LTL** formulae:

1. r should never occur.

$$\mathbf{G}\neg r$$

2. r should occur exactly once.

$$(\neg r)\,\mathcal{U}\,(r \wedge \mathbf{XG}\neg r)$$

3. At least once r should directly be followed by s.

$$\mathbf{F}(r \wedge \mathbf{X}s)$$

4. r is true at **exactly** all even states. $r \wedge \mathbf{G}(r \leftrightarrow \neg\mathbf{X}r)$
5. r is true at each even state (the odd states do not matter). Does $r \wedge \mathbf{G}(r \wedge \mathbf{XX}r)$ work? No. This is not expressible.

Relation to first-order logic (1)

1. The monadic first-order theory of (linear) order, **FO**$(\leq)$ is equivalent to **LTL**.

2. There is a translation from sentences of **LTL** to sentences of **FO**$(\leq)$ and vice versa, such that the **LTL** sentence is true in λ, π iff its **translation is true** in the associated first-order structure.

Relation to first-order logic (2)

1. More precisely: an infinite **path** λ is described as a first-order structure with **domain** $\mathbb{N}$ and predicates P_{p} for $\mathsf{p} \in \mathcal{P}rop$. The predicates stand for the set of timepoints where p is true. So each path λ can be represented as a structure
$\mathcal{N}_\lambda = \langle \mathbb{N}, \leq^{\mathbb{N}}, P_1^{\mathcal{N}}, P_2^{\mathcal{N}}, \ldots P_n^{\mathcal{N}} \rangle$.

   Then each **LTL** formula ϕ translates to a first-order formula $\alpha_\phi(x)$ with one free variable s.t.

   $$\phi \text{ is true in } \lambda[n,\infty] \text{ iff } \alpha_\phi(n) \text{ is true in } \mathcal{N}_\lambda.$$

   And conversely: for each first-order formula with a free variable there is a corresponding **LTL** formula s.t. the same condition holds.

The formulae GFp, FGp

1. What are their counterparts in **FO**$(\leq)$?
2. We will see later that **FG**p does not belong to **CTL**, but to **CTL***. It is not even equivalent to a **CTL** formula.
3. However, **GF**p is equivalent to a **CTL** formula: A**G**A**F**p

Some Remarks

1. A particular logic **LTL** is determined by the number n of propositional variables. Strictly speaking, this number should be a parameter of the logic. This also applies to the logics **CTL** and **ATL**.
2. While both **F** and **G** can be expressed using $\mathcal{U}$, the converse is not true: $\mathcal{U}$ can not be expressed by **F** and **G**.

Satisfiability of LTL formulae

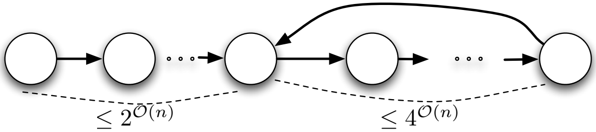
A formula is satisfiable, if there is a path where it is true. Can we **restrict the structure** of such paths? I.e. can we restrict to simple paths, for example paths that are periodic?

- If this is the case, then we might be able to **construct counterexamples** more easily, as we need only check very specific paths.
- It would be also useful to know **how long the period is** and **within which initial segment** of the path it starts, depending on the length of the formula φ.

Satisfiability of LTL formulae (cont.)

Theorem 2.5 (Periodic model theorem [Sistla and Clarke, 1985])

A formula $\varphi \in \mathcal{L}_{LTL}$ is satisfiable *iff there is a path λ which is* ultimately periodic, *and the period starts within $2^{1+|\varphi|}$ steps and has a length which is $\leq 4^{1+|\varphi|}$.*



2.2 CTL and Variants

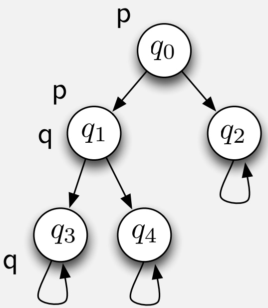
Branching Time

- **CTL**, **CTL***: Computation Tree Logics.
- Reasoning about possible computations of a system.
- Time is branching: We want all possible computations included!
- Models: states (time points, situations), transitions (changes). ($\rightsquigarrow$ Kripke models).
- Paths: courses of action, computations. ($\rightsquigarrow$ **LTL**)

- Path quantifiers: A (for all paths), E (there is a path);
- Temporal operators: **X** (nexttime), **F** (finally), **G** (globally) and $\mathcal{U}$ (until);

- Path quantifiers: A (for all paths), E (there is a path);
- Temporal operators: **X** (nexttime), **F** (finally), **G** (globally) and $\mathcal{U}$ (until);
- **CTL**: each temporal operator must be immediately preceded by exactly one path quantifier;
- **CTL***: no syntactic restrictions.

Example 2.6 (Branching Time)



In this structure, whenever p holds at some timepoint, then there is a path where q holds in the next step and there is (another) path where ¬q holds in the next step. And this holds along all paths (there are three infinite paths).

Definition 2.7 ($\mathcal{L}_{CTL^*}$ [Emerson and Halpern, 1986])

The language $\mathcal{L}_{\mathsf{CTL}^*}(\mathcal{P}rop)$ is given by all formulae generated by the following grammar:

$$\varphi ::= \mathsf{p} \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{E}\gamma$$

where

$$\gamma ::= \varphi \mid \neg\gamma \mid \gamma \vee \gamma \mid \gamma\,\mathcal{U}\,\gamma \mid \mathbf{X}\gamma$$

and $\mathsf{p} \in \mathcal{P}rop$. Formulae φ (resp. γ) are called state (resp. path) formulae.

We use the same abbreviations as for $\mathcal{L}_{LTL}$:

$\lambda, \pi \models \mathbf{F}\varphi$ iff

Definition 2.7 ($\mathcal{L}_{CTL^*}$ [Emerson and Halpern, 1986])

The language $\mathcal{L}_{CTL^*}(\mathcal{P}rop)$ is given by all formulae generated by the following grammar:

$$\varphi ::= \mathsf{p} \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{E}\gamma$$

where

$$\gamma ::= \varphi \mid \neg\gamma \mid \gamma \vee \gamma \mid \gamma\,\mathcal{U}\,\gamma \mid \mathbf{X}\gamma$$

and $\mathsf{p} \in \mathcal{P}rop$. Formulae φ (resp. γ) are called state (resp. path) formulae.

We use the same abbreviations as for $\mathcal{L}_{LTL}$:

$\lambda, \pi \models \mathbf{F}\varphi$ iff $\lambda[i,\infty], \pi \models \varphi$ for some $i \in \mathbb{N}_0$;
$\lambda, \pi \models \mathbf{G}\varphi$ iff

Definition 2.7 ($\mathcal{L}_{CTL^*}$ [Emerson and Halpern, 1986])

The language $\mathcal{L}_{CTL^*}(\mathcal{P}rop)$ is given by all formulae generated by the following grammar:

$$\varphi ::= \mathsf{p} \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{E}\gamma$$

where

$$\gamma ::= \varphi \mid \neg\gamma \mid \gamma \vee \gamma \mid \gamma\,\mathcal{U}\,\gamma \mid \mathbf{X}\gamma$$

and $\mathsf{p} \in \mathcal{P}rop$. Formulae φ (resp. γ) are called state (resp. path) formulae.

We use the same abbreviations as for $\mathcal{L}_{LTL}$:

$\lambda, \pi \models \mathbf{F}\varphi$ iff $\lambda[i,\infty], \pi \models \varphi$ for some $i \in \mathbb{N}_0$;
$\lambda, \pi \models \mathbf{G}\varphi$ iff $\lambda[i,\infty], \pi \models \varphi$ for all $i \in \mathbb{N}_0$;

- The $\mathcal{L}_{CTL^*}$-formula $\mathbf{EF}\varphi$, for instance, ensures that

- The $\mathcal{L}_{CTL^*}$-formula $\mathbf{EF}\varphi$, for instance, ensures that there is at least one path on which φ holds at some (future) time moment.

- The $\mathcal{L}_{CTL^*}$-formula $\mathsf{E}\mathbf{F}\varphi$, for instance, ensures that there is at least one path on which φ holds at some (future) time moment.
- The formula $\mathsf{A}\mathbf{FG}\varphi$ states that

- The $\mathcal{L}_{CTL^*}$-formula $\mathsf{E}\mathbf{F}\varphi$, for instance, ensures that there is at least one path on which φ holds at some (future) time moment.
- The formula $\mathsf{A}\mathbf{FG}\varphi$ states that φ holds almost everywhere . More precisely, on all paths it always holds from some future time moment.

- The $\mathcal{L}_{CTL^*}$-formula $\mathsf{E}\mathbf{F}\varphi$, for instance, ensures that there is at least one path on which φ holds at some (future) time moment.
- The formula $\mathsf{A}\mathbf{F}\mathbf{G}\varphi$ states that φ holds almost everywhere . More precisely, on all paths it always holds from some future time moment.
- $\mathcal{L}_{CTL^*}$-formulae do not only talk about temporal patterns on a given path, they also quantify (existentially or universally) over such paths.

- The $\mathcal{L}_{CTL^*}$-formula $\mathsf{EF}\varphi$, for instance, ensures that there is at least one path on which φ holds at some (future) time moment.
- The formula $\mathsf{AFG}\varphi$ states that φ holds almost everywhere . More precisely, on all paths it always holds from some future time moment.
- $\mathcal{L}_{CTL^*}$-formulae do not only talk about temporal patterns on a given path, they also quantify (existentially or universally) over such paths.
- The logic is complex! For practical purposes, a fragment with better computational properties is often sufficient.

Definition 2.8 ($\mathcal{L}_{CTL}$ [Clarke and Emerson, 1981])

The language $\mathcal{L}_{\mathsf{CTL}}(\mathcal{P}rop)$ is given by all formulae generated by the following grammar, where $\mathsf{p} \in \mathcal{P}rop$ is a proposition:

$$\varphi ::= \mathsf{p} \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{E}(\varphi\,\mathcal{U}\,\varphi) \mid \mathsf{E}\mathbf{X}\varphi \mid \mathsf{E}\mathbf{G}\varphi.$$

We introduce the following macros:

- $\mathbf{F}\varphi \equiv$,
- $\mathsf{A}\mathbf{X}\varphi \equiv$,
- $\mathsf{A}\mathbf{G}\varphi \equiv$, and
- $\mathsf{A}\varphi\,\mathcal{U}\,\psi \equiv$

Definition 2.8 ($\mathcal{L}_{CTL}$ [Clarke and Emerson, 1981])

The language $\mathcal{L}_{\mathsf{CTL}}(\mathcal{P}rop)$ is given by all formulae generated by the following grammar, where $\mathsf{p} \in \mathcal{P}rop$ is a proposition:

$$\varphi ::= \mathsf{p} \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathsf{E}(\varphi \,\mathcal{U}\, \varphi) \mid \mathsf{E}\mathbf{X}\varphi \mid \mathsf{E}\mathbf{G}\varphi.$$

We introduce the following macros:

- $\mathbf{F}\varphi \equiv$,
- $\mathsf{A}\mathbf{X}\varphi \equiv$,
- $\mathsf{A}\mathbf{G}\varphi \equiv$, and
- $\mathsf{A}\varphi\,\mathcal{U}\,\psi \equiv$ Exercise!

Definition 2.8 ($\mathcal{L}_{CTL}$ [Clarke and Emerson, 1981])

The language $\mathcal{L}_{CTL}(\mathcal{P}rop)$ is given by all formulae generated by the following grammar, where $p \in \mathcal{P}rop$ is a proposition:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid E(\varphi\,\mathcal{U}\,\varphi) \mid E\mathbf{X}\varphi \mid E\mathbf{G}\varphi.$$

We introduce the following macros:

- $\mathbf{F}\varphi \equiv \top\,\mathcal{U}\,\varphi$,
- $A\mathbf{X}\varphi \equiv$,
- $A\mathbf{G}\varphi \equiv$, and
- $A\varphi\,\mathcal{U}\,\psi \equiv$ Exercise!

Definition 2.8 ($\mathcal{L}_{CTL}$ [Clarke and Emerson, 1981])

The language $\mathcal{L}_{\mathrm{CTL}}(\mathcal{P}rop)$ is given by all formulae generated by the following grammar, where $\mathrm{p} \in \mathcal{P}rop$ is a proposition:

$$\varphi ::= \mathrm{p} \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathrm{E}(\varphi\,\mathcal{U}\,\varphi) \mid \mathrm{E}\mathbf{X}\varphi \mid \mathrm{E}\mathbf{G}\varphi.$$

We introduce the following macros:

- $\mathbf{F}\varphi \equiv \top\,\mathcal{U}\,\varphi$,
- $\mathrm{A}\mathbf{X}\varphi \equiv \neg\mathrm{E}\mathbf{X}\neg\varphi$,
- $\mathrm{A}\mathbf{G}\varphi \equiv$, and
- $\mathrm{A}\varphi\,\mathcal{U}\,\psi \equiv$ Exercise!

Definition 2.8 ($\mathcal{L}_{CTL}$ [Clarke and Emerson, 1981])

The language $\mathcal{L}_{CTL}(\mathcal{P}rop)$ is given by all formulae generated by the following grammar, where $p \in \mathcal{P}rop$ is a proposition:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid E(\varphi\,\mathcal{U}\,\varphi) \mid E\mathbf{X}\varphi \mid E\mathbf{G}\varphi.$$

We introduce the following macros:

- $\mathbf{F}\varphi \equiv \top\,\mathcal{U}\,\varphi$,
- $A\mathbf{X}\varphi \equiv \neg E\mathbf{X}\neg\varphi$,
- $A\mathbf{G}\varphi \equiv \neg E\mathbf{F}\neg\varphi$, and
- $A\varphi\,\mathcal{U}\,\psi \equiv$ Exercise!

Definition 2.8 ($\mathcal{L}_{CTL}$ [Clarke and Emerson, 1981])

The language $\mathcal{L}_{\mathrm{CTL}}(\mathcal{P}rop)$ is given by all formulae generated by the following grammar, where $\mathrm{p} \in \mathcal{P}rop$ is a proposition:

$$\varphi ::= \mathrm{p} \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathrm{E}(\varphi\,\mathcal{U}\,\varphi) \mid \mathrm{E}\mathbf{X}\varphi \mid \mathrm{E}\mathbf{G}\varphi.$$

We introduce the following macros:

- $\mathbf{F}\varphi \equiv \top\,\mathcal{U}\,\varphi$,
- $\mathrm{A}\mathbf{X}\varphi \equiv \neg\mathrm{E}\mathbf{X}\neg\varphi$,
- $\mathrm{A}\mathbf{G}\varphi \equiv \neg\mathrm{E}\mathbf{F}\neg\varphi$, and
- $\mathrm{A}\varphi\,\mathcal{U}\,\psi \equiv \ \ldots$ Exercise!

For example, A**G**E**X**p is a $\mathcal{L}_{CTL}$-formula whereas A**GF**p is not.

Example 2.9 (CTL* or CTL?)

Are the following **CTL*** or **CTL** formulae? What do they express?

1. E**F**A**X**shutdown

For example, A**G**E**X**p is a $\mathcal{L}_{CTL}$-formula whereas A**GF**p is not.

Example 2.9 (CTL* or CTL?)

Are the following **CTL*** or **CTL** formulae? What do they express?

1. E**F**A**X**shutdown
2. E**FX**shutdown

For example, A**G**E**X**p is a $\mathcal{L}_{CTL}$-formula whereas A**GF**p is not.

Example 2.9 (CTL* or CTL?)

Are the following **CTL*** or **CTL** formulae? What do they express?

1. E**F**A**X**shutdown
2. E**FX**shutdown
3. A**GF**rain

For example, A**G**E**X**p is a $\mathcal{L}_{CTL}$-formula whereas A**GF**p is not.

Example 2.9 (CTL* or CTL?)

Are the following **CTL*** or **CTL** formulae? What do they express?

1. E**F**A**X**shutdown
2. E**FX**shutdown
3. A**GF**rain
4. A**G**A**F**rain (Is it different from (3)?)

For example, A**G**E**X**p is a $\mathcal{L}_{CTL}$-formula whereas A**GF**p is not.

Example 2.9 (CTL* or CTL?)

Are the following **CTL*** or **CTL** formulae? What do they express?

1. E**F**A**X**shutdown
2. E**FX**shutdown
3. A**GF**rain
4. A**G**A**F**rain (Is it different from (3)?)
5. E**FG**broken

For example, $\mathsf{A}\mathbf{G}\mathsf{E}\mathbf{X}p$ is a $\mathcal{L}_{CTL}$-formula whereas $\mathsf{A}\mathbf{G}\mathbf{F}p$ is not.

Example 2.9 (CTL* or CTL?)

Are the following **CTL*** or **CTL** formulae? What do they express?

1. $\mathsf{E}\mathbf{F}\mathsf{A}\mathbf{X}shutdown$
2. $\mathsf{E}\mathbf{F}\mathbf{X}shutdown$
3. $\mathsf{A}\mathbf{G}\mathbf{F}rain$
4. $\mathsf{A}\mathbf{G}\mathsf{A}\mathbf{F}rain$ (Is it different from (3)?)
5. $\mathsf{E}\mathbf{F}\mathbf{G}broken$
6. $\mathsf{A}\mathbf{G}(p \rightarrow (\mathsf{E}\mathbf{X}q \wedge \mathsf{E}\mathbf{X}\neg q))$

The precise definition of Kripke structures is given in Section 4. To understand the following definitions it suffices to note that:

- Given a set of states St (each is a propositional model), a Kripke model $\mathcal{M}$ is simply a tuple $(St, \mathcal{R})$ where $\mathcal{R} \subseteq St \times St$ is a binary relation.
- $q_1 \mathcal{R} q_2$ (also written $(q_1, q_2) \in \mathcal{R}$ or $\mathcal{R}(q_1, q_2)$) means that state q_2 is reachable from state q_1 (by executing certain actions).
- The relation $\mathcal{R}$ is serial: for all q there is a q' such that $q\mathcal{R}q'$. This ensures that our paths are infinite.
- Given a state q in a Kripke model, by $\Lambda(q)$ we mean the set of all paths determined by the relation $\mathcal{R}$ starting in q: $q, q_1, q_2, \ldots, q_i, \ldots$ where $q\mathcal{R}q_1$, $\ldots q_i \mathcal{R} q_{i+1}, \ldots$

Definition 2.10 (Semantics $\models^{\mathsf{CTL}^*}$)

Let $\mathcal{M}$ be a Kripke model, $q \in St$ and $\lambda \in \Lambda$. The **semantics of $\mathcal{L}_{\mathsf{CTL}^*}$- and $\mathcal{L}_{\mathsf{CTL}}$-formulae** is given by the satisfaction relation $\models^{\mathsf{CTL}^*}$ for **state formulae** by

- $\mathcal{M}, q \models^{\mathsf{CTL}^*} \mathsf{p}$ iff $\lambda[0] \in \pi(\mathsf{p})$ and $\mathsf{p} \in \mathcal{P}rop$;

Definition 2.10 (Semantics $\models^{\mathsf{CTL}^*}$)

Let $\mathcal{M}$ be a Kripke model, $q \in St$ and $\lambda \in \Lambda$. The semantics of $\mathcal{L}_{\mathsf{CTL}^*}$- and $\mathcal{L}_{\mathsf{CTL}}$-formulae is given by the satisfaction relation $\models^{\mathsf{CTL}^*}$ for state formulae by

- $\mathcal{M}, q \models^{\mathsf{CTL}^*} \mathsf{p}$ iff $\lambda[0] \in \pi(\mathsf{p})$ and $\mathsf{p} \in \mathcal{P}rop$;
- $\mathcal{M}, q \models^{\mathsf{CTL}^*} \neg\varphi$ iff $\mathcal{M}, q \not\models^{\mathsf{CTL}^*} \varphi$;

Definition 2.10 (Semantics $\models^{\mathsf{CTL}^*}$)

Let $\mathcal{M}$ be a Kripke model, $q \in St$ and $\lambda \in \Lambda$. The semantics of $\mathcal{L}_{\mathsf{CTL}^*}$- and $\mathcal{L}_{\mathsf{CTL}}$-formulae is given by the satisfaction relation $\models^{\mathsf{CTL}^*}$ for state formulae by

- $\mathcal{M}, q \models^{\mathsf{CTL}^*} \mathsf{p}$ iff $\lambda[0] \in \pi(\mathsf{p})$ and $\mathsf{p} \in \mathcal{P}rop$;
- $\mathcal{M}, q \models^{\mathsf{CTL}^*} \neg\varphi$ iff $\mathcal{M}, q \not\models^{\mathsf{CTL}^*} \varphi$;
- $\mathcal{M}, q \models^{\mathsf{CTL}^*} \varphi \vee \psi$ iff $\mathcal{M}, q \models^{\mathsf{CTL}^*} \varphi$ or $\mathcal{M}, q \models^{\mathsf{CTL}^*} \psi$;

Definition 2.10 (Semantics $\models^{\mathsf{CTL}^*}$)

Let $\mathcal{M}$ be a Kripke model, $q \in St$ and $\lambda \in \Lambda$. The semantics of $\mathcal{L}_{\mathsf{CTL}^*}$- and $\mathcal{L}_{\mathsf{CTL}}$-formulae is given by the satisfaction relation $\models^{\mathsf{CTL}^*}$ for state formulae by

- $\mathcal{M}, q \models^{\mathsf{CTL}^*} \mathsf{p}$ iff $\lambda[0] \in \pi(\mathsf{p})$ and $\mathsf{p} \in \mathcal{P}rop$;
- $\mathcal{M}, q \models^{\mathsf{CTL}^*} \neg\varphi$ iff $\mathcal{M}, q \not\models^{\mathsf{CTL}^*} \varphi$;
- $\mathcal{M}, q \models^{\mathsf{CTL}^*} \varphi \vee \psi$ iff $\mathcal{M}, q \models^{\mathsf{CTL}^*} \varphi$ or $\mathcal{M}, q \models^{\mathsf{CTL}^*} \psi$;
- $\mathcal{M}, q \models^{\mathsf{CTL}^*} \mathsf{E}\varphi$ iff there is a path $\lambda \in \Lambda(q)$ such that $\mathcal{M}, \lambda \models^{\mathsf{CTL}^*} \varphi$;

and for path formulae by:

- $\mathcal{M}, \lambda \models^{\mathsf{CTL}^*} \varphi$ iff

and for path formulae by:

- $\mathcal{M}, \lambda \models^{\mathsf{CTL}^*} \varphi$ iff $\mathcal{M}, \lambda[0] \models^{\mathsf{CTL}^*} \varphi$;
- $\mathcal{M}, \lambda \models^{\mathsf{CTL}^*} \neg\gamma$ iff

and for path formulae by:

- $\mathcal{M}, \lambda \models^{\textsf{CTL}^*} \varphi$ iff $\mathcal{M}, \lambda[0] \models^{\textsf{CTL}^*} \varphi$;
- $\mathcal{M}, \lambda \models^{\textsf{CTL}^*} \neg\gamma$ iff $\mathcal{M}, \lambda \not\models^{\textsf{CTL}^*} \gamma$;
- $\mathcal{M}, \lambda \models^{\textsf{CTL}^*} \gamma \vee \delta$ iff

and for path formulae by:

- $\mathcal{M}, \lambda \models^{\mathsf{CTL}^*} \varphi$ iff $\mathcal{M}, \lambda[0] \models^{\mathsf{CTL}^*} \varphi$;
- $\mathcal{M}, \lambda \models^{\mathsf{CTL}^*} \neg\gamma$ iff $\mathcal{M}, \lambda \not\models^{\mathsf{CTL}^*} \gamma$;
- $\mathcal{M}, \lambda \models^{\mathsf{CTL}^*} \gamma \vee \delta$ iff $\mathcal{M}, \lambda \models^{\mathsf{CTL}^*} \gamma$ or $\mathcal{M}, \lambda \models^{\mathsf{CTL}^*} \delta$;
- $\mathcal{M}, \lambda \models^{\mathsf{CTL}^*} \mathbf{X}\gamma$ iff

and for path formulae by:

- $\mathcal{M}, \lambda \models^{\mathbf{CTL}^*} \varphi$ iff $\mathcal{M}, \lambda[0] \models^{\mathbf{CTL}^*} \varphi$;
- $\mathcal{M}, \lambda \models^{\mathbf{CTL}^*} \neg\gamma$ iff $\mathcal{M}, \lambda \not\models^{\mathbf{CTL}^*} \gamma$;
- $\mathcal{M}, \lambda \models^{\mathbf{CTL}^*} \gamma \vee \delta$ iff $\mathcal{M}, \lambda \models^{\mathbf{CTL}^*} \gamma$ or $\mathcal{M}, \lambda \models^{\mathbf{CTL}^*} \delta$;
- $\mathcal{M}, \lambda \models^{\mathbf{CTL}^*} \mathbf{X}\gamma$ iff $\lambda[1, \infty], \pi \models^{\mathbf{CTL}^*} \gamma$; and
- $\mathcal{M}, \lambda \models^{\mathbf{CTL}^*} \gamma \mathcal{U} \delta$ iff

and for path formulae by:

- $\mathcal{M}, \lambda \models^{\mathsf{CTL}^*} \varphi$ iff $\mathcal{M}, \lambda[0] \models^{\mathsf{CTL}^*} \varphi$;
- $\mathcal{M}, \lambda \models^{\mathsf{CTL}^*} \neg\gamma$ iff $\mathcal{M}, \lambda \not\models^{\mathsf{CTL}^*} \gamma$;
- $\mathcal{M}, \lambda \models^{\mathsf{CTL}^*} \gamma \vee \delta$ iff $\mathcal{M}, \lambda \models^{\mathsf{CTL}^*} \gamma$ or $\mathcal{M}, \lambda \models^{\mathsf{CTL}^*} \delta$;
- $\mathcal{M}, \lambda \models^{\mathsf{CTL}^*} \mathbf{X}\gamma$ iff $\lambda[1, \infty], \pi \models^{\mathsf{CTL}^*} \gamma$; and
- $\mathcal{M}, \lambda \models^{\mathsf{CTL}^*} \gamma \mathcal{U} \delta$ iff there is an $i \in \mathbb{N}_0$ such that $\mathcal{M}, \lambda[i, \infty] \models^{\mathsf{CTL}^*} \delta$ and $\mathcal{M}, \lambda[j, \infty] \models^{\mathsf{CTL}^*} \gamma$ for all $0 \leq j < i$.

and for path formulae by:

- $\mathcal{M}, \lambda \models^{\mathbf{CTL}^*} \varphi$ iff $\mathcal{M}, \lambda[0] \models^{\mathbf{CTL}^*} \varphi$;
- $\mathcal{M}, \lambda \models^{\mathbf{CTL}^*} \neg\gamma$ iff $\mathcal{M}, \lambda \not\models^{\mathbf{CTL}^*} \gamma$;
- $\mathcal{M}, \lambda \models^{\mathbf{CTL}^*} \gamma \vee \delta$ iff $\mathcal{M}, \lambda \models^{\mathbf{CTL}^*} \gamma$ or $\mathcal{M}, \lambda \models^{\mathbf{CTL}^*} \delta$;
- $\mathcal{M}, \lambda \models^{\mathbf{CTL}^*} \mathbf{X}\gamma$ iff $\lambda[1, \infty], \pi \models^{\mathbf{CTL}^*} \gamma$; and
- $\mathcal{M}, \lambda \models^{\mathbf{CTL}^*} \gamma \mathcal{U} \delta$ iff there is an $i \in \mathbb{N}_0$ such that $\mathcal{M}, \lambda[i, \infty] \models^{\mathbf{CTL}^*} \delta$ and $\mathcal{M}, \lambda[j, \infty] \models^{\mathbf{CTL}^*} \gamma$ for all $0 \leq j < i$.

Is this complicated semantics over paths necessary for **CTL**?

State-based semantics for CTL

- $\mathcal{M}, q \models^{\mathsf{CTL}} p \quad$ iff $q \in \pi(p)$;

State-based semantics for CTL

- $\mathcal{M}, q \models^{\text{CTL}} p \quad$ iff $q \in \pi(p)$;
- $\mathcal{M}, q \models^{\text{CTL}} \neg\varphi \quad$ iff $\mathcal{M}, q \not\models^{\text{CTL}} \varphi$;

State-based semantics for CTL

- $\mathcal{M}, q \models^{\textbf{CTL}} p$ iff $q \in \pi(p)$;
- $\mathcal{M}, q \models^{\textbf{CTL}} \neg\varphi$ iff $\mathcal{M}, q \not\models^{\textbf{CTL}} \varphi$;
- $\mathcal{M}, q \models^{\textbf{CTL}} \varphi \vee \psi$ iff $\mathcal{M}, q \models^{\textbf{CTL}} \varphi$ or $\mathcal{M}, q \models^{\textbf{CTL}} \psi$;

State-based semantics for CTL

- $\mathcal{M}, q \models^{\mathsf{CTL}} p$ iff $q \in \pi(p)$;
- $\mathcal{M}, q \models^{\mathsf{CTL}} \neg\varphi$ iff $\mathcal{M}, q \not\models^{\mathsf{CTL}} \varphi$;
- $\mathcal{M}, q \models^{\mathsf{CTL}} \varphi \vee \psi$ iff $\mathcal{M}, q \models^{\mathsf{CTL}} \varphi$ or $\mathcal{M}, q \models^{\mathsf{CTL}} \psi$;
- $\mathcal{M}, q \models^{\mathsf{CTL}} \mathsf{E}\mathbf{X}\varphi$ iff there is a path $\lambda \in \Lambda(q)$ such that $\mathcal{M}, \lambda[1] \models^{\mathsf{CTL}} \varphi$;

State-based semantics for CTL

- $\mathcal{M}, q \models^{\mathsf{CTL}} p$ iff $q \in \pi(p)$;
- $\mathcal{M}, q \models^{\mathsf{CTL}} \neg\varphi$ iff $\mathcal{M}, q \not\models^{\mathsf{CTL}} \varphi$;
- $\mathcal{M}, q \models^{\mathsf{CTL}} \varphi \vee \psi$ iff $\mathcal{M}, q \models^{\mathsf{CTL}} \varphi$ or $\mathcal{M}, q \models^{\mathsf{CTL}} \psi$;
- $\mathcal{M}, q \models^{\mathsf{CTL}} \mathsf{E}\mathbf{X}\varphi$ iff there is a path $\lambda \in \Lambda(q)$ such that $\mathcal{M}, \lambda[1] \models^{\mathsf{CTL}} \varphi$;
- $\mathcal{M}, q \models^{\mathsf{CTL}} \mathsf{E}\mathbf{G}\varphi$ iff there is a path $\lambda \in \Lambda(q)$ such that $\mathcal{M}, \lambda[i] \models^{\mathsf{CTL}} \varphi$ for every $i \geq 0$;

State-based semantics for CTL

- $\mathcal{M}, q \models^{\textsf{CTL}} p$ iff $q \in \pi(p)$;
- $\mathcal{M}, q \models^{\textsf{CTL}} \neg\varphi$ iff $\mathcal{M}, q \not\models^{\textsf{CTL}} \varphi$;
- $\mathcal{M}, q \models^{\textsf{CTL}} \varphi \vee \psi$ iff $\mathcal{M}, q \models^{\textsf{CTL}} \varphi$ or $\mathcal{M}, q \models^{\textsf{CTL}} \psi$;
- $\mathcal{M}, q \models^{\textsf{CTL}} \mathsf{E}\mathbf{X}\varphi$ iff there is a path $\lambda \in \Lambda(q)$ such that $\mathcal{M}, \lambda[1] \models^{\textsf{CTL}} \varphi$;
- $\mathcal{M}, q \models^{\textsf{CTL}} \mathsf{E}\mathbf{G}\varphi$ iff there is a path $\lambda \in \Lambda(q)$ such that $\mathcal{M}, \lambda[i] \models^{\textsf{CTL}} \varphi$ for every $i \geq 0$;
- $\mathcal{M}, q \models^{\textsf{CTL}} \mathsf{E}\varphi\mathcal{U}\psi$ iff there is a path $\lambda \in \Lambda(q)$ such that $\mathcal{M}, \lambda[i] \models^{\textsf{CTL}} \psi$ for some $i \geq 0$, and $\mathcal{M}, \lambda[j] \models^{\textsf{CTL}} \varphi$ for all $0 \leq j < i$.

LTL as subset of CTL*

LTL is interpreted over infinite chains (infinite words), but not over (serial) Kripke structures (which are branching).

- To consider **LTL** as a subset of **CTL***, one can just add the quantifier A in front of a **LTL** formula and use the semantics of **CTL***. For infinite chains, this semantics coincides with the **LTL** semantics.
- The theorem of *Clarke und Draghiescu* gives a nice characterization of those **CTL*** formulae that are equivalent to LTL formulae. Given a **CTL*** formula φ, we construct φ' by just forgetting all path operators. Then

φ is equivalent to a **LTL** formula

iff

φ and Aφ' are equivalent under the semantics of **CTL***.

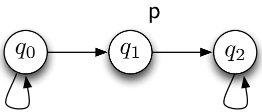
Application of Clarke and Draghiescu

We consider the **LTL** formula **GF**p. Viewed as a **CTL*** formula it becomes A**GF**p. But this is equivalent (in **CTL***) to A**G**A**F**p, a **CTL** formula.

Now we consider the **CTL** formula E**G**E**F**p. It is not equivalent to any **LTL** formula. This is because

E**G**E**F**p and A**GF**p

are not equivalent in **CTL***:



The first formula holds, the second does not.

LTL as subset of CTL* (2)

- How do **LTL** and **CTL** compare?
- The **CTL** formula $A\mathbf{G}(p \rightarrow (E\mathbf{X}q \wedge E\mathbf{X}\neg q))$ describes Kripke structures of the form in Example 2.6. No **LTL** formula can describe this class of Kripke structures.
- The **LTL** formula $A\mathbf{F}(p \wedge \mathbf{X}p)$ can not be expressed by a **CTL** formula. Check why neither $A\mathbf{F}(p \wedge A\mathbf{X}p)$ nor $A\mathbf{F}(p \wedge E\mathbf{X}p)$ are equivalent. Similarly, the **LTL** formula $A\mathbf{F}\mathbf{G}p$ can not be expressed by a **CTL** formula.
- There is a syntactic characterisation of formulae expressible in both **CTL** and **LTL**. Model checking in this class can be done more efficiently. We refer to [Maidl, 2000].

Example 2.11 (Robots and Carriage)

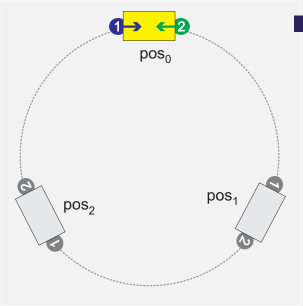


Figure 2 : Two robots and a carriage.

- Two robots push a carriage from opposite sides.

Example 2.11 (Robots and Carriage)

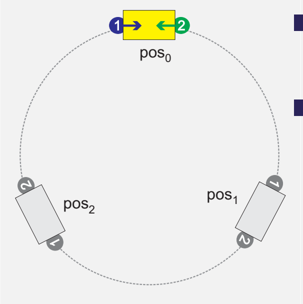


Figure 2 : Two robots and a carriage.

- Two robots push a carriage from opposite sides.
- Carriage can move clockwise or anticlockwise, or it can remain in the same place.

Example 2.11 (Robots and Carriage)

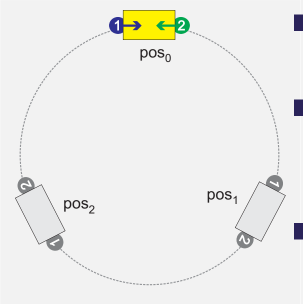


Figure 2 : Two robots and a carriage.

- Two robots push a carriage from opposite sides.
- Carriage can move clockwise or anticlockwise, or it can remain in the same place.
- 3 positions of the carriage.

Example 2.11 (Robots and Carriage)

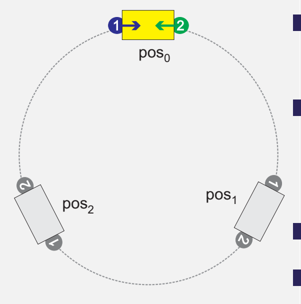


Figure 2 : Two robots and a carriage.

- Two robots push a carriage from opposite sides.
- Carriage can move clockwise or anticlockwise, or it can remain in the same place.
- 3 positions of the carriage.
- We label the states with propositions pos_0, pos_1, pos_2, respectively, to allow for referring to the current position of the carriage in the object language.

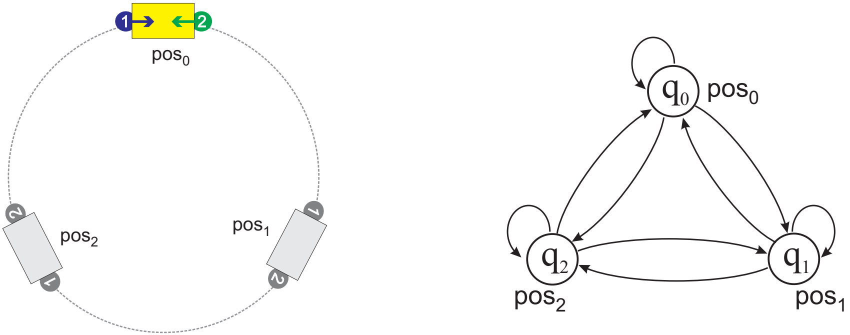
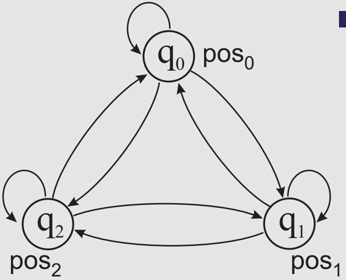
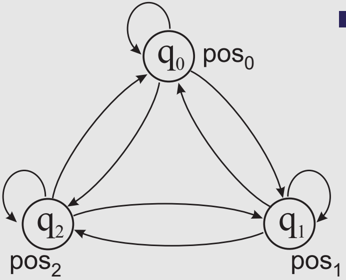


Figure 3 : Two robots and a carriage: A schematic view (left) and a transition system $\mathcal{M}_0$ that models the scenario (right).

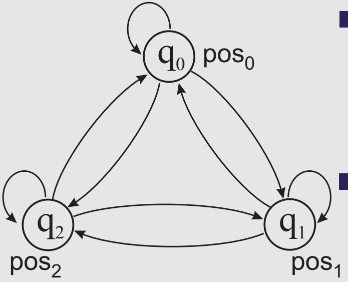


- $\mathcal{M}_0, q_0$ $\mathsf{E}\mathbf{F}\mathsf{pos}_1$:



- $\mathcal{M}_0, q_0 \models^{\mathsf{CTL}} \mathsf{E}\mathbf{F}\mathsf{pos}_1$: In state q_0, there is a path such that the carriage will reach position 1 sometime in the future.

$\mathcal{M}_0, q_0$ $\mathsf{A}\mathbf{F}\mathsf{pos}_1$.



- $\mathcal{M}_0, q_0 \models^{\mathsf{CTL}} \mathsf{E}\mathbf{F}\mathsf{pos}_1$: In state q_0, there is a path such that the carriage will reach position 1 sometime in the future.
- The same is not true for *all* paths, so we also have: $\mathcal{M}_0, q_0 \not\models^{\mathsf{CTL}} \mathsf{A}\mathbf{F}\mathsf{pos}_1$.

It becomes more interesting if abilities of agents are considered $\rightsquigarrow$ **ATL**.

Example: Rocket and Cargo

- A **rocket** and a **cargo**.

Example: Rocket and Cargo

- A **rocket** and a **cargo**.
- The rocket can be moved between London (proposition roL) and Paris (proposition roP).

Example: Rocket and Cargo

- A **rocket** and a **cargo**.
- The rocket can be moved between London (proposition roL) and Paris (proposition roP).
- The cargo can be in London (caL), Paris (caP), or inside the rocket (caR).

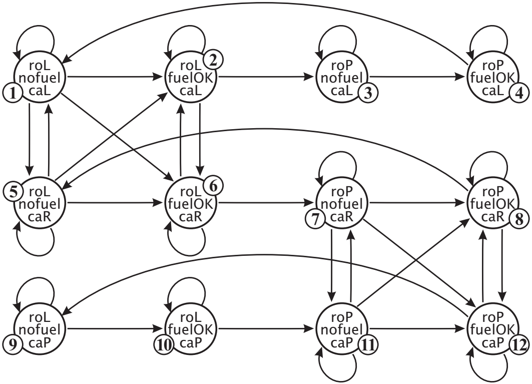
Example: Rocket and Cargo

- A **rocket** and a **cargo**.
- The rocket can be moved between London (proposition roL) and Paris (proposition roP).
- The cargo can be in London (caL), Paris (caP), or inside the rocket (caR).
- The rocket can be moved only if it has its fuel tank full (fuelOK).

