Logical Specifications for Reactive Systems

Rayna Dimitrova
University of Leicester

Midlands Graduate School 2019

Acknowledgement: Many slides courtesy of Bernd Finkbeiner.

Reactive systems

- Continuous interaction with environment
- Finite state space, focus on control, not data transformation
- Typical examples:
  - Hardware circuits
  - Reactive controllers of cyberphysical systems

Correctness specified by temporal properties.

Cyberphysical example: Autonomous driving

Reactive traffic planner decides

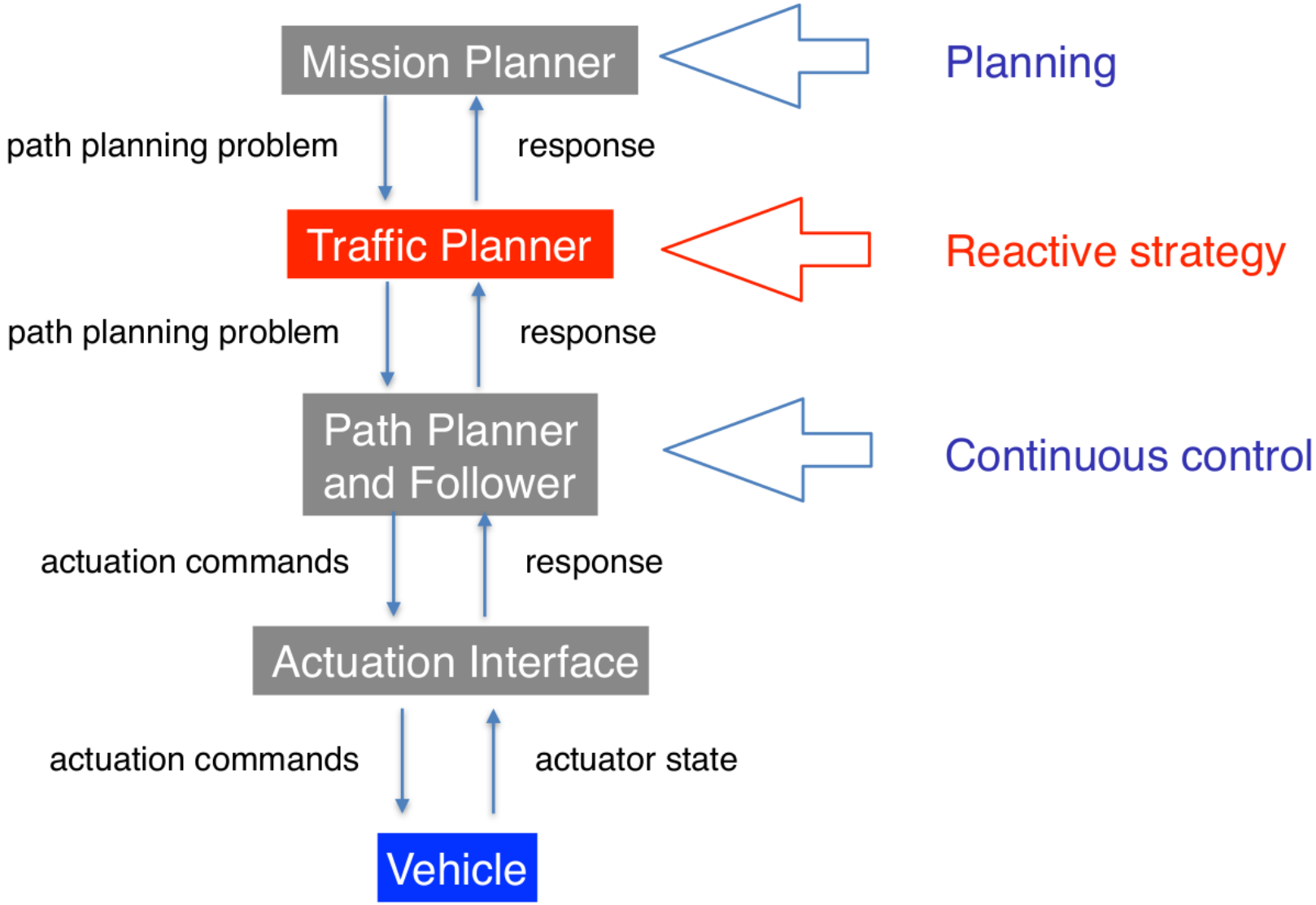
- whether vehicle should stay in the travel lane or perform a passing manoeuvre,
- whether it should go or stop,
- whether it can reverse, . . .

Specification consists of

- traffic rules: for example "no collisions", "obey speed limits"
- goals: for example "eventually the checkpoint should be reached"

