Reactive Synthesis for Cyber-Physical System Control

Rüdiger Ehlers
University of Bremen & DFKI GmbH

RiSE Workshop, Pöllauberg, Austria, September 2016

Synthesis of reactive systems

Specification

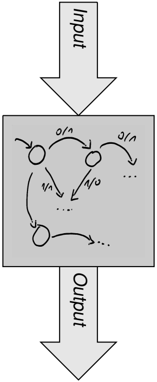
$+$

Input $= \{u, \ldots\}$
Output $= \{v, \ldots\}$

$\Rightarrow$

Realizable $\longrightarrow$

Not realizable



Synthesis of CPS controllers

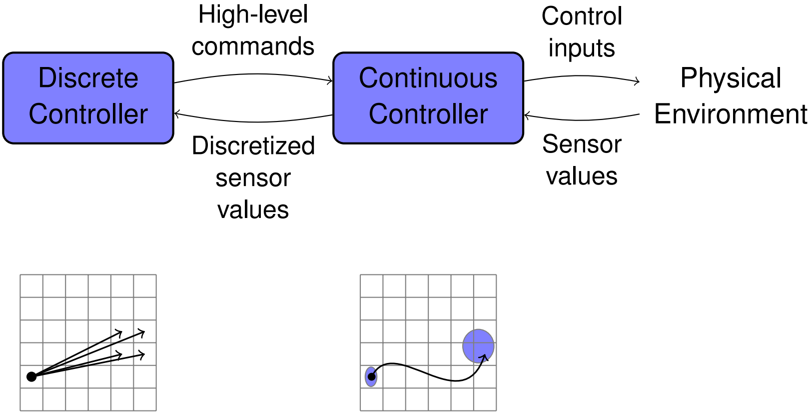
Main Question

How to deal with the **continuous** and **discrete** aspects of the problem at the same time?

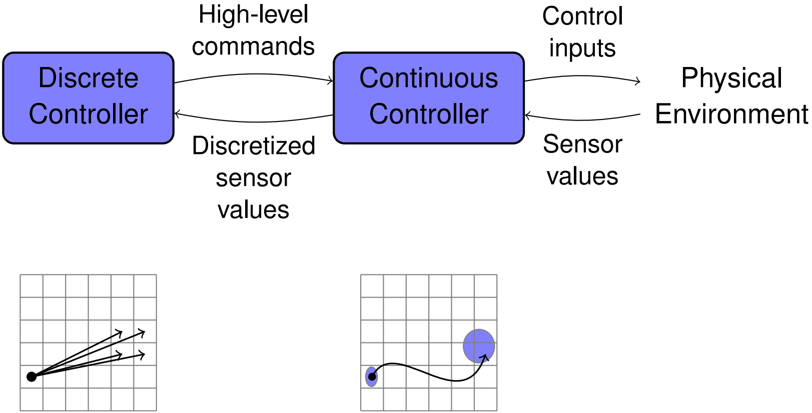
Approaches

1. Abstraction of the continuous world into the purely discrete world + discrete synthesis (e.g., Nilsson et al., 2016)
2. Simplifying the continuous parts (Linear hybrid automata / Timed automata) and using a specialized synthesis algorithm for the resulting mixed discrete/continuous model (e.g., Benerecetti et al., 2013; Papusha et al., 2016)
3. “Continuization” of the discrete parts and using purely continuous methods for controller computation