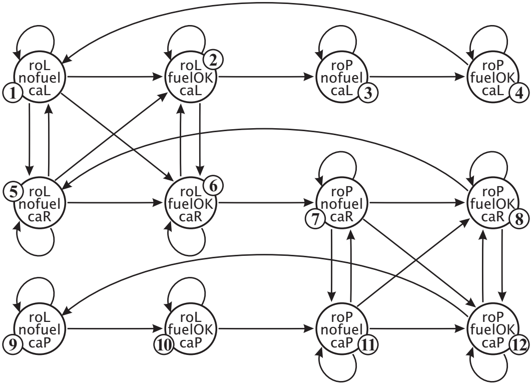
Example: Rocket and Cargo

- A **rocket** and a **cargo**.
- The rocket can be moved between London (proposition roL) and Paris (proposition roP).
- The cargo can be in London (caL), Paris (caP), or inside the rocket (caR).
- The rocket can be moved only if it has its fuel tank full (fuelOK).
- When it moves, it consumes fuel, and nofuel holds after each flight.

Example: Rocket and Cargo

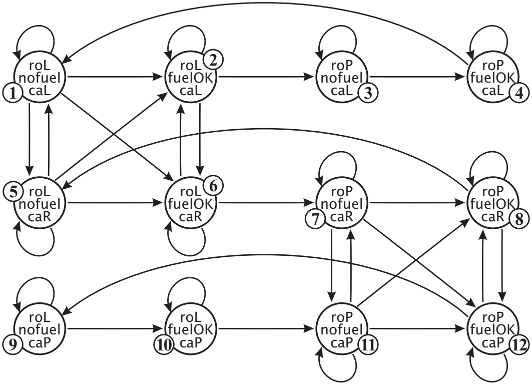


Example: Rocket and Cargo



$$roL \rightarrow E\Diamond roP$$

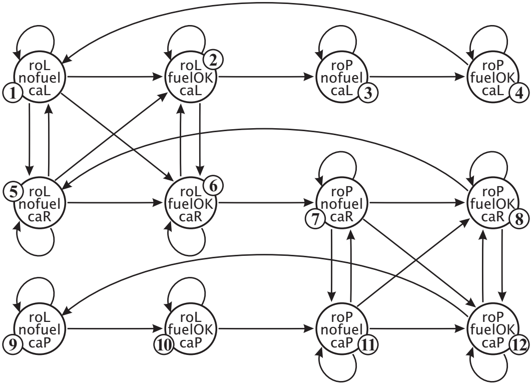
Example: Rocket and Cargo



$$\mathrm{roL} \to \mathsf{E}\Diamond \mathrm{roP}$$

$$\mathsf{A}\mathbf{G}(\mathrm{roL} \vee \mathrm{roP})$$

Example: Rocket and Cargo

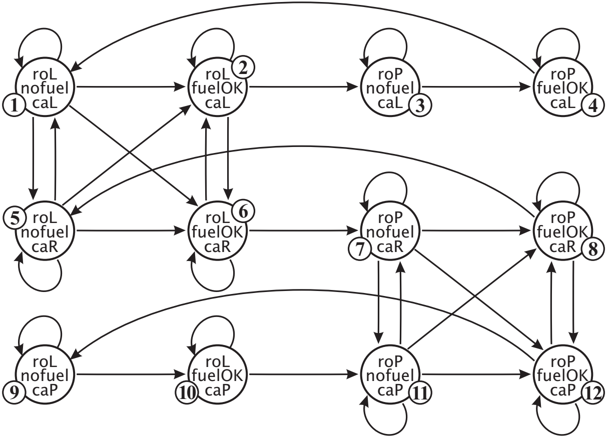


$$\mathrm{roL} \rightarrow \mathrm{E}\Diamond\mathrm{roP}$$

$$\mathrm{A}\mathbf{G}(\mathrm{roL} \vee \mathrm{roP})$$

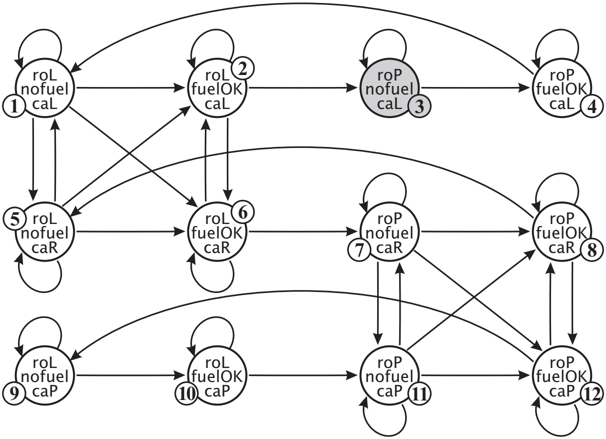
$$\mathrm{roL} \rightarrow \mathrm{A}\mathbf{X}(\mathrm{roP} \rightarrow \mathrm{nofuel})$$

Example: Rocket and Cargo



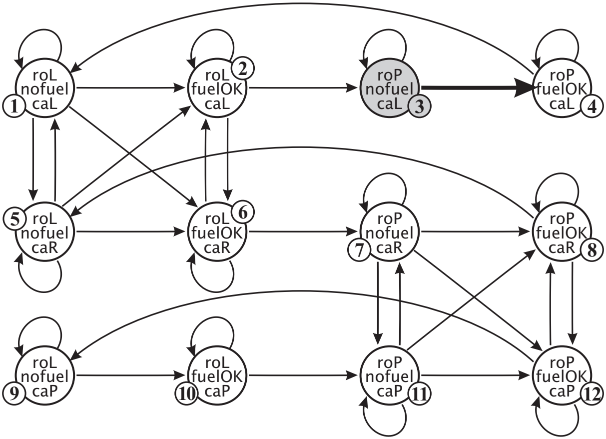
$E\Diamond caP$

Example: Rocket and Cargo



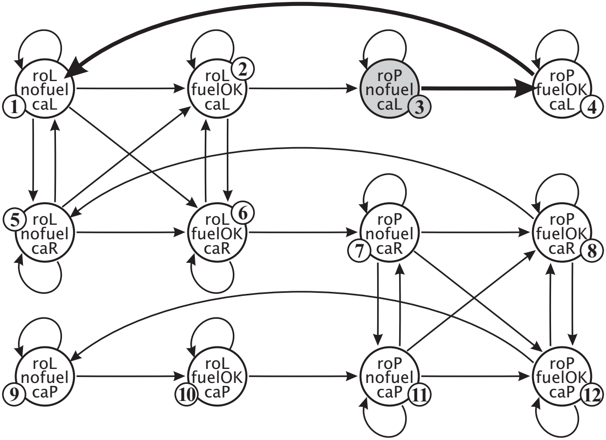
$E\Diamond caP$

Example: Rocket and Cargo



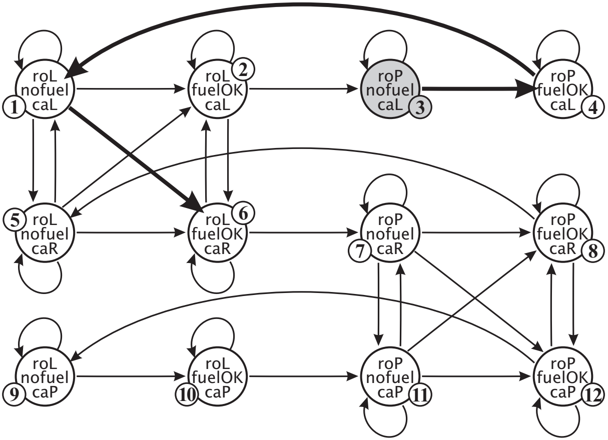
$E\Diamond caP$

Example: Rocket and Cargo



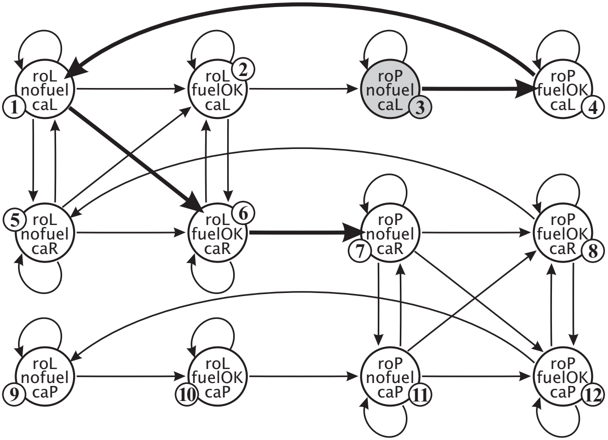
$E\Diamond caP$

Example: Rocket and Cargo



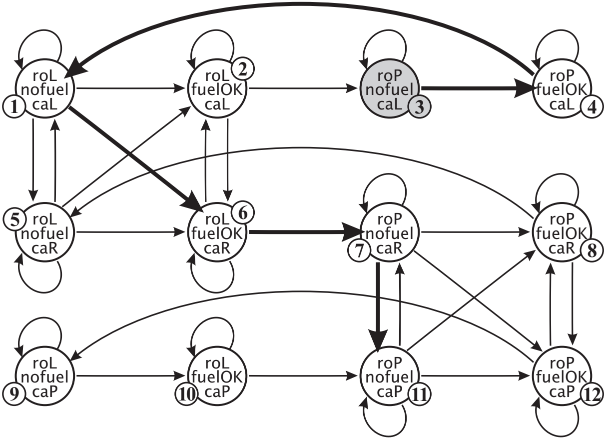
$E\Diamond caP$

Example: Rocket and Cargo



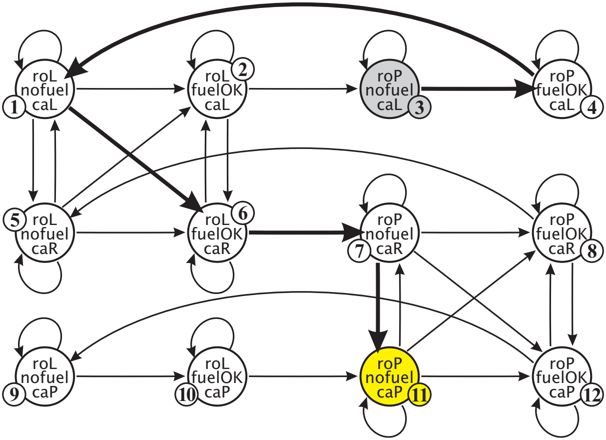
E◊caP

Example: Rocket and Cargo



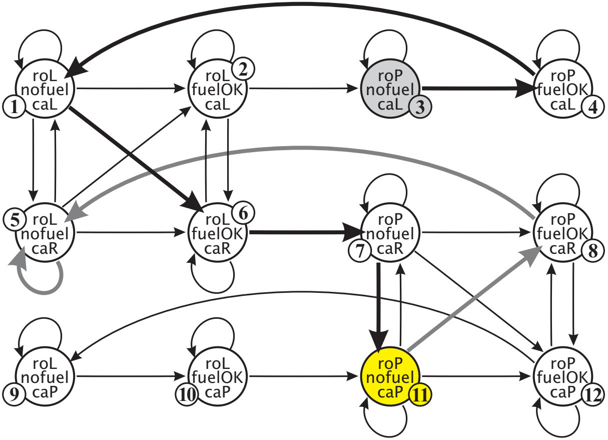
E◊caP

Example: Rocket and Cargo



$E\Diamond caP$

Example: Rocket and Cargo



E◊caP

- In our logics, we assumed a serial accessibility relation: no deadlocks are possible.
- One can also allow states with no outgoing transitions. In that case, in the semantical definition of E on Slide 138 one has to replace “there is a path” by “there is an infinite path or one which can not be extended”.
- Similar modifications are needed in the definition of **CTL**.
- One can also add to each state with no outgoing transitions a special transition leading to a new state that loops into itself.

How to express that there is no possibility of a deadlock?

- In our logics, we assumed a serial accessibility relation: no deadlocks are possible.
- One can also allow states with no outgoing transitions. In that case, in the semantical definition of E on Slide 138 one has to replace “there is a path” by “there is an infinite path or one which can not be extended”.
- Similar modifications are needed in the definition of **CTL**.
- One can also add to each state with no outgoing transitions a special transition leading to a new state that loops into itself.

How to express that there is no possibility of a deadlock?

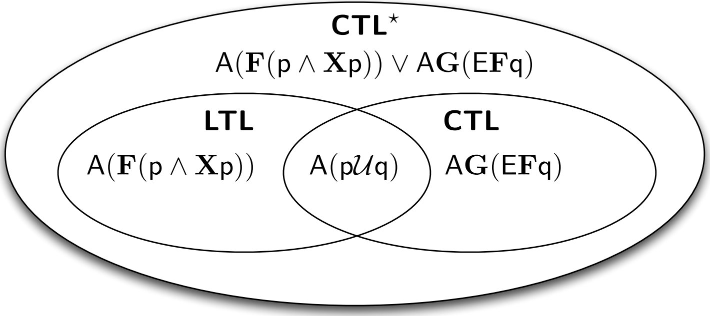
$$\mathsf{A}\mathbf{GX}\top \quad (\rightsquigarrow \mathbf{CTL}^*)$$

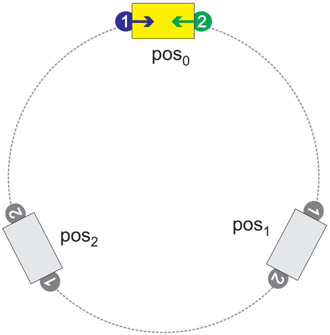
- In our logics, we assumed a serial accessibility relation: no deadlocks are possible.
- One can also allow states with no outgoing transitions. In that case, in the semantical definition of E on Slide 138 one has to replace “there is a path” by “there is an infinite path or one which can not be extended”.
- Similar modifications are needed in the definition of **CTL**.
- One can also add to each state with no outgoing transitions a special transition leading to a new state that loops into itself.

How to express that there is no possibility of a deadlock?

$\mathsf{A}\mathbf{G}\mathbf{X}\top \quad (\rightsquigarrow \textbf{CTL}^*) \qquad\qquad \mathsf{A}\mathbf{G}\mathsf{E}\mathbf{X}\top \quad (\rightsquigarrow \textbf{CTL})$

A Venn diagram showing typical formulae in the respective areas.





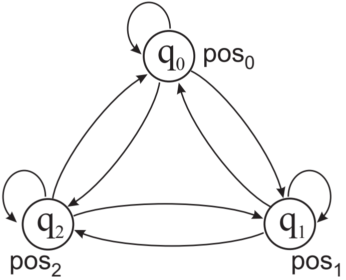


Figure 4 : Two robots and a carriage: a schematic view (left) and a transition system $\mathcal{M}_0$ that models the scenario (right).

2.3 ATL and variants

Alternating-time Temporal Logics

- **ATL, ATL**** [Alur et al. 1997]
- Temporal logic meets *game theory*
- Modeling abilities of *multiple agents*
- Main idea: *cooperation modalities*

Alternating-time Temporal Logics

- **ATL, ATL**** [Alur et al. 1997]
- Temporal logic meets *game theory*
- Modeling abilities of *multiple agents*
- Main idea: *cooperation modalities*

$\langle\!\langle A\rangle\!\rangle\varphi$: coalition A has a collective strategy to **enforce** φ

Enforcement is understood in the game-theoretical sense: There is a *winning strategy*.

The syntax is given as for the computation-tree logics.

Definition 2.12 (Language $\mathcal{L}_{ATL^*}$ [Alur et al., 1997])

The language $\mathcal{L}_{ATL^*}$ is given by all formulae generated by the following grammar:

$$\varphi ::= \mathrm{p} \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\langle A\rangle\rangle\gamma \quad \text{where}$$
$$\gamma ::= \varphi \mid \neg\gamma \mid \gamma \vee \gamma \mid \gamma\,\mathcal{U}\,\gamma \mid \bigcirc\gamma,$$

$A \subseteq \mathbb{A}\mathrm{gt}$, and $\mathrm{p} \in \mathcal{P}rop$. Formulae φ (resp. γ) are called state (resp. path) formulae.

Note that we are using now the symbol "$\bigcirc$" instead of "**X**" as it is more custom when dealing with **ATL**.

The language $\mathcal{L}_{ATL}$ restricts $\mathcal{L}_{ATL^*}$ in the same way as $\mathcal{L}_{CTL}$ restricts $\mathcal{L}_{CTL^*}$: Each temporal operator must be directly preceded by a cooperation modality.

Definition 2.13 (Language $\mathcal{L}_{ATL}$ [Alur et al., 1997])

The language $\mathcal{L}_{ATL}$ is given by all formulae generated by the following grammar:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\!\langle A \rangle\!\rangle \bigcirc \varphi \mid \langle\!\langle A \rangle\!\rangle \Box \varphi \mid \langle\!\langle A \rangle\!\rangle \varphi \mathcal{U} \varphi$$

where $A \subseteq \mathbb{A}\mathrm{gt}$ and p $\in \mathcal{P}rop$.

Note that we are using now the symbol "$\Box$" instead of "**G**" as it is more custom when dealing with **ATL**.

The language $\mathcal{L}_{ATL^+}$ restricts $\mathcal{L}_{ATL^*}$ but extends $\mathcal{L}_{ATL}$. It allows for Boolean combinations of path formulae.

Definition 2.14 (Language $\mathcal{L}_{ATL^+}$)

The language $\mathcal{L}_{ATL^+}$ is given by all formulae generated by the following grammar:

$$\varphi ::= \mathsf{p} \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\langle A\rangle\rangle\gamma, \qquad \gamma ::= \neg\gamma \mid \gamma \vee \gamma \mid \bigcirc\varphi \mid \varphi\,\mathcal{U}\,\varphi.$$

where $A \subseteq \mathbb{A}\mathrm{gt}$ and $\mathsf{p} \in \mathcal{P}rop$.

The language $\mathcal{L}_{ATL^+}$ restricts $\mathcal{L}_{ATL^*}$ but extends $\mathcal{L}_{ATL}$. It allows for Boolean combinations of path formulae.

Definition 2.14 (Language $\mathcal{L}_{ATL^+}$)

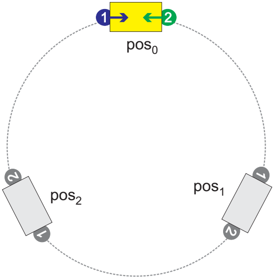
The language $\mathcal{L}_{ATL^+}$ is given by all formulae generated by the following grammar:

$$\varphi ::= \mathsf{p} \mid \neg\varphi \mid \varphi \vee \varphi \mid \langle\langle A \rangle\rangle\gamma, \qquad \gamma ::= \neg\gamma \mid \gamma \vee \gamma \mid \bigcirc\varphi \mid \varphi\,\mathcal{U}\,\varphi.$$

where $A \subseteq \mathbb{A}\mathrm{gt}$ and $\mathsf{p} \in \mathcal{P}rop$.

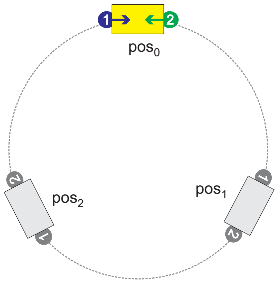
ATL Models: Concurrent Game Structures

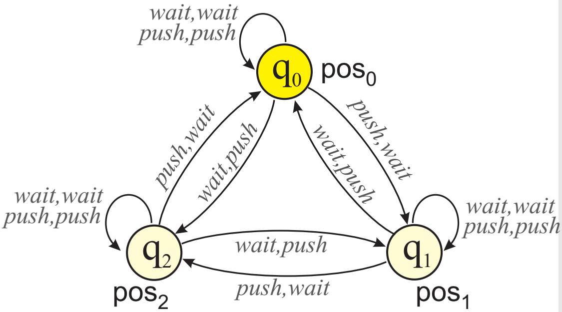
- **Agents**, **actions**, **transitions**, atomic **propositions**
- Atomic propositions + **interpretation**
- **Actions are abstract**



ATL Models: Concurrent Game Structures

- **Agents**, **actions**, **transitions**, atomic **propositions**
- Atomic propositions + **interpretation**
- **Actions are abstract**





Definition 2.15 (Concurrent Game Structure)

A concurrent game structure is a tuple
$\mathcal{M} = \langle \mathbb{A}\mathrm{gt}, St, \pi, Act, d, o \rangle$, where:

Definition 2.15 (Concurrent Game Structure)

A concurrent game structure is a tuple
$\mathcal{M} = \langle \mathbb{A}\mathrm{gt}, St, \pi, Act, d, o \rangle$, where:

- $\mathbb{A}\mathrm{gt}$: a finite set of all agents;

Definition 2.15 (Concurrent Game Structure)

A concurrent game structure is a tuple $\mathcal{M} = \langle \mathbb{A}\mathrm{gt}, St, \pi, Act, d, o \rangle$, where:

- $\mathbb{A}\mathrm{gt}$: a finite set of all agents;
- St: a set of states;

Definition 2.15 (Concurrent Game Structure)

A concurrent game structure is a tuple
$\mathcal{M} = \langle \mathbb{A}\mathrm{gt}, St, \pi, Act, d, o \rangle$, where:

- $\mathbb{A}\mathrm{gt}$: a finite set of all agents;
- St: a set of states;
- $\pi : St \to 2^{\mathcal{P}rop}$: a valuation of propositions;

Definition 2.15 (Concurrent Game Structure)

A concurrent game structure is a tuple
$\mathcal{M} = \langle \mathbb{A}\mathrm{gt}, St, \pi, Act, d, o \rangle$, where:

- $\mathbb{A}\mathrm{gt}$: a finite set of all agents;
- St: a set of states;
- $\pi : St \rightarrow 2^{\mathcal{P}rop}$: a valuation of propositions;
- Act: a finite set of (atomic) actions;

Definition 2.15 (Concurrent Game Structure)

A concurrent game structure is a tuple
$\mathcal{M} = \langle \mathbb{A}\mathrm{gt}, St, \pi, Act, d, o \rangle$, where:

- $\mathbb{A}\mathrm{gt}$: a finite set of all agents;
- St: a set of states;
- $\pi : St \rightarrow 2^{\mathcal{P}rop}$: a valuation of propositions;
- Act: a finite set of (atomic) actions;
- $d : \mathbb{A}\mathrm{gt} \times St \rightarrow 2^{Act}$ 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.

Recall and information

A strategy of agent a is a conditional plan that specifies what a is going to do in each situation.

Two types of "situations": Decisions are based on

Recall and information

A strategy of agent a is a conditional plan that specifies what a is going to do in each situation.

Two types of "situations": Decisions are based on

- the current state only ($\rightsquigarrow$ memoryless strategies)

$$s_a : St \to Act.$$

Recall and information

A strategy of agent a is a conditional plan that specifies what a is going to do in each situation.

Two types of "situations": Decisions are based on

- the current state only ($\leadsto$ memoryless strategies)

  $$s_a : St \to Act.$$

- on the whole history of events that have happened ($\leadsto$ perfect recall strategies)

  $$s_a : St^+ \to Act.$$

Recall and information

A strategy of agent a is a conditional plan that specifies what a is going to do in each situation.

Two types of “situations”: Decisions are based on

- the current state only ($\rightsquigarrow$ memoryless strategies)

  $$s_a : St \to Act.$$

- on the whole history of events that have happened ($\rightsquigarrow$ perfect recall strategies)

  $$s_a : St^+ \to Act.$$

We also distinguish between agents with

- perfect information (all states are distinguishable).
- imperfect information (some state are

Perfect Information Strategies

Definition 2.16 (*IR*- and *Ir*-strategies)

- A perfect information perfect recall strategy for agent a (*IR*-strategy for short) is a function

  $$s_a : St^+ \to Act \text{ such that } s_a(q_0 q_1 \dots q_n) \in d_a(q_n).$$

  The set of such strategies is denoted by Σ_a^{IR}.

Perfect Information Strategies

Definition 2.16 (*IR*- and *Ir*-strategies)

- A perfect information perfect recall strategy for agent a (*IR*-strategy for short) is a function

  $$s_a : St^+ \to Act \text{ such that } s_a(q_0q_1\dots q_n) \in d_a(q_n).$$

  The set of such strategies is denoted by Σ_a^{IR}.

- A perfect information memoryless strategy for agent a (*Ir*-strategy for short) is given by a function

  $$s_a : St \to Act \text{ where } s_{a(q)} \in d_{a(q)}.$$

  The set of such strategies is denoted by Σ_a^{Ir}.

Perfect Information Strategies

Definition 2.16 (*IR*- and *Ir*-strategies)

- A perfect information perfect recall strategy for agent a (*IR*-strategy for short) is a function

  $s_a : St^+ \to Act$ such that $s_a(q_0 q_1 \dots q_n) \in d_a(q_n)$.

  The set of such strategies is denoted by Σ_a^{IR}.

- A perfect information memoryless strategy for agent a (*Ir*-strategy for short) is given by a function

  $s_a : St \to Act$ where $s_{a(q)} \in d_{a(q)}$.

  The set of such strategies is denoted by Σ_a^{Ir}.

i (resp. *I*) stands for imperfect (resp. perfect) information and *r* (resp. *R*) for imperfect (resp. perfect) recall. [Schobbens, 2004]

Some Notation

The following holds for all kind of strategies:

- A collective strategy for a group of agents $A = \{a_1, \ldots, a_r\} \subseteq \mathbb{A}\mathrm{gt}$ is a set

  $$s_A = \{s_a \mid a \in A\}$$

  of strategies, one per agent from A.

Some Notation

The following holds for all kind of strategies:

- A collective strategy for a group of agents $A = \{a_1, \dots, a_r\} \subseteq \mathbb{A}\mathrm{gt}$ is a set

  $$s_A = \{s_a \mid a \in A\}$$

  of strategies, one per agent from A.
- $s_A|_a$, we denote agent a's part of the collective strategy s_A, $s_A|_a = s_A \cap \Sigma_a$.

Some Notation

The following holds for all kind of strategies:

- A collective strategy for a group of agents $A = \{a_1, \ldots, a_r\} \subseteq \mathbb{A}\mathrm{gt}$ is a set

  $$s_A = \{s_a \mid a \in A\}$$

  of strategies, one per agent from A.
- $s_A|_a$, we denote agent a's part of the collective strategy s_A, $s_A|_a = s_A \cap \Sigma_a$.
- $s_\emptyset = \emptyset$ denotes the strategy of the empty coalition.

Some Notation

The following holds for all kind of strategies:

- A collective strategy for a group of agents $A = \{a_1, \dots, a_r\} \subseteq \mathbb{A}\mathrm{gt}$ is a set
$$s_A = \{s_a \mid a \in A\}$$
of strategies, one per agent from A.
- $s_A|_a$, we denote agent a's part of the collective strategy s_A, $s_A|_a = s_A \cap \Sigma_a$.
- $s_\emptyset = \emptyset$ denotes the strategy of the empty coalition.
- Σ_A denotes the set of all collective strategies of A.

Some Notation

The following holds for all kind of strategies:

- A collective strategy for a group of agents $A = \{a_1, \ldots, a_r\} \subseteq \mathbb{A}\mathrm{gt}$ is a set
  $$s_A = \{s_a \mid a \in A\}$$
  of strategies, one per agent from A.
- $s_A|_a$, we denote agent a's part of the collective strategy s_A, $s_A|_a = s_A \cap \Sigma_a$.
- $s_\emptyset = \emptyset$ denotes the strategy of the empty coalition.
- Σ_A denotes the set of all collective strategies of A.
- $\Sigma = \Sigma_{\mathbb{A}\mathrm{gt}}$

Outcome of a strategy

$out(q, s_A)$= set of all paths that may occur
when agents A execute s_A from state q onward.

Definition 2.17 (Outcome)

$\lambda = q_0 q_1 \ldots \in St \in out(q, s_A) \subseteq St^\omega$ iff

1. $q_0 = q$

Outcome of a strategy

$out(q, s_A)$= set of all paths that may occur when agents A execute s_A from state q onward.

Definition 2.17 (Outcome)

$\lambda = q_0 q_1 \ldots \in St \in out(q, s_A) \subseteq St^\omega$ iff

1. $q_0 = q$
2. for each $i = 1, \ldots$ there is a tuple $(\alpha_1^{i-1}, \ldots, \alpha_k^{i-1}) \in Act^k$ such that

Outcome of a strategy

$out(q, s_A)$= set of all paths that may occur
when agents A execute s_A from state q onward.

Definition 2.17 (Outcome)

$\lambda = q_0 q_1 \ldots \in St \in out(q, s_A) \subseteq St^\omega$ iff

1. $q_0 = q$
2. for each $i = 1, \ldots$ there is a tuple $(\alpha_1^{i-1}, \ldots, \alpha_k^{i-1}) \in Act^k$ such that
   - $\alpha_a^{i-1} \in d_a(q_{i-1})$ for each $a \in \mathbb{A}\mathrm{gt}$,

Outcome of a strategy

$out(q, s_A)$= set of all paths that may occur when agents A execute s_A from state q onward.

Definition 2.17 (Outcome)

$\lambda = q_0 q_1 \ldots \in St \in out(q, s_A) \subseteq St^\omega$ iff

1. $q_0 = q$
2. for each $i = 1, \ldots$ there is a tuple $(\alpha_1^{i-1}, \ldots, \alpha_k^{i-1}) \in Act^k$ such that
   - $\alpha_a^{i-1} \in d_a(q_{i-1})$ for each $a \in \mathbb{A}\mathrm{gt}$,
   - $\alpha_a^{i-1} = s_A|_a(q_0 q_1 \ldots q_{i-1})$ for each $a \in A$, and

Outcome of a strategy

$out(q, s_A)$= set of all paths that may occur when agents A execute s_A from state q onward.

Definition 2.17 (Outcome)

$\lambda = q_0 q_1 \ldots \in St \in out(q, s_A) \subseteq St^\omega$ iff

1. $q_0 = q$
2. for each $i = 1, \ldots$ there is a tuple $(\alpha_1^{i-1}, \ldots, \alpha_k^{i-1}) \in Act^k$ such that
   - $\alpha_a^{i-1} \in d_a(q_{i-1})$ for each $a \in \mathbb{A}\mathrm{gt}$,
   - $\alpha_a^{i-1} = s_A|_a(q_0 q_1 \ldots q_{i-1})$ for each $a \in A$, and
   - $o(q_{i-1}, \alpha_1^{i-1}, \ldots, \alpha_k^{i-1}) = q_i$.

Outcome of a strategy

$out(q, s_A)$= set of all paths that may occur when agents A execute s_A from state q onward.

Definition 2.17 (Outcome)

$\lambda = q_0 q_1 \ldots \in St \in out(q, s_A) \subseteq St^\omega$ iff

1. $q_0 = q$
2. for each $i = 1, \ldots$ there is a tuple $(\alpha_1^{i-1}, \ldots, \alpha_k^{i-1}) \in Act^k$ such that
   - $\alpha_a^{i-1} \in d_a(q_{i-1})$ for each $a \in \mathbb{A}\mathrm{gt}$,
   - $\alpha_a^{i-1} = s_A|_a(q_0 q_1 \ldots q_{i-1})$ for each $a \in A$, and
   - $o(q_{i-1}, \alpha_1^{i-1}, \ldots, \alpha_k^{i-1}) = q_i$.

For an Ir-strategy replace "$s_A|_a(q_0 q_1 \ldots q_{i-1})$" by "$s_A|_a(q_{i-1})$".

Definition 2.18 (Perfect information semantics)

$\mathcal{M}, q \models_{\mathsf{Ix}} \langle\langle A \rangle\rangle \Phi$ iff there is a collective **Ix-strategy** s_A such that, for each path $\lambda \in out(q, s_A)$, we have $\mathcal{M}, \lambda \models_{\mathsf{Ix}} \Phi$.

Definition 2.18 (Perfect information semantics)

$\mathcal{M}, q \models_{\mathsf{Ix}} \langle\langle A \rangle\rangle \Phi$ iff there is a collective Ix-strategy s_A such that, for each path $\lambda \in out(q, s_A)$, we have $\mathcal{M}, \lambda \models_{\mathsf{Ix}} \Phi$.

$\mathcal{M}, \lambda \models_{\mathsf{Ix}} \bigcirc \varphi$ iff $\mathcal{M}, \lambda[1, \infty] \models_{\mathsf{Ix}} \varphi$;

Definition 2.18 (Perfect information semantics)

$\mathcal{M}, q \models_{\text{Ix}} \langle\!\langle A \rangle\!\rangle \Phi$ iff there is a collective **Ix-strategy** s_A such that, for each path $\lambda \in out(q, s_A)$, we have $\mathcal{M}, \lambda \models_{\text{Ix}} \Phi$.

$\mathcal{M}, \lambda \models_{\text{Ix}} \bigcirc \varphi$ iff $\mathcal{M}, \lambda[1, \infty] \models_{\text{Ix}} \varphi$;

$\mathcal{M}, \lambda \models_{\text{Ix}} \Diamond \varphi$ iff $\mathcal{M}, \lambda[i, \infty] \models_{\text{Ix}} \varphi$ for some $i \geq 0$;

Definition 2.18 (Perfect information semantics)

$\mathcal{M}, q \models_{\mathsf{Ix}} \langle\langle A \rangle\rangle \Phi$ iff there is a collective **Ix-strategy** s_A such that, for each path $\lambda \in out(q, s_A)$, we have $\mathcal{M}, \lambda \models_{\mathsf{Ix}} \Phi$.

$\mathcal{M}, \lambda \models_{\mathsf{Ix}} \bigcirc\varphi$ iff $\mathcal{M}, \lambda[1,\infty] \models_{\mathsf{Ix}} \varphi$;
$\mathcal{M}, \lambda \models_{\mathsf{Ix}} \Diamond\varphi$ iff $\mathcal{M}, \lambda[i,\infty] \models_{\mathsf{Ix}} \varphi$ for some $i \geq 0$;
$\mathcal{M}, \lambda \models_{\mathsf{Ix}} \Box\varphi$ iff $\mathcal{M}, \lambda[i,\infty] \models_{\mathsf{Ix}} \varphi$ for all $i \geq 0$;

Definition 2.18 (Perfect information semantics)

$\mathcal{M}, q \models_{\mathsf{Ix}} \langle\langle A \rangle\rangle \Phi$ iff there is a collective **Ix-strategy** s_A such that, for each path $\lambda \in out(q, s_A)$, we have $\mathcal{M}, \lambda \models_{\mathsf{Ix}} \Phi$.

$\mathcal{M}, \lambda \models_{\mathsf{Ix}} \bigcirc \varphi$ iff $\mathcal{M}, \lambda[1, \infty] \models_{\mathsf{Ix}} \varphi$;

$\mathcal{M}, \lambda \models_{\mathsf{Ix}} \Diamond \varphi$ iff $\mathcal{M}, \lambda[i, \infty] \models_{\mathsf{Ix}} \varphi$ for some $i \geq 0$;

$\mathcal{M}, \lambda \models_{\mathsf{Ix}} \Box \varphi$ iff $\mathcal{M}, \lambda[i, \infty] \models_{\mathsf{Ix}} \varphi$ for all $i \geq 0$;

$\mathcal{M}, \lambda \models_{\mathsf{Ix}} \varphi \mathcal{U} \psi$ iff $\mathcal{M}, \lambda[i, \infty] \models_{\mathsf{Ix}} \psi$ for some $i \geq 0$, and $\mathcal{M}, \lambda[j, \infty] \models_{\mathsf{Ix}} \varphi$ forall $0 \leq j \leq i$.

Definition 2.18 (Perfect information semantics)

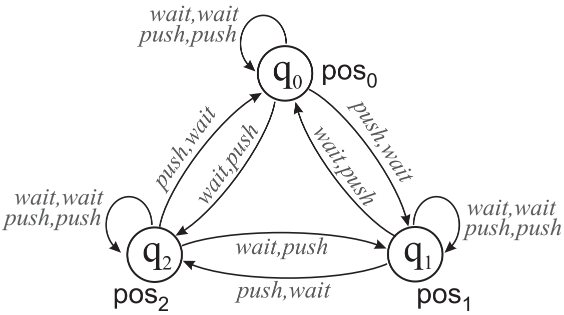
$\mathcal{M}, q \models_{\mathsf{Ix}} p$	iff p is in $\pi(q)$;
$\mathcal{M}, q \models_{\mathsf{Ix}} \varphi \vee \psi$	iff $\mathcal{M}, q \models_{\mathsf{Ix}} \varphi$ or $\mathcal{M}, q \models_{\mathsf{Ix}} \psi$;
$\mathcal{M}, q \models_{\mathsf{Ix}} \langle\langle A \rangle\rangle \Phi$	iff there is a collective **Ix-strategy** s_A such that, for each path $\lambda \in out(q, s_A)$, we have $\mathcal{M}, \lambda \models_{\mathsf{Ix}} \Phi$.
$\mathcal{M}, \lambda \models_{\mathsf{Ix}} \bigcirc \varphi$	iff $\mathcal{M}, \lambda[1, \infty] \models_{\mathsf{Ix}} \varphi$;
$\mathcal{M}, \lambda \models_{\mathsf{Ix}} \Diamond \varphi$	iff $\mathcal{M}, \lambda[i, \infty] \models_{\mathsf{Ix}} \varphi$ for some $i \geq 0$;
$\mathcal{M}, \lambda \models_{\mathsf{Ix}} \Box \varphi$	iff $\mathcal{M}, \lambda[i, \infty] \models_{\mathsf{Ix}} \varphi$ for all $i \geq 0$;
$\mathcal{M}, \lambda \models_{\mathsf{Ix}} \varphi \mathcal{U} \psi$	iff $\mathcal{M}, \lambda[i, \infty] \models_{\mathsf{Ix}} \psi$ for some $i \geq 0$, and $\mathcal{M}, \lambda[j, \infty] \models_{\mathsf{Ix}} \varphi$ forall $0 \leq j \leq i$.

Definition 2.18 (Perfect information semantics)

$\mathcal{M}, q \models_{\mathsf{Ix}} p$	iff p is in $\pi(q)$;
$\mathcal{M}, q \models_{\mathsf{Ix}} \varphi \vee \psi$	iff $\mathcal{M}, q \models_{\mathsf{Ix}} \varphi$ or $\mathcal{M}, q \models_{\mathsf{Ix}} \psi$;
$\mathcal{M}, q \models_{\mathsf{Ix}} \langle\!\langle A \rangle\!\rangle \Phi$	iff there is a collective **Ix-strategy** s_A such that, for each path $\lambda \in out(q, s_A)$, we have $\mathcal{M}, \lambda \models_{\mathsf{Ix}} \Phi$.
$\mathcal{M}, \lambda \models_{\mathsf{Ix}} \bigcirc \varphi$	iff $\mathcal{M}, \lambda[1, \infty] \models_{\mathsf{Ix}} \varphi$;
$\mathcal{M}, \lambda \models_{\mathsf{Ix}} \Diamond \varphi$	iff $\mathcal{M}, \lambda[i, \infty] \models_{\mathsf{Ix}} \varphi$ for some $i \geq 0$;
$\mathcal{M}, \lambda \models_{\mathsf{Ix}} \Box \varphi$	iff $\mathcal{M}, \lambda[i, \infty] \models_{\mathsf{Ix}} \varphi$ for all $i \geq 0$;
$\mathcal{M}, \lambda \models_{\mathsf{Ix}} \varphi \mathcal{U} \psi$	iff $\mathcal{M}, \lambda[i, \infty] \models_{\mathsf{Ix}} \psi$ for some $i \geq 0$, and $\mathcal{M}, \lambda[j, \infty] \models_{\mathsf{Ix}} \varphi$ forall $0 \leq j \leq i$.

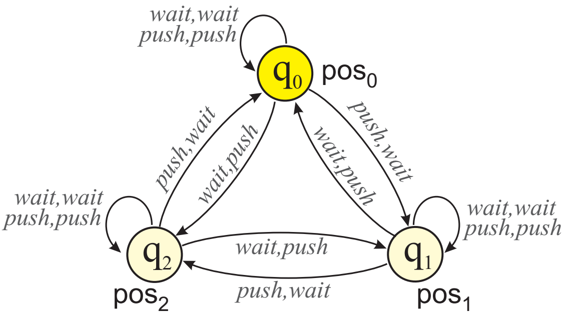
Note that temporal formulae and the Boolean connectives are handled as before.