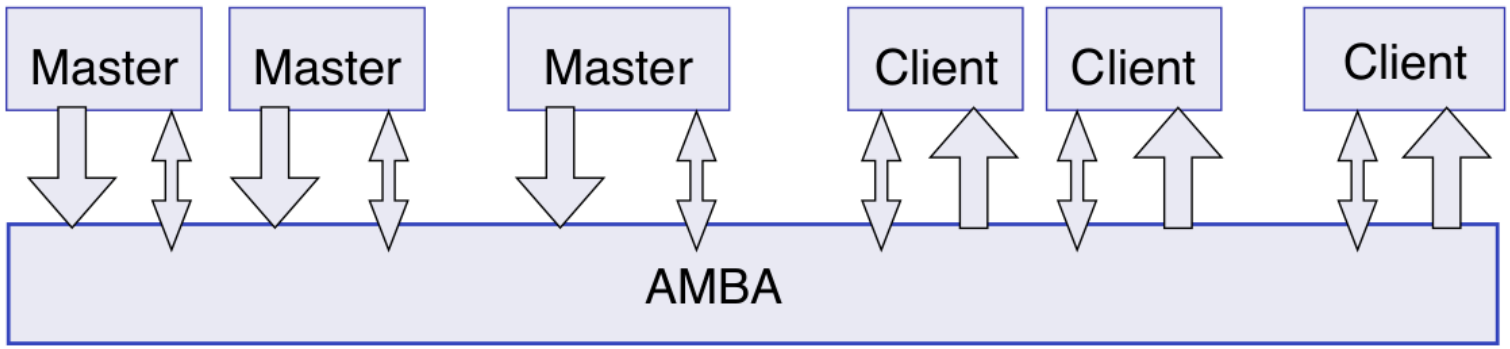
[Murray et al, 2012]

Hierarchical control



Hardware example: AMBA AHB Bus

- High-performance on-chip bus
- Data, address and control signals
- Up to 16 masters and 16 clients



Specification consists of

- 12 guarantees: for example "when a locked unspecified length burst starts, new access does not start until the current master i releases the bus by lowering $HBUSREQ_i$"
- 3 assumptions: for example "the clients indicate infinitely often that they have finished processing the data by lowering $HREADY$"

[Bloem et al, 2007]

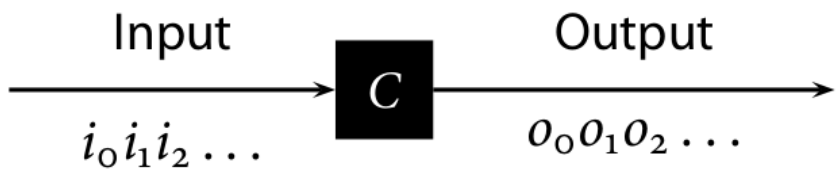
Motivation for synthesis

- In safety-critical systems and in hardware **bugs are expensive!**
- Finding and fixing bugs is **time-consuming!**

Construct correct systems automatically!

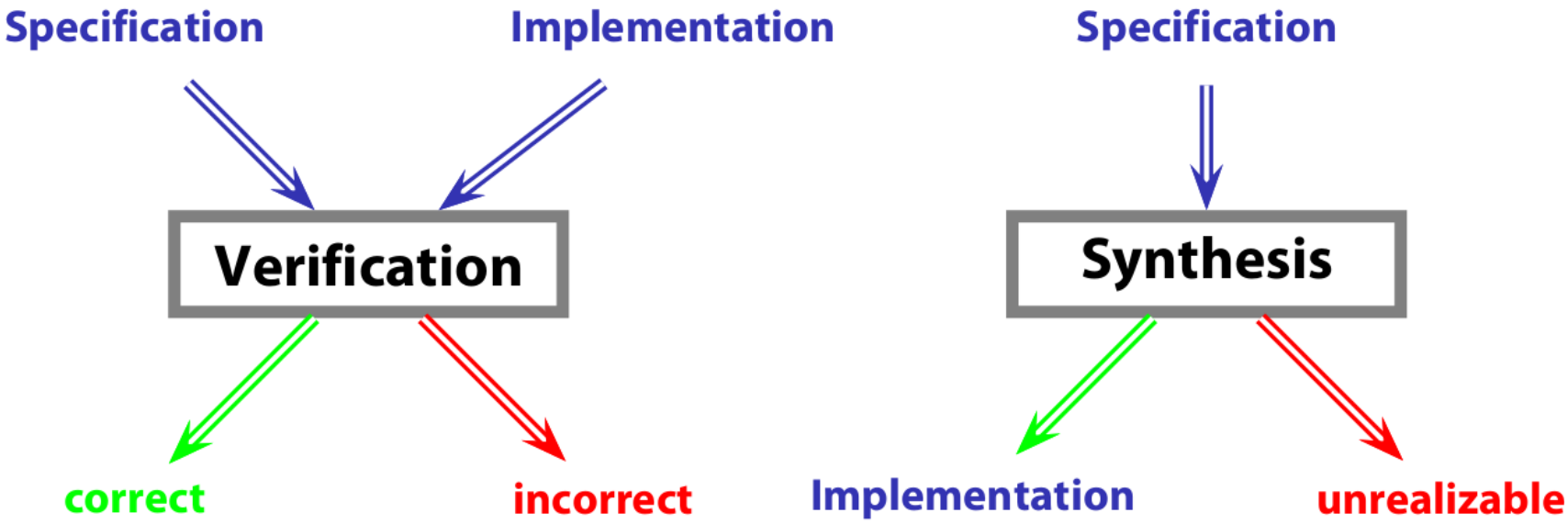
The reactive synthesis problem

Alonzo Church (1957)



Given a requirement φ
on the input-output behaviour of a Boolean circuit,
compute a circuit that satisfies φ

Verification vs synthesis



Realizability: Does there exists implementation satisfying the spec?

Synthesis: Construct such an implementation (if there is one).

Premise of synthesis

It is easier to say what a system should do
then how it should be done.

Example:

- **Eventually** the next waypoint should be reached.
- The vehicle should **never** collide with obstacles.

Specification language?