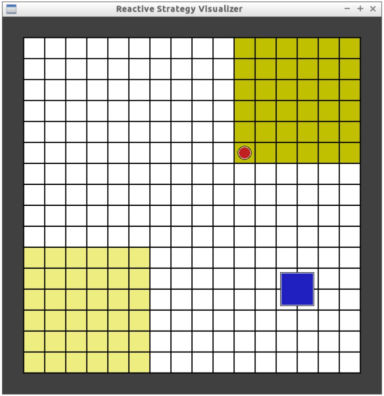
Using a discrete abstraction



Using a discrete abstraction

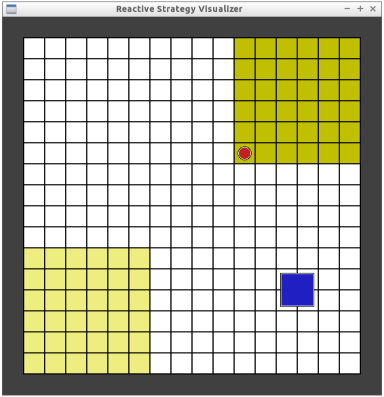


Discrete synthesis for CPS – Example



Based on joint work with Ufuk Topcu, HSCC 2014

Discrete synthesis for CPS – Example



Input/Output

Input:

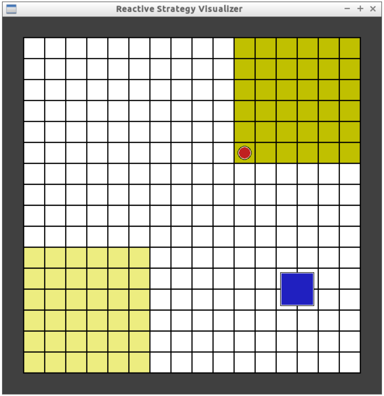
- (Sensed) positions of the robots
- Delivery requests

Output:

- Up/Left/Right/Down command of the red robot
- Pickup/drop actions of the red robot

Based on joint work with Ufuk Topcu, HSCC 2014

Discrete synthesis for CPS – Example

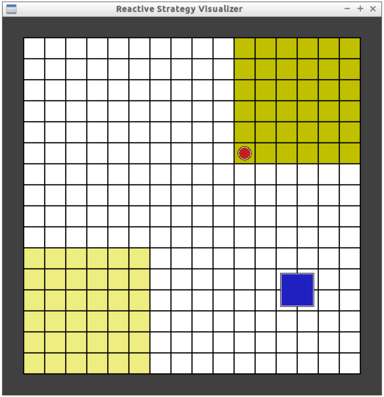


Guarantees

- Whenever a button is pressed, then the robot is eventually at the lower left region and performing a pick-up action, while later being in the top right region, performing a drop action, without performing a drop action in between.
- No crashes between the robots

Based on joint work with Ufuk Topcu, HSCC 2014

Discrete synthesis for CPS – Example

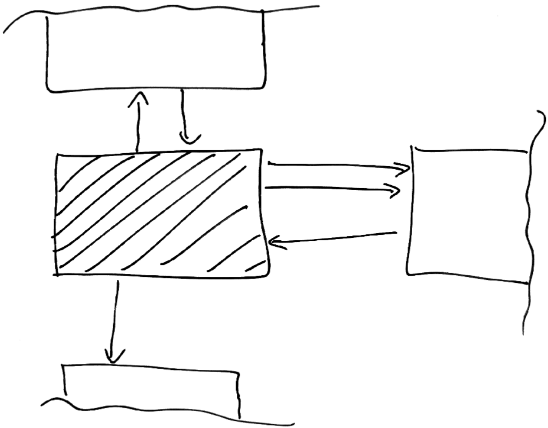


Assumptions

- Obstacle can only move in every second step
- Obstacle can only move by one cell per direction per step
- *x* position of the robot is updated according to its choice
- *y* position of the robot is updated according to its choice
- No robot jumps further that one cell are possible

Based on joint work with Ufuk Topcu, HSCC 2014

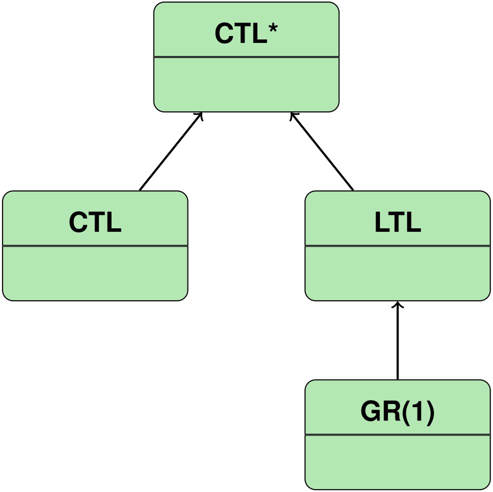
Assumptions and guarantees in specifications