Example: Robots and Carriage



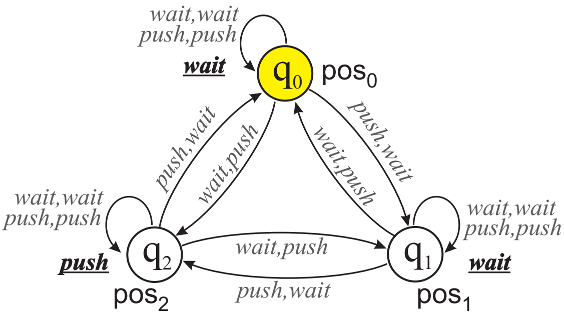
$$\mathsf{pos}_0 \rightarrow \langle\!\langle 1 \rangle\!\rangle \Box \neg \mathsf{pos}_1$$

Example: Robots and Carriage



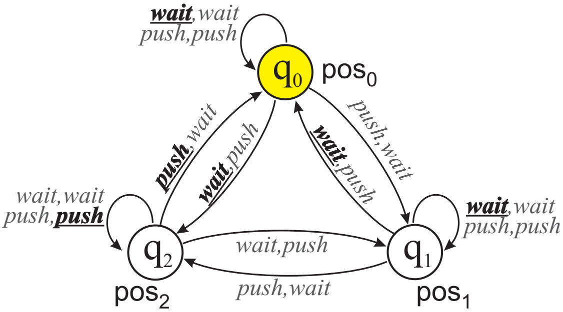
$$pos_0 \rightarrow \langle\langle 1 \rangle\rangle \Box \neg pos_1$$

Example: Robots and Carriage



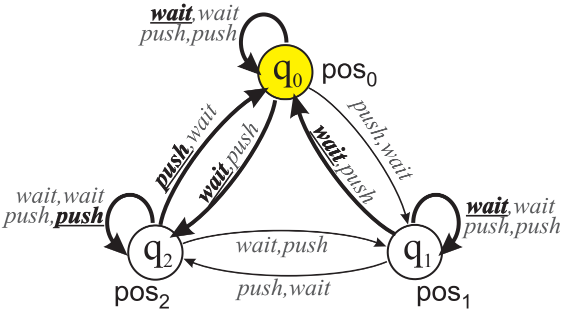
$$\mathsf{pos}_0 \rightarrow \langle\langle 1 \rangle\rangle \Box \neg \mathsf{pos}_1$$

Example: Robots and Carriage



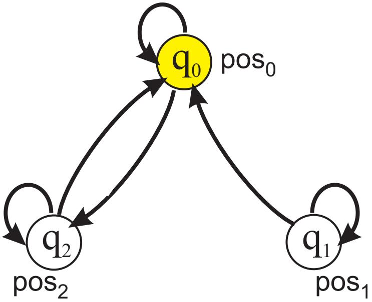
$$\mathsf{pos}_0 \rightarrow \langle\langle 1 \rangle\rangle \Box \neg \mathsf{pos}_1$$

Example: Robots and Carriage



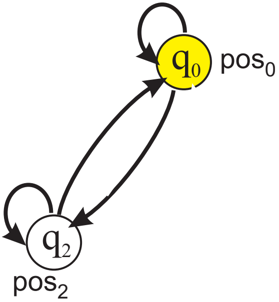
$$\mathsf{pos}_0 \rightarrow \langle\!\langle 1 \rangle\!\rangle \Box \neg \mathsf{pos}_1$$

Example: Robots and Carriage



$$\mathsf{pos}_0 \rightarrow \langle\langle 1 \rangle\rangle \Box \neg \mathsf{pos}_1$$

Example: Robots and Carriage



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

Definition 2.19 ($\mathbf{ATL}_{Ix}$, $\mathbf{ATL}^{+}_{Ix}$, $\mathbf{ATL}^{*}_{Ix}$, ATL, $\mathbf{ATL}^{*}$)

We define **$\mathbf{ATL}_{Ix}$**, **$\mathbf{ATL}^{+}_{Ix}$**, and **$\mathbf{ATL}^{*}_{Ix}$** as the logics $(\mathcal{L}_{\mathsf{ATL}}, \models_{Ix})$, $(\mathcal{L}_{\mathsf{ATL}^+}, \models_{Ix})$ and $(\mathcal{L}_{\mathsf{ATL}^*}, \models_{Ix})$ where $x \in \{r, R\}$, respectively. Moreover, we use **ATL** (resp. **$\mathbf{ATL}^{*}$**) as an abbreviation for **$\mathbf{ATL}_{IR}$** (resp. **$\mathbf{ATL}^{*}_{IR}$**).

Definition 2.19 ($\mathbf{ATL}_{Ix}$, $\mathbf{ATL}^{+}_{Ix}$, $\mathbf{ATL}^{*}_{Ix}$, ATL, $\mathbf{ATL}^{*}$)

We define **$\mathbf{ATL}_{Ix}$**, **$\mathbf{ATL}^{+}_{Ix}$**, and **$\mathbf{ATL}^{*}_{Ix}$** as the logics $(\mathcal{L}_{\mathsf{ATL}}, \models_{Ix})$, $(\mathcal{L}_{\mathsf{ATL}^{+}}, \models_{Ix})$ and $(\mathcal{L}_{\mathsf{ATL}^{*}}, \models_{Ix})$ where $x \in \{r, R\}$, respectively. Moreover, we use **ATL** (resp. **$\mathbf{ATL}^{*}$**) as an abbreviation for **$\mathbf{ATL}_{IR}$** (resp. **$\mathbf{ATL}^{*}_{IR}$**).

Intuitively, a logic is given by the set of all valid formulae.

Definition 2.19 ($\mathbf{ATL}_{Ix}$, $\mathbf{ATL}^+_{Ix}$, $\mathbf{ATL}^*_{Ix}$, ATL, $\mathbf{ATL}^*$)

We define **$\mathbf{ATL}_{Ix}$**, **$\mathbf{ATL}^+_{Ix}$**, and **$\mathbf{ATL}^*_{Ix}$** as the logics $(\mathcal{L}_{\mathsf{ATL}}, \models_{Ix})$, $(\mathcal{L}_{\mathsf{ATL}^+}, \models_{Ix})$ and $(\mathcal{L}_{\mathsf{ATL}^*}, \models_{Ix})$ where $x \in \{r, R\}$, respectively. Moreover, we use **ATL** (resp. **$\mathbf{ATL}^*$**) as an abbreviation for **$\mathbf{ATL}_{IR}$** (resp. **$\mathbf{ATL}^*_{IR}$**).

Intuitively, a logic is given by the set of all valid formulae.

Theorem 2.20

For $\mathcal{L}_{ATL}$, the perfect recall semantics is equivalent to the memoryless semantics under perfect information*, i.e., $\mathcal{M}, q \models_{IR} \varphi$ iff $\mathcal{M}, q \models_{Ir} \varphi$. Both semantics are different for $\mathcal{L}_{ATL^*}$. That is*

$$ATL = ATL_{Ir} = ATL_{IR}.$$

Theorem 2.20

For $\mathcal{L}_{ATL}$, the perfect recall semantics is equivalent to the memoryless semantics under perfect information, *i.e., $\mathcal{M}, q \models_{IR} \varphi$ iff $\mathcal{M}, q \models_{Ir} \varphi$. Both semantics are different for $\mathcal{L}_{ATL^*}$. That is*

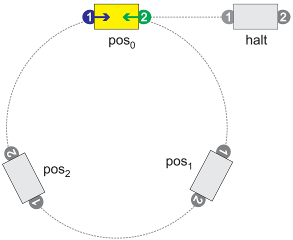
$$ATL = ATL_{Ir} = ATL_{IR}.$$

Proof idea.

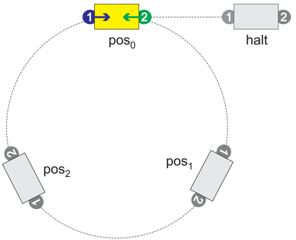
The first “non-looping part” of each path has to satisfy a formula. $\rightsquigarrow$ Exercise □

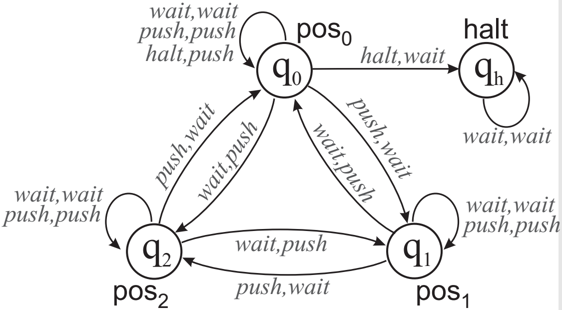
The property has been first observed in [Schobbens, 2004] but it follows from [Alur et al., 2002] in a straightforward way.

Example: Robots and Carriage (2)



Example: Robots and Carriage (2)



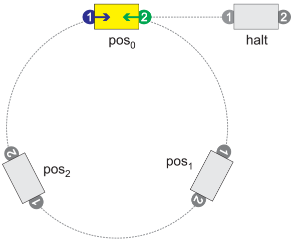


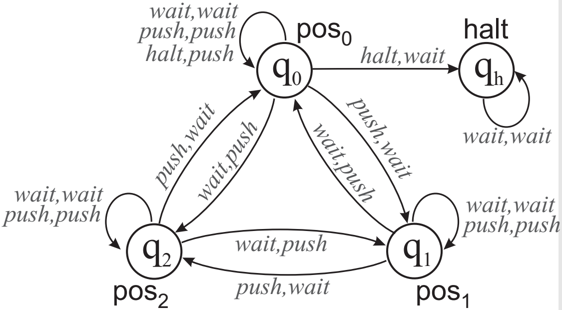
What about $\langle\langle 1,2\rangle\rangle(\Diamond \mathsf{pos}_1 \wedge \Diamond \mathsf{halt})$?

$\mathcal{M}, q_0 \models_{IR} \langle\langle 1,2\rangle\rangle(\Diamond \mathsf{pos}_1 \wedge \Diamond \mathsf{halt})$

$\mathcal{M}, q_0 \quad {}_{Ir} \langle\langle 1,2\rangle\rangle(\Diamond \mathsf{pos}_1 \wedge \Diamond \mathsf{halt})$

Example: Robots and Carriage (2)





What about $\langle\!\langle 1,2 \rangle\!\rangle(\Diamond \mathsf{pos}_1 \land \Diamond \mathsf{halt})$?

$$\mathcal{M}, q_0 \models_{IR} \langle\!\langle 1,2 \rangle\!\rangle(\Diamond \mathsf{pos}_1 \land \Diamond \mathsf{halt})$$
$$\mathcal{M}, q_0 \not\models_{Ir} \langle\!\langle 1,2 \rangle\!\rangle(\Diamond \mathsf{pos}_1 \land \Diamond \mathsf{halt})$$

2.4 Imperfect Information

Imperfect information

How can we reason about agents/extensive games with *imperfect information*?

Imperfect information

How can we reason about agents/extensive games with *imperfect information*?

We combine **ATL*** and epistemic logic.

- We extend CGSs with *indistinguishability relations* $\sim_a \subseteq St \times St$, one per agent. The relations are assumed to be equivalence relations.

Imperfect information

How can we reason about agents/extensive games with imperfect information?

We combine **ATL*** and epistemic logic.

- We extend CGSs with indistinguishability relations $\sim_a \subseteq St \times St$, one per agent. The relations are assumed to be equivalence relations.

- We interpret $\langle\langle A \rangle\rangle$ epistemically ($\rightsquigarrow \models_{iR}$ and $\models_{ir}$)

Definition 2.21 (CEGS)

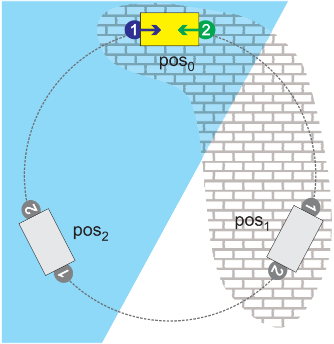
A concurrent epistemic game structure (CEGS) is a tuple

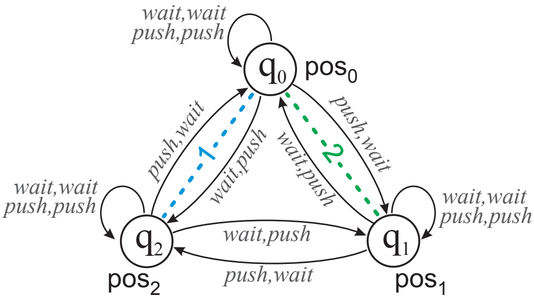
$$\mathcal{M} = (\mathbb{A}\mathrm{gt}, St, \Pi, \pi, Act, d, o, \{\sim_a \mid a \in \mathbb{A}\mathrm{gt}\})$$

with

- $(\mathbb{A}\mathrm{gt}, St, \Pi, \pi, Act, d, o)$ a CGS and
- $\sim_a \subseteq St \times St$ equivalence relations (indistinguishability relations).

Example: Robots and Carriage



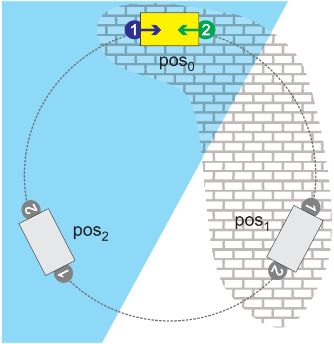


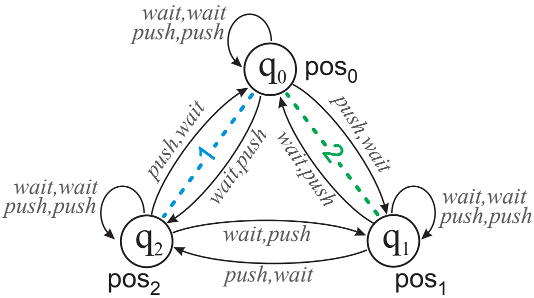
What about $\langle\langle \mathbb{A}\mathrm{gt} \rangle\rangle \bigcirc \mathsf{pos}_1$ in q_0?

$\mathcal{M}, q_0 \qquad {}_{Ir}\langle\langle \mathbb{A}\mathrm{gt} \rangle\rangle \bigcirc \mathsf{pos}_1$

$\mathcal{M}, q_0 \qquad {}_{ir}\langle\langle \mathbb{A}\mathrm{gt} \rangle\rangle \bigcirc \mathsf{pos}_1$

Example: Robots and Carriage



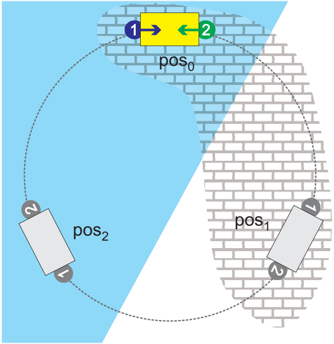


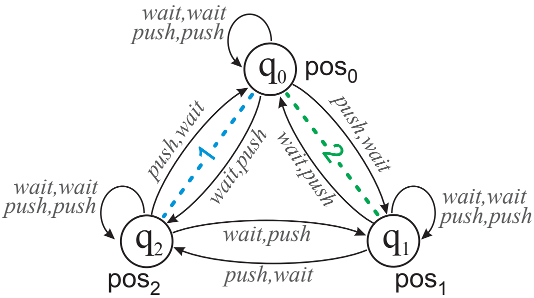
What about $\langle\langle \mathbb{A}gt\rangle\rangle \bigcirc \mathsf{pos}_1$ in q_0?

$\mathcal{M}, q_0 \models_{Ir} \langle\langle \mathbb{A}gt\rangle\rangle \bigcirc \mathsf{pos}_1$

$\mathcal{M}, q_0 \quad {}_{ir}\langle\langle \mathbb{A}gt\rangle\rangle \bigcirc \mathsf{pos}_1$

Example: Robots and Carriage





What about $\langle\!\langle \mathbb{A}\mathrm{gt} \rangle\!\rangle \bigcirc \mathsf{pos}_1$ in q_0?

$\mathcal{M}, q_0 \models_{Ir} \langle\!\langle \mathbb{A}\mathrm{gt} \rangle\!\rangle \bigcirc \mathsf{pos}_1$

$\mathcal{M}, q_0 \not\models_{ir} \langle\!\langle \mathbb{A}\mathrm{gt} \rangle\!\rangle \bigcirc \mathsf{pos}_1$

Problem:

Strategic and epistemic abilities are not independent!

Problem:

Strategic and epistemic abilities are not independent!

$\langle\langle A\rangle\rangle\Phi$ = A can enforce Φ

Problem:

Strategic and epistemic abilities are not independent!

$\langle\langle A \rangle\rangle \Phi$ = A can enforce Φ

It should at least mean that A are able to identify and execute the right strategy!

Problem:

Strategic and epistemic abilities are not independent!

$\langle\langle A\rangle\rangle\Phi$ = A can enforce Φ

It should at least mean that A are able to identify and execute the right strategy!

Executable strategies = uniform strategies

Definition 2.22 (Uniform strategy)

Strategy s_a is uniform iff it specifies the same choices for indistinguishable situations :

- Memoryless strategies:

$$\text{if } q \sim_a q' \text{ then } s_a(q) = s_a(q').$$

Definition 2.22 (Uniform strategy)

Strategy s_a is uniform iff it specifies the same choices for indistinguishable situations :

- Memoryless strategies:

$$\text{if } q \sim_a q' \text{ then } s_a(q) = s_a(q').$$

- Perfect recall:

$$\text{if } \lambda \approx_a \lambda' \text{ then } \Rightarrow s_a(\lambda) = s_a(\lambda'),$$

where $\lambda \approx_a \lambda'$ iff $\lambda[i] \sim_a \lambda'[i]$ for every i.

Definition 2.22 (Uniform strategy)

Strategy s_a is uniform iff it specifies the same choices for indistinguishable situations :

- Memoryless strategies:

$$\text{if } q \sim_a q' \text{ then } s_a(q) = s_a(q').$$

- Perfect recall:

$$\text{if } \lambda \approx_a \lambda' \text{ then} \Rightarrow s_a(\lambda) = s_a(\lambda'),$$

where $\lambda \approx_a \lambda'$ iff $\lambda[i] \sim_a \lambda'[i]$ for every i.

A collective strategy is uniform iff it consists only of uniform individual strategies.

Imperfect Information Strategies

Definition 2.23 (*IR*- and *Ir*-strategies)

- A imperfect information perfect recall strategy for agent a (*iR*-strategy for short) is a uniform *IR*-strategy.

Imperfect Information Strategies

Definition 2.23 (*IR*- and *Ir*-strategies)

- A imperfect information perfect recall strategy for agent a (*iR*-strategy for short) is a uniform *IR*-strategy.
- A imperfect information memoryless strategy for agent a (*ir*-strategy for short) is a uniform *Ir*-strategy.

Imperfect Information Strategies

Definition 2.23 (*IR*- and *Ir*-strategies)

- A imperfect information perfect recall strategy for agent a (*iR*-strategy for short) is a uniform *IR*-strategy.
- A imperfect information memoryless strategy for agent a (*ir*-strategy for short) is a uniform *Ir*-strategy.

The outcome is defined as before.

Imperfect Information Semantics

The imperfect information semantics is defined as before, only the clause for

$\mathcal{M}, q \models_{\mathsf{Ix}} \langle\langle A \rangle\rangle \varphi$ iff there is a collective Ix-strategy s_A such that, for each path $\lambda \in out(q, s_A)$, we have $\mathcal{M}, \lambda \models_{\mathsf{Ix}} \varphi$.

is replaced by

Imperfect Information Semantics

The imperfect information semantics is defined as before, only the clause for

$\mathcal{M}, q \models_{\mathsf{Ix}} \langle\langle A \rangle\rangle \varphi$ iff there is a collective Ix-strategy s_A such that, for each path $\lambda \in out(q, s_A)$, we have $\mathcal{M}, \lambda \models_{\mathsf{Ix}} \varphi$.

is replaced by

$\mathcal{M}, q \models_{\mathsf{ix}} \langle\langle A \rangle\rangle \varphi$ iff there is a uniform ix-strategy s_A such that, for each path $\lambda \in \bigcup_{q':q\sim_A q'} out(q', s_A)$, we have $\mathcal{M}, \lambda \models_{\mathsf{ix}} \varphi$

where $x \in \{r, R\}$ and $\sim_A := \cup_{a\in A} \sim_a$.

Remark 2.24

The last definition models that "everybody in A knows that φ".

Remark 2.24

The last definition models that "everybody in A knows that φ".

The fixed-point characterisation does not hold anymore!

Theorem 2.25

*The following formulae are not valid for **ATL$_{ir}$**:*

- $\langle\langle A\rangle\rangle\Box\varphi \quad \leftrightarrow \quad \varphi \wedge \langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\Box\varphi$
- $\langle\langle A\rangle\rangle\varphi_1\,\mathcal{U}\,\varphi_2 \quad \leftrightarrow \quad \varphi_2 \vee (\varphi_1 \wedge \langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\varphi_1\,\mathcal{U}\,\varphi_2)$.

Proof.

$\rightsquigarrow$: Exercise. $\square$

Proof idea

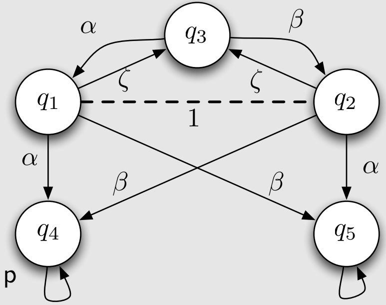
We construct a counterexample for

$$\langle\langle 1\rangle\rangle\Diamond p \quad \leftrightarrow \quad p \quad \vee \quad \langle\langle 1\rangle\rangle\bigcirc\langle\langle 1\rangle\rangle\Diamond p$$

Proof idea

We construct a counterexample for

$\langle\langle 1\rangle\rangle\Diamond \mathsf{p} \quad \leftrightarrow \quad \mathsf{p} \quad \vee$
$\langle\langle 1\rangle\rangle\bigcirc \langle\langle 1\rangle\rangle\Diamond \mathsf{p}$

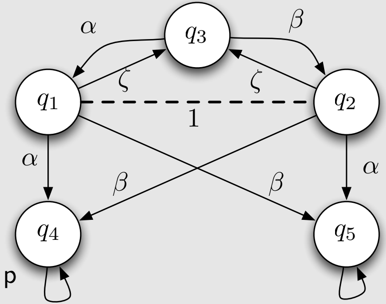


Proof idea

We construct a counterexample for

$\langle\langle 1 \rangle\rangle \Diamond \mathsf{p} \leftrightarrow \mathsf{p} \vee \langle\langle 1 \rangle\rangle \bigcirc \langle\langle 1 \rangle\rangle \Diamond \mathsf{p}$

q$\mathcal{M}, q_1 \not\models_{ir} \langle\langle 1 \rangle\rangle \Diamond \mathsf{p}$ iff

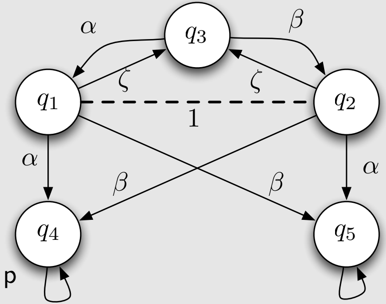


Proof idea

We construct a counterexample for

$\langle\!\langle 1\rangle\!\rangle\Diamond \mathsf{p} \quad \leftrightarrow \quad \mathsf{p} \;\vee\; \langle\!\langle 1\rangle\!\rangle\bigcirc \langle\!\langle 1\rangle\!\rangle\Diamond \mathsf{p}$

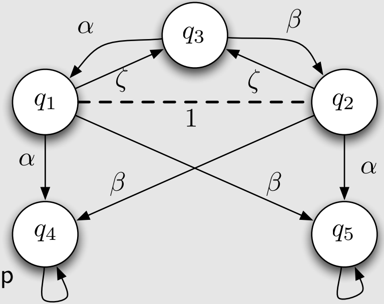
q$\mathcal{M}, q_1 \not\models_{ir} \langle\!\langle 1\rangle\!\rangle\Diamond \mathsf{p}$ iff
not ($\exists s \in \Sigma_u^{\mathrm{ir}}$



Proof idea

We construct a counterexample for

$\langle\langle 1\rangle\rangle\Diamond \mathsf{p} \quad \leftrightarrow \quad \mathsf{p} \quad \vee$
$\langle\langle 1\rangle\rangle\bigcirc\langle\langle 1\rangle\rangle\Diamond \mathsf{p}$

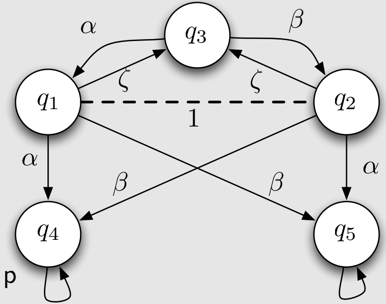


q$\mathcal{M}, q_1 \not\models_{ir} \langle\langle 1\rangle\rangle\Diamond \mathsf{p}$ iff
not ($\exists s \in \Sigma_u^{ir}\ \forall\lambda \in \bigcup_{q\in\{q_1,q_2\}} out(q,s)$

Proof idea

We construct a counterexample for

$\langle\!\langle 1\rangle\!\rangle\Diamond \mathsf{p} \leftrightarrow \mathsf{p} \vee \langle\!\langle 1\rangle\!\rangle\bigcirc\langle\!\langle 1\rangle\!\rangle\Diamond \mathsf{p}$

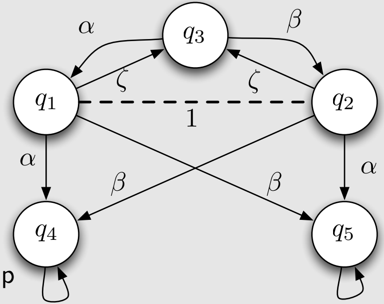


q$\mathcal{M}, q_1 \not\models_{ir} \langle\!\langle 1\rangle\!\rangle\Diamond$p iff
not ($\exists s \in \Sigma_u^{ir}\ \forall\lambda \in \bigcup_{q\in\{q_1,q_2\}} out(q,s) \exists i \in \mathbb{N}_0$:
$\mathcal{M}, \lambda[i] \models_{ir}$ p)

Proof idea

We construct a counterexample for

$\langle\!\langle 1\rangle\!\rangle\Diamond \mathsf{p} \quad \leftrightarrow \quad \mathsf{p} \;\vee\; \langle\!\langle 1\rangle\!\rangle\bigcirc\langle\!\langle 1\rangle\!\rangle\Diamond \mathsf{p}$



q$\mathcal{M}, q_1 \not\models_{ir} \langle\!\langle 1\rangle\!\rangle\Diamond \mathsf{p}$ iff
not ($\exists s \in \Sigma_u^{\mathsf{ir}}\ \forall\lambda \in \bigcup_{q\in\{q_1,q_2\}} out(q,s)\exists i \in \mathbb{N}_0$:
$\mathcal{M}, \lambda[i] \models_{ir} \mathsf{p}$)

$\mathcal{M}, q_1 \models_{ir} \mathsf{p} \vee \langle\!\langle 1\rangle\!\rangle\bigcirc\langle\!\langle 1\rangle\!\rangle\Diamond \mathsf{p}$

2.5 Dynamic Logics

1^{st} idea: Consider **actions** or **atomic programs** α. Each such α defines a transition (**accessibility relation**) from worlds into worlds.

1st idea: Consider **actions** or **atomic programs** α. Each such α defines a transition (**accessibility relation**) from worlds into worlds.

2nd idea: We need statements about the outcome of actions:

- $[\alpha]\varphi$: "after **each execution** of α, φ holds,
- $\langle\alpha\rangle\varphi$: "after **some executions** of α, φ holds.

1st idea: Consider **actions** or **atomic programs** α. Each such α defines a transition (**accessibility relation**) from worlds into worlds.

2nd idea: We need statements about the outcome of actions:

- $[\alpha]\varphi$: "after **each execution** of α, φ holds,
- $\langle\alpha\rangle\varphi$: "after **some executions** of α, φ holds.

As usual, $\langle\alpha\rangle\varphi \equiv \neg[\alpha]\neg\varphi$.

3^{rd} idea: Programs/actions can be combined (sequentially, nondeterministically, iteratively), e.g.:

$$[\alpha;\beta]\varphi$$

would mean "after each execution of α **and then** β, formula φ holds".

Can we combine these three ideas and come up with a language and logic where we can express all these features?

Dynamic Logic over arbitrary programs

Example 2.26 (Propositional Dynamic Logic)

Infinite collection of diamonds: $\mathcal{O}p = \{\pi \mid \pi \text{ is a program}\}$

What do the following operators express?

$\langle\pi\rangle\varphi$:

$[\pi]\varphi$:

Dynamic Logic over arbitrary programs

Example 2.26 (Propositional Dynamic Logic)

Infinite collection of diamonds: $\mathcal{O}p = \{\pi \mid \pi \text{ is a program}\}$

What do the following operators express?

$\langle\pi\rangle\varphi$: Some terminating execution of π leads to a state with information φ

$[\pi]\varphi$:

Dynamic Logic over arbitrary programs

Example 2.26 (Propositional Dynamic Logic)

Infinite collection of diamonds: $\mathcal{O}p = \{\pi \mid \pi$ is a program$\}$

What do the following operators express?

$\langle\pi\rangle\varphi$: Some terminating execution of π leads to a state with information φ

$[\pi]\varphi$: Each terminating execution of π leads to a state with information φ

Dynamic Logic over arbitrary programs

Example 2.26 (Propositional Dynamic Logic)

Infinite collection of diamonds: $\mathcal{O}p = \{\pi \mid \pi \text{ is a program}\}$

What do the following operators express?

$\langle\pi\rangle\varphi$: Some terminating execution of π leads to a state with information φ

$[\pi]\varphi$: Each terminating execution of π leads to a state with information φ

It would be nice if we could combine simple programs:

Dynamic Logic over arbitrary programs

Example 2.26 (Propositional Dynamic Logic)

Infinite collection of diamonds: $\mathcal{O}p = \{\pi \mid \pi \text{ is a program}\}$

What do the following operators express?

$\langle\pi\rangle\varphi$: Some terminating execution of π leads to a state with information φ

$[\pi]\varphi$: Each terminating execution of π leads to a state with information φ

It would be nice if we could combine simple programs:

$\pi \cup \pi'$: **Nondeterministic choice**

Dynamic Logic over arbitrary programs

Example 2.26 (Propositional Dynamic Logic)

Infinite collection of diamonds: $\mathcal{O}p = \{\pi \mid \pi \text{ is a program}\}$

What do the following operators express?

$\langle\pi\rangle\varphi$: Some terminating execution of π leads to a state with information φ

$[\pi]\varphi$: Each terminating execution of π leads to a state with information φ

It would be nice if we could combine simple programs:

$\pi \cup \pi'$: **Nondeterministic choice**

$\pi ; \pi'$: **Sequential composition**

Dynamic Logic over arbitrary programs

Example 2.26 (Propositional Dynamic Logic)

Infinite collection of diamonds: $\mathcal{O}p = \{\pi \mid \pi \text{ is a program}\}$

What do the following operators express?

$\langle\pi\rangle\varphi$: Some terminating execution of π leads to a state with information φ

$[\pi]\varphi$: Each terminating execution of π leads to a state with information φ

It would be nice if we could combine simple programs:

$\pi \cup \pi'$: **Nondeterministic choice**

$\pi; \pi'$: **Sequential composition**

π^* : **Iterative execution**

What do the following statements express?

$\langle \pi^* \rangle \varphi \leftrightarrow \varphi \vee \langle \pi; \pi^* \rangle \varphi$:

$[\pi^*](\varphi \to [\pi]\varphi) \to (\varphi \to [\pi^*]\varphi)$:

What do the following statements express?

$\langle \pi^* \rangle \varphi \leftrightarrow \varphi \vee \langle \pi ; \pi^* \rangle \varphi$: A state with information φ is reached by executing π a finite number of times iff the current state satisfies φ or we can execute π once and reach a state in which φ holds by executing π a finite number of times.

$[\pi^*](\varphi \rightarrow [\pi]\varphi) \rightarrow (\varphi \rightarrow [\pi^*]\varphi)$:

What do the following statements express?

$\langle\pi^*\rangle\varphi \leftrightarrow \varphi \vee \langle\pi;\pi^*\rangle\varphi$: A state with information φ is reached by executing π a finite number of times iff the current state satisfies φ or we can execute π once and reach a state in which φ holds by executing π a finite number of times.

$[\pi^*](\varphi \rightarrow [\pi]\varphi) \rightarrow (\varphi \rightarrow [\pi^*]\varphi)$: $\rightsquigarrow$ Exercise.

What do the following statements express?

$\langle\pi^*\rangle\varphi \leftrightarrow \varphi \vee \langle\pi;\pi^*\rangle\varphi$: A state with information φ is reached by executing π a finite number of times iff the current state satisfies φ or we can execute π once and reach a state in which φ holds by executing π a finite number of times.

$[\pi^*](\varphi \rightarrow [\pi]\varphi) \rightarrow (\varphi \rightarrow [\pi^*]\varphi)$: $\rightsquigarrow$ Exercise.

Do these formulae always hold?

How can we actually use this logic?

Dynamic Logic Models

A model is simply a Kripke structure where each atomic program constitutes an accessibility relation.

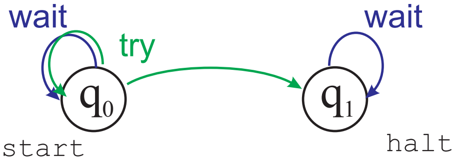
Definition 2.27 (Labelled Transition System)

A labelled transition system is a pair

$$\langle St, \{\xrightarrow{\alpha}: \ \alpha \in \mathbf{L}\}\rangle$$

where St is a non-empty set of states and $\mathbf{L}$ is a non-empty set of labels and for each $\alpha \in \mathbf{L}$: $\xrightarrow{\alpha} \subseteq St \times St$.

What are concrete examples of such systems?


wait
try
wait
q0
q1
start
halt

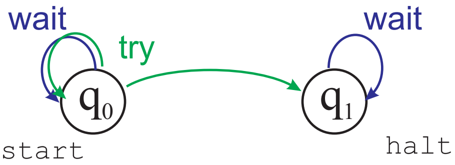
Definition 2.28 (Dynamic Logic Model)

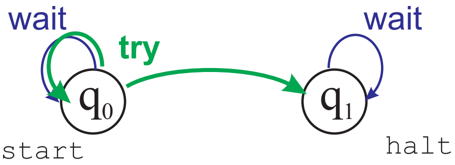
A model of propositional dynamic logic is given by a labelled transition systems and a valuation of propositions.

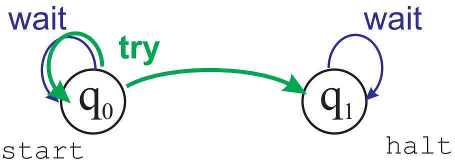
For atomic programs α, the semantics is easily defined:

Definition 2.29 (Semantics of DL)

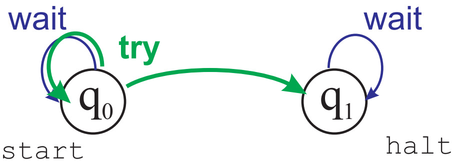
$\mathcal{M}, s \models [\alpha]\varphi$ iff for all t such that $s \xrightarrow{\alpha} t$, we have $\mathcal{M}, t \models \varphi$.


wait
try
wait
q0
q1
start
halt

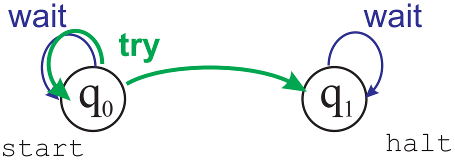

wait
try
wait
q_0
q_1
start
halt



$$\mathsf{start} \rightarrow \langle try \rangle \mathsf{halt}$$



$$\text{start} \rightarrow \langle try \rangle \text{halt}$$
$$\text{start} \rightarrow \neg[try]\text{halt}$$



$$\text{start} \rightarrow \langle try \rangle \text{halt}$$
$$\text{start} \rightarrow \neg [try] \text{halt}$$
$$\text{start} \rightarrow \langle try \rangle [wait] \text{halt}$$

But what if we want to consider **complex programs**? First of all, we have to make sure that we can build such programs.

Definition 2.30 (Composite labels)

We require that the set of labels forms a Kleene algebra $\langle \mathbf{L}, ;, \cup, * \rangle$. We also assume that the set of labels contains constructs of the form "φ?", whenever φ is a formula not involving any modalities.

What has this to do with programs?

- "$;$" means sequential composition,
- "$\cup$" means nondeterministic choice,
- "$*$" means finite iteration (regular expr.),
- "φ**?**" means test.

if φ **then** a **else** b	$(\varphi\mathbf{?;}a) \cup (\neg\varphi\mathbf{?;}b)$
while φ **do** a	$(\varphi\mathbf{?;}a)^* \,;\, (\neg\varphi\mathbf{?})$

Definition 2.31 (Condition on Labels)

We assume that the labels obey the following conditions:

- $s \xrightarrow{\alpha;\beta} t$ iff $s \xrightarrow{\alpha} s'$ and $s' \xrightarrow{\beta} t$,
- $s \xrightarrow{\alpha\cup\beta} t$ iff $s \xrightarrow{\alpha} t$ or $s \xrightarrow{\beta} t$,
- $s \xrightarrow{\alpha^*} t$ is the reflexive and transitive closure of $s \xrightarrow{\alpha} t$,
- $s \xrightarrow{\varphi?} t$ iff $s = t$ and $s \models_{\mathcal{M}} \varphi$.

We are now ready to define the semantics of DL for arbitrary complex expressions of labels.

Definition 2.32 (Semantics of DL)

We assume that the set of labels forms a Kleene algebra and that the conditions of Definition 2.31 hold. Then we define, as in Definition 2.29:

$\mathcal{M}, s \models [\alpha]\varphi$ iff for all t st. $s \xrightarrow{\alpha} t$, we have $\mathcal{M}, t \models \varphi$.

- One of the most appealing aspects of dynamic logic is the close link to **Hoare Logic**, and **partial correctness assertions** in general [Parikh, 1979].
- Thus, $\{p\}\alpha\{q\}$ in Hoare Logic can be expressed as $p \Rightarrow [\alpha]q$ in PDL, while termination of a program α can be expressed by $\langle\alpha\rangle\top$.
- These aspects make dynamic logic a viable alternative to temporal logic in providing the basis for agent specification formalisms.

3. From Specification to Implementation

We have seen how a logical formalism can be used to specify agent behaviour.

But there remains a gap between such a specification and an actual implemented agent system.

How might we bridge this gap?

We have seen how a logical formalism can be used to specify agent behaviour.

But there remains a gap between such a specification and an actual implemented agent system.

How might we bridge this gap? And bridge it reliably?

We have seen how a logical formalism can be used to specify agent behaviour.

But there remains a gap between such a specification and an actual implemented agent system.

How might we bridge this gap? And bridge it reliably?

Approaches we might use include:

- formal verification
- refinement
- synthesis
- direct execution

We have seen how a logical formalism can be used to specify agent behaviour.

But there remains a gap between such a specification and an actual implemented agent system.

How might we bridge this gap? And bridge it reliably?

Approaches we might use include:

- formal verification
- refinement
- synthesis
- direct execution

We will briefly review these next.

3.1 Checking Implementations

Towards Formal Verification

The most likely way for bridging the gap is for someone else to implement an agent.

In most cases such implementations will be developed by **informal approaches**, such as traditional software engineering methods.

In this case, a formal specification represents a **formal requirement** that we can check the implementations against.

3.2 Refinement

Refinement

φ_S provides some logical specification of agent behaviour

Refinement

φ_S provides some logical specification of agent behaviour

φ_S might be

- quite vague and high-level, and
- non-deterministic

Refinement

φ_S provides some logical specification of agent behaviour

φ_S might be

- quite vague and high-level, and
- non-deterministic

We can refine this to a new specification, φ_R

φ_R will typically be

- more detailed and specific,
- more deterministic,
- and closer to a 'real' implementation.

Refinement

φ_S provides some logical specification of agent behaviour

φ_S might be

- quite vague and high-level, and
- non-deterministic

We can refine this to a new specification, φ_R

φ_R will typically be

- more detailed and specific,
- more deterministic,
- and closer to a 'real' implementation.

Crucially any behaviour allowed by our refined specification, φ_R, must be allowed within the original specification, φ_S.

Example 1

Originally, we specify the system behaviour to be '$a \vee b$'.

But then refine it (becoming more deterministic) to just 'b'.

Example 1

Originally, we specify the system behaviour to be '$a \lor b$'.

But then refine it (becoming more deterministic) to just 'b'.

Example 2

Imagine we specify a Mammal.

We might later refine this to specify a Dog!

This removes some irrelevant possibilities (e.g. "two-legged") but all behaviours of a dog are still possible behaviours of a mammal.

Formal Aspects of Refinement

In refining φ_S to φ_R, it is typical (and expected) that

$$\vdash \varphi_R \Rightarrow \varphi_S$$

Formal Aspects of Refinement

In refining φ_S to φ_R, it is typical (and expected) that

$$\vdash \varphi_R \Rightarrow \varphi_S$$

So, implementations satisfying φ_R will also still satisfy φ_S.

Formal Aspects of Refinement

In refining φ_S to φ_R, it is typical (and expected) that

$$\vdash \varphi_R \Rightarrow \varphi_S$$

So, implementations satisfying φ_R will also still satisfy φ_S.