Linear-time temporal logic (LTL)

modal logic over infinite sequences [Pnueli 1977]

- **Propositional logic**
  - $a \in AP$ — atomic proposition
  - $\neg\phi$ and $\phi \wedge \psi$ — negation and conjunction
- **Temporal operators**
  - $\mathsf{X}\phi$ — next state fulfills ϕ
  - $\phi \,\mathsf{U}\, \psi$ — ϕ holds Until a ψ-state is reached

derived operators

$\mathsf{F}\,\phi \equiv true\,\mathsf{U}\,\phi$ — "some time in the future"

$\mathsf{G}\,\phi \equiv \neg\,\mathsf{F}\,\neg\,\phi$ — "from now on forever"

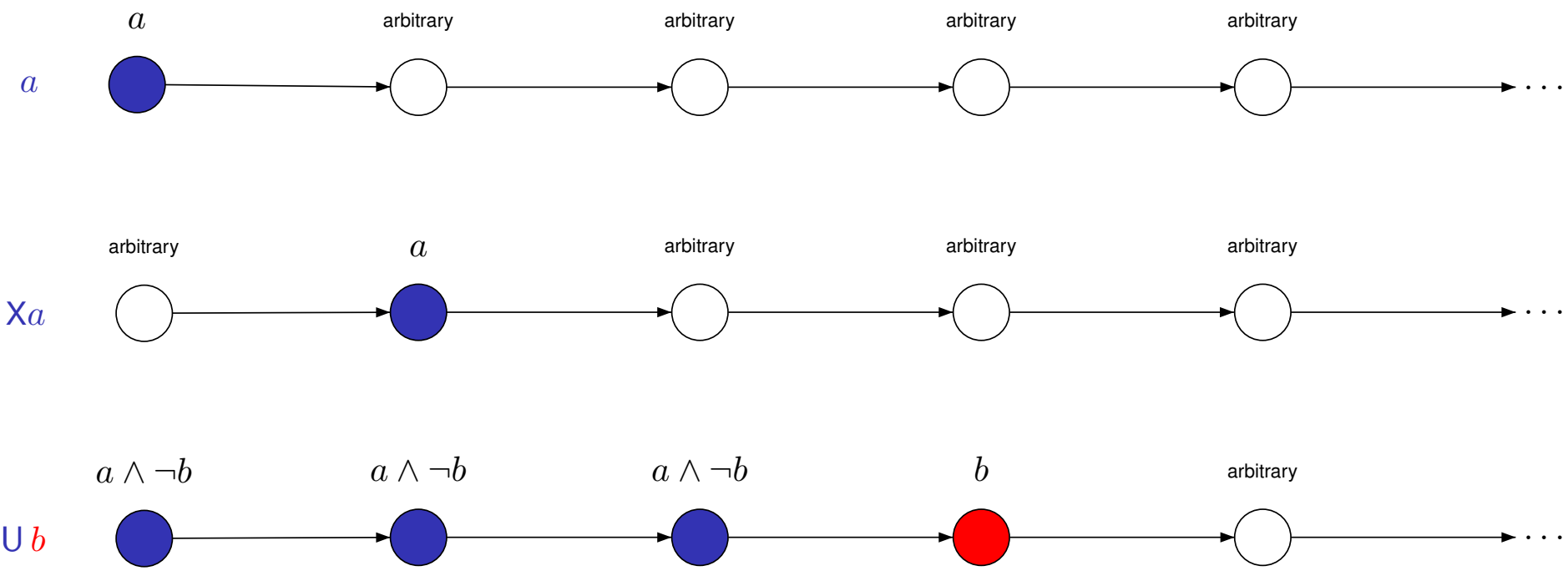
Semantics of LTL

LTL formulas are interpreted over **infinite sequences of labels**.

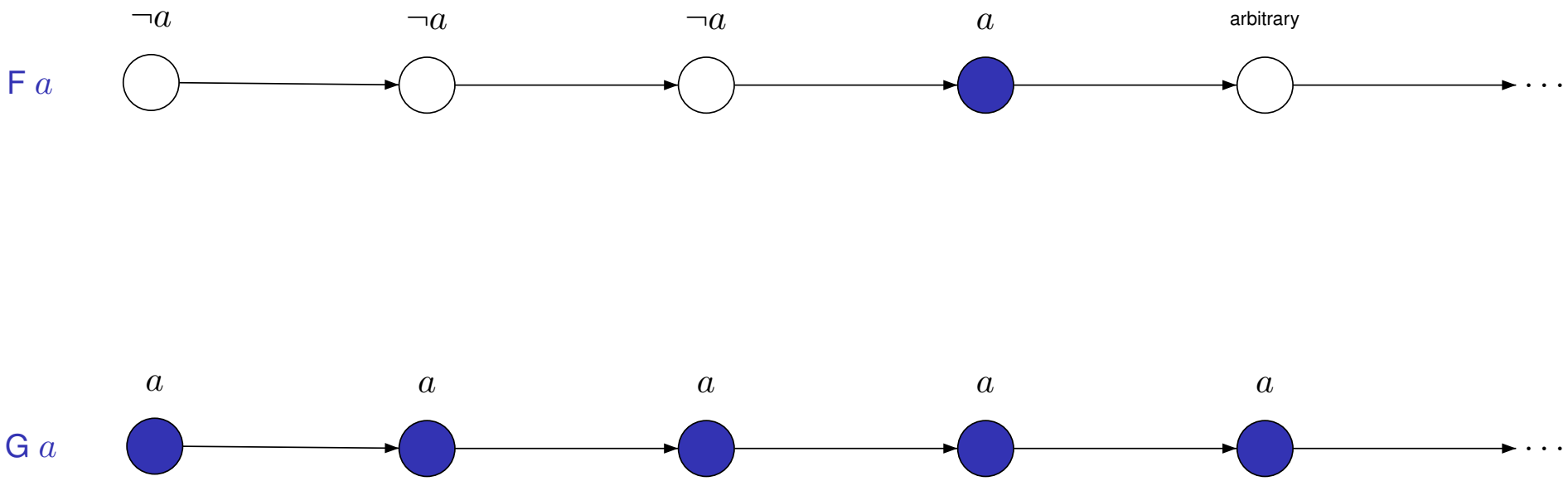
A **label** is a set of atomic propositions (that hold true at a given time).