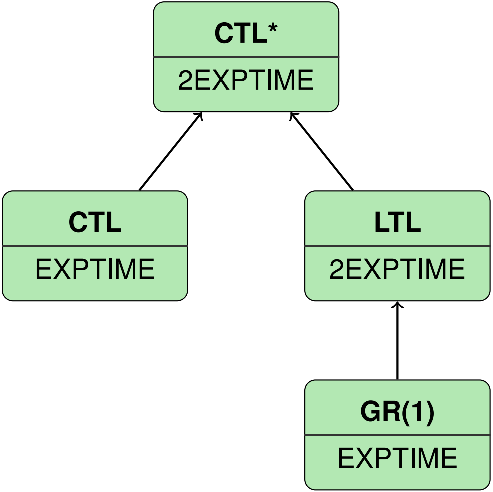
Specification shape

$$\left(\bigwedge \text{Assumptions}\right) \rightarrow \left(\bigwedge \text{Guarantees}\right)$$

Reactive synthesis – Complexity vs. expressivity



Reactive synthesis – Complexity vs. expressivity



Reactive synthesis – Complexity vs. expressivity

GR(1) synthesis applications

- On-chip bus arbiter (Bloem et al., 2007b,a; Godhal et al., 2011)
- High-level robot control (Kress-Gazit et al., 2009; Raman et al., 2013; Jing et al., 2013)
- Vehicle power management (Ozay et al., 2011a)
- Camera network control (Ozay et al., 2011b)
- ...

GR(1)

EXPTIME

So is that the end of the story?

And everyone lived happily ever after...

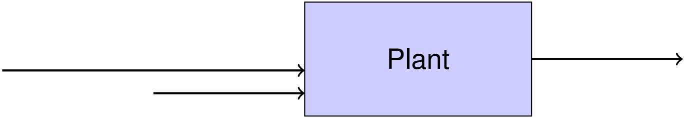
So is that the end of the story?

And everyone lived happily ever after...

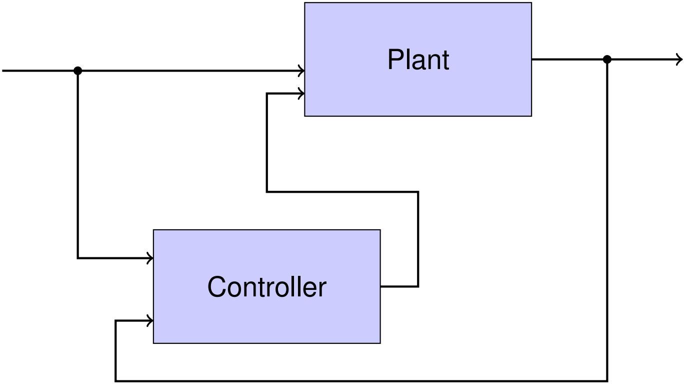
...well, not quite. There is also:

- Noise
- Imprecise modelling of the environment
- Incomplete information
- Scalability
- Robustness / Error-resilience
- Quirks of the synthesis algorithm
- ...