But there may well be some implementations allowed by φ_S that are now **disallowed** by φ_R.

Formal Aspects of Refinement

In refining φ_S to φ_R, it is typical (and expected) that

$$\vdash \varphi_R \Rightarrow \varphi_S$$

So, implementations satisfying φ_R will also still satisfy φ_S.

But there may well be some implementations allowed by φ_S that are now **disallowed** by φ_R.

Two things are important here:

1. whatever logical properties we established of φ_S can, because we know that $\varphi_R \Rightarrow \varphi_S$, also be established of φ_R; and

Formal Aspects of Refinement

In refining φ_S to φ_R, it is typical (and expected) that

$$\vdash \varphi_R \Rightarrow \varphi_S$$

So, implementations satisfying φ_R will also still satisfy φ_S.

But there may well be some implementations allowed by φ_S that are now **disallowed** by φ_R.

Two things are important here:

1. whatever logical properties we established of φ_S can, because we know that $\varphi_R \Rightarrow \varphi_S$, also be established of φ_R; and
2. φ_R is **more detailed**, more deterministic, or at least closer to a possible implementation on the agent.

Example

Our original specification, φ_M, is for a Mammal.

We develop a refinement, φ_D, specifying a Dog.

All dogs are mammals, so we know $\varphi_D \Rightarrow \varphi_M$.

Now we refine still further to give, φ_P, specifying a Poodle.

Since all poodles are dogs, then $\varphi_P \Rightarrow \varphi_D$.

Example

Our original specification, φ_M, is for a Mammal.

We develop a refinement, φ_D, specifying a Dog.

All dogs are mammals, so we know $\varphi_D \Rightarrow \varphi_M$.

Now we refine still further to give, φ_P, specifying a Poodle.

Since all poodles are dogs, then $\varphi_P \Rightarrow \varphi_D$.

We might have proved a property of mammals, for example having "warm-blood" but do not have to prove this again for poodles, since we know

$$\varphi_P \Rightarrow \varphi_D \qquad \varphi_D \Rightarrow \varphi_M \qquad \varphi_M \Rightarrow \text{“warm-blood”}$$

Refinement Process

$$\begin{array}{c} \varphi_S \\ \Uparrow \\ \varphi_R \\ \Uparrow \\ \varphi_{R_1} \\ \Uparrow \\ \varphi_{R_2} \\ \vdots \\ \varphi_{R_N} \end{array}$$

Thus, we can develop a series of refinements, φ_{R_1}, φ_{R_2}, φ_{R_3}, $\ldots$, successively moving us towards an implementation in a formally defined way [Mili et al., 1986].

Any of these refinements satisfies the logical properties of the original specification.

Refinement Process

$$\begin{array}{c} \varphi_S \\ \Uparrow \\ \varphi_R \\ \Uparrow \\ \varphi_{R_1} \\ \Uparrow \\ \varphi_{R_2} \\ \vdots \\ \varphi_{R_N} \end{array}$$

Thus, we can develop a series of refinements, φ_{R_1}, φ_{R_2}, φ_{R_3}, $\ldots$, successively moving us towards an implementation in a formally defined way [Mili et al., 1986].

Any of these refinements satisfies the logical properties of the original specification.

There still remains the problem of **getting from a logical specification**, say φ_{R_i}, to an *actual agent implementation*.

3.3 Synthesis

Program Synthesis

Generally, within formal approaches to program development, we are given a program/system, S, and a logical requirement, R, and asked

does S always satisfy R?

Program Synthesis

Generally, within formal approaches to program development, we are given a program/system, S, and a logical requirement, R, and asked

> does S always satisfy R?

With synthesis we are just given R and asked

> can we construct an S that always satisfies R?

Program Synthesis

Generally, within formal approaches to program development, we are given a program/system, S, and a logical requirement, R, and asked

> does S always satisfy R?

With synthesis we are just given R and asked

> can we construct an S that always satisfies R?

Or, even more appealing:

> can we automatically construct an S that always satisfies R?

Program Synthesis

Generally, within formal approaches to program development, we are given a program/system, S, and a logical requirement, R, and asked

> does S always satisfy R?

With synthesis we are just given R and asked

> can we construct an S that always satisfies R?

Or, even more appealing:

> can we automatically construct an S that always satisfies R?

Sounds very appealing ..... but can be very complex.

Agent Synthesis?

Ideally, we would like to automatically synthesise an agent program directly from an agent specification.

This sounds ideal, especially if we can guarantee that the agent will definitely implement its specification.

This is, of course, a very appealing direction in traditional formal methods [Manna and Waldinger, 1971].

A typical approach is to synthesise a finite state automaton from a logical (usually temporal) specification [Pnueli and Rosner, 1989b].

In some cases this can be automatic and effective.

However: the complexity of this is often very large, e.g. 2-EXPTIME.

3.4 Specifications as Programs

Executions

A formal specification essentially characterises a set of models of the entity being specified.

In the case of agents, a logical agent specification describes a **set of agent executions** that satisfy the specification.

So, if we have some process for extracting one (or more) of these models/executions from the specification then this effectively gives us a way of **implementing the formal specification**.

Recap: Logic Programming

Logic Programming provides a mechanism for trying to build a model (execution) of a set of Horn Clauses.

Indeed, we could use many other methods for model-building from a set of Horn Clauses [Kowalski, 1979].

Recap: Logic Programming

Logic Programming provides a mechanism for trying to build a model (execution) of a set of Horn Clauses.

Indeed, we could use many other methods for model-building from a set of Horn Clauses [Kowalski, 1979].

If we wish to do something similar for agent specifications, then we must invoke suitable model-building procedures for the logics underlying these specifications.

Fortunately, the basis for many agent specifications is linear temporal logic and the models of this logic are linear sequences of states which corresponds closely to program executions.

Executable Agent Specifications

1. begin by building models from temporal specifications

Executable Agent Specifications

1. begin by building models from temporal specifications
2. then extending this to agent specifications

Executable Agent Specifications

1. begin by building models from temporal specifications
2. then extending this to agent specifications

But how?

Executable Agent Specifications

1. begin by building models from temporal specifications
2. then extending this to agent specifications

But how?

An obvious first step is to extend the resolution approach that is central to Logic Programming to the temporal logic case.

Executable Agent Specifications

1. begin by building models from temporal specifications
2. then extending this to agent specifications

But how?

An obvious first step is to extend the resolution approach that is central to Logic Programming to the temporal logic case. Unfortunately

$\rightarrow$ this is quite complex and,

Executable Agent Specifications

1. begin by building models from temporal specifications
2. then extending this to agent specifications

But how?

An obvious first step is to extend the resolution approach that is central to Logic Programming to the temporal logic case. Unfortunately

→ this is quite complex and,
→ sometimes gives counter-intuitive results.

Notable languages:

- **Templog** [Abadi and Manna, 1989]; and
- **Chronolog** [Orgun and Wadge, 1992]

Executable Agent Specifications

1. begin by building models from temporal specifications
2. then extending this to agent specifications

But how?

An obvious first step is to extend the resolution approach that is central to Logic Programming to the temporal logic case. Unfortunately

$\rightarrow$ this is quite complex and,
$\rightarrow$ sometimes gives counter-intuitive results.

Notable languages:

- **Templog** [Abadi and Manna, 1989]; and
- **Chronolog** [Orgun and Wadge, 1992]

Both execute a subset of temporal Horn clauses using TSLD-resolution, an extension of classical SLD-resolution.

Basic METATEM Execution

METATEM [Fisher and Hepple, 2009]

- executes temporal specifications, and
- builds the underlying temporal models in the intuitive order, i.e. from the beginning onwards.

Basic METATEM Execution

METATEM [Fisher and Hepple, 2009]

- executes temporal specifications, and
- builds the underlying temporal models in the intuitive order, i.e. from the beginning onwards.

METATEM execution uses a lightweight forward chaining procedure which builds an execution sequence that is a model for the temporal specification.

Basic METATEM Execution

METATEM [Fisher and Hepple, 2009]

- executes temporal specifications, and
- builds the underlying temporal models in the intuitive order, i.e. from the beginning onwards.

METATEM execution uses a lightweight forward chaining procedure which builds an execution sequence that is a model for the temporal specification.

"Imperative Future" approach [Barringer et al., 1996]

$\rightarrow$ built from the beginning, i.e.

$\rightarrow$ the model is constructed step by step, starting from the initial state.

Basic METATEM Execution

METATEM [Fisher and Hepple, 2009]

- executes temporal specifications, and
- builds the underlying temporal models in the intuitive order, i.e. from the beginning onwards.

METATEM execution uses a lightweight forward chaining procedure which builds an execution sequence that is a model for the temporal specification.

"Imperative Future" approach [Barringer et al., 1996]

→ built from the beginning, i.e.

→ the model is constructed step by step, starting from the initial state.

In the basic case this is complete in that the temporal specification for an agent can be executed if, and only if, the specification is satisfiable.

Concurrent METATEM

A temporal specification on its own is not enough, so the basic specification is extended with both beliefs and motivations.

Concurrent METATEM

A temporal specification on its own is not enough, so the basic specification is extended with both beliefs and motivations.

Beliefs provide the information the agent decides upon.

Concurrent METATEM

A temporal specification on its own is not enough, so the basic specification is extended with both beliefs and motivations.

Beliefs provide the information the agent decides upon.

In addition, two varieties of motivations are developed:

- the temporal '$\Diamond$' modality, which provides a very strong motivation since the semantics of '$\Diamond g$' require that g will definitely happen; and
- the combination '$B\Diamond$', where 'B' is the belief operator, which provides a weaker motivation for the agent.

Concurrent METATEM

A temporal specification on its own is not enough, so the basic specification is extended with both beliefs and motivations.

Beliefs provide the information the agent decides upon.

In addition, two varieties of motivations are developed:

- the temporal '$\Diamond$' modality, which provides a very strong motivation since the semantics of '$\Diamond g$' require that g will definitely happen; and
- the combination '$B\Diamond$', where 'B' is the belief operator, which provides a weaker motivation for the agent.

Concurrent MetateM takes a set of such agents, each executing their own formal specifications asynchronously and allows them to communicate, cooperate and self-organize [Fisher, 2011].

4. Formal Verification

4 Formal Verification

- What is Formal Verification?
- Deductive Verification
- Algorithmic Verification
- Program verification
- Run-time verification

Program Analysis: From Testing....

Once we decide to analyze a system with respect to a formal property, there are a number of ways to achieve this.

One, particularly popular, approach is to carry out testing [Ammann and Offutt, 2008].

Program Analysis: From Testing....

Once we decide to analyze a system with respect to a formal property, there are a number of ways to achieve this.

One, particularly popular, approach is to carry out *testing* [Ammann and Offutt, 2008].

$\rightarrow$ the system/program is executed under a specific set of conditions and the execution produced is compared to an expected outcome.

Program Analysis: From Testing....

Once we decide to analyze a system with respect to a formal property, there are a number of ways to achieve this.

One, particularly popular, approach is to carry out *testing* [Ammann and Offutt, 2008].

→ the system/program is executed under a specific set of conditions and the execution produced is compared to an expected outcome.

The skill in testing is to carry this out for enough different conditions so that the developer can be relatively confident that the program/system is indeed correct.

. . . to Formal Verification

While testing is, of course, very useful it only examines a subset of all the possible executions.

What if we want to be sure that the logical specification is met whichever way the program/system executes?

. . . to Formal Verification

While testing is, of course, very useful it only examines a subset of all the possible executions.

What if we want to be sure that the logical specification is met whichever way the program/system executes?

Assessing whether this is the case or not is the core of formal verification.

4.1 What is Formal Verification?

Definitions: Formal Verification

The Latin origin of ‘verification’ is veritas facere: “making something true”. A more recent dictionary definition is

Verification: **additional proof that something that was believed (some fact or hypothesis or theory) is correct**

Definitions: Formal Verification

The Latin origin of 'verification' is veritas facere: "making something true". A more recent dictionary definition is

Verification: **additional proof that something that was believed (some fact or hypothesis or theory) is correct**

Moving on to "formal verification" we find,

Formal Verification: **the act of proving or disproving the correctness of a system with respect to a certain formal specification or property**

Varieties of Formal Verification

So, we essentially want to examine *all* possible executions of our system/program in order to assess whether they all satisfy our formal requirements.

Varieties of Formal Verification

So, we essentially want to examine all possible executions of our system/program in order to assess whether they all satisfy our formal requirements.

- finite set of different executions
  $\rightarrow$ enumerate them all and check their properties

Varieties of Formal Verification

So, we essentially want to examine all possible executions of our system/program in order to assess whether they all satisfy our formal requirements.

- finite set of different executions
  $\rightarrow$ enumerate them all and check their properties

- infinite number of possible executions
  $\rightarrow$ then we must do something more sophisticated

Varieties of Formal Verification

So, we essentially want to examine all possible executions of our system/program in order to assess whether they all satisfy our formal requirements.

- finite set of different executions
  $\rightarrow$ enumerate them all and check their properties

- infinite number of possible executions
  $\rightarrow$ then we must do something more sophisticated

We next overview some alternative formal verification approaches before moving on to these in an agent context.

4.2 Deductive Verification

Deductive Verification

If we have a system with an infinite (or very large) number of possible executions, then a typical approach is to use some logical description to capture the behaviour of our system.

This logical formula, say Sys, is likely to have been devised from the formal semantics of the system/program.

Deductive Verification

If we have a system with an infinite (or very large) number of possible executions, then a typical approach is to use some logical description to capture the behaviour of our system.

This logical formula, say Sys, is likely to have been devised from the formal semantics of the system/program.

If we then have a formal specification of our requirements, say Req given in the same logic, then the aim of **deductive verification** is to prove

$$\vdash \; Sys \Rightarrow Req$$

If this is proved then all executions, characterized by Sys, satisfy the required property, Req.

Of course, logical proof can be difficult.

1. If we are lucky, Sys and Req can be described in a quite simple logic and the formula $Sys \Rightarrow Req$ can be decided in a fast and automated way.

Of course, logical proof can be difficult.

1. If we are lucky, Sys and Req can be described in a quite simple logic and the formula $Sys \Rightarrow Req$ can be decided in a fast and automated way.

   $\rightarrow$ often essential to invoke human intervention and so utilize semi-automated theorem-provers such as
   - **Isabelle** [Paulson, 1994] and
   - **PVS** [Owre et al., 1998].

Of course, logical proof can be difficult.

1. If we are lucky, Sys and Req can be described in a quite simple logic and the formula $Sys \Rightarrow Req$ can be decided in a fast and automated way.
   - $\rightarrow$ often essential to invoke human intervention and so utilize semi-automated theorem-provers such as
     - **Isabelle** [Paulson, 1994] and
     - **PVS** [Owre et al., 1998].
2. More likely either the proof process cannot be fully automated or, even if it can, it is likely to be very slow.

Of course, logical proof can be difficult.

1. If we are lucky, Sys and Req can be described in a quite simple logic and the formula $Sys \Rightarrow Req$ can be decided in a fast and automated way.
   - $\rightarrow$ often essential to invoke human intervention and so utilize semi-automated theorem-provers such as
     - **Isabelle** [Paulson, 1994] and
     - **PVS** [Owre et al., 1998].
2. More likely either the proof process cannot be fully automated or, even if it can, it is likely to be very slow.
   - $\rightarrow$ more sophisticated heuristics and abstractions are typically used.

4.3 Algorithmic Verification

Model Checking (1)

If we want to establish some property of all executions of a system, and if there is only a finite number of such executions, then an obvious approach is to enumerate the executions and check the property on each in turn.

While this is a gross simplification, it is essentially the basis of the model checking approach to algorithmic verification that has been so successful and influential [Clarke et al., 1986].

Here, a mathematical model, $\mathcal{M}$, of the system in question is produced such that the model captures all relevant system executions.

Such a model is typically generated from an operational semantics for the system.

Model Checking (2)

The formal requirement, R, is usually given in a form of temporal logic.

Model Checking (2)

The formal requirement, R, is usually given in a form of temporal logic.

A **model checker** exhaustively checks that all paths through $\mathcal{M}$ (and, therefore, runs through the system) satisfy R.

Model Checking (2)

The formal requirement, R, is usually given in a form of **temporal logic**.

A **model checker** exhaustively checks that all paths through $\mathcal{M}$ (and, therefore, runs through the system) satisfy R.

- All paths satisfy the logical requirement
  $\rightarrow \mathcal{M} \models R$
  $\rightarrow$ system is reported as being correct with respect to its specification.

Model Checking (2)

The formal requirement, R, is usually given in a form of temporal logic.

A **model checker** exhaustively checks that all paths through $\mathcal{M}$ (and, therefore, runs through the system) satisfy R.

- All paths satisfy the logical requirement
  $\rightarrow \mathcal{M} \models R$
  $\rightarrow$ system is reported as being correct with respect to its specification.
- If a path, σ, fails to satisfy the specification
  $\rightarrow \sigma \not\models R$
  $\rightarrow$ provides an execution that violates the formal requirement.

Automata-theoretic View of Model-Checking

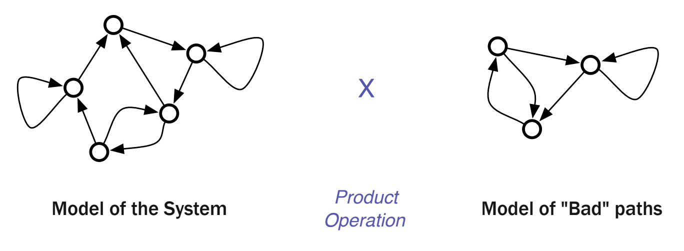
The typical way of visualizing such algorithmic verification is in terms of finite state automata, in particular Büchi Automata.

A Büchi Automaton is essentially a finite state automaton with infinite runs.

The basic idea with model-checking is to capture all the possible executions of the system to be verified as a Büchi Automaton and generate a separate Büchi Automaton describing all bad runs, i.e. executions that do not satisfy the property being verified.

Then we take the synchronous product of these two Büchi Automata [Sistla et al., 1987, Vardi and Wolper, 1994].

Automata-theoretic View of Model-Checking



Automata-theoretic View of Model-Checking

- If the product automaton is empty
  - $\rightarrow$ no sequence which is a legal run of the system while at the same time satisfying the "bad" property.

Automata-theoretic View of Model-Checking

- If the product automaton is empty
  - $\rightarrow$ no sequence which is a legal run of the system while at the same time satisfying the "bad" property.
- If the product automaton is non-empty
  - $\rightarrow$ **identifies** a sequence which is a legal run of the system while at the same time satisfying our "bad" property.

Automata-theoretic View of Model-Checking

- If the product automaton is empty
  - → no sequence which is a legal run of the system while at the same time satisfying the "bad" property.

- If the product automaton is non-empty
  - → **identifies** a sequence which is a legal run of the system while at the same time satisfying our "bad" property.
  - → This highlights a **failing run** of the system.

Model Checking

The model-checking approach has been extremely successful, not only in analyzing hardware systems and protocols, but increasingly in software systems [Clarke et al., 1999, Ball and Rajamani, 2001, Baier and Katoen, 2008].

While the basic idea is quite simple, the success of the technology is, to a large part, due to the improvements in implementation and efficiency that have occurred over the last 25 years.

As well as the above characterization in terms of automata, on the fly [Gerth et al., 1995], symbolic [McMillan, 1993] and SAT-based [Prasad et al., 2005] techniques have all improved the efficacy of model-checkers.

"On the Fly" Model-Checking

Recall: basic automata-theoretic view of model-checking involves constructing the product of two Büchi Automata.

"On the Fly" Model-Checking

Recall: basic automata-theoretic view of model-checking involves constructing the product of two Büchi Automata.

In many practical cases, this product turns out to be much too large to realistically construct.

"On the Fly" Model-Checking

Recall: basic automata-theoretic view of model-checking involves constructing the product of two Büchi Automata.

In many practical cases, this product turns out to be much too large to realistically construct.

- So, rather than constructing the actual product automaton, the idea with the "on the fly approach" is to explore paths through this product automaton without actually constructing it!

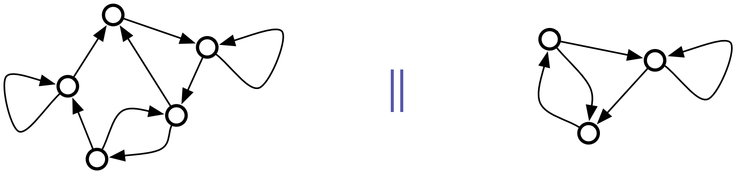
"On the Fly" Model-Checking

Recall: basic automata-theoretic view of model-checking involves constructing the product of two Büchi Automata.

In many practical cases, this product turns out to be much too large to realistically construct.

- So, rather than constructing the actual product automaton, the idea with the "on the fly approach" is to explore paths through this product automaton without actually constructing it!
- This is achieved by exploring the two automata in parallel.

“On the Fly” exploration of the product automaton



Model of the System *Parallel Exploration* **Model of "Bad" paths**

“On the Fly” (1)

In the “on the fly” approach, we explore the ‘system’ automaton, ensuring that every transition we take is mirrored by a simultaneous transition in the “bad” automaton.

"On the Fly" (1)

In the "on the fly" approach, we explore the 'system' automaton, ensuring that every transition we take is mirrored by a simultaneous transition in the "bad" automaton.

We keep exploring this pair synchronously until either

1. a path has been found which satisfies both

"On the Fly" (1)

In the "on the fly" approach, we explore the 'system' automaton, ensuring that every transition we take is mirrored by a simultaneous transition in the "bad" automaton.

We keep exploring this pair synchronously until either

1. a path has been found which satisfies both
   $\rightarrow$ we have found our "bad" path

"On the Fly" (1)

In the "on the fly" approach, we explore the 'system' automaton, ensuring that every transition we take is mirrored by a simultaneous transition in the "bad" automaton.

We keep exploring this pair synchronously until either

1. a path has been found which satisfies both
   $\rightarrow$ we have found our "bad" path
2. exploration of the 'system' automaton can go no further

“On the Fly” (1)

In the “on the fly” approach, we explore the ‘system’ automaton, ensuring that every transition we take is mirrored by a simultaneous transition in the "bad" automaton.

We keep exploring this pair synchronously until either

1. a path has been found which satisfies both
   $\rightarrow$ we have found our "bad" path
2. exploration of the ‘system’ automaton can go no further
   $\rightarrow$ we roll back our execution to any previous choice point in the ‘system’ automaton and continue exploration

“On the Fly” (2)

If we have explored all possible paths through the ‘system’ automaton and none of them have yielded a run of the “bad” automaton, then we can assert that no execution has the “bad” property.

“On the Fly” (2)

If we have explored all possible paths through the ‘system’ automaton and none of them have yielded a run of the “bad” automaton, then we can assert that no execution has the “bad” property.

In order to be able to utilize this “on the fly” approach, the model checking implementation needs to have

"On the Fly" (2)

If we have explored all possible paths through the 'system' automaton and none of them have yielded a run of the "bad" automaton, then we can assert that no execution has the "bad" property.

In order to be able to utilize this "on the fly" approach, the model checking implementation needs to have

1. a way to synchronously step through two representations, and

“On the Fly” (2)

If we have explored all possible paths through the ‘system’ automaton and none of them have yielded a run of the “bad” automaton, then we can assert that no execution has the “bad” property.

In order to be able to utilize this “on the fly” approach, the model checking implementation needs to have

1. a way to synchronously step through two representations, and
2. a mechanism for backtracking the execution.

"On the Fly" (2)

If we have explored all possible paths through the 'system' automaton and none of them have yielded a run of the "bad" automaton, then we can assert that no execution has the "bad" property.

In order to be able to utilize this "on the fly" approach, the model checking implementation needs to have

1. a way to synchronously step through two representations, and
2. a mechanism for backtracking the execution.

The predominant model checker exhibiting this technology is the Spin model-checker [Holzmann, 2003].

4.4 Program verification

Checking Programs Directly

Traditionally, in model-checking, a ‘model’ of the executions of the system is built and then that model is explored and checked with respect to the property.

However, if the system we are to verify is a program, then why not use the program itself as the model?

In this approach, often termed “software model-checking” or “program model-checking”, a logical property is directly checked against the program code [Holzmann and Smith, 1999b, Holzmann and Smith, 1999a, Visser et al., 2003].

This is actually similar to the “on the fly” approach.

Program Model-Checking

Recall: for the "on the fly" approach, we need

1. a way of synchronously stepping through a program at the same time as checking a property, and
2. a mechanism for backtracking execution of the program.

Program Model-Checking

Recall: for the "on the fly" approach, we need

1. a way of synchronously stepping through a program at the same time as checking a property, and
2. a mechanism for backtracking execution of the program.

So, as long as we have implementation technology that allows these two, we can implement program verification.

Program Model-Checking

Recall: for the "on the fly" approach, we need

1. a way of synchronously stepping through a program at the same time as checking a property, and
2. a mechanism for backtracking execution of the program.

So, as long as we have implementation technology that allows these two, we can implement program verification.

The program to be checked is run (e.g. through symbolic execution) and the execution is dynamically assessed against the requirement.

Program Model-Checking

Recall: for the "on the fly" approach, we need

1. a way of synchronously stepping through a program at the same time as checking a property, and
2. a mechanism for backtracking execution of the program.

So, as long as we have implementation technology that allows these two, we can implement program verification.

The program to be checked is run (e.g. through symbolic execution) and the execution is dynamically assessed against the requirement.

Once checked, the program is forced to explore an alternative execution path which is again checked. And so on.

Program Model-Checkers

This has led to the development of model checkers for various high-level languages such as JAVA and C.

Program Model-Checkers

This has led to the development of model checkers for various high-level languages such as JAVA and C.

- JAVA PATHFINDER system implements this approach for model-checking JAVA programs [Visser et al., 2003].

Program Model-Checkers

This has led to the development of model checkers for various high-level languages such as JAVA and C.

- JAVA PATHFINDER system implements this approach for model-checking JAVA programs [Visser et al., 2003].
  1. utilizes a modified JAVA virtual machine which can backtrack, and

Program Model-Checkers

This has led to the development of model checkers for various high-level languages such as JAVA and C.

- JAVA PATHFINDER system implements this approach for model-checking JAVA programs [Visser et al., 2003].
  1. utilizes a modified JAVA virtual machine which can backtrack, and
  2. it uses synchronous listener threads.

Program Model-Checkers

This has led to the development of model checkers for various high-level languages such as JAVA and C.

- JAVA PATHFINDER system implements this approach for model-checking JAVA programs [Visser et al., 2003].

  1. utilizes a modified JAVA virtual machine which can backtrack, and
  2. it uses synchronous listener threads.

We will see later how JAVA PATHFINDER forms the basis for a model-checking system for JAVA-based rational agent programs.

4.5 Run-time verification

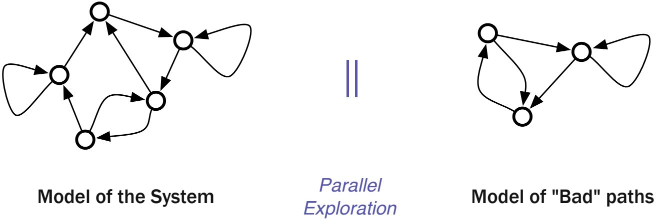
Run-time Verification

Once we have the idea that a form of model-checking can be invoked directly on the program, by forcing it to run numerous times, then this leads us on to thinking about run-time verification [Havelund and Rosu, 2001].

The idea here is to use (lightweight) formal verification technology to check executions as they are being created.

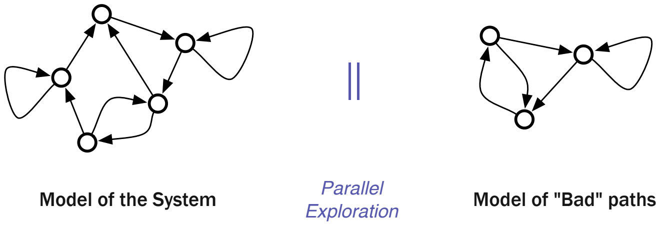
In this way, errors are also spotted at run-time.

Recap; “on the fly”



Here, all the possible program executions are checking against a parallel automaton looking for “bad” runs.

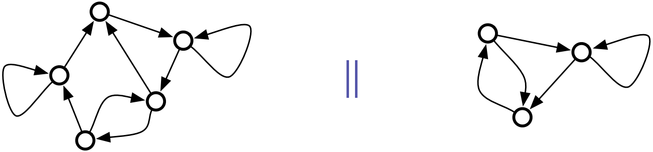
Recap; "on the fly"



Here, all the possible program executions are checking against a parallel automaton looking for "bad" runs.

$\rightarrow$ just take this automaton and just use it to check the current execution as it is being created.

Recap; “on the fly”

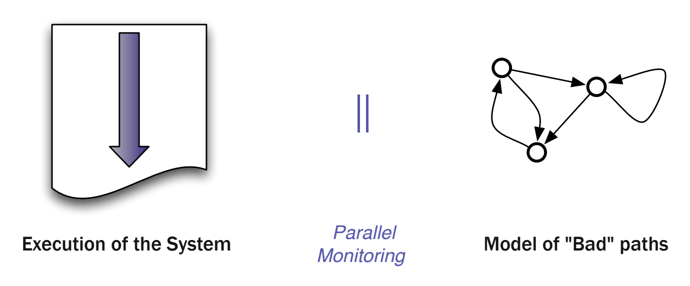


Model of the System | *Parallel Exploration* | **Model of "Bad" paths**

Here, all the possible program executions are checking against a parallel automaton looking for “bad” runs.

- $\rightarrow$ just take this automaton and just use it to check the current execution as it is being created.
- $\rightarrow$ in this way we can monitor the execution and recognize when a quite complex error condition has occurred.

General View of Run-Time Model-Checking



Now we turn to approaches that have been specifically developed for the formal verification of agents and multi-agent systems.

Now we turn to approaches that have been specifically developed for the formal verification of agents and multi-agent systems.

There are, as yet, few run-time verification tools explicitly developed for agents and so we will primarily consider

Now we turn to approaches that have been specifically developed for the formal verification of agents and multi-agent systems.

There are, as yet, few run-time verification tools explicitly developed for agents and so we will primarily consider

- the deductive verification of agents,

Now we turn to approaches that have been specifically developed for the formal verification of agents and multi-agent systems.

There are, as yet, few run-time verification tools explicitly developed for agents and so we will primarily consider

- the deductive verification of agents,
- the algorithmic verification of agent models, and

Now we turn to approaches that have been specifically developed for the formal verification of agents and multi-agent systems.

There are, as yet, few run-time verification tools explicitly developed for agents and so we will primarily consider

- the deductive verification of agents,
- the algorithmic verification of agent models, and
- the direct algorithmic verification of agent programs.

5. Deductive Verification of Agents

Deductive Verification

The essence of deductive verification is to provide a logical description capturing the full behaviour of our agent, say 'Ag'.

Then, if we wish to verify some property of our agent, such as the agent will eventually terminate, we describe this property as another logical formula, Req, and then attempt to prove

$$\vdash \ Ag \Rightarrow Req$$

Deductive Verification

The essence of deductive verification is to provide a logical description capturing the full behaviour of our agent, say 'Ag'.

Then, if we wish to verify some property of our agent, such as the agent will eventually terminate, we describe this property as another logical formula, Req, and then attempt to prove

$$\vdash \ Ag \Rightarrow Req$$

If we succeed with this proof, then Req is true of all possible behaviours of the agent.

5.1 Problems

While the deductive approach is very appealing, there are some difficulties to be overcome when using it:

1. For our particular agent, what logic should 'Ag' be described in, and how do we actually generate 'Ag'?

While the deductive approach is very appealing, there are some difficulties to be overcome when using it:

1. For our particular agent, what logic should 'Ag' be described in, and how do we actually generate 'Ag'?
2. What logic should Req be described in, and can we be sure this is sufficient to allow us to say what we want?

While the deductive approach is very appealing, there are some difficulties to be overcome when using it:

1. For our particular agent, what logic should 'Ag' be described in, and how do we actually generate 'Ag'?
2. What logic should Req be described in, and can we be sure this is sufficient to allow us to say what we want?
3. Given Ag and Req, then will it be *possible* to prove

$$\vdash \ Ag \Rightarrow Req$$

And will we be able to automate this proof process?

While the deductive approach is very appealing, there are some difficulties to be overcome when using it:

1. For our particular agent, what logic should 'Ag' be described in, and how do we actually generate 'Ag'?
2. What logic should Req be described in, and can we be sure this is sufficient to allow us to say what we want?
3. Given Ag and Req, then will it be *possible* to prove

   $$\vdash \; Ag \Rightarrow Req$$

   And will we be able to automate this proof process?
4. If we fail to prove $\vdash \; Ag \Rightarrow Req$, then what does that mean?

While the deductive approach is very appealing, there are some difficulties to be overcome when using it:

1. For our particular agent, what logic should 'Ag' be described in, and how do we actually generate 'Ag'?
2. What logic should Req be described in, and can we be sure this is sufficient to allow us to say what we want?
3. Given Ag and Req, then will it be *possible* to prove

   $$\vdash \; Ag \Rightarrow Req$$

   And will we be able to automate this proof process?
4. If we fail to prove $\vdash \; Ag \Rightarrow Req$, then what does that mean?

Some of these are, of course, quite difficult and fundamental questions.

Describing Agents

For any formal method then we need some variety of formal semantics which provides a formal (often logical) representation of all the behaviours of the agent.

Describing Agents

For any formal method then we need some variety of formal semantics which provides a formal (often logical) representation of all the behaviours of the agent.

If agents are described in terms of enhanced finite-state machines then this is fairly straightforward.

If, however, we have an agent program then we require a semantics for the agent programming language.

Describing Agents

For any formal method then we need some variety of formal semantics which provides a formal (often logical) representation of all the behaviours of the agent.

If agents are described in terms of enhanced finite-state machines then this is fairly straightforward.

If, however, we have an agent program then we require a semantics for the agent programming language.

In the case of deductive verification we consider here we specifically need a logical semantics for the agent programming language.

Describing Agents

For any formal method then we need some variety of formal semantics which provides a formal (often logical) representation of all the behaviours of the agent.

If agents are described in terms of enhanced finite-state machines then this is fairly straightforward.

If, however, we have an agent program then we require a semantics for the agent programming language.

In the case of deductive verification we consider here we specifically need a logical semantics for the agent programming language.

As with traditional formal methods, other varieties of formal semantics, notably operational semantics, are popular.

Requirements and Proof

Any decision about what logical basis to be used must clearly be driven by the requirements of both the logical semantics

i.e. what logic the semantics is provided in

and the formal requirements

i.e. what logic allows us to state the questions we wish to ask

Requirements and Proof

Any decision about what logical basis to be used must clearly be driven by the requirements of both the logical semantics

i.e. what logic the semantics is provided in

and the formal requirements

i.e. what logic allows us to state the questions we wish to ask

Logics combining a temporal/dynamic dimension with at least a knowledge/belief dimension (and probably a motivational dimension) are often used.

5.2 Examples of Direct Proof

IMPACT

Systems are specified in IMPACT through agent programs.

IMPACT

Systems are specified in IMPACT through agent programs.

$\rightarrow$ programs contain clauses with negation, as in logic programming.

IMPACT

Systems are specified in IMPACT through agent programs.

- → programs contain clauses with negation, as in logic programming.
- → the semantics is given by the well-known fixpoint semantics (least Herbrand model in the case of Horn clauses or stable semantics in the case of rules with negation-as-failure).

IMPACT

Systems are specified in IMPACT through agent programs.

- → programs contain clauses with negation, as in logic programming.
- → the semantics is given by the well-known fixpoint semantics (least Herbrand model in the case of Horn clauses or stable semantics in the case of rules with negation-as-failure).

While the basic language of IMPACT does not allow us to formalise mental attitudes, or temporal or probabilistic reasoning all these features have been subsequently investigated [Dix et al., 2001, Dix et al., 2006], and can be modeled with annotated logic programs.

Golog and SITCALC

The Cognitive Agent Specification Language (CASL) is, as GOLOG, based on the situation calculus, but is extended with knowledge and goal operators [Shapiro et al., 2002].

Golog and SITCALC

The Cognitive Agent Specification Language (CASL) is, as GOLOG, based on the situation calculus, but is extended with knowledge and goal operators [Shapiro et al., 2002].

Alongside this, the authors described CASLve, a verification environment for CASL that translates a CASL specification into a problem for the PVS verification system.

2APL, 3APL

In [Alechina et al., 2011] the authors consider a fragment of 3APL and define a series of propositional dynamic logics that can be used to prove safety and liveness properties of programs in this fragment under different deliberation strategies.

2APL, 3APL

In [Alechina et al., 2011] the authors consider a fragment of 3APL and define a series of propositional dynamic logics that can be used to prove safety and liveness properties of programs in this fragment under different deliberation strategies.

- done by relating the operational semantics of programs to models in the appropriate logic.

2APL, 3APL

In [Alechina et al., 2011] the authors consider a fragment of 3APL and define a series of propositional dynamic logics that can be used to prove safety and liveness properties of programs in this fragment under different deliberation strategies.

- done by relating the operational semantics of programs to models in the appropriate logic.

  $\rightarrow$ the axiomatisation of fully interleaved strategies.

METATEM

As described earlier, METATEM is a little unusual, having no explicit motivational dimension but using combinations of temporal and belief operators to achieve such ‘goals’.

METATEM

As described earlier, METATEM is a little unusual, having no explicit motivational dimension but using combinations of temporal and belief operators to achieve such ‘goals’.

- Can prove some (simple) properties of METATEM programs using deductive proof methods for temporal logics of belief [Dixon et al., 2002].

METATEM

As described earlier, METATEM is a little unusual, having no explicit motivational dimension but using combinations of temporal and belief operators to achieve such ‘goals’.

- Can prove some (simple) properties of METATEM programs using deductive proof methods for temporal logics of belief [Dixon et al., 2002].

However, this is non-standard and true “BDI-like” agents usually require a logic with some explicit motivational dimension, such as intentions or goals.

5.3 Use of Logic Programming

Logic Programming

If our agent language is based on logic programming then there may be several advantages.

Logic Programming

If our agent language is based on logic programming then there may be several advantages.

1. it is traditional that declarative as well as operational and fixed point semantics are provided for logic programming languages
   - $\rightarrow$ generating a logical formula describing the agent behaviour is often more straightforward

Logic Programming

If our agent language is based on logic programming then there may be several advantages.

1. it is traditional that declarative as well as operational and fixed point semantics are provided for logic programming languages
   - $\rightarrow$ generating a logical formula describing the agent behaviour is often more straightforward
2. the underlying execution mechanism is essentially deductive (often some variety of SLD-resolution)
   - $\rightarrow$ we might use the execution system itself to carry out the deductive verification we are interested in

Logic Programming

If our agent language is based on logic programming then there may be several advantages.

1. it is traditional that declarative as well as operational and fixed point semantics are provided for logic programming languages
   - → generating a logical formula describing the agent behaviour is often more straightforward
2. the underlying execution mechanism is essentially deductive (often some variety of SLD-resolution)
   - → we might use the execution system itself to carry out the deductive verification we are interested in
   - → in some cases this can be expressive and efficient.

Logic Programming

If our agent language is based on logic programming then there may be several advantages.

1. it is traditional that declarative as well as operational and fixed point semantics are provided for logic programming languages
   - → generating a logical formula describing the agent behaviour is often more straightforward
2. the underlying execution mechanism is essentially deductive (often some variety of SLD-resolution)
   - → we might use the execution system itself to carry out the deductive verification we are interested in
   - → in some cases this can be expressive and efficient.

However, it is often the case that not all the aspects we might wish for from "BDI-like" languages are present.

Abductive Logic Programming

Extend standard logic programs with abducible predicates.

Abductive Logic Programming

Extend standard logic programs with abducible predicates.

These are predicates whose values can be set in such a way to explain certain observations.

Abductive Logic Programming

Extend standard logic programs with abducible predicates.

These are predicates whose values can be set in such a way to explain certain observations.

- Given a program and a set of observations, an abduction process is used to suggest which abducible predicates explain the observations.

Abductive Logic Programming

Extend standard logic programs with abducible predicates.

These are predicates whose values can be set in such a way to explain certain observations.

- Given a program and a set of observations, an abduction process is used to suggest which abducible predicates explain the observations.

This is particularly useful where agents have only partial knowledge of their environment and so must work out what is the most reasonable explanation for the things it perceives. Importantly, an abductive proof procedure is used as part of this process [Kakas et al., 1993].

KGP and SCIFF

The $\mathcal{KGP}$ agent approach [Sadri and Toni, 2006] is based on logic programming but extended with specific agent aspects: **K**nowledge; **G**oals; and **P**lans.

Abductive logic programming is used via the $\mathcal{S}CIFF$ procedure for interactive verification.

KGP and SCIFF

The $\mathcal{KGP}$ agent approach [Sadri and Toni, 2006] is based on logic programming but extended with specific agent aspects: **K**nowledge; **G**oals; and **P**lans.