A reactive system satisfies an LTL formula φ if each of its executions (the corresponding sequences of labels) satisfies the formula φ.

Intuitive semantics of LTL



Intuitive semantics of LTL (cont.)



Example: Traffic light properties

- Once red, the light cannot become green immediately:

Example: Traffic light properties

- Once red, the light cannot become green immediately:

$$\mathsf{G}\,(\mathsf{red} \;\rightarrow\; \neg\,\mathsf{X}\,\mathsf{green})$$

Example: Traffic light properties

- Once red, the light cannot become green immediately:

$$\mathsf{G}\,(\mathsf{red} \;\rightarrow\; \neg\,\mathsf{X}\,\mathsf{green})$$

- The light becomes green eventually:

Example: Traffic light properties

- Once red, the light cannot become green immediately:

$$\mathsf{G}\,(\mathsf{red} \;\rightarrow\; \neg\,\mathsf{X}\,\mathsf{green})$$

- The light becomes green eventually:

$$\mathsf{F}\,\mathsf{green}$$

Example: Traffic light properties

- Once red, the light cannot become green immediately:

$$\mathsf{G}\,(\mathsf{red} \;\rightarrow\; \neg\,\mathsf{X}\,\mathsf{green})$$

- The light becomes green eventually:

$$\mathsf{F}\,\mathsf{green}$$

- Once red, the light always becomes green eventually:

Example: Traffic light properties

- Once red, the light cannot become green immediately:

$$\mathsf{G}\,(\mathsf{red} \rightarrow \neg\,\mathsf{X}\,\mathsf{green})$$

- The light becomes green eventually:

$$\mathsf{F}\,\mathsf{green}$$

- Once red, the light always becomes green eventually:

$$\mathsf{G}\,(\mathsf{red} \rightarrow \mathsf{F}\,\mathsf{green})$$

Example: Traffic light properties

- Once red, the light cannot become green immediately:

$$\mathsf{G}\,(\mathsf{red} \rightarrow \neg\,\mathsf{X}\,\mathsf{green})$$

- The light becomes green eventually:

$$\mathsf{F}\,\mathsf{green}$$

- Once red, the light always becomes green eventually:

$$\mathsf{G}\,(\mathsf{red} \rightarrow \mathsf{F}\,\mathsf{green})$$

- Once red, the light always becomes green eventually after being yellow for some time inbetween:

Example: Traffic light properties

- Once red, the light cannot become green immediately:

$$\mathsf{G}\,(\mathsf{red}\ \rightarrow\ \neg\,\mathsf{X}\,\mathsf{green})$$

- The light becomes green eventually:

$$\mathsf{F}\,\mathsf{green}$$

- Once red, the light always becomes green eventually:

$$\mathsf{G}\,(\mathsf{red}\ \rightarrow \mathsf{F}\,\mathsf{green})$$

- Once red, the light always becomes green eventually after being yellow for some time inbetween:

$$\mathsf{G}(\mathsf{red}\ \rightarrow \mathsf{X}(\mathsf{red}\ \mathsf{U}\,(\mathsf{yellow}\ \wedge\ \mathsf{X}(\mathsf{yellow}\ \mathsf{U}\,\mathsf{green}))))$$

Example: Synthesis of an arbiter circuit

An arbiter circuit

- receives **requests** r_1, r_2 from two clients and
- produces **grants** g_1 and g_2 for the two clients.

The specification of the arbiter is the conjunction of the properties:

- **Mutual exclusion**

  At no point in time there should be both g_1 and g_2 in the output.

- **Response**

  Every request r_i from the client (for $i \in \{1, 2\}$) should eventually be followed by a grant g_i for client i.

Example: Synthesis of an arbiter circuit

An arbiter circuit

- receives **requests** r_1, r_2 from two clients and
- produces **grants** g_1 and g_2 for the two clients.

The specification of the arbiter is the conjunction of the properties:

- **Mutual exclusion**

  At no point in time there should be both g_1 and g_2 in the output.

$$\mathsf{G}\,\neg(g_1 \wedge g_2)$$

- **Response**

  Every request r_i from the client (for $i \in \{1, 2\}$) should eventually be followed by a grant g_i for client i.

Example: Synthesis of an arbiter circuit

An arbiter circuit

- receives **requests** r_1, r_2 from two clients and
- produces **grants** g_1 and g_2 for the two clients.