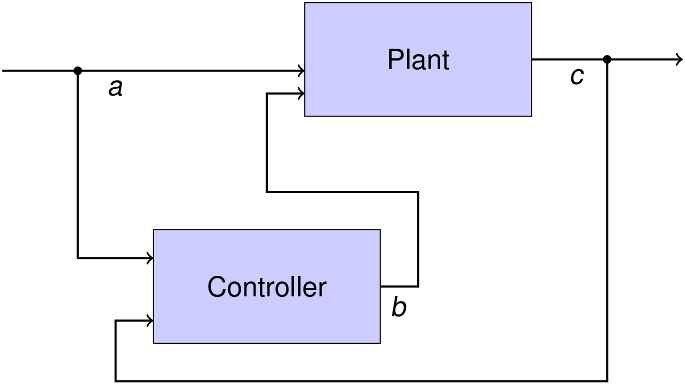
Controlling (Cyber-)physical systems



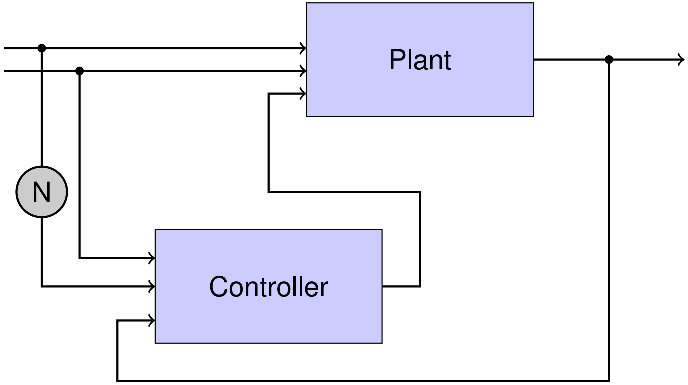
Controlling (Cyber-)physical systems



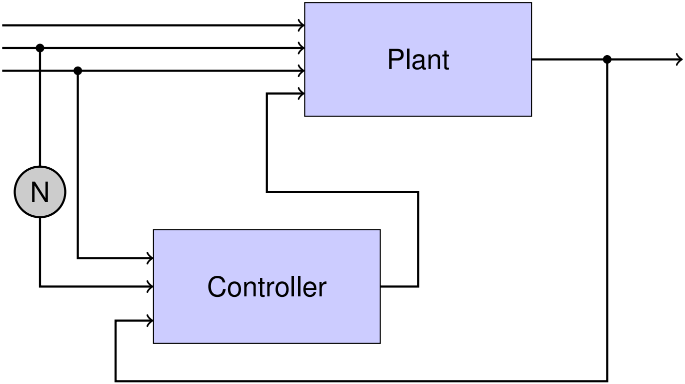
Controlling (Cyber-)physical systems



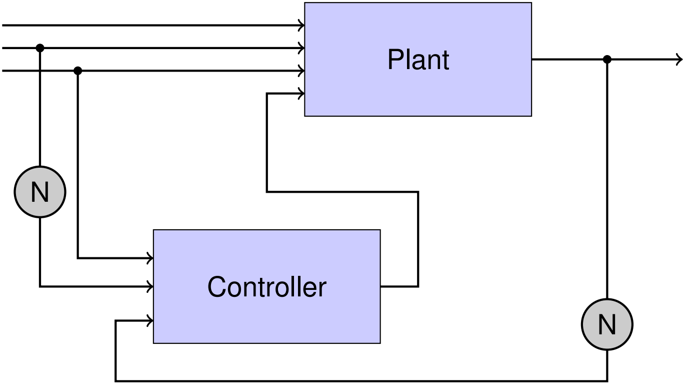
Controlling (Cyber-)physical systems



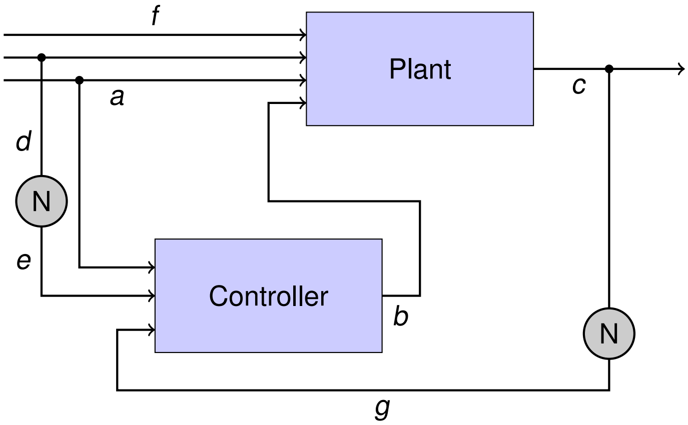
Controlling (Cyber-)physical systems



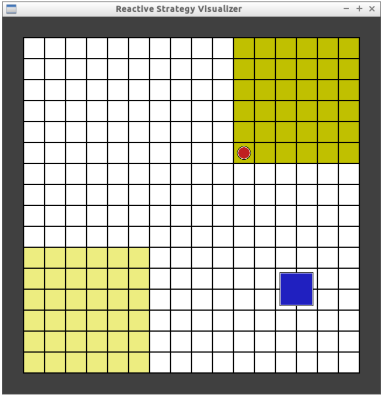
Controlling (Cyber-)physical systems



Controlling (Cyber-)physical systems



Example for error-resilience



Focus of this talk

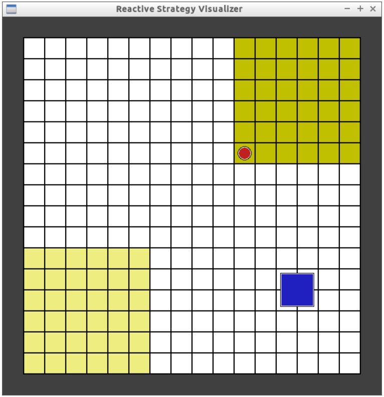
A few answers to how we can deal with...

- Noise
- Imprecise modelling of the environment
- Incomplete information