Abductive logic programming is used via the $\mathcal{S}CIFF$ procedure for interactive verification.

$\mathcal{S}CIFF$ was originally developed to verify the compliance of agent to interaction protocols and it uses

KGP and SCIFF

The $\mathcal{KGP}$ agent approach [Sadri and Toni, 2006] is based on logic programming but extended with specific agent aspects: **K**nowledge; **G**oals; and **P**lans.

Abductive logic programming is used via the $\mathcal{S}CIFF$ procedure for interactive verification.

$\mathcal{S}CIFF$ was originally developed to verify the compliance of agent to interaction protocols and it uses

1. abducibles to represent hypotheses about agent behaviour,

KGP and SCIFF

The $\mathcal{KGP}$ agent approach [Sadri and Toni, 2006] is based on logic programming but extended with specific agent aspects: **K**nowledge; **G**oals; and **P**lans.

Abductive logic programming is used via the $\mathcal{S}CIFF$ procedure for interactive verification.

$\mathcal{S}CIFF$ was originally developed to verify the compliance of agent to interaction protocols and it uses

1. abducibles to represent hypotheses about agent behaviour,
2. CLP constraints, and

KGP and SCIFF

The $\mathcal{KGP}$ agent approach [Sadri and Toni, 2006] is based on logic programming but extended with specific agent aspects: **K**nowledge; **G**oals; and **P**lans.

Abductive logic programming is used via the $\mathcal{S}CIFF$ procedure for interactive verification.

$\mathcal{S}CIFF$ was originally developed to verify the compliance of agent to interaction protocols and it uses

1. abducibles to represent hypotheses about agent behaviour,
2. CLP constraints, and
3. existentially quantified variables in integrity constraints.

Action logics

In a series of papers, e.g. [Giordano et al., 2007], the problem of specifying and verifying systems of communicating agents and interaction protocols (e.g. verification of a priori conformance to the agreed protocol) is tackled.

Action logics

In a series of papers, e.g. [Giordano et al., 2007], the problem of specifying and verifying systems of communicating agents and interaction protocols (e.g. verification of a priori conformance to the agreed protocol) is tackled.

- applies to the case where protocols are specified with finite-state automata or when the policies can be implemented in DyLOG, a **computational logic**.

Action logics

In a series of papers, e.g. [Giordano et al., 2007], the problem of specifying and verifying systems of communicating agents and interaction protocols (e.g. verification of a priori conformance to the agreed protocol) is tackled.

- applies to the case where protocols are specified with finite-state automata or when the policies can be implemented in DYLOG, a **computational logic**.

$\rightarrow$ The [Giordano et al., 2007] approach is based on a Dynamic Linear Time Temporal Logic.

5.4 Example

Recall the example of two robots working together to manufacture an artifact, introduced elsewhere in this book.

We considered some of the requirements of such a scenario earlier.

Recall the example of two robots working together to manufacture an artifact, introduced elsewhere in this book.

We considered some of the requirements of such a scenario earlier.

Now, if we wish to apply deductive verification to assess some of these requirements, we need a logical description of the system in question.

Typically, this would contain logical representations of all the steps of the robots, for example

Recall the example of two robots working together to manufacture an artifact, introduced elsewhere in this book.

We considered some of the requirements of such a scenario earlier.

Now, if we wish to apply deductive verification to assess some of these requirements, we need a logical description of the system in question.

Typically, this would contain logical representations of all the steps of the robots, for example

$$\left[\begin{array}{l} K_{\mathsf{robot}_1}\, \mathit{infrontof}(\mathit{robot}_1, A) \wedge \\ K_{\mathsf{robot}_1}\, \mathit{infrontof}(\mathit{robot}_1, B) \wedge \\ \qquad\quad \mathit{do}(\mathit{robot}_1, \mathit{load}(A, B)) \end{array}\right] \Rightarrow \bigcirc\, \mathit{infrontof}(\mathit{robot}_1, AB)$$

Deduction

Once we have a suitable specification of the system (say Sys), possibly comprising formulae such as the above, then we can verify this with respect to some of the formal requirements (say Req) in the way described earlier, i.e

$$\vdash \ Sys \Rightarrow Req$$

Deduction

Once we have a suitable specification of the system (say Sys), possibly comprising formulae such as the above, then we can verify this with respect to some of the formal requirements (say Req) in the way described earlier, i.e

$$\vdash \ Sys \Rightarrow Req$$

Of course, we require suitable, preferably automated, proof systems for the relevant logics.

Deduction

Once we have a suitable specification of the system (say Sys), possibly comprising formulae such as the above, then we can verify this with respect to some of the formal requirements (say Req) in the way described earlier, i.e

$$\vdash Sys \Rightarrow Req$$

Of course, we require suitable, preferably automated, proof systems for the relevant logics.

For example, the above will need at least proof in temporal logics of knowledge [Fagin et al., 1995].

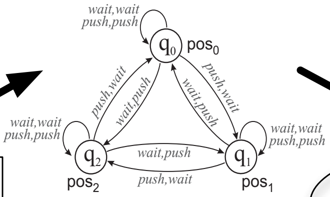
6. Algorithmic Verification of Models

6 Algorithmic Verification of Models

- Representation
- MC of CTL
- MC of LTL
- MC of CTL*
- MC of ATL
- MC of MAS with Imperfect Information/Recall
- Summary of Complexity Results
- Model Checking Agent Language Models

What is Model Checking

Formal model



Problem
(e.g. mobile phone)
+
(Safety) Property
(e.g. deadlock free)

$$\mathfrak{M} \overset{?}{\models} \langle\!\langle\{1,2\}\rangle\!\rangle\Box \bigcirc \top$$

Let's model ckeck...

Computational Complexity?

$$\varphi = \langle\!\langle\{1,2\}\rangle\!\rangle\Box \bigcirc \top$$

Logical (formal) specification

What is Model Checking? (1)

- Model checking refers to the problem to determine whether a given formula φ is satisfied in a state q of model $\mathcal{M}$.

What is Model Checking? (1)

- Model checking refers to the problem to determine whether a given formula φ is satisfied in a state q of model $\mathcal{M}$.
- Local model checking is the decision problem that determines membership in the set
  $\mathsf{MC}(\mathcal{L}, \mathsf{Struc}, \models) := \{(\mathcal{M}, q, \varphi) \in \mathsf{Struc} \times \mathcal{L} \mid \mathcal{M}, q \models \varphi\}$,
  where
  - $\mathcal{L}$ is a logical language,
  - Struc is a class of (pointed) models for $\mathcal{L}$ (i.e. a tuple consisting of a model and a state), and
  - $\models$ is a semantic satisfaction relation compatible with $\mathcal{L}$ and Struc.

What is Model Checking? (2)

- Global model checking: Determine all states in which φ is true.
- Here: The complexities of local and global model checking coincide.
- We are interested in the decidability and the computational complexity of determining whether an input instance $(\mathcal{M}, q, \varphi)$ belongs to $\mathrm{MC}(\ldots)$.

6.1 Representation

- How do we measure the size of a given model?
- Should we simply consider the **number of states**?
- Should we assume the model is given **explicitly** and we just count the number of symbols that are necessary to represent it?

Example 6.1 (Explicit versus implicit representation)

We here consider the famed primality problem: checking whether a given natural number n is prime. A very simple and well-known algorithm uses $\sqrt{n}$-many divisions (starting with 2, then 3, etc. until $\sqrt{n}$) and thus runs in less than linear time **when the input is represented in unary**. But a symbolic representation of n needs only $\log(n)$ bits and thus the above algorithm runs in *exponential* time: $\sqrt{n}$ is exponential as a function of $\log(n)$.

Input size

- Size of the model ($|\mathcal{M}|$): number of (states and) transitions in the $\mathcal{M}$
- Size of the formula ($|\varphi|$): given by its length (i.e., the number of elements it is composed of, apart from parentheses).

For example, the formula $\mathsf{A}\bigcirc(\mathsf{pos}_0 \vee \mathsf{pos}_1)$ has length 5.

Be careful . . .

. . . if numbers are involved!

So the indeces have to be represented as well (these could be arbitrary numbers).

Measuring complexity

We distinguish between the following approaches:

Explicit: The input size is given by the number of transitions in the model and the length of the formula. Thus we assume the model is given explicitly.

Implicit: We assume that the transition function is implicitly encoded in a sufficiently small way. The input size can then be viewed as a function of the number of states and the number of agents (and the length of the formula).

Measuring complexity (cont.)

Highly compact: For many systems, some symbolic and thus very compact representations are possible. The model can be defined in terms of a compact high-level representation, plus an **unfolding procedure** that defines the precise relationship between representations and explicit models of the logic. Of course, unfolding a higher-level description to an explicit model involves usually an exponential blowup in its size.

- Taking only the number of states into account would give a misleading measure.
- Let n be the number of states in a concurrent game structure $\mathcal{M}$, let k denote the number of agents, and d the maximal number of available decisions (moves) per agent per state. Then,

$$m = \mathbf{O}(nd^k).$$

- If we consider explicit models, the size of the input is measured as nd^k.
- If we consider implicit models, then the size of the input is viewed as a function of n and k.
- Therefore many model checking algorithms (e.g. from [Alur et al., 2002]) are **polynomial** in nd^k but they run in exponential time if the number of agents is a parameter of the problem (implicit models).

Model Checking LTL/CTL

Let $\mathcal{M}$ be a Kripke model and q be a state in the model.

- Model checking a $\mathcal{L}_{CTL}/\mathcal{L}_{CTL^*}$-formula φ in $\mathcal{M}, q$ means to determine whether $\mathcal{M}, q \models \varphi$, i.e., whether φ holds in $\mathcal{M}, q$.

Model Checking LTL/CTL

Let $\mathcal{M}$ be a Kripke model and q be a state in the model.

- Model checking a $\mathcal{L}_{CTL}/\mathcal{L}_{CTL^*}$-formula φ in $\mathcal{M}, q$ means to determine whether $\mathcal{M}, q \models \varphi$, i.e., whether φ holds in $\mathcal{M}, q$.

Consider the path $\lambda = q_{i_1} q_{i_2} \dots$ with $i_1.i_2 i_3 i_4 \dots = 3.14159265\dots$. How can we represent such a path? We need a finite representation.

- For **LTL**, checking $\mathcal{M}, q \models \varphi$ means that we check whether φ holds on all the paths in $\mathcal{M}$ which start from q.

Model Checking LTL/CTL

Let $\mathcal{M}$ be a Kripke model and q be a state in the model.

- Model checking a $\mathcal{L}_{CTL}/\mathcal{L}_{CTL^*}$-formula φ in $\mathcal{M}, q$ means to determine whether $\mathcal{M}, q \models \varphi$, i.e., whether φ holds in $\mathcal{M}, q$.

Consider the path $\lambda = q_{i_1} q_{i_2} \ldots$ with $i_1.i_2 i_3 i_4 \cdots = 3.14159265\ldots$. How can we represent such a path? We need a finite representation.

- For **LTL**, checking $\mathcal{M}, q \models \varphi$ means that we check whether φ holds on all the paths in $\mathcal{M}$ which start from q.
- That is, it is equivalent to CTL* model checking of a formula Aφ in $\mathcal{M}, q$.

6.2 MC of CTL

Model Checking CTL

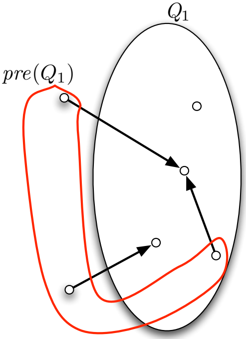
The next algorithm is based on the following fixed-point characterisations:

$$\begin{aligned} \mathsf{E}\Box\varphi &\leftrightarrow \varphi \wedge \mathsf{E}\bigcirc \mathsf{E}\Box\varphi, \\ \mathsf{E}\varphi_1 \,\mathcal{U}\, \varphi_2 &\leftrightarrow \varphi_2 \vee (\varphi_1 \wedge \mathsf{E}\bigcirc \mathsf{E}\varphi_1 \,\mathcal{U}\, \varphi_2). \end{aligned}$$

Paths can be constructed step-by-step.

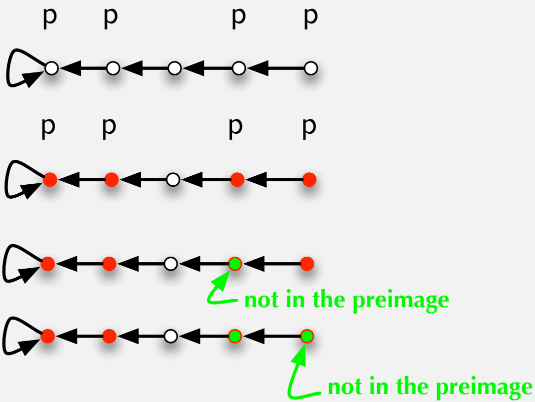
Model Checking CTL

- Let the function $pre(Q')$ return all states such that there is a transition leading to a state in Q' .
- Formally: Given a set of states $Q' \subseteq St$ the preimage of Q', $pre(Q')$, consists of all states q'' such that there is a state $q' \in St'$ with $(q'', q') \in R$.

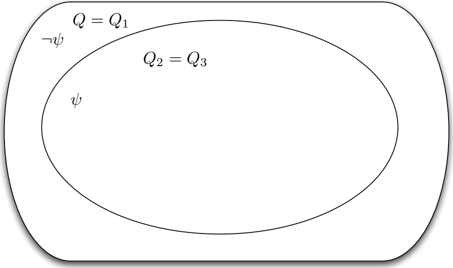


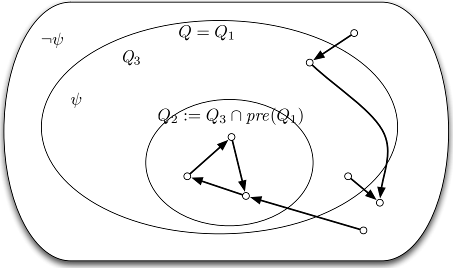
Example 6.2

Model Check E□p in the following model:



Model checking $\mathsf{E}\Box\psi$





Model Checking CTL

function $mcheck(\mathfrak{M}, \varphi)$.

case $\varphi \equiv p$ **:** return $\{q \in St \mid p \in \pi(q)\}$

end case

Figure 5 : **CTL**-model checking algorithm

Model Checking CTL

function $mcheck(\mathfrak{M}, \varphi)$.

case $\varphi \equiv p$ **:** return $\{q \in St \mid p \in \pi(q)\}$

case $\varphi \equiv \neg\psi$ **:** return $St \setminus mcheck(\mathfrak{M}, \psi)$

end case

Figure 5 : **CTL**-model checking algorithm

Model Checking CTL

function $mcheck(\mathfrak{M}, \varphi)$.

case $\varphi \equiv p$ **:** return $\{q \in St \mid p \in \pi(q)\}$
case $\varphi \equiv \neg\psi$ **:** return $St \setminus mcheck(\mathfrak{M}, \psi)$
case $\varphi \equiv \psi_1 \wedge \psi_2$ **:** return $mcheck(\mathfrak{M}, \psi_1) \cap mcheck(\mathfrak{M}, \psi_2)$

end case

Figure 5 : **CTL**-model checking algorithm

Model Checking CTL

function $mcheck(\mathfrak{M}, \varphi)$.

case $\varphi \equiv p$ **:** return $\{q \in St \mid p \in \pi(q)\}$
case $\varphi \equiv \neg\psi$ **:** return $St \setminus mcheck(\mathfrak{M}, \psi)$
case $\varphi \equiv \psi_1 \wedge \psi_2$ **:** return $mcheck(\mathfrak{M}, \psi_1) \cap mcheck(\mathfrak{M}, \psi_2)$
case $\varphi \equiv \mathsf{E}\bigcirc\psi$ **:** return $pre(mcheck(\mathfrak{M}, \psi))$

end case

Figure 5 : **CTL**-model checking algorithm

Model Checking CTL

function $mcheck(\mathfrak{M}, \varphi)$.

case $\varphi \equiv p$ **:** return $\{q \in St \mid p \in \pi(q)\}$
case $\varphi \equiv \neg\psi$ **:** return $St \setminus mcheck(\mathfrak{M}, \psi)$
case $\varphi \equiv \psi_1 \wedge \psi_2$ **:** return $mcheck(\mathfrak{M}, \psi_1) \cap mcheck(\mathfrak{M}, \psi_2)$
case $\varphi \equiv \mathsf{E}\bigcirc\psi$ **:** return $pre(mcheck(\mathfrak{M}, \psi))$
case $\varphi \equiv \mathsf{E}\Box\psi$ **:**
$Q_1 := Q$; $Q_2 := Q_3 := mcheck(\mathfrak{M}, \psi)$;
while $Q_1 \not\subseteq Q_2$ **do** $Q_1 := Q_1 \cap Q_2$; $Q_2 := pre(Q_1) \cap Q_3$ **od**;
return Q_1

end case

Figure 5 : **CTL**-model checking algorithm

Model Checking CTL

function $mcheck(\mathfrak{M}, \varphi)$.

case $\varphi \equiv p$ **:** return $\{q \in St \mid p \in \pi(q)\}$
case $\varphi \equiv \neg\psi$ **:** return $St \setminus mcheck(\mathfrak{M}, \psi)$
case $\varphi \equiv \psi_1 \wedge \psi_2$ **:** return $mcheck(\mathfrak{M}, \psi_1) \cap mcheck(\mathfrak{M}, \psi_2)$
case $\varphi \equiv \mathsf{E}\bigcirc\psi$ **:** return $pre(mcheck(\mathfrak{M}, \psi))$
case $\varphi \equiv \mathsf{E}\Box\psi$ **:**
$Q_1 := Q$; $Q_2 := Q_3 := mcheck(\mathfrak{M}, \psi)$;
while $Q_1 \not\subseteq Q_2$ **do** $Q_1 := Q_1 \cap Q_2$; $Q_2 := pre(Q_1) \cap Q_3$ **od**;
return Q_1
case $\varphi \equiv \mathsf{E}\psi_1 \mathcal{U} \psi_2$ **:**
$Q_1 := \emptyset$; $Q_2 := mcheck(\mathfrak{M}, \psi_2)$; $Q_3 := mcheck(\mathfrak{M}, \psi_1)$;
while $Q_2 \not\subseteq Q_1$ **do** $Q_1 := Q_1 \cup Q_2$; $Q_2 := pre(Q_1) \cap Q_3$ **od**;
return Q_1
end case

Figure 5 : **CTL**-model checking algorithm

Model Checking CTL

Theorem 6.3 (CTL [Clarke et al., 1986, Schnoebelen, 2003])

Model checking **CTL** *is* P-complete, *and can be done in time* $\mathbf{O}(|\mathcal{M}| \cdot |\varphi|)$, *where* $|\mathcal{M}|$ *is given by the number of transitions.*

Proof

The algorithm determining the states in a model at which a given formula holds is presented in Figure 5 on Slide 493.

The lower bound (P-hardness) can be for instance proven by a reduction of the Circuit-Value-Problem [Schnoebelen, 2003].

6.3 MC of LTL

Model Checking LTL and CTL

We are mainly interested in the complexity class (and an abstract algorithm) of the model checking problem.

Is there a more convenient way to determine the complexity without working out the algorithm?

Model Checking LTL and CTL

We are mainly interested in the complexity class (and an abstract algorithm) of the model checking problem.

Is there a more convenient way to determine the complexity without working out the algorithm?

- Automata-theory to build algorithms.
- Unified approach.
- Automata are well studied.
- Simplifies complexity analysis.
- Usually, one is only interested in a complexity class. It is very time-demanding to come up with a good algorithm.

Automata and Model Checking

How can we use automata for the model checking problem?

The basic idea is the following:

1. We build an automaton $A_{\mathcal{M},q_0}$ accepting the paths of model $\mathcal{M}, q_0$.
2. We build an automaton A_φ accepting all paths satisfying φ.
3. Then, we have:

$$\mathcal{M} \models \varphi \text{ iff}$$

Automata and Model Checking

How can we use automata for the model checking problem?

The basic idea is the following:

1. We build an automaton $A_{\mathcal{M},q_0}$ accepting the paths of model $\mathcal{M}, q_0$.
2. We build an automaton A_φ accepting all paths satisfying φ.
3. Then, we have:

$$\mathcal{M} \models \varphi \text{ iff } L(A_{\mathcal{M},q_0}) \subseteq L(A_\varphi).$$

Automata and Model Checking

How can we use automata for the model checking problem?

The basic idea is the following:

1. We build an automaton $A_{\mathcal{M},q_0}$ accepting the paths of model $\mathcal{M}, q_0$.
2. We build an automaton A_φ accepting all paths satisfying φ.
3. Then, we have:

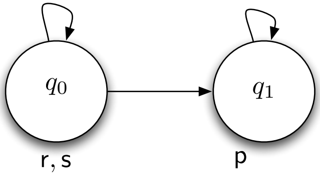
$$\mathcal{M} \models \varphi \text{ iff } L(A_{\mathcal{M},q_0}) \subseteq L(A_\varphi).$$

Remark 6.4

Büchi automata are finite automata which accept infinite words (cf. pages 705).

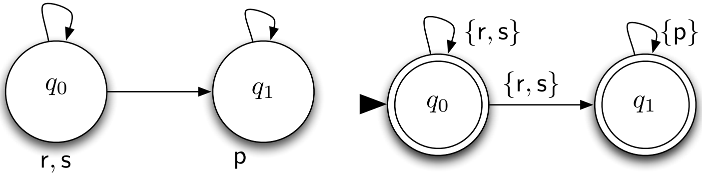
Büchi Automata and Kripke Models

We can relate a Kripke model $\mathcal{M} = (St, \mathcal{R}, \pi)$ and a state $q_0 \in St$ to a Büchi automaton $A_{\mathcal{M},q_0} = (\Sigma, St, q_0, \Delta, St)$



Büchi Automata and Kripke Models

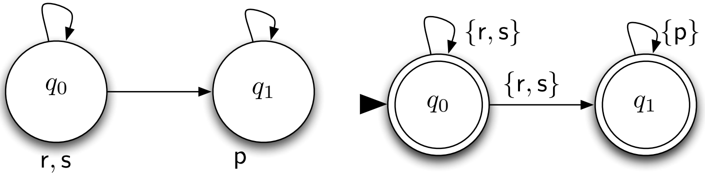
We can relate a Kripke model $\mathcal{M} = (St, \mathcal{R}, \pi)$ and a state $q_0 \in St$ to a Büchi automaton $A_{\mathcal{M},q_0} = (\Sigma, St, q_0, \Delta, St)$



Büchi Automata and Kripke Models

We can relate a Kripke model $\mathcal{M} = (St, \mathcal{R}, \pi)$ and a state $q_0 \in St$ to a Büchi automaton $A_{\mathcal{M},q_0} = (\Sigma, St, q_0, \Delta, St)$

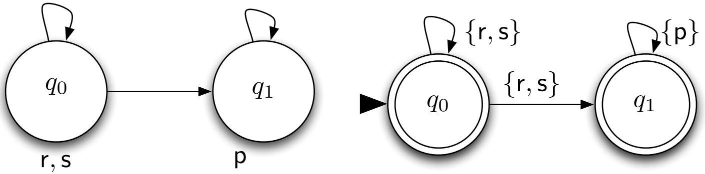
- $\Sigma = \mathcal{P}(\mathcal{P}rop)$: Each input symbol is a set of propositions,
- $q' \in \Delta(q, w)$ iff $((q, q') \in \mathcal{R}$ and $w = \pi(q))$,
- all states being accepting states (i.e. each infinite run of the automaton is accepting).



Büchi Automata and Kripke Models

We can relate a Kripke model $\mathcal{M} = (St, \mathcal{R}, \pi)$ and a state $q_0 \in St$ to a Büchi automaton $A_{\mathcal{M},q_0} = (\Sigma, St, q_0, \Delta, St)$

- $\Sigma = \mathcal{P}(\mathcal{P}rop)$: Each input symbol is a set of propositions,
- $q' \in \Delta(q, w)$ iff $((q, q') \in \mathcal{R}$ and $w = \pi(q))$,
- all states being accepting states (i.e. each infinite run of the automaton is accepting).



Note: The automaton accepts words over $2^{\mathcal{P}rop}$ but paths are sequences of states! What now?

LTL Semantics Revisited

The truth of $\lambda, \pi \models \varphi$ does **only** depend on the **propositions** true at states.

LTL Semantics Revisited

The truth of $\lambda, \pi \models \varphi$ does **only** depend on the **propositions** true at states. Clearly, for path, λ, λ' we have the following:

If for all $i \in \mathbb{N}_0$

$$\pi(\lambda[i]) = \pi(\lambda'[i]) \text{ then } \qquad .$$

LTL Semantics Revisited

The truth of $\lambda, \pi \models \varphi$ does **only** depend on the **propositions** true at states. Clearly, for path, λ, λ' we have the following:

If for all $i \in \mathbb{N}_0$

$$\pi(\lambda[i]) = \pi(\lambda'[i]) \text{ then } \lambda, \pi \models \varphi \text{ iff } \lambda', \pi \models \varphi.$$

LTL Semantics Revisited

The truth of $\lambda, \pi \models \varphi$ does only depend on the propositions true at states. Clearly, for path, λ, λ' we have the following:

If for all $i \in \mathbb{N}_0$

$$\pi(\lambda[i]) = \pi(\lambda'[i]) \text{ then } \lambda, \pi \models \varphi \text{ iff } \lambda', \pi \models \varphi.$$

Hence, we can also use the infinite word

$$\lambda^{\pi} := \pi(\lambda[0])\pi(\lambda[1])\pi(\lambda[2]) \cdots \in 2^{\mathcal{P}rop^{\omega}}$$

to give truth to **LTL**-formulae.

Alternative LTL Semantics

The original clauses had the following form:

- $\lambda, \pi \models^{\mathbf{LTL}} \mathsf{p}$ iff $\lambda[0] \in \pi(\mathsf{p})$;
- $\lambda, \pi \models^{\mathbf{LTL}} \neg\varphi$ iff $\lambda, \pi \not\models^{\mathbf{LTL}} \varphi$;
- $\lambda, \pi \models^{\mathbf{LTL}} \varphi \wedge \psi$ iff $\lambda, \pi \models^{\mathbf{LTL}} \varphi$ and $\lambda, \pi \models^{\mathbf{LTL}} \psi$.

What happens if we use λ^{π} instead of λ, π?

Alternative LTL Semantics

The original clauses had the following form:

- $\lambda, \pi \models^{\mathsf{LTL}} \mathsf{p}$ iff $\lambda[0] \in \pi(\mathsf{p})$;
- $\lambda, \pi \models^{\mathsf{LTL}} \neg\varphi$ iff $\lambda, \pi \not\models^{\mathsf{LTL}} \varphi$;
- $\lambda, \pi \models^{\mathsf{LTL}} \varphi \wedge \psi$ iff $\lambda, \pi \models^{\mathsf{LTL}} \varphi$ and $\lambda, \pi \models^{\mathsf{LTL}} \psi$.

What happens if we use λ^π instead of λ, π?

We simply replace "λ, π" by "λ^π" everywhere and modify the clause for propositions as follows:

Alternative LTL Semantics

The original clauses had the following form:

- $\lambda, \pi \models^{\mathsf{LTL}} \mathsf{p}$ iff $\lambda[0] \in \pi(\mathsf{p})$;
- $\lambda, \pi \models^{\mathsf{LTL}} \neg\varphi$ iff $\lambda, \pi \not\models^{\mathsf{LTL}} \varphi$;
- $\lambda, \pi \models^{\mathsf{LTL}} \varphi \wedge \psi$ iff $\lambda, \pi \models^{\mathsf{LTL}} \varphi$ and $\lambda, \pi \models^{\mathsf{LTL}} \psi$.

What happens if we use λ^π instead of λ, π?

We simply **replace "λ, π" by "λ^π"** everywhere and modify the clause for propositions as follows:

- $\lambda^\pi \models^{\mathsf{LTL}} \mathsf{p}$ iff $\mathsf{p} \in \lambda^\pi[0]$.

We use the same notations for λ^π as for paths any may also omit superscript π if clear from context.

We can state the relation between $\Lambda_{\mathcal{M}}$, $\mathcal{M}$, q and $A_{\mathcal{M},q}$ precisely.

Proposition 6.5

Let $\mathcal{M} = (St, \mathcal{R}, \pi)$ and $q_0 \in St$. The automaton $A_{\mathcal{M},q_0}$ accepts the language

$$\{\lambda^\pi \mid \lambda \in \Lambda_{\mathcal{M}}(q_0)\}.$$

Proof.

Exercise! $\square$

In the following we define the automaton A_φ accepting exactly those infinite words w over $2^{\mathcal{P}rop}$ such that $w \models \varphi$. Then, we have:

$$\mathcal{M}, q \models \varphi \text{ iff } L(A_{\mathcal{M},q}) \subseteq L(A_\varphi) \text{ iff } L(A_{\mathcal{M},q}) \cap \overline{L(A_\varphi)} = \emptyset.$$

How can we avoid the complementation of the Büchi automaton (this operation is expensive)? We have:

So: model checking is reduced to emptiness checking Büchi automata.

In the following we define the automaton A_φ accepting exactly those infinite words w over $2^{\mathcal{P}rop}$ such that $w \models \varphi$. Then, we have:

$$\mathcal{M}, q \models \varphi \text{ iff } L(A_{\mathcal{M},q}) \subseteq L(A_\varphi) \text{ iff } L(A_{\mathcal{M},q}) \cap \overline{L(A_\varphi)} = \emptyset.$$

How can we avoid the complementation of the Büchi automaton (this operation is expensive)? We have:

$$L(A_{\mathcal{M},q}) \cap \overline{L(A_\varphi)} = \emptyset \text{ iff } L(A_{\mathcal{M},q}) \cap L(A_{\neg\varphi}) = \emptyset.$$

So: model checking is reduced to emptiness checking Büchi automata.

The Automaton A_φ

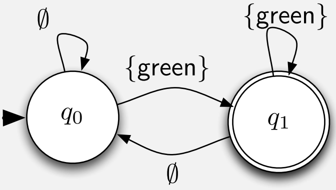
Example 6.6 (Automaton for $\Box\Diamond$green)

Construct a Büchi automaton which accepts all path satisfying $\Box\Diamond\mathrm{green}$ over $\mathcal{P}rop = \{\mathrm{green}\}$. Thus, the autmaton can read $\emptyset$ or $\{\mathrm{green}\}$.

The Automaton A_φ

Example 6.6 (Automaton for $\Box\Diamond$green)

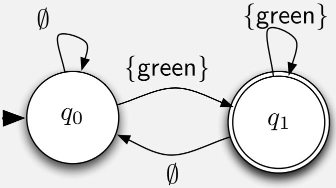
Construct a Büchi automaton which accepts all path satisfying $\Box\Diamond$green over $\mathcal{P}rop = \{\text{green}\}$. Thus, the autmaton can read $\emptyset$ or $\{\text{green}\}$.



The Automaton A_φ

Example 6.6 (Automaton for $\Box\Diamond$green)

Construct a Büchi automaton which accepts all path satisfying $\Box\Diamond$green over $\mathcal{P}rop = \{\text{green}\}$. Thus, the autmaton can read $\emptyset$ or $\{\text{green}\}$.



The automaton accepts e.g.

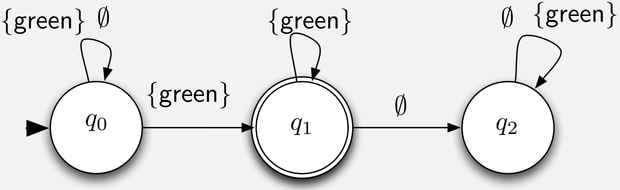
- $\emptyset\emptyset\emptyset(\{\text{green}\})^\omega \quad \hat{=} \quad q_0q_0q_0(q_1)^\omega$
- $(\emptyset\{\text{green}\})^\omega \quad \hat{=} \quad (q_0q_1)^\omega$

Example 6.7 (Automaton for $\Diamond\Box$green)

Construct a Büchi automaton which accepts all path satisfying $\Diamond\Box$green over $\mathcal{P}rop = \{\text{green}\}$.

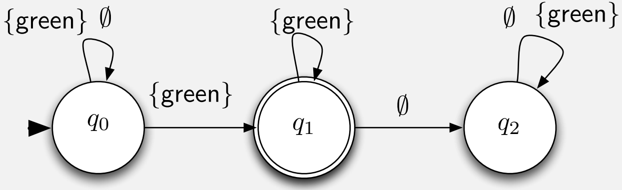
Example 6.7 (Automaton for $\Diamond\Box$green)

Construct a Büchi automaton which accepts all path satisfying $\Diamond\Box$green over $\mathcal{P}rop = \{\text{green}\}$.



Example 6.7 (Automaton for $\Diamond\Box$green)

Construct a Büchi automaton which accepts all path satisfying $\Diamond\Box$green over $\mathcal{P}rop = \{\text{green}\}$.



Note, that this automaton is non-deterministic.

In the following describe how the automaton A_φ can be constructed systematically.

Theorem 6.8 ([Sistla and Clarke, 1985, Lichtenstein and Pnueli, 1985, Vardi and Wolper, 1986])

For a given $\mathcal{L}_{LTL}$-formula φ a Büchi Automaton $A_\varphi = (S, \Sigma, \Delta, S_0, F)$ accepting exactly the words satisfying φ can be constructed *where $\Sigma = \mathcal{P}(\mathcal{P}rop)$ and $|S| \leq 2^{(\mathcal{O}(|\varphi|))}$.*

In the following we introduce additional notation and construct the automaton.

How does the automaton look like?

- States will consist of subformulae of φ (or their negations).
- A run $\rho = S_1 S_2 \dots$ of the automaton is an infinite sequence of such sets ofsubformulae.

How does the automaton look like?

- States will consist of subformulae of φ (or their negations).
- A run $\rho = S_1 S_2 \ldots$ of the automaton is an infinite sequence of such sets ofsubformulae.

Given a word $\lambda^\pi = w_1 w_2 \ldots$ with $\lambda^\pi \models \varphi$ we would like to enrich each (propositional) w_i with subformulae to S_i such that

How does the automaton look like?

- States will consist of subformulae of φ (or their negations).
- A run $\rho = S_1 S_2 \dots$ of the automaton is an infinite sequence of such sets ofsubformulae.

Given a word $\lambda^\pi = w_1 w_2 \dots$ with $\lambda^\pi \models \varphi$ we would like to enrich each (propositional) w_i with subformulae to S_i such that

$$\lambda^\pi[i, \infty] \models \psi \quad \textbf{iff} \quad \psi \in S_i$$

for all subformulae ψ of φ.

How does the automaton look like?

- States will consist of subformulae of φ (or their negations).
- A run $\rho = S_1 S_2 \ldots$ of the automaton is an infinite sequence of such sets ofsubformulae.

Given a word $\lambda^\pi = w_1 w_2 \ldots$ with $\lambda^\pi \models \varphi$ we would like to enrich each (propositional) w_i with subformulae to S_i such that

$$\lambda^\pi[i, \infty] \models \psi \quad \textbf{iff} \quad \psi \in S_i$$

for all subformulae ψ of φ.

Intuitively, each S_i encodes the formulae which should be true at this moment.

The basic idea is that a run of the automaton simulates the **LTL** semantics.

Definition 6.9 (Closure $cl(\varphi)$)

The **closure** $cl(\varphi)$ is defined as follows:

1. $\varphi \in cl(\varphi)$,

Definition 6.9 (Closure $cl(\varphi)$)

The closure $cl(\varphi)$ is defined as follows:

1. $\varphi \in cl(\varphi)$,
2. $\phi \wedge \psi \in cl(\varphi)$ implies $\phi, \psi \in cl(\varphi)$,

Definition 6.9 (Closure $cl(\varphi)$)

The **closure** $cl(\varphi)$ is defined as follows:

1. $\varphi \in cl(\varphi)$,
2. $\phi \wedge \psi \in cl(\varphi)$ implies $\phi, \psi \in cl(\varphi)$,
3. $\neg\psi \in cl(\varphi)$ implies $\psi \in cl(\varphi)$,

Definition 6.9 (Closure $cl(\varphi)$)

The **closure** $cl(\varphi)$ is defined as follows:

1. $\varphi \in cl(\varphi)$,
2. $\phi \land \psi \in cl(\varphi)$ implies $\phi, \psi \in cl(\varphi)$,
3. $\neg\psi \in cl(\varphi)$ implies $\psi \in cl(\varphi)$,
4. $\psi \in cl(\varphi)$ and $\psi \neq \neg\phi$ implies $\neg\psi \in cl(\varphi)$,

Definition 6.9 (Closure $cl(\varphi)$)

The **closure** $cl(\varphi)$ is defined as follows:

1. $\varphi \in cl(\varphi)$,
2. $\phi \wedge \psi \in cl(\varphi)$ implies $\phi, \psi \in cl(\varphi)$,
3. $\neg\psi \in cl(\varphi)$ implies $\psi \in cl(\varphi)$,
4. $\psi \in cl(\varphi)$ and $\psi \neq \neg\phi$ implies $\neg\psi \in cl(\varphi)$,
5. $\bigcirc\psi \in cl(\varphi)$ implies $\psi \in cl(\varphi)$,

Definition 6.9 (Closure $cl(\varphi)$)

The closure $cl(\varphi)$ is defined as follows:

1. $\varphi \in cl(\varphi)$,
2. $\phi \wedge \psi \in cl(\varphi)$ implies $\phi, \psi \in cl(\varphi)$,
3. $\neg\psi \in cl(\varphi)$ implies $\psi \in cl(\varphi)$,
4. $\psi \in cl(\varphi)$ and $\psi \neq \neg\phi$ implies $\neg\psi \in cl(\varphi)$,
5. $\bigcirc\psi \in cl(\varphi)$ implies $\psi \in cl(\varphi)$,
6. $\psi \mathcal{U} \phi \in cl(\varphi)$ implies $\psi, \phi \in cl(\varphi)$.

Note, that it holds that

Definition 6.9 (Closure $cl(\varphi)$)

The closure $cl(\varphi)$ is defined as follows:

1. $\varphi \in cl(\varphi)$,
2. $\phi \wedge \psi \in cl(\varphi)$ implies $\phi, \psi \in cl(\varphi)$,
3. $\neg\psi \in cl(\varphi)$ implies $\psi \in cl(\varphi)$,
4. $\psi \in cl(\varphi)$ and $\psi \neq \neg\phi$ implies $\neg\psi \in cl(\varphi)$,
5. $\bigcirc\psi \in cl(\varphi)$ implies $\psi \in cl(\varphi)$,
6. $\psi\mathcal{U}\phi \in cl(\varphi)$ implies $\psi, \phi \in cl(\varphi)$.

Note, that it holds that $|cl(\varphi)| \leq 2|\varphi|$.

Example 6.10 (Closure)

How does the closure for $\varphi = \mathsf{r}\,\mathcal{U}\,(\mathsf{s} \vee \mathsf{t})$ look like?

Example 6.10 (Closure)

How does the closure for $\varphi = \mathsf{r}\,\mathcal{U}\,(\mathsf{s} \vee \mathsf{t})$ look like?
The closure $cl(\varphi)$ consists of the following formulae:

1. φ
2. $s \vee t$
3. r
4. s
5. t

and their negations!

What other properties should such sets fulfill? Note, that we are interested in a correspondence to runs.

Definition 6.11 (Logically consistent)

We call $B \subseteq cl(\varphi)$ propositionally consistent iff for all $\varphi_1 \wedge \varphi_2, \psi \in cl(\varphi)$:

Definition 6.11 (Logically consistent)

We call $B \subseteq cl(\varphi)$ propositionally consistent iff for all $\varphi_1 \wedge \varphi_2, \psi \in cl(\varphi)$:

1. $\varphi_1 \wedge \varphi_2 \in B$ iff
2. $\psi \in B$ implies
3. $\top \in cl(\varphi)$ implies

We identify $\neg\neg\varphi$ with φ.

Definition 6.11 (Logically consistent)

We call $B \subseteq cl(\varphi)$ propositionally consistent iff for all $\varphi_1 \wedge \varphi_2, \psi \in cl(\varphi)$:

1. $\varphi_1 \wedge \varphi_2 \in B$ iff $\varphi_1 \in B$ and $\varphi_2 \in B$,
2. $\psi \in B$ implies
3. $\top \in cl(\varphi)$ implies

We identify $\neg\neg\varphi$ with φ.

Definition 6.11 (Logically consistent)

We call $B \subseteq cl(\varphi)$ propositionally consistent iff for all $\varphi_1 \wedge \varphi_2, \psi \in cl(\varphi)$:

1. $\varphi_1 \wedge \varphi_2 \in B$ iff $\varphi_1 \in B$ and $\varphi_2 \in B$,
2. $\psi \in B$ implies $\neg\psi \notin B$,
3. $\top \in cl(\varphi)$ implies

We identify $\neg\neg\varphi$ with φ.