The specification of the arbiter is the conjunction of the properties:

- **Mutual exclusion**

  At no point in time there should be both g_1 and g_2 in the output.

$$\mathsf{G}\,\neg(g_1 \wedge g_2)$$

- **Response**

  Every request r_i from the client (for $i \in \{1, 2\}$) should eventually be followed by a grant g_i for client i.

$$\mathsf{G}\,\big((r_1 \rightarrow \mathsf{F}\,g_1) \wedge (r_2 \rightarrow \mathsf{F}\,g_2)\big)$$

Semantics over sequences

Notation:

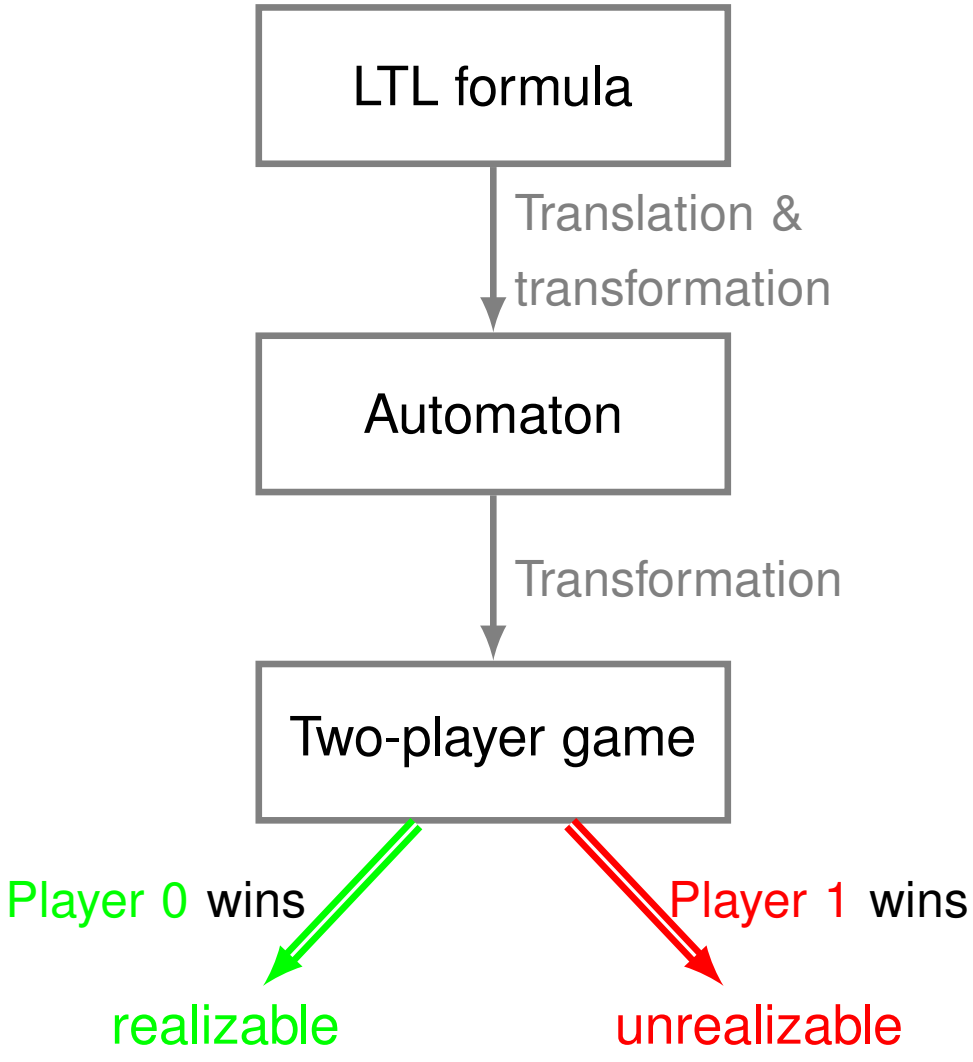
- AP: set of **atomic propositions**
- Alphabet $\Sigma = 2^{AP}$: subsets of AP.
  For $p \in AP, s \in 2^{AP}$ we write $s \models p$ iff $p \in s$.
- Σ^ω: set of **infinite sequences** over alphabet Σ.

Let $\sigma = s_0 s_1 s_2 \ldots \in \Sigma^\omega$ be a sequence of labels.