- Scalability
- Robustness / Error-resilience
- Quirks of the synthesis algorithm
- ...

Synthesizing *error-resilient* implementations

Based on joint work with Ufuk Topcu, HSCC 2014

Example for (discrete) robustness (revisited)

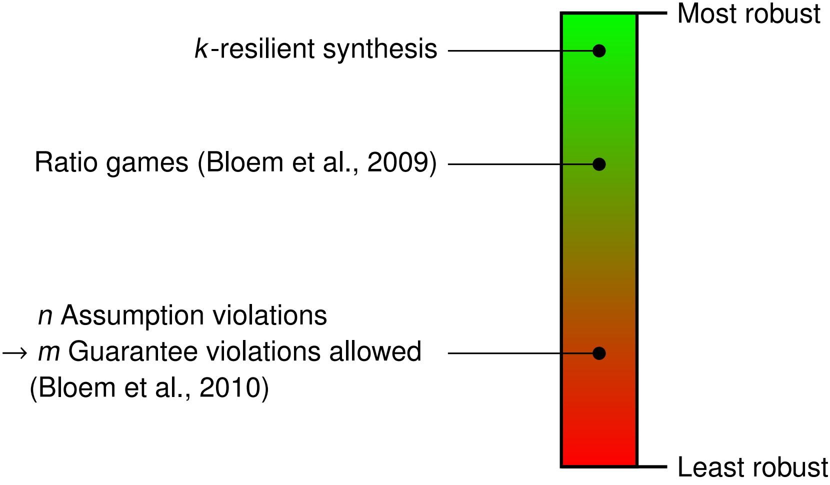


Assumptions

- **Obstacle can only move in every second step**
- Obstacle can only move by one cell per direction per step
- *x* position of the robot is updated according to its choice
- *y* position of the robot is updated according to its choice
- No robot jumps further that one cell are possible

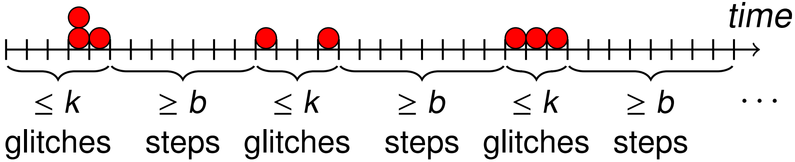
Based on joint work with Ufuk Topcu, HSCC 2014

Spectrum of robustness (in discrete synthesis)