Definition 6.11 (Logically consistent)

We call $B \subseteq cl(\varphi)$ propositionally consistent iff for all $\varphi_1 \wedge \varphi_2, \psi \in cl(\varphi)$:

1. $\varphi_1 \wedge \varphi_2 \in B$ iff $\varphi_1 \in B$ and $\varphi_2 \in B$,
2. $\psi \in B$ implies $\neg\psi \notin B$,
3. $\top \in cl(\varphi)$ implies $\top \in B$.

We identify $\neg\neg\varphi$ with φ.

Definition 6.11 (Logically consistent)

We call $B \subseteq cl(\varphi)$ **propositionally consistent** iff for all $\varphi_1 \wedge \varphi_2, \psi \in cl(\varphi)$:

1. $\varphi_1 \wedge \varphi_2 \in B$ iff $\varphi_1 \in B$ and $\varphi_2 \in B$,
2. $\psi \in B$ implies $\neg\psi \notin B$,
3. $\top \in cl(\varphi)$ implies $\top \in B$.

We identify $\neg\neg\varphi$ with φ.

Definition 6.12 (Locally consistent)

We call $B \subseteq cl(\varphi)$ **locally consistent** iff for all $\varphi_1 \,\mathcal{U}\, \varphi_2 \in cl(\varphi)$:

Definition 6.11 (Logically consistent)

We call $B \subseteq cl(\varphi)$ propositionally consistent iff for all $\varphi_1 \wedge \varphi_2, \psi \in cl(\varphi)$:

1. $\varphi_1 \wedge \varphi_2 \in B$ iff $\varphi_1 \in B$ and $\varphi_2 \in B$,
2. $\psi \in B$ implies $\neg\psi \notin B$,
3. $\top \in cl(\varphi)$ implies $\top \in B$.

We identify $\neg\neg\varphi$ with φ.

Definition 6.12 (Locally consistent)

We call $B \subseteq cl(\varphi)$ locally consistent iff for all $\varphi_1 \mathcal{U} \varphi_2 \in cl(\varphi)$:

1. $\varphi_2 \in B$ implies .
2. $\varphi_1 \mathcal{U} \varphi_2 \in B$ and $\varphi_2 \notin B$ implies .

Definition 6.11 (Logically consistent)

We call $B \subseteq cl(\varphi)$ propositionally consistent iff for all $\varphi_1 \wedge \varphi_2, \psi \in cl(\varphi)$:

1. $\varphi_1 \wedge \varphi_2 \in B$ iff $\varphi_1 \in B$ and $\varphi_2 \in B$,
2. $\psi \in B$ implies $\neg\psi \notin B$,
3. $\top \in cl(\varphi)$ implies $\top \in B$.

We identify $\neg\neg\varphi$ with φ.

Definition 6.12 (Locally consistent)

We call $B \subseteq cl(\varphi)$ locally consistent iff for all $\varphi_1 \, \mathcal{U} \, \varphi_2 \in cl(\varphi)$:

1. $\varphi_2 \in B$ implies $\varphi_1 \, \mathcal{U} \, \varphi_2 \in B$.
2. $\varphi_1 \, \mathcal{U} \, \varphi_2 \in B$ and $\varphi_2 \notin B$ implies .

Definition 6.11 (Logically consistent)

We call $B \subseteq cl(\varphi)$ propositionally consistent iff for all $\varphi_1 \wedge \varphi_2, \psi \in cl(\varphi)$:

1. $\varphi_1 \wedge \varphi_2 \in B$ iff $\varphi_1 \in B$ and $\varphi_2 \in B$,
2. $\psi \in B$ implies $\neg\psi \notin B$,
3. $\top \in cl(\varphi)$ implies $\top \in B$.

We identify $\neg\neg\varphi$ with φ.

Definition 6.12 (Locally consistent)

We call $B \subseteq cl(\varphi)$ locally consistent iff for all $\varphi_1 \mathcal{U} \varphi_2 \in cl(\varphi)$:

1. $\varphi_2 \in B$ implies $\varphi_1 \mathcal{U} \varphi_2 \in B$.
2. $\varphi_1 \mathcal{U} \varphi_2 \in B$ and $\varphi_2 \notin B$ implies $\varphi_1 \in B$.

Definition 6.13 (Maximal consistent)

We call $B \subseteq cl(\varphi)$ **maximal** iff for all $\psi \in cl(\varphi)$

$$\psi \notin B \quad \text{implies} \quad \neg\psi \in B.$$

We identify $\neg\neg\varphi$ with φ.

Definition 6.13 (Maximal consistent)

We call $B \subseteq cl(\varphi)$ **maximal** iff for all $\psi \in cl(\varphi)$

$$\psi \notin B \quad \text{implies} \quad \neg\psi \in B.$$

We identify $\neg\neg\varphi$ with φ.

Definition 6.14 (Elementary, $\mathcal{EL}(\varphi)$)

We call $B \subseteq cl(\varphi)$ **elementary** iff B is propositionally and locally consistent and maximal.

Definition 6.13 (Maximal consistent)

We call $B \subseteq cl(\varphi)$ maximal iff for all $\psi \in cl(\varphi)$

$$\psi \notin B \quad \text{implies} \quad \neg\psi \in B.$$

We identify $\neg\neg\varphi$ with φ.

Definition 6.14 (Elementary, $\mathcal{EL}(\varphi)$)

We call $B \subseteq cl(\varphi)$ elementary iff B is propositionally and locally consistent and maximal.
We define $\mathcal{EL}(\varphi)$ as the set of all elementary subsets of $cl(\varphi)$.

In the following we construct infinite words over $\mathcal{EL}(\varphi)$ that corresponds to accepting paths.

The closure of $\varphi = \mathsf{r}\,\mathcal{U}\,\mathsf{s}$ is given by $\{\varphi, \neg\varphi, \mathsf{r}, \mathsf{s}, \neg\mathsf{r}, \neg\mathsf{s}\}$.
Which of the following sets are elementary?

1. $\emptyset$
2. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \mathsf{s}\}$
3. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}\}$
4. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \neg\mathsf{r}, \neg\mathsf{s}\}$
5. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \neg\mathsf{r}, \mathsf{s}\}$
6. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \neg\mathsf{s}\}$
7. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \neg\mathsf{r}, \neg\mathsf{s}\}$
8. $\{\neg(\mathsf{r}\,\mathcal{U}\,\mathsf{s}), \mathsf{r}, \neg\mathsf{s}\}$
9. $\{\neg(\mathsf{r}\,\mathcal{U}\,\mathsf{s}), \neg\mathsf{r}, \neg\mathsf{s}\}$

The closure of $\varphi = \mathsf{r}\,\mathcal{U}\,\mathsf{s}$ is given by $\{\varphi, \neg\varphi, \mathsf{r}, \mathsf{s}, \neg\mathsf{r}, \neg\mathsf{s}\}$.
Which of the following sets are elementary?

1. $\emptyset$ not maximal
2. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \mathsf{s}\}$
3. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}\}$
4. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \neg\mathsf{r}, \neg\mathsf{s}\}$
5. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \neg\mathsf{r}, \mathsf{s}\}$
6. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \neg\mathsf{s}\}$
7. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \neg\mathsf{r}, \neg\mathsf{s}\}$
8. $\{\neg(\mathsf{r}\,\mathcal{U}\,\mathsf{s}), \mathsf{r}, \neg\mathsf{s}\}$
9. $\{\neg(\mathsf{r}\,\mathcal{U}\,\mathsf{s}), \neg\mathsf{r}, \neg\mathsf{s}\}$

The closure of $\varphi = \mathsf{r}\,\mathcal{U}\,\mathsf{s}$ is given by $\{\varphi, \neg\varphi, \mathsf{r}, \mathsf{s}, \neg\mathsf{r}, \neg\mathsf{s}\}$. Which of the following sets are elementary?

1. $\emptyset$ not maximal
2. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \mathsf{s}\}$ yes
3. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}\}$
4. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \neg\mathsf{r}, \neg\mathsf{s}\}$
5. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \neg\mathsf{r}, \mathsf{s}\}$
6. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \neg\mathsf{s}\}$
7. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \neg\mathsf{r}, \neg\mathsf{s}\}$
8. $\{\neg(\mathsf{r}\,\mathcal{U}\,\mathsf{s}), \mathsf{r}, \neg\mathsf{s}\}$
9. $\{\neg(\mathsf{r}\,\mathcal{U}\,\mathsf{s}), \neg\mathsf{r}, \neg\mathsf{s}\}$

The closure of $\varphi = \mathsf{r}\,\mathcal{U}\,\mathsf{s}$ is given by $\{\varphi, \neg\varphi, \mathsf{r}, \mathsf{s}, \neg\mathsf{r}, \neg\mathsf{s}\}$. Which of the following sets are elementary?

1. $\emptyset$ not maximal
2. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \mathsf{s}\}$ yes
3. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}\}$ not maximal
4. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \neg\mathsf{r}, \neg\mathsf{s}\}$
5. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \neg\mathsf{r}, \mathsf{s}\}$
6. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \neg\mathsf{s}\}$
7. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \neg\mathsf{r}, \neg\mathsf{s}\}$
8. $\{\neg(\mathsf{r}\,\mathcal{U}\,\mathsf{s}), \mathsf{r}, \neg\mathsf{s}\}$
9. $\{\neg(\mathsf{r}\,\mathcal{U}\,\mathsf{s}), \neg\mathsf{r}, \neg\mathsf{s}\}$

The closure of $\varphi = \mathsf{r}\,\mathcal{U}\,\mathsf{s}$ is given by $\{\varphi, \neg\varphi, \mathsf{r}, \mathsf{s}, \neg\mathsf{r}, \neg\mathsf{s}\}$. Which of the following sets are elementary?

1. $\emptyset$ not maximal
2. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \mathsf{s}\}$ yes
3. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}\}$ not maximal
4. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \neg\mathsf{r}, \neg\mathsf{s}\}$ not locally consistent
5. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \neg\mathsf{r}, \mathsf{s}\}$
6. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \neg\mathsf{s}\}$
7. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \neg\mathsf{r}, \neg\mathsf{s}\}$
8. $\{\neg(\mathsf{r}\,\mathcal{U}\,\mathsf{s}), \mathsf{r}, \neg\mathsf{s}\}$
9. $\{\neg(\mathsf{r}\,\mathcal{U}\,\mathsf{s}), \neg\mathsf{r}, \neg\mathsf{s}\}$

The closure of $\varphi = \mathsf{r}\,\mathcal{U}\,\mathsf{s}$ is given by $\{\varphi, \neg\varphi, \mathsf{r}, \mathsf{s}, \neg\mathsf{r}, \neg\mathsf{s}\}$. Which of the following sets are elementary?

1. $\emptyset$ not maximal
2. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \mathsf{s}\}$ yes
3. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}\}$ not maximal
4. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \neg\mathsf{r}, \neg\mathsf{s}\}$ not locally consistent
5. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \neg\mathsf{r}, \mathsf{s}\}$ yes
6. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \neg\mathsf{s}\}$
7. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \neg\mathsf{r}, \neg\mathsf{s}\}$
8. $\{\neg(\mathsf{r}\,\mathcal{U}\,\mathsf{s}), \mathsf{r}, \neg\mathsf{s}\}$
9. $\{\neg(\mathsf{r}\,\mathcal{U}\,\mathsf{s}), \neg\mathsf{r}, \neg\mathsf{s}\}$

The closure of $\varphi = \mathsf{r}\,\mathcal{U}\,\mathsf{s}$ is given by $\{\varphi, \neg\varphi, \mathsf{r}, \mathsf{s}, \neg\mathsf{r}, \neg\mathsf{s}\}$. Which of the following sets are elementary?

1. $\emptyset$ not maximal
2. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \mathsf{s}\}$ yes
3. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}\}$ not maximal
4. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \neg\mathsf{r}, \neg\mathsf{s}\}$ not locally consistent
5. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \neg\mathsf{r}, \mathsf{s}\}$ yes
6. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \neg\mathsf{s}\}$ yes
7. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \neg\mathsf{r}, \neg\mathsf{s}\}$
8. $\{\neg(\mathsf{r}\,\mathcal{U}\,\mathsf{s}), \mathsf{r}, \neg\mathsf{s}\}$
9. $\{\neg(\mathsf{r}\,\mathcal{U}\,\mathsf{s}), \neg\mathsf{r}, \neg\mathsf{s}\}$

The closure of $\varphi = \mathsf{r}\,\mathcal{U}\,\mathsf{s}$ is given by $\{\varphi, \neg\varphi, \mathsf{r}, \mathsf{s}, \neg\mathsf{r}, \neg\mathsf{s}\}$.
Which of the following sets are elementary?

1. $\emptyset$ not maximal
2. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \mathsf{s}\}$ yes
3. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}\}$ not maximal
4. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \neg\mathsf{r}, \neg\mathsf{s}\}$ not locally consistent
5. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \neg\mathsf{r}, \mathsf{s}\}$ yes
6. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \neg\mathsf{s}\}$ yes
7. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \neg\mathsf{r}, \neg\mathsf{s}\}$ not propositionally consistent
8. $\{\neg(\mathsf{r}\,\mathcal{U}\,\mathsf{s}), \mathsf{r}, \neg\mathsf{s}\}$
9. $\{\neg(\mathsf{r}\,\mathcal{U}\,\mathsf{s}), \neg\mathsf{r}, \neg\mathsf{s}\}$

The closure of $\varphi = \mathsf{r}\,\mathcal{U}\,\mathsf{s}$ is given by $\{\varphi, \neg\varphi, \mathsf{r}, \mathsf{s}, \neg\mathsf{r}, \neg\mathsf{s}\}$.
Which of the following sets are elementary?

1. $\emptyset$ not maximal
2. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \mathsf{s}\}$ yes
3. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}\}$ not maximal
4. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \neg\mathsf{r}, \neg\mathsf{s}\}$ not locally consistent
5. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \neg\mathsf{r}, \mathsf{s}\}$ yes
6. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \neg\mathsf{s}\}$ yes
7. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \neg\mathsf{r}, \neg\mathsf{s}\}$ not propositionally consistent
8. $\{\neg(\mathsf{r}\,\mathcal{U}\,\mathsf{s}), \mathsf{r}, \neg\mathsf{s}\}$ yes
9. $\{\neg(\mathsf{r}\,\mathcal{U}\,\mathsf{s}), \neg\mathsf{r}, \neg\mathsf{s}\}$

The closure of $\varphi = \mathsf{r}\,\mathcal{U}\,\mathsf{s}$ is given by $\{\varphi, \neg\varphi, \mathsf{r}, \mathsf{s}, \neg\mathsf{r}, \neg\mathsf{s}\}$.
Which of the following sets are elementary?

1. $\emptyset$ not maximal
2. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \mathsf{s}\}$ yes
3. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}\}$ not maximal
4. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \neg\mathsf{r}, \neg\mathsf{s}\}$ not locally consistent
5. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \neg\mathsf{r}, \mathsf{s}\}$ yes
6. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \neg\mathsf{s}\}$ yes
7. $\{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \neg\mathsf{r}, \neg\mathsf{s}\}$ not propositionally consistent
8. $\{\neg(\mathsf{r}\,\mathcal{U}\,\mathsf{s}), \mathsf{r}, \neg\mathsf{s}\}$ yes
9. $\{\neg(\mathsf{r}\,\mathcal{U}\,\mathsf{s}), \neg\mathsf{r}, \neg\mathsf{s}\}$ yes

Example 6.15 (Elementary sets)

The closure of $\varphi = \mathsf{r}\,\mathcal{U}\,\mathsf{s}$ is given by

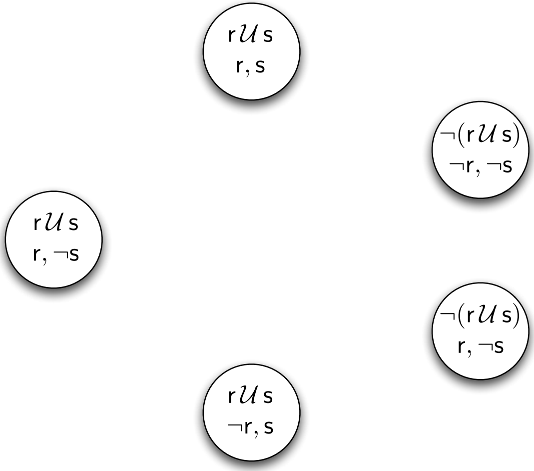
$$cl(\varphi) = \{\varphi, \neg\varphi, \mathsf{r}, \mathsf{s}, \neg\mathsf{r}, \neg\mathsf{s}\}$$

The following list contains all elementary sets of φ:

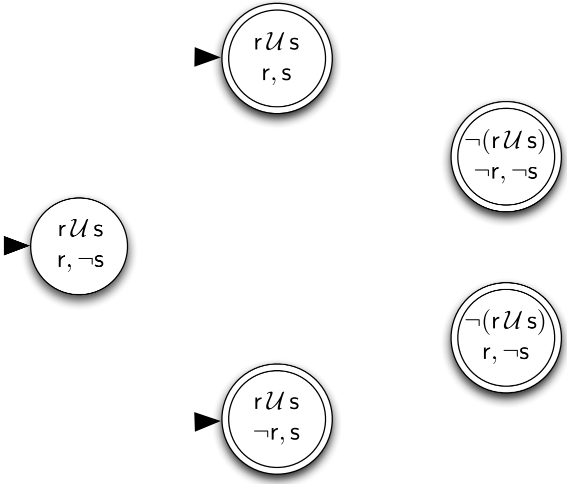
1. $E_1 = \{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \mathsf{s}\}$
2. $E_2 = \{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \neg\mathsf{r}, \mathsf{s}\}$
3. $E_3 = \{\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \neg\mathsf{s}\}$
4. $E_4 = \{\neg\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \mathsf{r}, \neg\mathsf{s}\}$
5. $E_5 = \{\neg\mathsf{r}\,\mathcal{U}\,\mathsf{s}, \neg\mathsf{r}, \neg\mathsf{s}\}$

In the following, we construct the Büchi atuomaton A_φ for $\varphi = \mathsf{r}\,\mathcal{U}\,\mathsf{s}$.

Constructing the Automaton for r $\mathcal{U}$ s

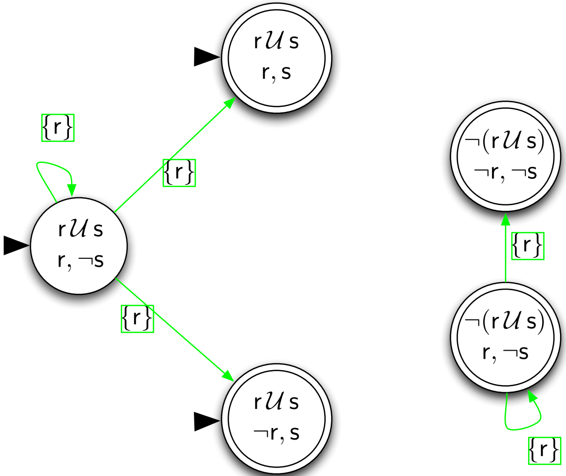


- Initial states?
  $\{s \in S \mid \varphi \in s\}$
- Accepting states?
  If $\varphi_1 \mathcal{U} \varphi_2 \in cl(\varphi)$ then
  $\varphi_1 \mathcal{U} \varphi_2 \notin s$ or
  $\varphi_2 \in s$



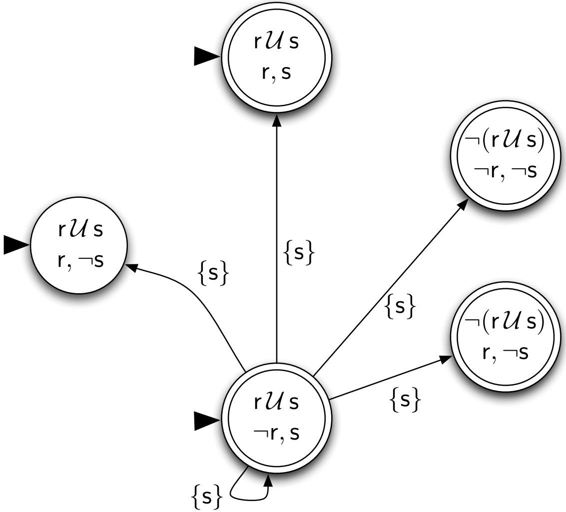
- Initial states?
  $\{s \in S \mid \varphi \in s\}$
- Accepting states?
  If $\varphi_1 \mathcal{U} \varphi_2 \in cl(\varphi)$ then
  $\varphi_1 \mathcal{U} \varphi_2 \notin s$ or
  $\varphi_2 \in s$
- $\rightsquigarrow A$ reads $\{r\}$

$(s, a, t) \in \Delta$ then $\forall \mathrm{r}\mathcal{U}\mathrm{s} \in cl(\varphi)$:
$\mathrm{r}\mathcal{U}\mathrm{s} \in s$ iff $(\mathrm{s} \in s$ or $(\mathrm{r} \in s$ and $\mathrm{r}\mathcal{U}\mathrm{s} \in t))$



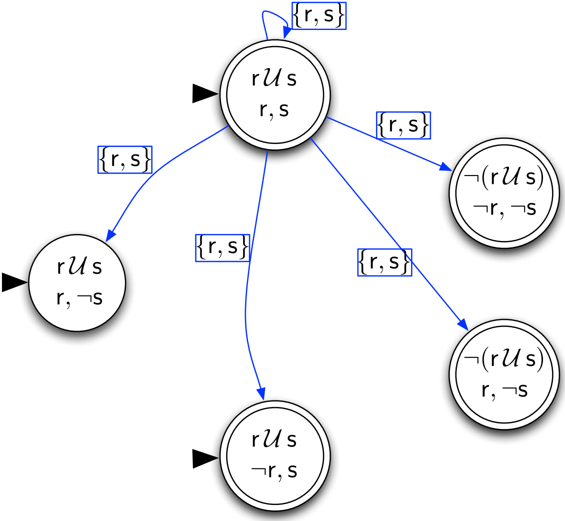
- A reads {r}
- $\rightsquigarrow$ A reads {s}

$(s, a, t) \in \Delta$ then $\forall \mathsf{r}\,\mathcal{U}\,\mathsf{s} \in cl(\varphi)$:
$\mathsf{r}\,\mathcal{U}\,\mathsf{s} \in s$ iff ($\mathsf{s} \in s$ or ($\mathsf{r} \in s$ and $\mathsf{r}\,\mathcal{U}\,\mathsf{s} \in t$))



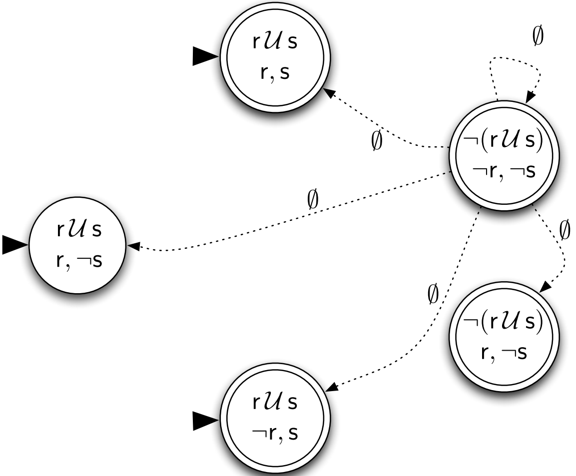
- A reads {s}
- $\rightsquigarrow$ A reads {r, s}

$(s, a, t) \in \Delta$ then $\forall \mathsf{r}\,\mathcal{U}\,\mathsf{s} \in cl(\varphi)$:
$\mathsf{r}\,\mathcal{U}\,\mathsf{s} \in s$ iff $(\mathsf{s} \in s$ or $(\mathsf{r} \in s$ and $\mathsf{r}\,\mathcal{U}\,\mathsf{s} \in t))$



- A reads $\{r, s\}$
- $\rightsquigarrow A$ reads $\emptyset$

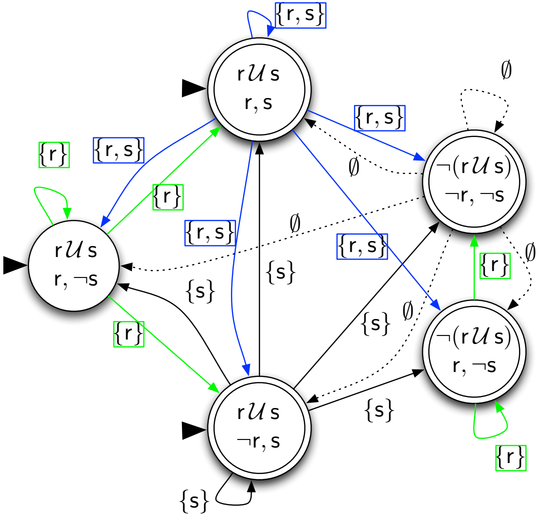
$(s, a, t) \in \Delta$ then $\forall r\,\mathcal{U}\,s \in cl(\varphi)$:
$r\,\mathcal{U}\,s \in s$ iff $(s \in s$ or $(r \in s$ and $r\,\mathcal{U}\,s \in t))$



- A reads $\emptyset$

$(s, a, t) \in \Delta$ then $\forall \mathsf{r}\,\mathcal{U}\,\mathsf{s} \in cl(\varphi)$:

$\mathsf{r}\,\mathcal{U}\,\mathsf{s} \in s$ iff ($\mathsf{s} \in s$ or ($\mathsf{r} \in s$ and $\mathsf{r}\,\mathcal{U}\,\mathsf{s} \in t$))



- The complete automaton

$(s, a, t) \in \Delta$ then $\forall \mathsf{r}\,\mathcal{U}\,\mathsf{s} \in cl(\varphi)$:
$\mathsf{r}\,\mathcal{U}\,\mathsf{s} \in s$ iff $(\mathsf{s} \in s$ or $(\mathsf{r} \in s$ and $\mathsf{r}\,\mathcal{U}\,\mathsf{s} \in t))$

Theorem 6.16 (LTL [Sistla and Clarke, 1985, Lichtenstein and Pnueli, 1985, Vardi and Wolper, 1986])

Model checking **LTL is $PSPACE$-complete**, *and can be done in time $2^{\mathbf{O}(|\varphi|)}\mathbf{O}(|\mathcal{M}|)$, where $|\mathcal{M}|$ is given by the number of transitions.*

Proof: Upper Bound

Given an $\mathcal{L}_{LTL}$-formula φ.

1. Construct Büchi automaton $\mathcal{A}_{\neg\varphi}$ of size $2^{\mathbf{O}(|\varphi|)}$ accepting exactly the words satisfying $\neg\varphi$.

Proof: Upper Bound

Given an $\mathcal{L}_{LTL}$-formula φ.

1. Construct Büchi automaton $\mathcal{A}_{\neg\varphi}$ of size $2^{\mathbf{O}(|\varphi|)}$ accepting exactly the words satisfying $\neg\varphi$.
2. Kripke model $\mathcal{M}, q$ can directly be interpreted as a Büchi automaton $\mathcal{A}_{\mathcal{M},q}$ of size $\mathbf{O}(|\mathcal{M}|)$ accepting all possible words in the Kripke model starting in q.

Proof: Upper Bound

Given an $\mathcal{L}_{LTL}$-formula φ.

1. Construct Büchi automaton $\mathcal{A}_{\neg\varphi}$ of size $2^{\mathbf{O}(|\varphi|)}$ accepting exactly the words satisfying $\neg\varphi$.
2. Kripke model $\mathcal{M}, q$ can directly be interpreted as a Büchi automaton $\mathcal{A}_{\mathcal{M},q}$ of size $\mathbf{O}(|\mathcal{M}|)$ accepting all possible words in the Kripke model starting in q.
3. The model checking problem reduces to the emptiness check of $L(\mathcal{A}_{\mathcal{M},q}) \cap L(\mathcal{A}_{\neg\varphi})$ which can be done in polynomial time wrt the size of the automaton (cf.pp. 769). That is, in time $\mathbf{O}(|\mathcal{M}|) \cdot 2^{\mathbf{O}(|\varphi|)}$ by constructing the product automaton.

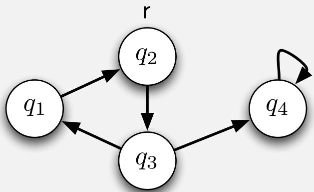
6.4 MC of CTL*

Theorem 6.17 (CTL* [Clarke et al., 1986, Emerson and Lei, 1987])

Model checking CTL is $PSPACE$-complete, and can be done in time $2^{\mathbf{O}(|\varphi|)}\mathbf{O}(|\mathcal{M}|)$, where $|\mathcal{M}|$ is given by the number of transitions.*

Example 6.18 (LTL mcheck for CTL mcheck)

In which states does $\varphi = \mathsf{E}\Diamond\Box\mathsf{A}\Box\Diamond\neg\mathsf{r}$ hold? How to use **LTL** model checking?



Proof.

Upper bound: Combine **CTL** and **LTL** model checking.

- Consider $\mathcal{L}_{CTL^*}$-formula φ containing $\mathsf{E}\psi$ where ψ is a pure $\mathcal{L}_{LTL}$-formula.

Proof.

Upper bound: Combine **CTL** and **LTL** model checking.

- Consider $\mathcal{L}_{CTL^*}$-formula φ containing $\mathsf{E}\psi$ where ψ is a pure $\mathcal{L}_{LTL}$-formula.
- Determine all states which satisfy $\mathsf{E}\psi$ (these are all states q with $\mathcal{M}, q \not\models^{\mathbf{LTL}} \neg\psi$), Complexity: $PSPACE$.

Proof.

Upper bound: Combine **CTL** and **LTL** model checking.

- Consider $\mathcal{L}_{CTL^*}$-formula φ containing $\mathsf{E}\psi$ where ψ is a pure $\mathcal{L}_{LTL}$-formula.
- Determine all states which satisfy $\mathsf{E}\psi$ (these are all states q with $\mathcal{M}, q \not\models^{\mathbf{LTL}} \neg\psi$), Complexity: $PSPACE$.
- Label them by a fresh proposition, say p, and replace $\mathsf{E}\psi$ in φ by p: $\mathsf{E}\bigcirc\overbrace{(\mathsf{r} \wedge \underbrace{\mathsf{E}\Diamond\mathsf{s}}_{\mathsf{p}_1})}^{\mathsf{p}_2} \rightsquigarrow \mathsf{E}\bigcirc(\mathsf{p}_2 \wedge \mathsf{p}_1)$

Proof.

Upper bound: Combine **CTL** and **LTL** model checking.

- Consider $\mathcal{L}_{CTL^*}$-formula φ containing Eψ where ψ is a pure $\mathcal{L}_{LTL}$-formula.
- Determine all states which satisfy Eψ (these are all states q with $\mathcal{M}, q \not\models^{\mathsf{LTL}} \neg\psi$), Complexity: $PSPACE$.
- Label them by a fresh proposition, say p, and replace Eψ in φ by p: $\mathsf{E}\bigcirc\left(\overbrace{\mathsf{r} \wedge \underbrace{\mathsf{E}\Diamond\mathsf{s}}_{\mathsf{p}_1}}^{\mathsf{p}_2}\right) \rightsquigarrow \mathsf{E}\bigcirc(\mathsf{p}_2 \wedge \mathsf{p}_1)$

Applying this procedure recursively yields a pure $\mathcal{L}_{CTL}$-formula which can be verified in polynomial time.
Complexity: $P^{PSPACE} = PSPACE$

Hardness: immediate from Theorem 6.16. $\square$

Summary

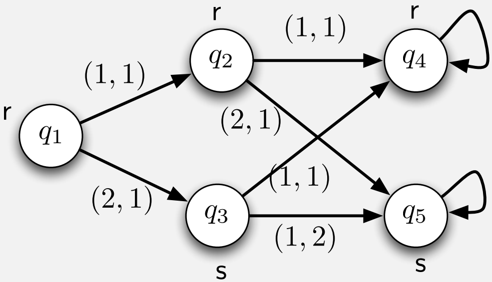
- Model checking **CTL is P-complete**.
- Model checking **LTL is $PSPACE$-complete**. The algorithm has been constructed from Büchi automata.
- Model checking **CTL* is also $PSPACE$-complete**. The algorithm is obtained by the ones for **CTL** and **LTL**.

6.5 MC of ATL

Example 6.19

Which formulae are true in the model?

1. $\mathcal{M}, q_1 \models \langle\langle 1 \rangle\rangle \Box \mathsf{r}$
2. $\mathcal{M}, q_1 \models \langle\langle 1 \rangle\rangle \Box \mathsf{s}$
3. $\mathcal{M}, q_1 \models \langle\langle 1 \rangle\rangle \bigcirc \langle\langle 1 \rangle\rangle \Box \mathsf{r}$



The **ATL** model checking algorithm employs the well-known fixpoint characterisations :

$$\begin{aligned}
\langle\langle A\rangle\rangle\Box\varphi &\leftrightarrow \varphi \wedge \langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\Box\varphi,\\
\langle\langle A\rangle\rangle\varphi_1\,\mathcal{U}\,\varphi_2 &\leftrightarrow \varphi_2 \vee \varphi_1 \wedge \langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\varphi_1\,\mathcal{U}\,\varphi_2.
\end{aligned}$$

The **ATL** model checking algorithm employs the well-known fixpoint characterisations :

$$\begin{aligned} \langle\langle A\rangle\rangle\Box\varphi &\leftrightarrow \varphi \wedge \langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\Box\varphi, \\ \langle\langle A\rangle\rangle\varphi_1\,\mathcal{U}\,\varphi_2 &\leftrightarrow \varphi_2 \vee \varphi_1 \wedge \langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\varphi_1\,\mathcal{U}\,\varphi_2. \end{aligned}$$

Do these characterisations also hold for incomplete information?

The **ATL** model checking algorithm employs the well-known fixpoint characterisations :

$$\begin{aligned}\langle\langle A\rangle\rangle\Box\varphi &\leftrightarrow \varphi\wedge\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\Box\varphi,\\ \langle\langle A\rangle\rangle\varphi_1\,\mathcal{U}\,\varphi_2 &\leftrightarrow \varphi_2\vee\varphi_1\wedge\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\varphi_1\,\mathcal{U}\,\varphi_2.\end{aligned}$$

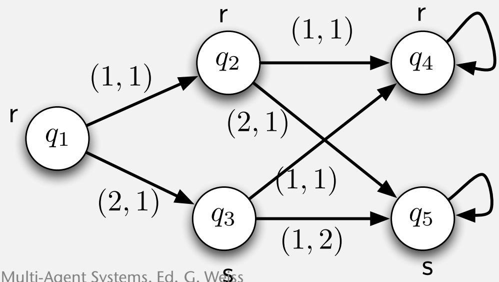
Do these characterisations also hold for incomplete information?

No! A choice of an action at a state q has non-local consequences: It automatically fixes choices at all states q' indistinguishable from q for the coalition A.

Again, crucial for model checking is the notion of preimage.

Example 6.20 (Preimage operator for ATL)

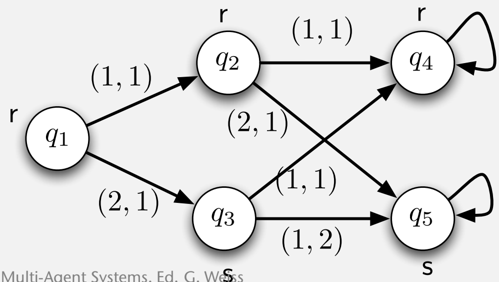
1. What is the preimage of $\{q_2, q_3\}$?
2. What is the preimage of $\{q_2\}$?



Example 6.20 (Preimage operator for ATL)

1. What is the preimage of $\{q_2, q_3\}$?
2. What is the preimage of $\{q_2\}$?

Careful: The preimage depends on a group of agents which try to reach a given region.

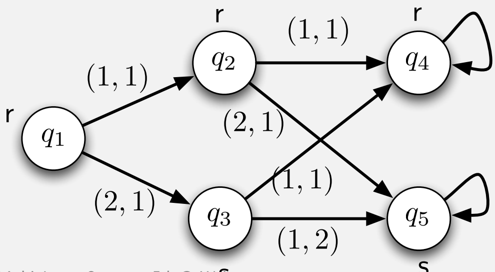


Example 6.20 (Preimage operator for ATL)

1. What is the **preimage** of $\{q_2, q_3\}$?
2. What is the **preimage** of $\{q_2\}$?

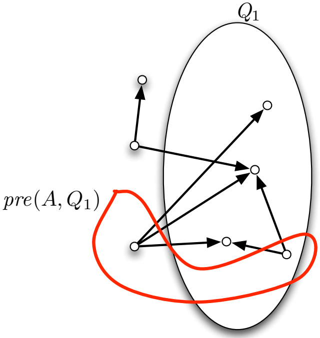
Careful: The preimage **depends on a group of agents** which try to reach a given region.

1. What is the **preimage** of $\{q_2, q_3\}$ wrt. any group A?
2. What is the **preimage** of $\{q_2\}$ wrt. $\{1\}$ and $\{2\}$?



function $pre(M, A, Q)$.
Auxiliary function; returns the exact set of states Q' such that, when the system is in a state $q \in Q'$, agents A can cooperate and enforce the next state to be in Q.
return $\{q \mid \exists \alpha_A \forall \alpha_{\mathbb{A}\mathrm{gt} \setminus A}\ o(q, \alpha_A, \alpha_{\mathbb{A}\mathrm{gt} \setminus A}) \in Q\}$

The function follows the same idea as the pre-image function of **CTL** model checking.



Note that: **ATL** $=$ **ATL$_{Ir}$** $=$ **ATL$_{IR}$** (cf. Theorem 2.20)

Theorem 6.21 (ATL$_{Ir}$ and ATL$_{IR}$ [Alur et al., 2002])

Model checking ATL$_{Ir}$ and ATL$_{IR}$ is P-complete, and can be done in time $\mathrm{O}(|\mathcal{M}| \cdot |\varphi|)$, where $|\mathcal{M}|$ is given by the number of transitions in $\mathcal{M}$.

Note that: **ATL** $=$ **ATL**$_{Ir}$ $=$ **ATL**$_{IR}$ (cf. Theorem 2.20)

Theorem 6.21 (ATL$_{Ir}$ and ATL$_{IR}$ [Alur et al., 2002])

Model checking ATL$_{Ir}$ and ATL$_{IR}$ is P-complete, and can be done in time $\mathrm{O}(|\mathcal{M}| \cdot |\varphi|)$, where $|\mathcal{M}|$ is given by the number of transitions in $\mathcal{M}$.

Note, that the size of $\mathcal{M}$ is exponential in the number of states and agents!

Note that: **ATL** $=$ **ATL**$_{Ir}$ $=$ **ATL**$_{IR}$ (cf. Theorem 2.20)

Theorem 6.21 (ATL$_{Ir}$ and ATL$_{IR}$ [Alur et al., 2002])

Model checking ATL$_{Ir}$ and ATL$_{IR}$ is P-complete, and can be done in time $\mathrm{O}(|\mathcal{M}| \cdot |\varphi|)$, where $|\mathcal{M}|$ is given by the number of transitions in $\mathcal{M}$.

Note, that the size of $\mathcal{M}$ is exponential in the number of states and agents!

Besides the new definition of the preimage function the algorithm is the same as for **CTL**:

function $mcheck(M, \varphi)$.

Returns states q with $\mathcal{M}, q \models \varphi$.

case $\varphi \in \Pi$: return $\pi(p)$
case $\varphi = \neg\psi$: return $St \setminus mcheck(M, \psi)$
case $\varphi = \psi_1 \vee \psi_2$: return $mcheck(M, \psi_1) \cup mcheck(M, \psi_2)$
case $\varphi = \langle\langle A\rangle\rangle \bigcirc \psi$: return $pre(M, A, mcheck(M, \psi))$
case $\varphi = \langle\langle A\rangle\rangle \Box \psi$:
  $Q_1 := St$; $Q_2 := mcheck(M, \psi)$; $Q_3 := Q_2$;
  while $Q_1 \not\subseteq Q_2$
  do $Q_1 := Q_2$; $Q_2 := pre(M, A, Q_1) \cap Q_3$ **od**;
  return Q_1
case $\varphi = \langle\langle A\rangle\rangle \psi_1 \mathcal{U} \psi_2$:
  $Q_1 := \varnothing$; $Q_2 := mcheck(M, \psi_1)$;
  $Q_3 := mcheck(M, \psi_2)$;
  while $Q_3 \not\subseteq Q_1$
  do $Q_1 := Q_1 \cup Q_3$; $Q_3 := pre(M, A, Q_1) \cap Q_2$ **od**;
  return Q_1
end case

And-Or-Graph Reachability

For the lower bound, we reduce reachability in and-or-graphs.

An and-or graph [Immerman, 1981]

- is a tuple (E, V, l) such that $G = (E, V)$ is a directed acyclic graph and $l : V \rightarrow \{\wedge, \vee\}$ a labeling function.