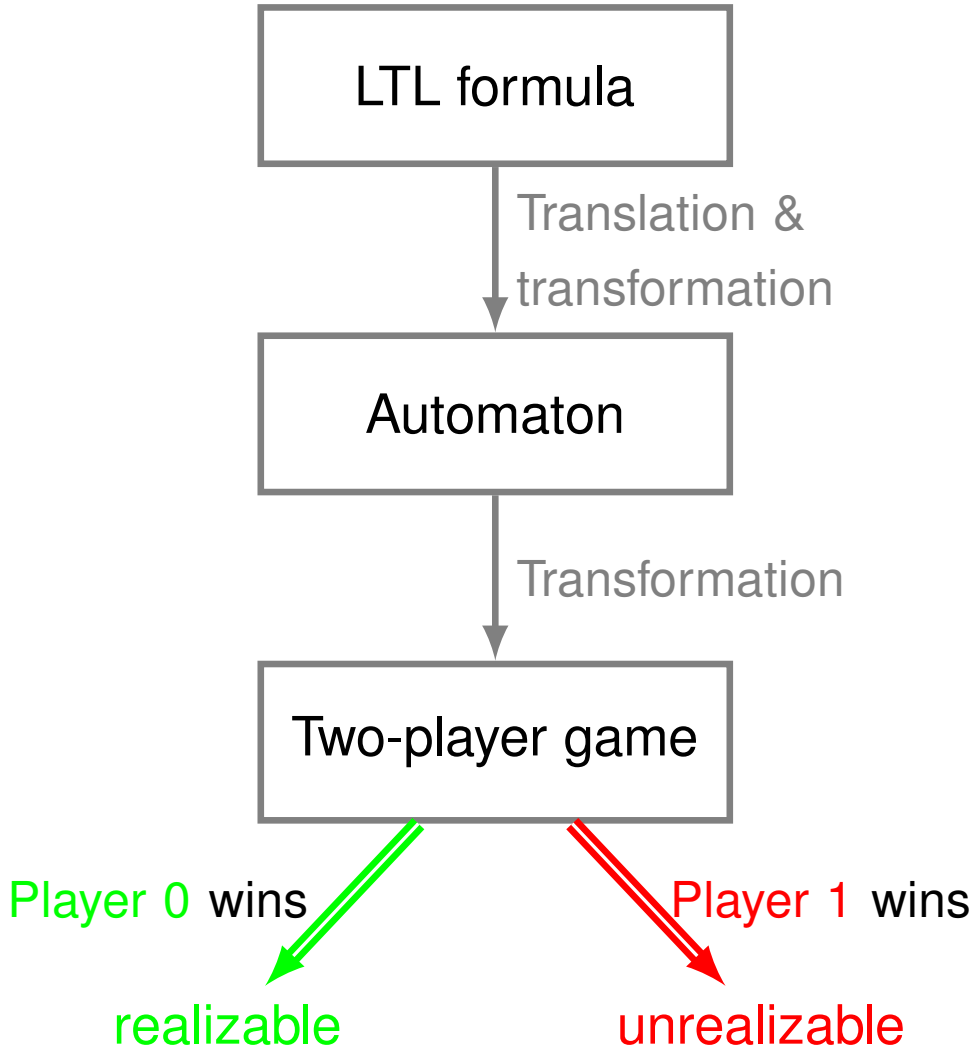
$$
\begin{array}{lcll}
\sigma & \models & true & \\
\sigma & \models & a & \text{iff} \quad a \in s_0 \quad (\text{i.e., } s_0 \models a) \\
\sigma & \models & \varphi_1 \wedge \varphi_2 & \text{iff} \quad \sigma \models \varphi_1 \text{ and } \sigma \models \varphi_2 \\
\sigma & \models & \neg\, \varphi & \text{iff} \quad \sigma \not\models \varphi \\
\sigma & \models & \mathsf{X}\varphi & \text{iff} \quad \sigma[1..] = s_1 s_2 s_3 \ldots \models \varphi \\
\sigma & \models & \varphi_1 \,\mathsf{U}\, \varphi_2 & \text{iff} \quad \exists j \geq 0.\ \sigma[j..] \models \varphi_2 \text{ and } \sigma[i..] \models \varphi_1,\ 0 \leq i < j
\end{array}
$$

where $\sigma[i..] = s_i s_{i+1} s_{i+2} \ldots$.