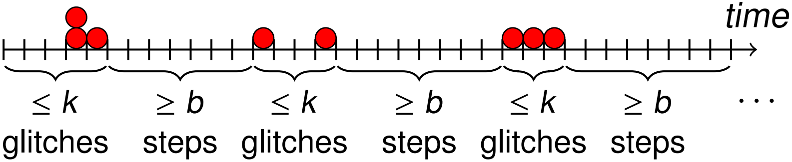


This figure is certainly not complete and only lists *some* approaches.

k-Resilient synthesis (Def.: Huang et al., 2012)



k-Resilient synthesis (Def.: Huang et al., 2012)



Algorithmic approach

Given a GR(1) specification ψ and k, we can encode the k-resilient synthesis problem of ψ by translating ψ to a modified GR(1) specification ψ' and perform GR(1) synthesis for ψ'.

The Reduction from GR(1) to GR(1) in a nutshell

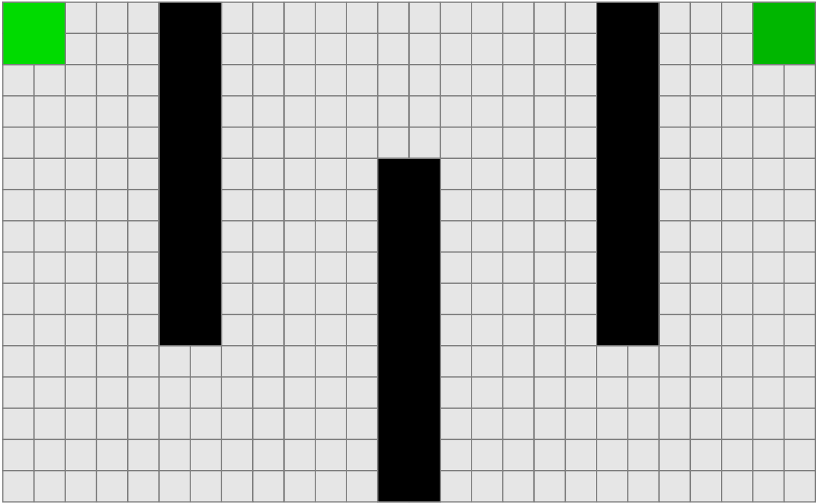
A counter for k

- We ask the system to be synthesized to output a **counter** that says how many *glitches* can be tolerated in the near future.
- If glitches stop occurring, then the system must eventually set the counter back to k.

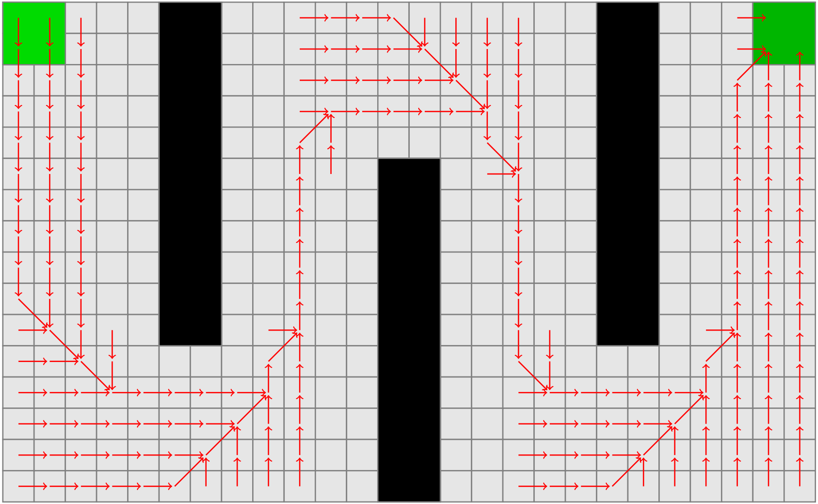
Effect

- Quantification over b is abstracted by a liveness property
- Idea can be implemented by altering the specification.