And-Or-Graph Reachability

For the lower bound, we reduce reachability in and-or-graphs.

An and-or graph [Immerman, 1981]

- is a tuple (E, V, l) such that $G = (E, V)$ is a directed acyclic graph and $l : V \to \{\wedge, \vee\}$ a labeling function.

Let $x_1, \dots, x_n$ denote all successor nodes of u. v is said to be reachable from u iff

1. $u = v$; or
2. $l(u) = \wedge$, $n \geq 1$, and v is reachable from all x_i's; or,
3. $l(u) = \vee$, $n \geq 1$, and v is reachable from some x_i.

And-Or-Graph Reachability

For the lower bound, we reduce reachability in and-or-graphs.

An and-or graph [Immerman, 1981]

- is a tuple (E, V, l) such that $G = (E, V)$ is a directed acyclic graph and $l : V \to \{\wedge, \vee\}$ a labeling function.

Let $x_1, \ldots, x_n$ denote all successor nodes of u. v is said to be reachable from u iff

1. $u = v$; or
2. $l(u) = \wedge$, $n \geq 1$, and v is reachable from all x_i's; or,
3. $l(u) = \vee$, $n \geq 1$, and v is reachable from some x_i.

Theorem 6.22 ([Immerman, 1981])

The and-or-graph reachability *problem is* P-complete.

Theorem 6.22 ([Immerman, 1981])

The and-or-graph reachability *problem is* P-complete.

Proof: Lower Bound

Hardness is shown by a reduction of reachability in And-Or-Graphs:

- Transform and-or-graph to a CGS;

Theorem 6.22 ([Immerman, 1981])

The and-or-graph reachability problem is P-complete.

Proof: Lower Bound

Hardness is shown by a reduction of reachability in And-Or-Graphs:

- Transform and-or-graph to a CGS;
- Player 1 owns or-states;

Theorem 6.22 ([Immerman, 1981])

The and-or-graph reachability problem is P-complete.

Proof: Lower Bound

Hardness is shown by a reduction of reachability in And-Or-Graphs:

- Transform and-or-graph to a CGS;
- Player 1 owns or-states;
- Player 2 owns and-states;

Theorem 6.22 ([Immerman, 1981])

The and-or-graph reachability problem is P-complete.

Proof: Lower Bound

Hardness is shown by a reduction of reachability in And-Or-Graphs:

- Transform and-or-graph to a CGS;
- Player 1 owns or-states;
- Player 2 owns and-states;
- v reachable from a iff $\mathcal{M}, a \models \langle\langle 1 \rangle\rangle \Diamond \mathsf{l}_v$.

ATL* with perfect recall

For perfect recall, we cannot simply guess a strategy $St^+ \to Act$.

For model checking an automata theoretic approach is used. Consider the formula $\langle\langle A \rangle\rangle \psi$ where $\psi \in \mathcal{L}_{LTL}$ and CGS $\mathcal{M}$ and a state q.

1. A tree automaton $A_{\mathcal{M},q,A}$ is used to accept all possible executions in $\mathcal{M}$ which can be enforced by A following some strategy.

   (Note: $\langle\langle A \rangle\rangle \psi$ says that there is some "tree" such that ψ holds along all branches).

ATL* with perfect recall

For perfect recall, we cannot simply guess a strategy $St^{+} \to Act$.

For model checking an automata theoretic approach is used. Consider the formula $\langle\!\langle A\rangle\!\rangle\psi$ where $\psi \in \mathcal{L}_{LTL}$ and CGS $\mathcal{M}$ and a state q.

1. A tree automaton $A_{\mathcal{M},q,A}$ is used to accept all possible executions in $\mathcal{M}$ which can be enforced by A following some strategy.

   (Note: $\langle\!\langle A\rangle\!\rangle\psi$ says that there is some "tree" such that ψ holds along all branches).
2. A tree automaton A_{ψ} is constructed to accept all (tree-like) models satisfying the $\mathcal{L}_{CTL^*}$-formula Aψ.

ATL* with perfect recall

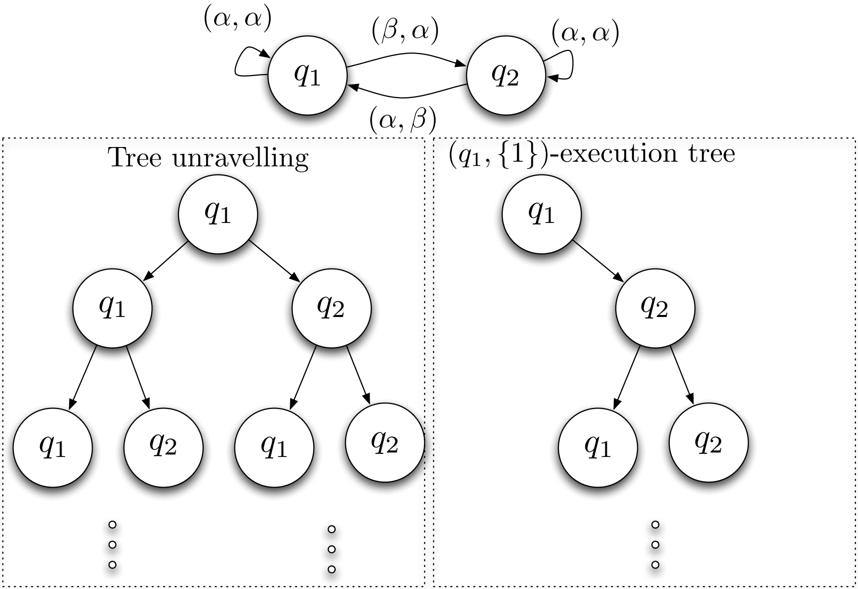
For perfect recall, we cannot simply guess a strategy $St^+ \to Act$.

For model checking an automata theoretic approach is used. Consider the formula $\langle\langle A\rangle\rangle\psi$ where $\psi \in \mathcal{L}_{LTL}$ and CGS $\mathcal{M}$ and a state q.

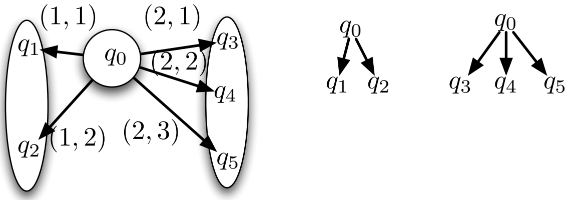
1. A tree automaton $A_{\mathcal{M},q,A}$ is used to accept all possible executions in $\mathcal{M}$ which can be enforced by A following some strategy.

   (Note: $\langle\langle A\rangle\rangle\psi$ says that there is some "tree" such that ψ holds along all branches).
2. A tree automaton A_ψ is constructed to accept all (tree-like) models satisfying the $\mathcal{L}_{CTL^*}$-formula Aψ.
3. We have: $\mathcal{M}, q \models \langle\langle A\rangle\rangle\psi$ **iff** $L(A_{\mathcal{M},q,A}) \cap L(A_\psi) \neq \emptyset$.

Execution trees



- An (q, A)-execution tree is induced by $out(q, s_A)$ for some strategy s_A of A.
- Intuitively, the transition relation of $A_{\mathcal{M},q,A}$ in a state q_0 is constructed from the different choices which A can enforce at q_0.



Theorem 6.23 (ATL*$_{IR}$ [Alur et al., 2002])

Model checking ATL$_{IR}$ is $2EXPTIME$-complete in the number of transitions in the model and the length of the formula.*

Complexity: Size of the automata and checking emptiness.

6.6 MC of MAS with Imperfect Information/Recall

Complexity Classes

Deterministic Turing machine (DTM)

- infinite (readable and writable) tape
- finitely many states
- deterministic moves

Complexity Classes

Deterministic Turing machine (DTM)

- infinite (readable and writable) tape
- finitely many states
- deterministic moves

Non-deterministic Turing machine (NTM)

Like a DTM but non-deterministic moves are allowed.

Orcale Machine (OTM)

- Let A be a language . An A-oracle machine is a DTM or NTM with a subroutine which allows to decide in one step whether $w \in A$ for some word w.

Orcale Machine (OTM)

- Let A be a language . An A-oracle machine is a DTM or NTM with a subroutine which allows to decide in one step whether $w \in A$ for some word w.
- For a complexity class $\mathcal{C}$ a $\mathcal{C}$-oracle machine is a A-oracle machine for any $A \in \mathcal{C}$.

Complexity Classes Σ_2^P, Δ_2^P, Δ_3^P

- Σ_i^P: problems solvable in polynomial time by a non-deterministic Turing machine making adaptive queries to a Σ_{i-1}^P oracle; i.e. by Σ_{i-1}^P-oracle polynomial time NTMs.

Complexity Classes Σ_2^P, Δ_2^P, Δ_3^P

- Σ_i^P: problems solvable in polynomial time by a non-deterministic Turing machine making adaptive queries to a Σ_{i-1}^P oracle; i.e. by Σ_{i-1}^P-oracle polynomial time NTMs.
- $\Sigma_2^P = NP^{NP}$: problems solvable in polynomial time by a non-deterministic Turing machine making adaptive queries to an NP oracle.

Complexity Classes $\Sigma_2^{\mathbf{P}}$, $\Delta_2^{\mathbf{P}}$, $\Delta_3^{\mathbf{P}}$

- $\Sigma_{\mathbf{i}}^{\mathbf{P}}$: problems solvable in polynomial time by a non-deterministic Turing machine making adaptive queries to a $\Sigma_{\mathbf{i-1}}^{\mathbf{P}}$ oracle; i.e. by $\Sigma_{\mathbf{i-1}}^{\mathbf{P}}$-oracle polynomial time NTMs.
- $\Sigma_{\mathbf{2}}^{\mathbf{P}} =$ NP$^{\text{NP}}$: problems solvable in polynomial time by a non-deterministic Turing machine making adaptive queries to an NP oracle.
- $\Delta_{\mathbf{2}}^{\mathbf{P}} =$ P$^{\text{NP}}$: A problem is in $\Delta_2^P = P^{NP}$ if it can be solved in deterministic polynomial time with subcalls to an NP-oracle. We also have $\Delta_3^P := P^{[NP^{NP}]}$ and $\Delta_1^P = P$.

$$P = \Delta_1^P \subseteq \Sigma_1^P = NP \subseteq \Delta_2^P \subseteq \Sigma_2^P \subseteq \cdots \subseteq PH \subseteq PSPACE.$$

Number of Strategies

We have introduced four types of strategies:

1. *ir*-strategies;
2. *Ir*-strategies;
3. *IR*-strategies;
4. *iR*-strategies.

How many strategies are there for each type?

Number of Strategies

We have introduced four types of strategies:

1. *ir*-strategies;
2. *Ir*-strategies;
3. *IR*-strategies;
4. *iR*-strategies.

How many strategies are there for each type?

1. exponentially many;

Number of Strategies

We have introduced four types of strategies:

1. *ir*-strategies;
2. *Ir*-strategies;
3. *IR*-strategies;
4. *iR*-strategies.

How many strategies are there for each type?

1. exponentially many;
2. exponentially many;

Number of Strategies

We have introduced four types of strategies:

1. *ir*-strategies;
2. *Ir*-strategies;
3. *IR*-strategies;
4. *iR*-strategies.

How many strategies are there for each type?

1. exponentially many;
2. exponentially many;
3. infinitely many;

Number of Strategies

We have introduced four types of strategies:

1. *ir*-strategies;
2. *Ir*-strategies;
3. *IR*-strategies;
4. *iR*-strategies.

How many strategies are there for each type?

1. exponentially many;
2. exponentially many;
3. infinitely many;
4. infinitely many.

Number of Strategies

We have introduced four types of strategies:

1. *ir*-strategies;
2. *Ir*-strategies;
3. *IR*-strategies;
4. *iR*-strategies.

How many strategies are there for each type?

1. exponentially many;
2. exponentially many;
3. infinitely many;
4. infinitely many.

Exponentially many wrt the size of the input! $\approx |\mathbf{Act}|^{|\mathbb{A}\mathrm{gt}|\cdot|\mathbf{St}|}$

Assume we are looking for a "good" Ir-strategy wrt some property P. How complex is this task? (Upper bound)

It is in NP, provided $P \in P$!

Assume we are looking for a "good" Ir-strategy wrt some property P. How complex is this task? (Upper bound)

It is in NP, provided $P \in P$!

1. Guess s_A;
2. check whether s_A satisfies P.

Assume we are looking for a "good" Ir-strategy wrt some property P. How complex is this task? (Upper bound)

It is in NP, provided $P \in P$!

1. Guess s_A;
2. check whether s_A satisfies P.

And the case for "good" ir-strategies?

It is also in NP, provided $P \in P$! Why? What about uniformity?

Assume we are looking for a "good" Ir-strategy wrt some property P. How complex is this task? (Upper bound)

It is in NP, provided $P \in P$!

1. Guess s_A;
2. check whether s_A satisfies P.

And the case for "good" ir-strategies?

It is also in NP, provided $P \in P$! Why? What about uniformity?

1. Guess *Ir*-strategy s_A;
2. check whether it is an *ir*-strategy, i.e. for uniformity (St is finite!);
3. check whether s_A satisfies P.

What if P is verifiable in $\mathcal{C}$ for an arbitrary complexity class $\mathcal{C}$?

What if P is verifiable in $\mathcal{C}$ for an arbitrary complexity class $\mathcal{C}$?

Finding *ir*- and *Ir*-strategies is in

What if P is verifiable in $\mathcal{C}$ for an arbitrary complexity class $\mathcal{C}$?

Finding *ir*- and *Ir*-strategies is in $NP^{\mathcal{C}}$.

What if P is verifiable in $\mathcal{C}$ for an arbitrary complexity class $\mathcal{C}$?

Finding *ir*- and *Ir*-strategies is in $NP^{\mathcal{C}}$.

What about perfect recall strategies?

What if P is verifiable in $\mathcal{C}$ for an arbitrary complexity class $\mathcal{C}$?

Finding *ir*- and *Ir*-strategies is in $NP^{\mathcal{C}}$.

What about perfect recall strategies?

There are infinitely many: So there is no general method!

Imperfect Information

Agent's ability to identify a strategy as winning also varies throughout the game in an arbitrary way (agents can learn as well as forget). This suggests that winning strategies cannot be synthesized incrementally.Indeed the fixpoint characterisations do not hold! :

$$\langle\langle A\rangle\rangle\Box\varphi \quad\not\leftrightarrow\quad \varphi\wedge\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\Box\varphi,$$

$$\langle\langle A\rangle\rangle\varphi_1\,\mathcal{U}\,\varphi_2 \quad\not\leftrightarrow\quad \varphi_2\vee\varphi_1\wedge\langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\varphi_1\,\mathcal{U}\,\varphi_2.$$

Imperfect Information

Agent's ability to identify a strategy as winning also varies throughout the game in an arbitrary way (agents can learn as well as forget). This suggests that winning strategies cannot be synthesized incrementally.Indeed the fixpoint characterisations do not hold! :

$$\langle\langle A\rangle\rangle\Box\varphi \quad \not\leftrightarrow \quad \varphi \wedge \langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\Box\varphi,$$
$$\langle\langle A\rangle\rangle\varphi_1\,\mathcal{U}\,\varphi_2 \quad \not\leftrightarrow \quad \varphi_2 \vee \varphi_1 \wedge \langle\langle A\rangle\rangle\bigcirc\langle\langle A\rangle\rangle\varphi_1\,\mathcal{U}\,\varphi_2.$$

How to model check a formula $\mathcal{M}, q \models \langle\langle A\rangle\rangle\gamma$ where γ includes no nested cooperation modalities ?

Imperfect Information

Agent's ability to identify a strategy as winning also varies throughout the game in an arbitrary way (agents can learn as well as forget). This suggests that winning strategies cannot be synthesized incrementally. Indeed the fixpoint characterisations do not hold! :

$$\begin{aligned} \langle\!\langle A \rangle\!\rangle \Box \varphi \quad &\not\leftrightarrow \quad \varphi \wedge \langle\!\langle A \rangle\!\rangle \bigcirc \langle\!\langle A \rangle\!\rangle \Box \varphi, \\ \langle\!\langle A \rangle\!\rangle \varphi_1 \mathcal{U} \varphi_2 \quad &\not\leftrightarrow \quad \varphi_2 \vee \varphi_1 \wedge \langle\!\langle A \rangle\!\rangle \bigcirc \langle\!\langle A \rangle\!\rangle \varphi_1 \mathcal{U} \varphi_2. \end{aligned}$$

How to model check a formula $\mathcal{M}, q \models \langle\!\langle A \rangle\!\rangle \gamma$ where γ includes no nested cooperation modalities ?

Theorem 6.24 ($\mathbf{ATL}_{ir}$)

Model checking $\mathbf{ATL}_{ir}$ *is* Δ_2^P*-complete.*

The lower bound is proven by a reduction of SNSAT_1.

Recall: $\Delta_2^P = P^{NP}$

Proof: Upper Bound

Let $\langle\!\langle A \rangle\!\rangle\gamma$ be given where γ includes no nested cooperation modalities.

1. Guess a strategy s_A of A.

Recall: $\Delta_2^P = P^{NP}$

Proof: Upper Bound

Let $\langle\langle A \rangle\rangle\gamma$ be given where γ includes no nested cooperation modalities.

1. Guess a strategy s_A of A.
2. "Prune" $\mathcal{M}$ to $\mathcal{M}|_{s_A}$; i.e. remove transitions that cannot occur according to s_A.

Recall: $\Delta_2^P = P^{NP}$

Proof: Upper Bound

Let $\langle\langle A\rangle\rangle\gamma$ be given where γ includes no nested cooperation modalities.

1. Guess a strategy s_A of A.
2. "Prune" $\mathcal{M}$ to $\mathcal{M}|_{s_A}$; i.e. remove transitions that cannot occur according to s_A.
3. Remove labels from $\mathcal{M}|_{s_A}$ and interpret it as Kripke structure $\mathcal{M}'|_{s_A}$

Recall: $\Delta_2^P = P^{NP}$

Proof: Upper Bound

Let $\langle\langle A\rangle\rangle\gamma$ be given where γ includes no nested cooperation modalities.

1. Guess a strategy s_A of A.
2. "Prune" $\mathcal{M}$ to $\mathcal{M}|_{s_A}$; i.e. remove transitions that cannot occur according to s_A.
3. Remove labels from $\mathcal{M}|_{s_A}$ and interpret it as Kripke structure $\mathcal{M}'|_{s_A}$
4. Then,

$$\mathcal{M}, q \models \langle\langle A\rangle\rangle\gamma \text{ iff } \mathcal{M}'|_{s_A}, q \models^{\text{CTL}} \mathsf{A}\gamma$$

Recall: $\Delta_2^P = P^{NP}$

Proof: Upper Bound

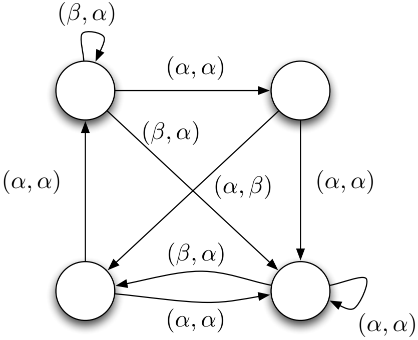
Let $\langle\langle A\rangle\rangle\gamma$ be given where γ includes no nested cooperation modalities.

1. Guess a strategy s_A of A.
2. "Prune" $\mathcal{M}$ to $\mathcal{M}|_{s_A}$; i.e. remove transitions that cannot occur according to s_A.
3. Remove labels from $\mathcal{M}|_{s_A}$ and interpret it as Kripke structure $\mathcal{M}'|_{s_A}$
4. Then,

$$\mathcal{M}, q \models \langle\langle A\rangle\rangle\gamma \text{ iff } \mathcal{M}'|_{s_A}, q \models^{\text{CTL}} \mathsf{A}\gamma$$

The basic idea is to guess a strategy and apply CTL model checking.

ATL and CTL: Pruning



Guess the strategy s_1 in which 1 always plays α .

$\langle\langle 1\rangle\rangle\Diamond\gamma \quad \rightsquigarrow$ **guess** s_1 , check A$\Diamond\gamma$ in the **pruned** model

Model Checking ATL* with memoryless strategies

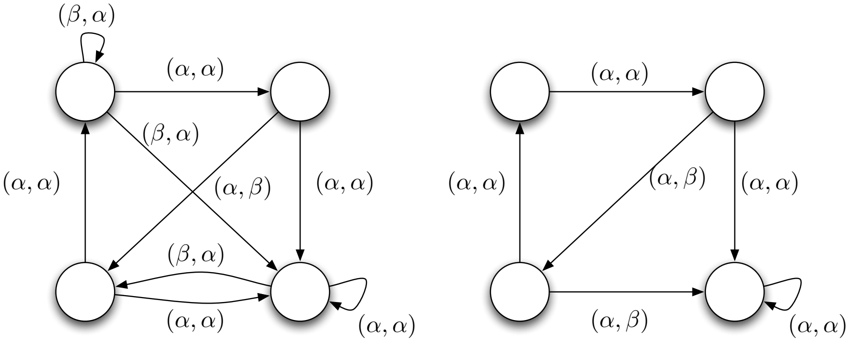
To solve the model checking problem for **ATL*$_{Ir}$** we make use of **CTL*** model checking.

The basic idea for model checking $\langle\langle A\rangle\rangle\psi$ is as follows:

1. Guess a strategy $s_A : St \rightarrow Act^{|A|}$ (in NP).
2. Prune the model; i.e. remove transitions which cannot occur.
3. CTL* model check Aψ in the resulting model.

Pruning the model

We can reduce model checking to model checking **CTL***:



Guess the strategy s_1 in which 1 always plays α .

$\langle\langle 1\rangle\rangle\Box\Diamond\gamma \quad \rightsquigarrow$ guess s_1 , check $\mathsf{A}\Box\Diamond\gamma$ in the pruned mod

s_1: agent 1 plays α in all states.

Theorem 6.25 (ATL*$_{ir}$ and ATL*$_{Ir}$ [Schobbens, 2004])

*Model checking **ATL***$_{ir}$ *and **ATL***$_{Ir}$ is PSPACE-**complete** *in the number of transitions in the model and the length of the formula.*

Proof: Lower Bound

LTL model checking is a special case of $\mathcal{L}_{ATL^*}$ model checking: PSPACE-hard.

Proof: Upper Bound

Let $\langle\!\langle A\rangle\!\rangle\psi$ where ψ is an $\mathcal{L}_{LTL}$-formula.

1. **Guess** an *Ir*-strategy (resp. *ir*-strategy) s_A of A.

Proof: Upper Bound

Let $\langle\langle A\rangle\rangle\psi$ where ψ is an $\mathcal{L}_{LTL}$-formula.

1. Guess an *Ir*-strategy (resp. *ir*-strategy) s_A of A.
2. "Prune" $\mathcal{M}$ to $\mathcal{M}|_{s_A}$; i.e. remove transitions that cannot occur according to s_A.

Proof: Upper Bound

Let $\langle\langle A\rangle\rangle\psi$ where ψ is an $\mathcal{L}_{LTL}$-formula.

1. Guess an *Ir*-strategy (resp. *ir*-strategy) s_A of A.
2. "Prune" $\mathcal{M}$ to $\mathcal{M}|_{s_A}$; i.e. remove transitions that cannot occur according to s_A.
3. Remove transition labels from $\mathcal{M}|_{s_A}$ and interpret it as Kripke structure $\mathcal{M}'|_{s_A}$

Proof: Upper Bound

Let $\langle\langle A\rangle\rangle\psi$ where ψ is an $\mathcal{L}_{LTL}$-formula.

1. **Guess** an *Ir*-strategy (resp. *ir*-strategy) s_A of A.
2. "**Prune**" $\mathcal{M}$ to $\mathcal{M}|_{s_A}$; i.e. remove transitions that cannot occur according to s_A.
3. **Remove transition labels** from $\mathcal{M}|_{s_A}$ and interpret it as **Kripke structure** $\mathcal{M}'|_{s_A}$
4. Then,

$$\mathcal{M}, q \models \langle\langle A\rangle\rangle\gamma \qquad \text{iff} \qquad \mathcal{M}'|_{s_A}, q \models^{\mathsf{CTL}^*} \mathsf{A}\gamma$$

Proof: Upper Bound

Let $\langle\langle A\rangle\rangle\psi$ where ψ is an $\mathcal{L}_{LTL}$-formula.

1. Guess an *Ir*-strategy (resp. *ir*-strategy) s_A of A.
2. "Prune" $\mathcal{M}$ to $\mathcal{M}|_{s_A}$; i.e. remove transitions that cannot occur according to s_A.
3. Remove transition labels from $\mathcal{M}|_{s_A}$ and interpret it as Kripke structure $\mathcal{M}'|_{s_A}$
4. Then,

$$\mathcal{M}, q \models \langle\langle A\rangle\rangle\gamma \qquad \text{iff} \qquad \mathcal{M}'|_{s_A}, q \models^{\text{CTL}^*} \mathsf{A}\gamma$$

This procedure can be performed in NP^{PSPACE}, which renders the complexity of the whole language to be in $P^{NP^{PSPACE}} = PSPACE$.

Imperfect Information and Perfect Recall

Conjecture 6.26 (ATL_{iR})

Model checking ATL_{iR} is undecidable.

Recently, a proof has been proposed by Dima and Tiplea (June 2010).

Imperfect Information and Perfect Recall

Conjecture 6.26 (ATL_{iR})

Model checking ATL_{iR} is undecidable.

Recently, a proof has been proposed by Dima and Tiplea (June 2010).

Conjecture 6.27 (ATL^*_{iR})

Model checking ATL^*_{iR} is undecidable.

Conjecture 6.28 (ATL^+_{iR})

Model checking ATL^+_{iR} is undecidable.

6.7 Summary of Complexity Results

- Nice results: model checking **CTL** and **ATL** is tractable.

- Nice results: model checking **CTL** and **ATL** is tractable.
- But: the result is relative to the size of the model and the formula

- Nice results: model checking **CTL** and **ATL** is tractable.
- But: the result is relative to the size of the model and the formula
- Well known catch (CTL): size of models is exponential wrt a higher-level description

- Nice results: model checking **CTL** and **ATL** is tractable.
- But: the result is relative to the size of the model and the formula
- Well known catch (CTL): size of models is exponential wrt a higher-level description
- Another problem: transitions are labelled
- So: the number of transitions can be exponential in the number of agents.

	Ir	*IR*	*ir*	*iR*
$\mathcal{L}_{ATL}$	P	P	Δ_2^P	Undecidable†
$\mathcal{L}_{ATL^+}$	Δ_3^P	$PSPACE$	Δ_3^P	Undecidable†
$\mathcal{L}_{ATL^*}$	$PSPACE$	$2EXPTIME$	$PSPACE$	Undecidable†

Figure 6 : † These problems are believed to be undecidable.

6.8 Model Checking Agent Language Models

- An operational semantics describes the configurations the system/program can be in and gives rules for transforming between these configurations.
- It provides an abstract view of the potential execution (i.e. sequence of configuration changes) of any program.
- Given a specific program, we can work through the program and, by examining the operational semantics, can build a model of all the potential configurations that the particular program can generate.
- This model can then be checked against a logical requirement.

Promela and Spin.

- In [Wooldridge et al., 2006] simple agent programs were verified via a translation to SPIN.
- In [Bordini et al., 2003], AgentSpeak programs were translated to the PROMELA language and then the SPIN model-checker is used to verify its properties.
- Note that subsequent work translated to JAVA and used JPF.

GOAL.

- In [Jongmans et al., 2010], the operational semantics of the GOAL agent programming language is used to describe all the possible executions of a specific GOAL program.
- The on-the-fly algorithmic verification techniques are used to explore all these potential executions.
- This provides quite an efficient verification mechanism for GOAL programs.

Rewriting

- Given that the formal semantics of an agent language is often given in terms of rewrite rules (especially if it is an operational semantics) then an alternative way to tackle verification would be to base it on some underlying *rewrite* system.
- This clearly has some link to the use of an underlying logic programming system as well as a link to the *model-checking* approaches based on operational semantics that we consider here.

MAUDE System

- The predominant rewrite system is MAUDE which provides an efficient and flexible rewriting basis [Clavel et al., 2003].
- Indeed, the operational semantics of several agent languages have been translated to MAUDE input [van Riemsdijk et al., 2006, Farwer and Dennis, 2007].

7. Algorithmic Verification of Programs

As we have seen, it is certainly possible to verify an agent program by building a model of its execution and then algorithmically verifying this model with respect to some requirement.

As we have seen, it is certainly possible to verify an agent program by building a model of its execution and then *algorithmically verifying* this model with respect to some requirement.

- Yet, a very appealing approach to verification is to verify the actual program rather than a model of it.

As we have seen, it is certainly possible to verify an agent program by building a model of its execution and then algorithmically verifying this model with respect to some requirement.

- Yet, a very appealing approach to verification is to verify the actual program rather than a model of it.

  → But, is this possible for agent programs?

As we have seen, it is certainly possible to verify an agent program by building a model of its execution and then *algorithmically verifying* this model with respect to some requirement.

- Yet, a very appealing approach to verification is to verify the actual program rather than a model of it.

  → But, is this possible for agent programs?

  → If so, how does this work?

As we have seen, it is certainly possible to verify an agent program by building a model of its execution and then *algorithmically verifying* this model with respect to some requirement.

- Yet, a very appealing approach to verification is to verify the actual program rather than a model of it.

  → But, is this possible for agent programs?

  → If so, how does this work?

  → And will it work for many different agent programs?

General Problem

So, we wish to verify an agent program by exploring its executions directly, rather than building a model (typically a finite-state automaton) and checking that.

General Problem

So, we wish to verify an agent program by exploring its executions directly, rather than building a model (typically a finite-state automaton) and checking that.

Once we have an operational semantics then, in principle, we should be able to achieve such program checking.

However, this is far from simple to implement!

General Problem

So, we wish to verify an agent program by exploring its executions directly, rather than building a model (typically a finite-state automaton) and checking that.

Once we have an operational semantics then, in principle, we should be able to achieve such program checking.

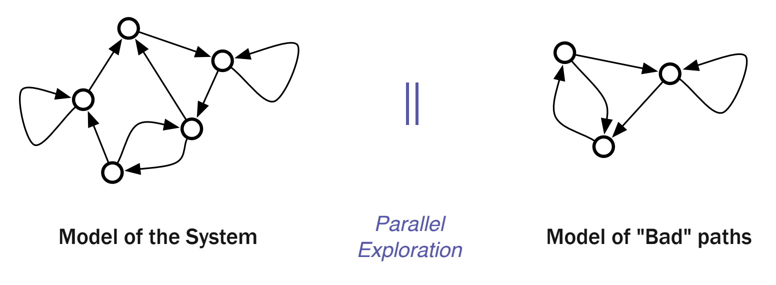
However, this is far from simple to implement!

Consequently, the agent program verification system we describe here takes advantage of sophisticated program verification systems for non-agent programs.

Specifically, it extends the JAVA PATHFINDER system for checking JAVA programs.

Checking Agent Programs

Recall how program verification works, based on the "on the fly" model-checking seen earlier.



Checking Agent Programs (cont.)

In the particular case of **JAVA PATHFINDER**, a modified JAVA virtual machine has been developed which allows both the parallel checking of properties and the backtracking of system executions.

The **MCAPL** framework [Dennis et al., 2012] comprises the **AIL** semantic toolkit, the **MCAPL** interface, and the **AJPF** model-checker.

7.1 AIL Semantic Toolkit

Operational Semantics: Creation

What do we do when we write an operational semantics for our favourite agent programming language?

- We decide on the essential configurations in the system, for example in a BDI-like language we might record the current beliefs, current intentions, suspended intentions, applicable plans, etc.

Operational Semantics: Creation

What do we do when we write an operational semantics for our favourite agent programming language?

- We decide on the essential configurations in the system, for example in a BDI-like language we might record the current beliefs, current intentions, suspended intentions, applicable plans, etc.
- Then we define allowable transitions between these configurations, corresponding to how the language works. A basic transition could be

$$\frac{\texttt{add\_belief}(\texttt{b})}{\langle \mathit{Beliefs}, \mathit{Intentions}, \ldots \rangle \longrightarrow \langle \mathit{Beliefs} \cup \{\texttt{b}\}, \mathit{Intentions}, \ldots \rangle}$$

where the set of beliefs is updated with the new belief, ‘b’, to generate a new configuration.

Operational Semantics: Use

We must generate many, usually more complex, transition rules in order to provide the operational semantics of our language.

Operational Semantics: Use

We must generate many, usually more complex, transition rules in order to provide the operational semantics of our language.

Then there are two particular ways in which we might use the operational semantics.

1. To provide an implementation
   Since such an operational semantics essentially describes a language interpreter then the language can be implemented just by encoding the operational semantic rules.

Operational Semantics: Use

We must generate many, usually more complex, transition rules in order to provide the operational semantics of our language.

Then there are two particular ways in which we might use the operational semantics.

1. To provide an implementation
   Since such an operational semantics essentially describes a language interpreter then the language can be implemented just by encoding the operational semantic rules.
2. As part of verification
   As we saw earlier, we might use the operational semantics as the basis for a model-checker.

Support

However, every time we tackle a new agent programming language we must go through this process again.

Support

However, every time we tackle a new agent programming language we must go through this process again.

A particularly awkward aspect is defining how the model-checking procedure accesses/evaluates beliefs, intentions, etc., within the agent execution.

Support

However, every time we tackle a new agent programming language we must go through this process again.

A particularly awkward aspect is defining how the model-checking procedure accesses/evaluates beliefs, intentions, etc., within the agent execution.

Since many agent languages are actually very similar, then there is surely scope for some re-use of the above aspects.

Support

However, every time we tackle a new agent programming language we must go through this process again.

A particularly awkward aspect is defining how the model-checking procedure accesses/evaluates beliefs, intentions, etc., within the agent execution.

Since many agent languages are actually very similar, then there is surely scope for some re-use of the above aspects.

Agent Infrastructure Layer (AIL)

The AIL is essentially a toolkit that aids the development of all the above aspects for BDI-like, JAVA-based, agent programming languages [Dennis et al., 2012].

AIL Semantic Toolkit (1)

When you have an idea for a new agent programming language, you can access the AIL toolkit to build an operational semantics for the language.

AIL Semantic Toolkit (1)

When you have an idea for a new agent programming language, you can access the AIL toolkit to build an operational semantics for the language.

Once such a semantics is built, the AIL toolkit naturally provides a JAVA implementation (since the semantic elements are all objects/classes within JAVA) and also provides ways in which a special model-checker (called AJPF) can access the components of the semantics.

AIL Semantic Toolkit (1)

When you have an idea for a new agent programming language, you can access the AIL toolkit to build an operational semantics for the language.

Once such a semantics is built, the AIL toolkit naturally provides a JAVA implementation (since the semantic elements are all objects/classes within JAVA) and also provides ways in which a special model-checker (called AJPF) can access the components of the semantics.

Although AIL provides a wide range of "ready made" semantic components and rules corresponding to typical BDI language features, the developer still has the capability to write new semantic rules (so long as they respect the interfaces and interactions required).

AIL Semantic Toolkit (2)

When we run a program in our new agent programming language

→ run it in an AIL-based interpreter which utilizes special AIL data structures to store the agent's internal configuration (typically, beliefs, intentions, plans, etc).

AIL Semantic Toolkit (2)

When we run a program in our new agent programming language

$\rightarrow$ run it in an AIL-based interpreter which utilizes special AIL data structures to store the agent's internal configuration (typically, beliefs, intentions, plans, etc).

AIL also provides support for describing the agent's *reasoning cycle* within the operational semantics.

$\rightarrow$ defines how the agent's practical reasoning progresses, depending on its current internal configuration.

AIL Semantic Toolkit (2)

When we run a program in our new agent programming language

→ run it in an AIL-based interpreter which utilizes special AIL data structures to store the agent's internal configuration (typically, beliefs, intentions, plans, etc).

AIL also provides support for describing the agent's reasoning cycle within the operational semantics.

→ defines how the agent's practical reasoning progresses, depending on its current internal configuration.

→ AIL provides support for constructing reasoning cycles along with a number of rules that typically appear in the operational semantics of agent programming languages.

7.2 Multiple Semantics

Heterogeneous Multi-Agent Systems

By using a common semantic base, we are able to define the formal semantics for many agent programming languages.

Heterogeneous Multi-Agent Systems

By using a common semantic base, we are able to define the formal semantics for many agent programming languages.

For example, in [Dennis and Fisher, 2008], the AIL is used to provide semantics for

- **GOAL** [de Boer et al., 2007],
- **SAAPL** [Winikoff, 2007], and
- **Gwendolen** [Dennis and Farwer, 2008].

Heterogeneous Multi-Agent Systems

By using a common semantic base, we are able to define the formal semantics for many agent programming languages.

For example, in [Dennis and Fisher, 2008], the AIL is used to provide semantics for

- **GOAL** [de Boer et al., 2007],
- **SAAPL** [Winikoff, 2007], and
- **Gwendolen** [Dennis and Farwer, 2008].

Not only can such agents be developed and verified separately, but the fact that the semantics for all three are built on a common basis means that heterogeneous multi-agent systems can be verified.

Heterogeneous Multi-Agent Systems

By using a common semantic base, we are able to define the formal semantics for many agent programming languages.

For example, in [Dennis and Fisher, 2008], the AIL is used to provide semantics for

- GOAL [de Boer et al., 2007],
- SAAPL [Winikoff, 2007], and
- Gwendolen [Dennis and Farwer, 2008].

Not only can such agents be developed and verified separately, but the fact that the semantics for all three are built on a common basis means that heterogeneous multi-agent systems can be verified.

A system comprising GOAL, SAAPL, and Gwendolen agents communicating together can be verified.

7.3 AJPF Model Checking

AJPF Internals

Since we have not yet explained how agents built using AIL semantic definitions are verified, we will turn to this next.

The AIL toolkit collects together **Java** classes that can be verified through **AJPF**, an extended version of the **Java Pathfinder** system.

When a language interpreter that has been developed using AIL is executed, then the interpreter communicates with the AJPF model checker.

In particular, the interpreter will notify AJPF each time a new state is reached that is relevant to the verification, while AJPF can, through the AIL structures, access all the internal details of the agent's execution.

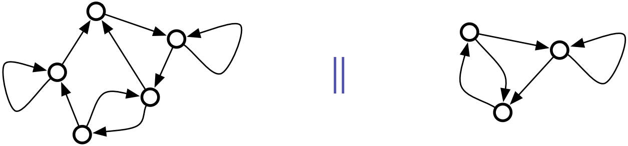
AJPF Exploration

Since AJPF is based on the **JPF** system it exhaustively explores the execution of the agent, backtracking if necessary through the underlying virtual machine.

AJPF Exploration

Since AJPF is based on the **JPF** system it exhaustively explores the execution of the agent, backtracking if necessary through the underlying virtual machine.

In parallel a **Java** listener object 'watches' for important steps through the execution (where 'important' is defined within the AIL semantic definitions) and tries to match its internal automaton to the execution it is seeing.

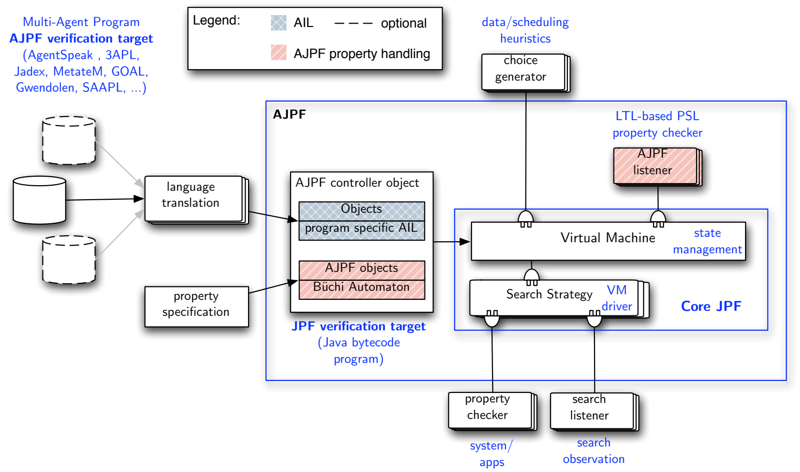


Java Interpretation of the Agent Program

Parallel Exploration

Java listener object encapsulating a model of the "Bad" possible paths

Schematic Diagram of the AJPF Architecture



Speed Issues

Program model checking is significantly slower than standard model-checking applied to models of the program execution.

Thus, verifications in AJPF take minutes and hours, rather than seconds with tools such as SPIN or NUSMV.

In spite of this, agent program verification is clearly very useful.

Future

Not only does AIL make it easier to develop agent programming language interpreters, but it also provides easy access to sophisticated model-checking capabilities.

Importantly, the program that is model-checked is the program that is run.