Synthesis from LTL specifications

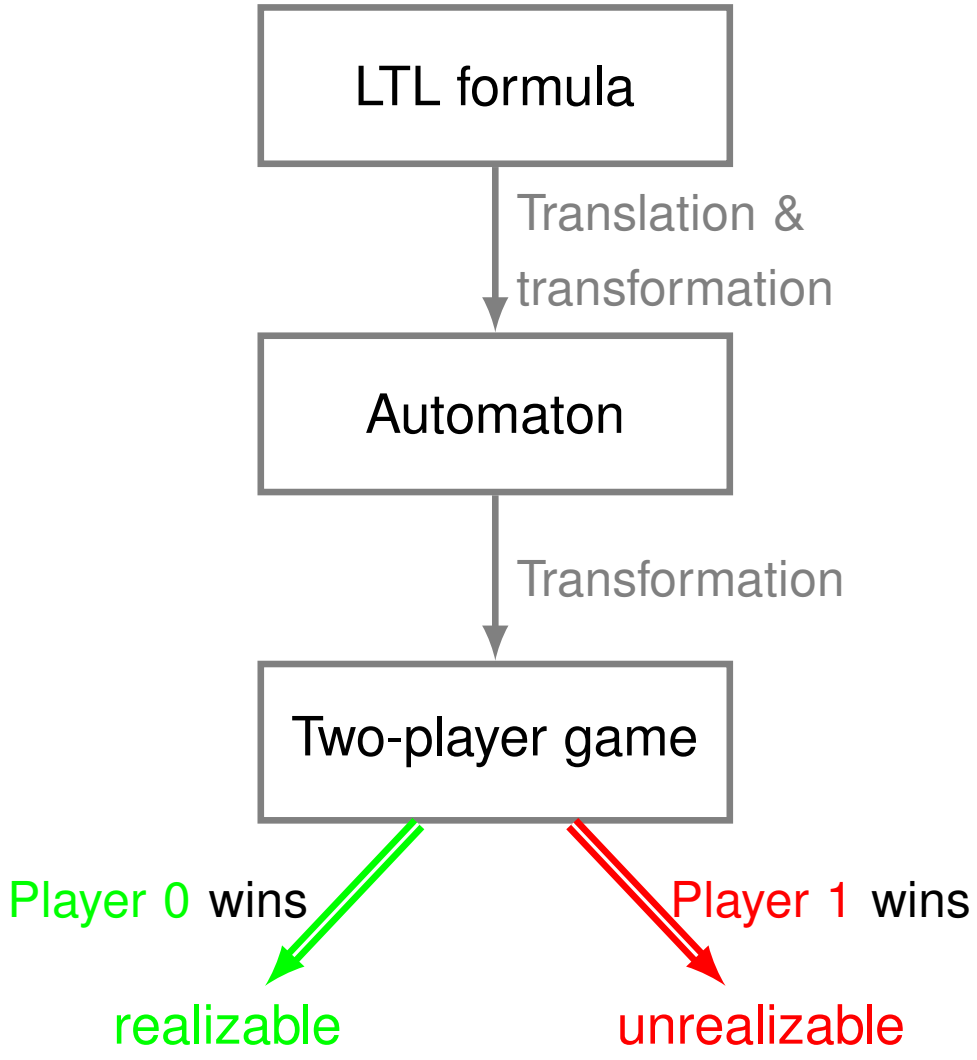


Synthesis from LTL specifications



Monday

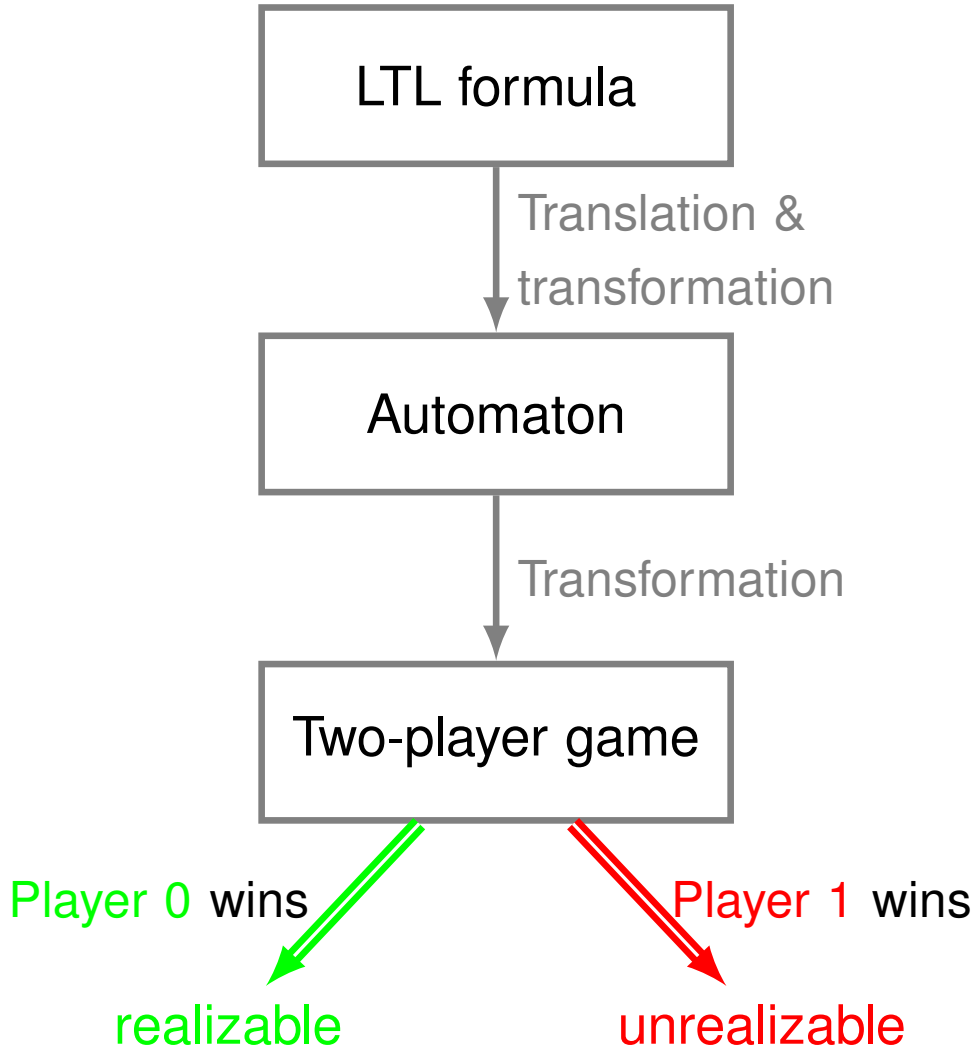
Synthesis from LTL specifications



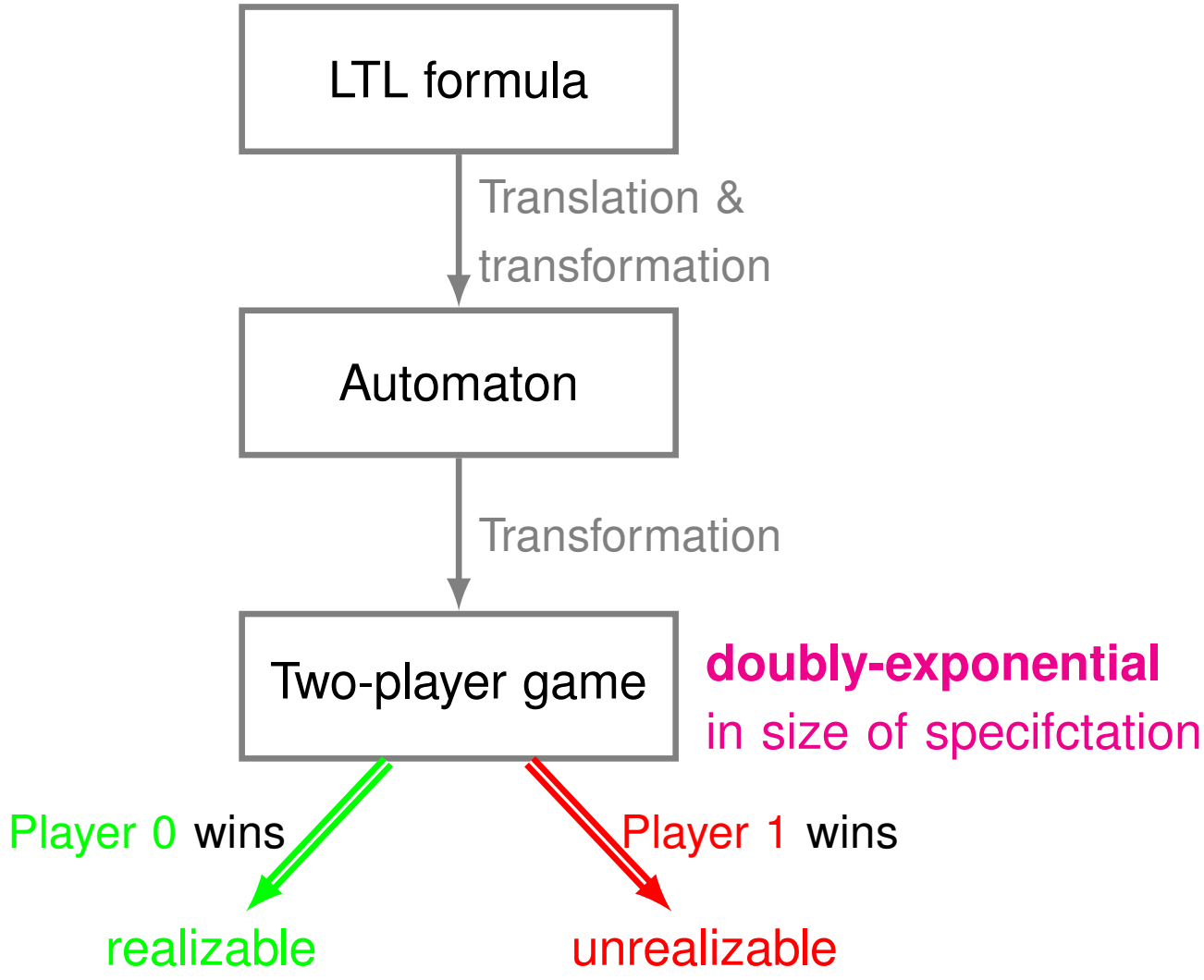
Monday

Tuesday

Complexity of synthesis from LTL specifications



Complexity of synthesis from LTL specifications



LTL synthesis

LTL synthesis is 2EXPTIME-complete.

GR(1): an efficient fragment of LTL

Generalized Reactivity(1) restricts specifications to specific form

[Piterman et al, 2006]

$$A_1 \wedge A_2 \wedge \ldots \wedge A_m \rightarrow G_1 \wedge G_2 \wedge \ldots \wedge G_n$$

Assumptions $A_1, A_2, \ldots, A_m$, **Guarantees** $G_1, G_2, \ldots, G_n$

Assumptions and guarantees are restricted to formulas of the form:

- **initialization properties:** Boolean combinations of propositions
- **safety properties** of the form $\mathsf{G}(\phi \rightarrow \mathsf{X}\,\psi)$
- **liveness properties** of the form $\mathsf{G}\,\mathsf{F}\,\phi$

Example: $(\mathsf{G}\,\mathsf{F}\,r) \rightarrow (\mathsf{G}\,\mathsf{F}\,g)$

GR(1): an efficient fragment of LTL

GR(1) games are comparatively small.

GR(1) synthesis

GR(1) synthesis can be done in exponential time. [Piterman et al, 2006]

However, not all LTL properties can be expressed in GR(1).

Nevertheless, very useful in practical applications.

Summary

In this lecture:

- Reactive systems and the synthesis problem
- LTL as a specification formalism for reactive systems
- The GR(1) fragment of LTL

In the rest of the course

- Monday: ω-automata as a specification formalism
- Tuesday: Synthesis games
- Wednesday: Approaches and Applications

Summary

In this lecture:

- Reactive systems and the synthesis problem
- LTL as a specification formalism for reactive systems
- The GR(1) fragment of LTL

In the rest of this course

- Monday: ω-automata as a specification formalism
- Tuesday: Synthesis games
- Wednesday: Approaches and Applications

Exercises

Exercise 1 Let p and q be atomic propositions. Show that:

a) $(\mathsf{G}\, p) \wedge (\mathsf{G}\, q)$ is equivalent to $\mathsf{G}(p \wedge q)$

b) $(\mathsf{G}\,\mathsf{F}\, p) \wedge (\mathsf{G}\,\mathsf{F}\, q)$ is **not** equivalent to $\mathsf{G}\,\mathsf{F}\,(p \wedge q)$

c) $(\mathsf{G}\,\mathsf{F}\, p) \rightarrow (\mathsf{G}\,\mathsf{F}\, q)$ is **not** equivalent to $\mathsf{G}(p \rightarrow \mathsf{F}\, q)$

Exercises

Exercise 2 Consider an elevator controller serving 3 floors, whose

- **input** is modelled via the Boolean variables

$$req_0, req_1, req_2$$

  expressing requests from the different floors

- **output** is modelled via the Boolean variables

$$loc_0, loc_1, loc_2$$

  expressing the current location of the elevator

Formulate in LTL the following assumption:

- If there is a request for a floor different from the floor on which the elevator is currently located, then the request for that floor is not set to false before the elevator visits that floor.

Exercises

Exercise 2 (continued)

Formulate in LTL the following guarantees:

- At each point in time, the elevator is located on exactly one floor.
- The elevator starts on the ground floor.
- The elevator can move one floor at a time.
- If there is a request for a floor different from the floor on which the elevator is currently located, then eventually the elevator is on that floor and the request for that floor is true.