This allows the MCAPL (i.e. AIL+AJPF) framework to be used in increasingly practical scenarios.

For example, in [Webster et al., 2011] this approach is used to verify key parts of the control for an unmanned air vehicle.

8. Appendix: Automata Theory

8 Appendix: Automata Theory

- Büchi Automata
- Generalized Büchi Automata
- Tree automata
- Emptiness Checking
- Determinization

8.1 Büchi Automata

Büchi Automata

- We would like to use finite automata to solve the model checking problem.
- Finite automata (on finite words) accept only finite words but paths are infinite.
- We need to extend the model to finite automata that accept infinite words.

Büchi Automata

- We would like to use finite automata to solve the model checking problem.
- Finite automata (on finite words) accept only finite words but paths are infinite.
- We need to extend the model to finite automata that accept infinite words.

How can we accept infinite words?

Definition 8.1 (ω-automaton)

An ω-automaton is a tuple

$$A = (Q, \Sigma, \Delta, q_I, C)$$

where

1. Q is a finite set of states;

Definition 8.1 (ω-automaton)

An ω-automaton is a tuple

$$A = (Q, \Sigma, \Delta, q_I, C)$$

where

1. Q is a finite set of states;
2. Σ is a finite alphabet;

Definition 8.1 (ω-automaton)

An ω-**automaton** is a tuple

$$A = (Q, \Sigma, \Delta, q_I, C)$$

where

1. Q is a finite set of **states**;
2. Σ is a **finite alphabet**;
3. $\Delta \subseteq Q \times \Sigma \times Q$ a **transition relation** ;

Definition 8.1 (ω-automaton)

An ω-automaton is a tuple

$$A = (Q, \Sigma, \Delta, q_I, C)$$

where

1. Q is a finite set of states;
2. Σ is a finite alphabet;
3. $\Delta \subseteq Q \times \Sigma \times Q$ a transition relation ;
4. q_I is the initial state; and

Definition 8.1 (ω-automaton)

An ω-automaton is a tuple

$$A = (Q, \Sigma, \Delta, q_I, C)$$

where

1. Q is a finite set of states;
2. Σ is a finite alphabet;
3. $\Delta \subseteq Q \times \Sigma \times Q$ a transition relation ;
4. q_I is the initial state; and
5. C an acceptance component (which is specialised in the following).

Definition 8.1 (ω-automaton)

An ω-automaton is a tuple

$$A = (Q, \Sigma, \Delta, q_I, C)$$

where

1. Q is a finite set of states;
2. Σ is a finite alphabet;
3. $\Delta \subseteq Q \times \Sigma \times Q$ a transition relation ;
4. q_I is the initial state; and
5. C an acceptance component (which is specialised in the following).

The crucial point is the acceptance component!

Definition 8.2 (Run)

A run $\rho = \rho(0)\rho(1)\cdots \in Q^\omega$ of A on a word $w = w_1 w_2 \cdots \in \Sigma^\omega$ is an infinite sequence of states of A such that:

1. $\rho(0) = q_I$
2. $\rho(i) \in \Delta(\rho(i-1), w_i)$ for $i \geq 1$.

Definition 8.2 (Run)

A **run** $\rho = \rho(0)\rho(1)\cdots \in Q^\omega$ **of** A **on a word** $w = w_1 w_2 \cdots \in \Sigma^\omega$ is an infinite sequence of states of A such that:

1. $\rho(0) = q_I$
2. $\rho(i) \in \Delta(\rho(i-1), w_i)$ for $i \geq 1$.

How could we accept the following language?

$L = \{w \in \{a,b\}^\omega \mid w$ contains infinitely many a and only finitely many $b\}$.

Is it sufficient to reach a final state once?

We define $Inf(\rho)$ as the set of all states that occur infinitely often on ρ; that is,

$$Inf(\rho) = \{q \in St \mid \forall i \exists j (j > i \land \rho(j) = q)\}$$

We define $Inf(\rho)$ as the set of all states that occur infinitely often on ρ; that is,

$$Inf(\rho) = \{q \in St \mid \forall i \exists j (j > i \wedge \rho(j) = q)\}$$

Definition 8.3 (Büchi automaton)

A Büchi automaton is an ω-automaton

$$A = (Q, \Sigma, \Delta, q_I, F)$$

where $F \subseteq Q$ with the following acceptance condition: A accepts $w \in \Sigma^\omega$ if, and only if, there is a run ρ of A such that

$$Inf(\rho) \cap F \neq \emptyset.$$

This automaton accepts all words s.t.some state from F is visited infinitely often on a corresponding run.

Definition 8.4 (Acceptable language)

The language accepted by A, $L(A)$, consists of all words accepted by A. That is,

$$L(A) = \{w \in \Sigma^\omega \mid A \text{ accepts } w\}.$$

A language is said to be (Büchi) acceptable if there is a Büchi automaton that accepts it.

Definition 8.4 (Acceptable language)

The language accepted by A, $L(A)$, consists of all words accepted by A. That is,

$$L(A) = \{w \in \Sigma^\omega \mid A \text{ accepts } w\}.$$

A language is said to be (Büchi) acceptable if there is a Büchi automaton that accepts it.

Remark 8.5 (Other automata types)

Other acceptance conditions yield different automata types: Rabin automata, Muller automata.

Example 8.6

Is there a Büchi Automaton that accepts the following language L over $\Sigma = \{a, b, c\}$?

$$L = \{w \in \Sigma^\omega \mid w \text{ contains infinitely many } a \text{ or } b \text{ and only finitely many } c\}$$

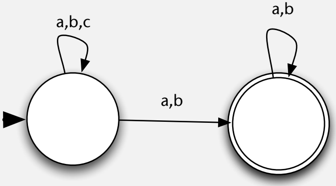
$\rightsquigarrow$ blackboard

Example 8.6

Is there a Büchi Automaton that accepts the following language L over $\Sigma = \{a, b, c\}$?

$$L = \{w \in \Sigma^\omega \mid w \text{ contains infinitely many } a \text{ or } b \text{ and only finitely many } c\}$$

$\rightsquigarrow$ blackboard



Example 8.7

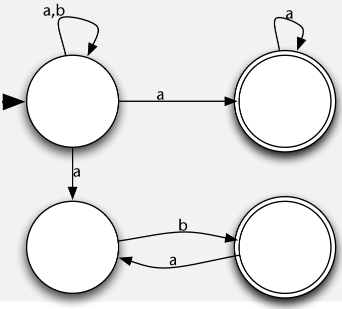
Is there a Büchi Automaton that accepts the following language L over $\Sigma = \{a, b\}$?

$$L = \{w \in \Sigma^\omega \mid w \text{ ends with } a^\omega \text{ or } (ab)^\omega\}$$

Example 8.7

Is there a Büchi Automaton that accepts the following language L over $\Sigma = \{a, b\}$?

$$L = \{w \in \Sigma^\omega \mid w \text{ ends with } a^\omega \text{ or } (ab)^\omega\}$$



Proposition 8.8 (Closure propeties)

1. *Büchi acceptable languages are closed under union, intersection, and negation.*

Proposition 8.8 (Closure propeties)

1. *Büchi acceptable languages are closed under union, intersection, and negation.*
2. *If A is a regular language with $\epsilon \notin A$, then, A^ω is Büchi acceptable.*

Proposition 8.8 (Closure propeties)

1. *Büchi acceptable languages are closed under union, intersection, and negation.*

2. *If A is a regular language with $\epsilon \notin A$, then, A^ω is Büchi acceptable.*

3. *If A is a regular language and B is Büchi recognizable, then AB is Büchi acceptable.*

Proof sketch

1. Union:

   Intersection:

   Complement:

2. A^{ω}:

3. AB:

Proof sketch

1. **Union**: Nondeterministically guess which automata should be executed. $\rightsquigarrow$ Exercise
   Intersection:

   Complement:

2. A^ω:

3. AB:

Proof sketch

1. **Union**: Nondeterministically guess which automata should be executed. $\rightsquigarrow$ Exercise
   Intersection: Product automaton yields a generalised Büchi automaton. The acceptance set is given by $\{F_1 \times S_2, S_1 \times F_2\}$. $\rightsquigarrow$ Exercise
   Complement:
2. A^ω:
3. AB:

Proof sketch

1. **Union**: Nondeterministically guess which automata should be executed. ⇝ Exercise
   Intersection: Product automaton yields a generalised Büchi automaton. The acceptance set is given by $\{F_1 \times S_2, S_1 \times F_2\}$. ⇝ Exercise
   Complement: This part is non-trivial and cannot be done in the scope of this lecture.
2. A^ω:
3. AB:

Proof sketch

1. Union: Nondeterministically guess which automata should be executed. $\rightsquigarrow$ Exercise
   Intersection: Product automaton yields a generalised Büchi automaton. The acceptance set is given by $\{F_1 \times S_2, S_1 \times F_2\}$. $\rightsquigarrow$ Exercise
   Complement: This part is non-trivial and cannot be done in the scope of this lecture.
2. A^ω: Connect transitions to final states also with the initial state $\rightsquigarrow$ Exercise
3. AB:

Proof sketch

1. **Union**: Nondeterministically guess which automata should be executed. $\leadsto$ Exercise
   Intersection: Product automaton yields a generalised Büchi automaton. The acceptance set is given by $\{F_1 \times S_2, S_1 \times F_2\}$. $\leadsto$ Exercise
   Complement: This part is non-trivial and cannot be done in the scope of this lecture.
2. A^ω: Connect transitions to final states also with the initial state $\leadsto$ Exercise
3. AB: Connect transitions to final states of the finite automaton with the initial state of the Büchi automaton. $\leadsto$ Exercise

Theorem 8.9 (Characterization Theorem)

A language L is Büchi acceptable if, and only if, there are finitely many regular languages $U_1, \ldots, U_n$ and $V_1, \ldots, V_n$ such that

$$L = \bigcup_{i=1,\ldots,n} U_i(V_i)^\omega$$

Theorem 8.9 (Characterization Theorem)

A language L is Büchi acceptable if, and only if, there are finitely many regular languages $U_1, \ldots, U_n$ and $V_1, \ldots, V_n$ such that

$$L = \bigcup_{i=1,\ldots,n} U_i(V_i)^\omega$$

This shows that any language $L \neq \emptyset$ acceptable by a Büchi automaton contains an ultimately periodic word.

Theorem 8.9 (Characterization Theorem)

A language L is Büchi acceptable if, and only if, there are finitely many regular languages $U_1, \ldots, U_n$ and $V_1, \ldots, V_n$ such that

$$L = \bigcup_{i=1,\ldots,n} U_i(V_i)^\omega$$

This shows that any language $L \neq \emptyset$ acceptable by a Büchi automaton contains an ultimately periodic word.

Example 8.10

For the language $L = \{w \in \Sigma^\omega \mid w \text{ ends with } a^\omega \text{ or } (ab)^\omega\}$ from Example 8.7 we have that $L =$.

Theorem 8.9 (Characterization Theorem)

A language L is Büchi acceptable if, and only if, there are finitely many regular languages $U_1, \dots, U_n$ and $V_1, \dots, V_n$ such that

$$L = \bigcup_{i=1,\dots,n} U_i (V_i)^\omega$$

This shows that any language $L \neq \emptyset$ acceptable by a Büchi automaton contains an ultimately periodic word.

Example 8.10

For the language $L = \{w \in \Sigma^\omega \mid w \text{ ends with } a^\omega \text{ or } (ab)^\omega\}$ from Example 8.7 we have that $L = \Sigma^*\{a\}^\omega \cup \Sigma^*\{ab\}^\omega$.

Proof of Theorem 8.9

"$\Rightarrow$": Let W(q,q') $= \{w \in \Sigma^* \mid q \rightarrow^w q'\}$.

Proof of Theorem 8.9

"$\Rightarrow$": Let W(q,q') $= \{w \in \Sigma^* \mid q \to^w q'\}$. Each language $W(q, q')$ is regular.

Proof of Theorem 8.9

"$\Rightarrow$": Let W(q,q') $= \{w \in \Sigma^* \mid q \to^w q'\}$. Each language $W(q,q')$ is **regular**. Then,

$$L(A) = \bigcup_{q \in Q_f} W(q_I, q)(W(q,q))^\omega.$$

Proof of Theorem 8.9

"$\Rightarrow$": Let W(q,q') $= \{w \in \Sigma^* \mid q \to^w q'\}$. Each language $W(q,q')$ is **regular**. Then,

$$L(A) = \bigcup_{q \in Q_f} W(q_I, q)(W(q,q))^\omega.$$

"$\Leftarrow$": Let $L = \bigcup_{i=1,\ldots,n} U_i(V_i)^\omega$ where each U_i, V_i is regular.

Proof of Theorem 8.9

"$\Rightarrow$": Let W(q,q') $= \{w \in \Sigma^* \mid q \rightarrow^w q'\}$. Each language $W(q,q')$ is regular. Then,

$$L(A) = \bigcup_{q \in Q_f} W(q_I, q)(W(q,q))^\omega.$$

"$\Leftarrow$": Let $L = \bigcup_{i=1,\ldots,n} U_i(V_i)^\omega$ where each U_i, V_i is regular. By Proposition 8.8 we have that $(V_i)^\omega$ and $U_i(V_i)^\omega$ are Büchi recognizable. Thus also their finite union.

8.2 Generalized Büchi Automata

Definition 8.11 (Generalised Büchi automaton)

A *generalised Büchi automaton* is an ω-automaton

$$A = (Q, \Sigma, \Delta, q_I, F)$$

where $F \subseteq 2^Q$ with the following *acceptance condition*: A *accepts* $w \in \Sigma^\omega$ if, and only if, there is a run ρ of A such that for each $F_i \in F$

$$Inf(\rho) \cap F_i \neq \emptyset.$$

Thus, such an automaton accepts all words such that some state from each F_i is visited infinitely often on a corresponding run.

We will use generalised Büchi automata for model checking **LTL**. How is the relation between Büchi and generalised Büchi automata?

We will use generalised Büchi automata for model checking **LTL**. How is the relation between Büchi and generalised Büchi automata?

Proposition 8.12 (Generalised Büchi $\rightsquigarrow$ Büchi)

For each generalised Büchi automaton one can construct an equivalent Büchi automaton.

We will use generalised Büchi automata for model checking **LTL**. How is the relation between Büchi and generalised Büchi automata?

Proposition 8.12 (Generalised Büchi $\rightsquigarrow$ Büchi)

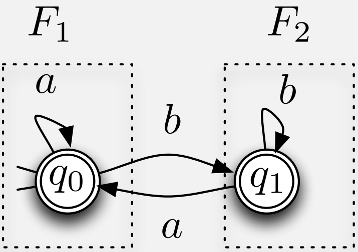
For each generalised Büchi automaton one can construct an equivalent Büchi automaton.

Proof.

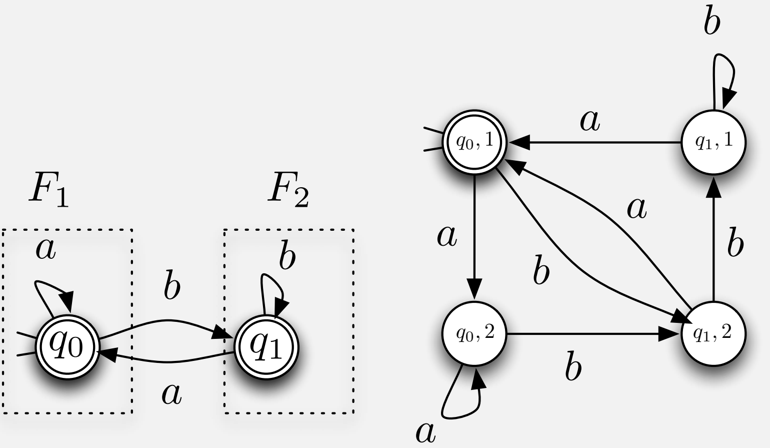
Idea: Consider state-tuples: $S \times \{1, \dots, k\}$. If the GBA moves to the next acceptance set a counter is incremented (modulo k). Then, a run visits states from each F_i infinitely often iff states from $F_1 \times \{1\}$ appear infinitely often. $\square$

We first consider an example:

Example 8.13



Example 8.13



Proof ctd.

Let $A = (\Sigma, S, \Delta, S_0, \{F_1, \ldots, F_n\})$ be a generalised Büchi automaton. We construct the Büchi Automaton $A' = (\Sigma, S', \Delta', S'_0, F')$:

- $S' = S \times \{1, \ldots, n\}$;

Proof ctd.

Let $A = (\Sigma, S, \Delta, S_0, \{F_1, \ldots, F_n\})$ be a generalised Büchi automaton. We construct the Büchi Automaton $A' = (\Sigma, S', \Delta', S'_0, F')$:

- $S' = S \times \{1, \ldots, n\}$;
- $S'_0 = S_0 \times \{1\}$;

Proof ctd.

Let $A = (\Sigma, S, \Delta, S_0, \{F_1, \ldots, F_n\})$ be a generalised Büchi automaton. We construct the Büchi Automaton $A' = (\Sigma, S', \Delta', S'_0, F')$:

- $S' = S \times \{1, \ldots, n\}$;
- $S'_0 = S_0 \times \{1\}$;
- $((s, j), a, (t, i)) \in \Delta'$ iff

Proof ctd.

Let $A = (\Sigma, S, \Delta, S_0, \{F_1, \ldots, F_n\})$ be a generalised Büchi automaton. We construct the Büchi Automaton $A' = (\Sigma, S', \Delta', S'_0, F')$:

- $S' = S \times \{1, \ldots, n\}$;
- $S'_0 = S_0 \times \{1\}$;
- $((s, j), a, (t, i)) \in \Delta'$ iff

$$(s, a, t) \in \Delta \text{ and } \begin{cases} i = j & \text{, if } s \notin F_j; \\ i = (j+1) \mod k & \text{, if } s \in F_j; \end{cases}$$

Proof ctd.

Let $A = (\Sigma, S, \Delta, S_0, \{F_1, \ldots, F_n\})$ be a generalised Büchi automaton. We construct the Büchi Automaton $A' = (\Sigma, S', \Delta', S'_0, F')$:

- $S' = S \times \{1, \ldots, n\}$;
- $S'_0 = S_0 \times \{1\}$;
- $((s, j), a, (t, i)) \in \Delta'$ iff

$$(s, a, t) \in \Delta \text{ and } \begin{cases} i = j & \text{, if } s \notin F_j; \\ i = (j+1) \mod k & \text{, if } s \in F_j; \end{cases}$$

- $F' = F_1 \times \{1\}$.

Proof ctd.

It remains to prove that both automata accept the same languages. We present the main ideas.
“$\Rightarrow$“: Let A be a GBA that accepts the word w.

Proof ctd.

It remains to prove that both automata accept the same languages. We present the main ideas.
“$\Rightarrow$“: Let A be a GBA that accepts the word w. Then, there is a run ρ such that states from each F_i, $i = 1, \ldots, k$, occur infinitely often on ρ.

Proof ctd.

It remains to prove that both automata accept the same languages. We present the main ideas.

"$\Rightarrow$": Let A be a GBA that accepts the word w. Then, there is a run ρ such that states from each F_i, $i = 1, \ldots, k$, occur infinitely often on ρ. That is, there is an infinite subsequence $(q_1 \ldots q_k)^\omega$ of ρ such that $q_i \in F_i$.

Proof ctd.

It remains to prove that both automata accept the same languages. We present the main ideas.

"⇒": Let A be a GBA that accepts the word w. Then, there is a run ρ such that states from each F_i, $i = 1, \ldots, k$, occur infinitely often on ρ. That is, there is an infinite subsequence $(q_1 \ldots q_k)^\omega$ of ρ such that $q_i \in F_i$. Hence, the state $(q_1, 1)$ is visited infinitely often in the automaton A'.

Proof ctd.

It remains to prove that both automata accept the same languages. We present the main ideas.

"$\Rightarrow$": Let A be a GBA that accepts the word w. Then, there is a run ρ such that states from each F_i, $i = 1, \ldots, k$, occur infinitely often on ρ. That is, there is an infinite subsequence $(q_1 \ldots q_k)^\omega$ of ρ such that $q_i \in F_i$. Hence, the state $(q_1, 1)$ is visited infinitely often in the automaton A'.

"$\Leftarrow$": Let A' accept the word w. Then, some state $(q_1, 1)$ with $q_1 \in F_1$ is visited infinitely often.

Proof ctd.

It remains to prove that both automata accept the same languages. We present the main ideas.
"$\Rightarrow$": Let A be a GBA that accepts the word w. Then, there is a run ρ such that states from each F_i, $i = 1, \ldots, k$, occur infinitely often on ρ. That is, there is an infinite subsequence $(q_1 \ldots q_k)^\omega$ of ρ such that $q_i \in F_i$. Hence, the state $(q_1, 1)$ is visited infinitely often in the automaton A'.

"$\Leftarrow$": Let A' accept the word w. Then, some state $(q_1, 1)$ with $q_1 \in F_1$ is visited infinitely often. After it has been visited once the automaton is in a state $(q, 2)$

Proof ctd.

It remains to prove that both automata accept the same languages. We present the main ideas.
"$\Rightarrow$": Let A be a GBA that accepts the word w. Then, there is a run ρ such that states from each F_i, $i = 1, \ldots, k$, occur infinitely often on ρ. That is, there is an infinite subsequence $(q_1 \ldots q_k)^\omega$ of ρ such that $q_i \in F_i$. Hence, the state $(q_1, 1)$ is visited infinitely often in the automaton A'.

"$\Leftarrow$": Let A' accept the word w. Then, some state $(q_1, 1)$ with $q_1 \in F_1$ is visited infinitely often. After it has been visited once the automaton is in a state $(q, 2)$ and can only return to $(q', 1)$ if some state $q \in F_2$ is visited,

Proof ctd.

It remains to prove that both automata accept the same languages. We present the main ideas.
"$\Rightarrow$": Let A be a GBA that accepts the word w. Then, there is a run ρ such that states from each F_i, $i = 1, \dots, k$, occur infinitely often on ρ. That is, there is an infinite subsequence $(q_1 \dots q_k)^\omega$ of ρ such that $q_i \in F_i$. Hence, the state $(q_1, 1)$ is visited infinitely often in the automaton A'.

"$\Leftarrow$": Let A' accept the word w. Then, some state $(q_1, 1)$ with $q_1 \in F_1$ is visited infinitely often. After it has been visited once the automaton is in a state $(q, 2)$ and can only return to $(q', 1)$ if some state $q \in F_2$ is visited, some from F_3 and so on is visited.

8.3 Tree automata

As before let Σ be a finite alphabet and k a natural number. A **k-ary Σ-tree $t = (dom_t, L)$** is a tree with maximal branching k and in which each node is labelled by an element from Σ. That is

$$L : dom_t \to \Sigma$$

where $dom_t \subseteq \{0, \ldots, k-1\}^*$ denotes the **domain** of the tree. It is required that dom_t is closed under prefixes, i.e.

$$wx \in dom_t \to \forall y(0 \leq y < x \to wy \in dom_t).$$

A k-ary ω-tree automaton over the alphabet Σ is an automaton that accepts infinite k-ary Σ-trees.

Definition 8.14 (k-ary ω-tree automaton)

A k-ary ω-tree automaton over the alphabet Σ is given by a tuple

$$A = (St, q_I, \Delta, C)$$

where

- St is a set of states,
- $q_I \in St$ the initial state,
- $\Delta : St \times \Sigma \times \{1, \ldots, k\} \to 2^{\bigcup_{i=1\ldots k} St^i}$ with $\Delta(q, a, i) \subseteq St^i$ a transition relation, and
- C an acceptance component (which is specified in the following).

Definition 8.15 (Run, path, successful, accepting)

A **run** of a k-ary ω-tree automaton A on an infinite k-ary Σ-tree $t = (dom_t, L_t)$ is an infinite k-ary St-tree $r = (dom_r, L_r)$ such that

1. $dom_r = dom_t$,
2. $L_r(\emptyset) = q_I$ and
3. $\forall w \in dom_t : (L_r(w0), \ldots, L_r(wi)) \in \Delta(L_r(w), L_t(w), i)$ where $i = \max\{j \mid wj \in dom_t\}$.

A **path** of the run r is an infinite linearly ordered subset of dom_r (i.e. it denotes a branch in the tree). We say that run r is **successful** if each path of r satisfies the accepting condition C. An input tree t is **accepted** by A if there is a successful run.

Definition 8.16 (Büchi tree automaton)

A Büchi tree automaton is given by an ω-tree automaton $A = (St, q_I, \Delta, F)$ where $F \subseteq St$ is a set of final states. A run $r = (dom_r, L)$ is **successful** if, and only if, for each path p on r there is a state that occurs infinitely often on p; i.e. for all paths p of r we have that

$$Inf(L|_p) \cap F \neq \emptyset.$$

$L|_p$ denotes the set of states in L which do also appear on p.

Definition 8.17 (Rabin tree automaton)

A Rabin tree automaton (or pairs tree automaton) is given by an ω-tree automaton $A = (St, q_I, \Delta, \Omega)$ where

$$\Omega = \{(L_1, U_1), \dots, (L_n, U_n)\}$$

where each pair $(L_i, U_i) \subseteq St \times St$ is a set of "accepting" pairs (these pairs are called Rabin pairs). A run $r = (dom_r, L)$ is successful if, and only if, for each path p on r there is an index $i \in \{1, \dots, n\}$ such that no state (resp. a state) from L_i (resp. from U_i) occurs infinitely often on p; i.e.

$$Inf(L|_p) \cap L_i = \emptyset \quad \text{and} \quad Inf(L|_p) \cap U_i \neq \emptyset$$

Theorem 8.18 ([Rabin, 1970])

There is a set of trees that is acceptable by a Rabin tree automaton but not by any Büchi tree automaton.

8.4 Emptiness Checking

Checking Emptiness

For the model checking algorithms we need to check whether the language of a Büchi automaton is empty.

Definition 8.19 (Graph reachability)

Let $G = (V, E)$ be graph. Given two vertices $u, v \in V$ the graph-reachability problem is the question whether v is reachable from u.

Checking Emptiness

For the model checking algorithms we need to check whether the language of a Büchi automaton is empty.

Definition 8.19 (Graph reachability)

Let $G = (V, E)$ be graph. Given two vertices $u, v \in V$ the **graph-reachability problem** is the question whether v is reachable from u.

Theorem 8.20 ([Jones, 1977, Jones, 1975])

The **graph-reachability problem** *is* $NLOGSPACE$**-complete** *under logspace-reductions.*

Theorem 8.21 ([Emerson and Lei, 1987])

The emptiness problem for Büchi automata is solvable in linear time and in nondeterministic logarithmic space .

Proof

We check whether there is some ultimately periodic word by finding an accepting state reachable from the initial state and from itself.

Theorem 8.21 ([Emerson and Lei, 1987])

The emptiness problem for Büchi automata is solvable in linear time and in nondeterministic logarithmic space .

Proof

We check whether there is some ultimately periodic word by finding an accepting state reachable from the initial state and from itself. The following algorithm runs in non-deterministic logarithmic space:

1. Guess an accepting state r, and
2. check whether $reach(r, r)$.

$\rightsquigarrow$: Back to **LTL** model checking, pp. 571.

How does $reach(x, y)$ work?

1. Chose some x-successor x' (non-determinism!).
2. Return "yes", if $x' = y$ else $reach(x', y)$.

How does $reach(x, y)$ work?

1. Chose some x-successor x' (non-determinism!).
2. Return “yes”, if $x' = y$ else $reach(x', y)$.

Hardness is shown by a reduction of the $NLOGSPACE$-complete problem of graph reachability from Definition 8.19. Given G, u, v, transform G to a Büchi automaton with initial state u and final state v and add a loop to v. Then:

v reachable from u in G iff automaton non-empty.

$\square$

Theorem 8.22 ([Rabin, 1970, Vardi and Wolper, 1984])

The emptiness problem for Büchi tree automata is decidable and P-complete under logarithmic space reductions.

Theorem 8.23 ([Emerson and Jutla, 1988, Pnueli and Rosner, 1989a])

The non-emptiness problem for Rabin tree automata is decidable and complete for NP.

Theorem 8.24 ([Emerson and Jutla, 1999])

The non-emptiness problem for pairs tree automata is decidable in deterministic time $(mn)^{\mathcal{O}(n)}$ where m is the number of states and n the number of pairs in the automaton.

8.5 Determinization

Determinization of Automata

Theorem 8.25 (Safra's construction [Safra, 1988])

Let A be a nondeterministic Büchi automaton with n states. Then, there is an equivalent deterministic Rabin automaton with $2^{\mathcal{O}(n \log n)}$ states.

9. Acknowledgements

Acknowledgements

- Thanks to Nils Bulling for providing us with some of the slides and pictures on modelchecking.

10. References

Abadi, M. and Manna, Z. (1989).
Temporal Logic Programming.
Journal of Symbolic Computation, 8: 277–295.

Alechina, N., Dastani, M., Logan, B., and Meyer, J.-J. C. (2011).
Reasoning about Agent Deliberation.
Autonomous Agents and Multi-Agent Systems, 22(2):356–381.

Alur, R., Henzinger, T. A., and Kupferman, O. (1997).
Alternating-Time Temporal Logic.
In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 100–109. IEEE Computer Society Press.

Alur, R., Henzinger, T. A., and Kupferman, O. (2002).
Alternating-Time Temporal Logic.
Journal of the ACM, 49:672–713.

Ammann, P. and Offutt, J. (2008).
Introduction to Software Testing.
Cambridge University Press.

Baier, C. and Katoen, J.-P. (2008).
Principles of Model Checking.
MIT Press.

Ball, T. and Rajamani, S. K. (2001).

The SLAM Toolkit.
In *Proc. 13th International Conference on Computer Aided Verification (CAV)*, volume 2102 of *LNCS*, pages 260–264. Springer.

Barringer, H., Fisher, M., Gabbay, D., Owens, R., and Reynolds, M., editors (1996).
The Imperative Future: Principles of Executable Temporal Logics.
Research Studies Press.

Bordini, R. H., Fisher, M., Pardavila, C., and Wooldridge, M. (2003).
Model Checking AgentSpeak.
In *Proc. 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS-2003)*.

Clarke, E. and Emerson, E. (1981).
Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic.
In *Proc. Logics of Programs Workshop*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71.

Clarke, E., Emerson, E., and Sistla, A. (1986).
Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications.
ACM Transactions on Programming Languages and Systems, 8(2):244–263.

Clarke, E. M., Grumberg, O., and Peled, D. (1999).
Model Checking.
MIT Press.

Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., and Talcott, C. (2003).
The Maude 2.0 System.

In Nieuwenhuis, R., editor, *Rewriting Techniques and Applications (RTA 2003)*, number 2706 in Lecture Notes in Computer Science, pages 76–87. Springer-Verlag.

de Boer, F. S., Hindriks, K. V., van der Hoek, W., and Meyer, J.-J. C. (2007).
A Verification Framework for Agent Programming with Declarative Goals.
J. Applied Logic, 5(2):277–302.

Dennis, L. A. and Farwer, B. (2008).
Gwendolen: A BDI Language for Verifiable Agents.
In Löwe, B., editor, *Proc. AISB'08 Workshop on Logic and the Simulation of Interaction and Reasoning*, Aberdeen. AISB.

Dennis, L. A. and Fisher, M. (2008).
Programming Verifiable Heterogeneous Agent Systems.
In *Proc. 6th International Workshop on Programming in Multi-agent Systems (ProMAS)*, volume 5442 of *LNCS*, pages 40–55. Springer Verlag.

Dennis, L. A., Fisher, M., Webster, M., and Bordini, R. H. (2012).
Model Checking Agent Programming Languages.
Automated Software Engineering, 19(1):5–63.

Dix, J., Kraus, S., and Subrahmanian, V. S. (2001).
Temporal Agent Programs.
Artificial Intelligence, 127(1):87–135.

Dix, J., Kraus, S., and Subrahmanian, V. S. (2006).

Heterogeneous Temporal Probabilistic Agents.
ACM Trans. Comput. Log., 7(1):151–198.

Dixon, C., Fisher, M., and Bolotov, A. (2002).
Resolution in a Logic of Rational Agency.
Artificial Intelligence, 139(1):47–89.

Emerson, E. and Halpern, J. (1986).
"Sometimes" and "Not Never" Revisited: On Branching versus Linear Time Temporal Logic.
Journal of the ACM, 33(1):151–178.

Emerson, E. A. and Jutla, C. S. (1988).
The complexity of tree automata and logics of programs.
In *SFCS '88: Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 328–337, Washington, DC, USA. IEEE Computer Society.

Emerson, E. A. and Jutla, C. S. (1999).
The complexity of tree automata and logics of programs.
SIAM J. Comput., 29:132–158.

Emerson, E. A. and Lei, C.-L. (1987).
Modalities for Model Checking: Branching Time Logic Strikes Back.
Science of Computer Programming, 8(3):275–306.

Fagin, R., Halpern, J. Y., Moses, Y., and Vardi, M. Y. (1995).
Reasoning about Knowledge.

MIT Press: Cambridge, MA.

Farwer, B. and Dennis, L. A. (2007).
Translating into an Intermediate Agent Layer: A Prototype in Maude.
In *Proc. International Workshop on Concurrency, Specification and Programming (CS&P)*, Lagow, Poland.

Fisher, M. (2011).
An Introduction to Practical Formal Methods Using Temporal Logic.
Wiley.

Fisher, M. and Hepple, A. (2009).
Executing Logical Agent Specifications.
In Bordini, R. H., Dastani, M., Dix, J., and El Fallah-Seghrouchni, A., editors, *Multi-agent Programming: Languages, Tools and Applications*, pages 1–27. Springer.

Gerth, R., Peled, D., Vardi, M. Y., and Wolper, P. (1995).
Simple On-the-fly Automatic Verification of Linear Temporal Logic.
In *Proc. 15th Workshop on Protocol Specification Testing and Verification (PSTV)*, pages 3–18. Chapman & Hall.

Giordano, L., Martelli, A., and Schwind, C. (2007).
Specifying and Verifying Interaction Protocols in a Temporal Action Logic.
Journal of Applied Logic, 5(2):214–234.

Havelund, K. and Rosu, G. (2001).
Monitoring Programs Using Rewriting.

In *Proc. 16th IEEE International Conference on Automated Software Engineering (ASE)*, pages 135–143. IEEE Computer Society Press.

Holzmann, G. J. (2003).
The Spin Model Checker: Primer and Reference Manual.
Addison-Wesley.

Holzmann, G. J. and Smith, M. H. (1999a).
A Practical Method for Verifying Event-Driven Software.
In *Proc. International Conference on Software Engineering (ICSE)*, pages 597–607.

Holzmann, G. J. and Smith, M. H. (1999b).
Software Model Checking.
In *Proc. Formal Description Techniques (FORTE)*, pages 481–497.

Immerman, N. (1981).
Number of quantifiers is better than number of tape cells.
Journal of Computer and System Sciences, 22(3):384 – 406.

Jones, N. D. (1975).
Space-bounded reducibility among combinatorial problems.
Journal of Computer and System Sciences, 11(1):68 – 85.

Jones, N. D. (1977).
Corrigendum: Space-bounded reducibility among combinatorial problems.
J. Comput. Syst. Sci., 15(2):241.

Jongmans, S.-S. T. Q., Hindriks, K. V., and van Riemsdijk, M. B. (2010).
Model Checking Agent Programs by Using the Program Interpreter.
In *Proc. 11th International Workshop on Computational Logic in Multi-agent Systems (CLIMA)*, volume 6245 of *LNCS*, pages 219–237. Springer.

Kakas, A. C., Kowalski, R. A., and Toni, F. (1993).
Abductive Logic Programming.
Journal of Logic and Computation, 2(6):719–770.

Kowalski, R. (1979).
Algorithm=Logic+Control.
Communications of the ACM, 22(7):424–436.

Lichtenstein, O. and Pnueli, A. (1985).
Checking that finite state concurrent programs satisfy their linear specification.
In *POPL '85: Proceedings of the 12th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 97–107, New York, NY, USA. ACM.

Maidl, M. (2000).
The common fragment of ctl and ltl.
In *FOCS*, pages 643–652. IEEE Computer Society.

Manna, Z. and Waldinger, R. J. (1971).
Toward Automatic Program Synthesis.
Communications of the ACM, 14(3):151–165.

McMillan, K. L. (1993).
Symbolic Model Checking.
Kluwer Academic Publishers.

Mili, A., Desharnais, J., and Gagné, J. R. (1986).
Formal Models of Stepwise Refinements of Programs.
ACM Computer Surveys, 18(3):231–276.

Orgun, M. A. and Wadge, W. W. (1992).
Towards a Unified Theory of Intensional Logic Programming.
Journal of Logic Programming, 13(1–4):413–440.

Owre, S., Rushby, J., Shankar, N., and Stringer-Calvert, D. (1998).
PVS: An Experience Report.
In Hutter, D., Stephan, W., Traverso, P., and Ullman, M., editors, *Applied Formal Methods*, volume 1641 of *Lecture Notes in Computer Science*, pages 338–345. Springer.

Parikh, R. (1979).
Propositional Dynamic Logics of Programs: A Survey.
Lecture Notes in Computer Science, 125:102–144.

Paulson, L. C. (1994).
A Generic Theorem Prover, volume 828 of *Lecture Notes in Computer Science*.
Springer.

Pnueli, A. (1977).

The Temporal Logic of Programs.
In *Proceedings of FOCS*, pages 46–57.

Pnueli, A. and Rosner, R. (1989a).
On the synthesis of a reactive module.
In *POPL '89: Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 179–190, New York, NY, USA. ACM.

Pnueli, A. and Rosner, R. (1989b).
On the Synthesis of an Asynchronous Reactive Module.
In *Proc. 16th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 372 of *LNCS*, pages 652–671. Springer.

Prasad, M. R., Biere, A., and Gupta, A. (2005).
A Survey of Recent Advances in SAT-based Formal Verification.
International Journal on Software Tools for Technology Transfer, 7(2):156–173.

Rabin, M. (1970).
Weakly definable relations and special automata.
Mathematical Logic and Foundations of Set Theory, pages 1–23.

Sadri, F. and Toni, F. (2006).
A Formal Analysis of KGP Agents.
In *Proc. European Conference on Logics in Artificial Intelligence (JELIA)*, volume 4160 of *Lecture Notes in Artificial Intelligence*, pages 413–425, Heidelberg, Germany. Springer-Verlag.

Safra, S. (1988).
On the complexity of omega -automata.
In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 319–327, Washington, DC, USA. IEEE Computer Society.

Schnoebelen, P. (2003).
The complexity of temporal model checking.
In *Advances in Modal Logics, Proceedings of AiML 2002*. World Scientific.

Schobbens, P. Y. (2004).
Alternating-Time Logic with Imperfect Recall.
Electronic Notes in Theoretical Computer Science, 85(2).

Shapiro, S., Lespérance, Y., and Levesque, H. (2002).
The Cognitive Agents Specification Language and Verification Environment for Multiagent Systems.
In *Proc. 1st International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 19–26, New York, NY, USA. ACM Press.

Sistla, A. P. and Clarke, E. M. (1985).
The complexity of propositional linear temporal logics.
J. ACM, 32(3):733–749.

Sistla, A. P., Vardi, M., and Wolper, P. (1987).
The Complementation Problem for Büchi Automata with Applications to Temporal Logic.
Theoretical Computer Science, 49:217–237.

van Riemsdijk, B., de Boer, F. S., Dastani, M., and Meyer, J.-J. C. (2006).
Prototyping 3APL in the Maude Term Rewriting Language.
In *Proc. 5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1279–1281. ACM.

Vardi, M. Y. and Wolper, P. (1984).
Automata theoretic techniques for modal logics of programs: (extended abstract).
In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 446–456, New York, NY, USA. ACM.

Vardi, M. Y. and Wolper, P. (1986).
An automata-theoretic approach to automatic program verification (preliminary report).
In *Proceedings of the First Annual IEEE Symposium on Logic in Computer Science (LICS 1986)*, pages 332–344. IEEE Computer Society Press.

Vardi, M. Y. and Wolper, P. (1994).
Reasoning About Infinite Computations.
Information and Computation, 115(1):1–37.

Visser, W., Havelund, K., Brat, G. P., Park, S., and Lerda, F. (2003).
Model Checking Programs.
Automated Software Engineering, 10(2):203–232.

Webster, M., Fisher, M., Cameron, N., and Jump, M. (2011).
Model Checking and the Certification of Autonomous Unmanned Aircraft Systems.
In *Proc. 30th International Conference on Computer Safety, Reliability and Security (SAFECOMP)*.

Winikoff, M. (2007).
Implementing Commitment-Based Interactions.
In *Proc. 6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 1–8, New York, NY, USA. ACM.

Wooldridge, M., Fisher, M., Huget, M.-P., and Parsons, S. (2006).
Model Checking for Multiagent Systems: The MABLE Language and its Applications.
International Journal of Artificial Intelligence Tools, 15(2):195–225.