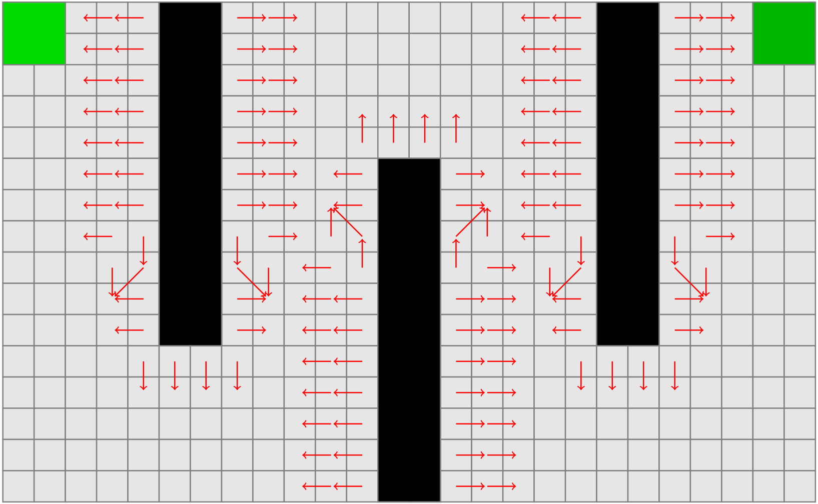
Example: Robot patrolling



Example: Robot patrolling



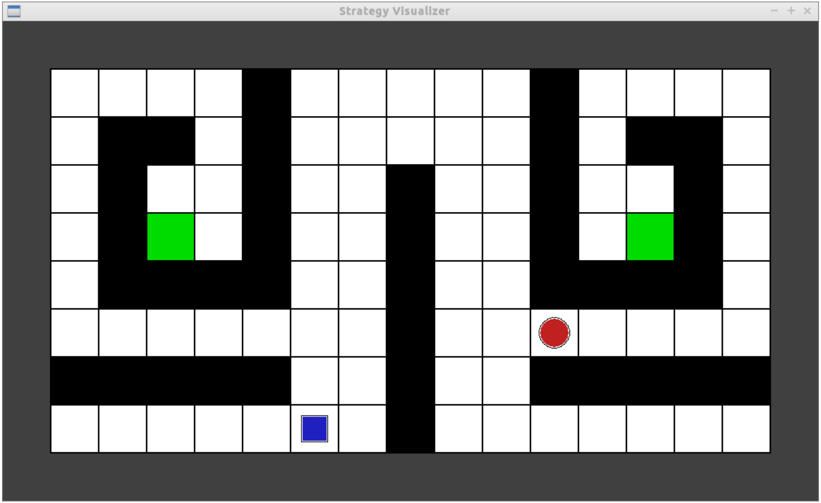
Example: Robot patrolling



Synthesizing *cooperative* controllers

Based on joint work with Roderick Bloem and Robert Könighofer, IROS 2015

Demo



Main problem

Main problem

How can we synthesize *cooperative* high-level robot controllers?

Naive approaches to solving this problem

- Require the system to satisfy the specification even if the environment is *naughty*

Main problem

Main problem

How can we synthesize *cooperative* high-level robot controllers?

Naive approaches to solving this problem

- Require the system to satisfy the specification even if the environment is *naughty* → assumptions are needed

Main problem

Main problem

How can we synthesize *cooperative* high-level robot controllers?

Naive approaches to solving this problem

- Require the system to satisfy the specification even if the environment is *naughty* → assumptions are needed
- Change the specification to prevent uncooperative behavior

Main problem

Main problem

How can we synthesize *cooperative* high-level robot controllers?

Naive approaches to solving this problem

- Require the system to satisfy the specification even if the environment is *naughty* → assumptions are needed
- Change the specification to prevent uncooperative behavior → defeats the purpose of synthesis

Main problem

Main problem

How can we synthesize *cooperative* high-level robot controllers?

Naive approaches to solving this problem

- Require the system to satisfy the specification even if the environment is *naughty* → assumptions are needed
- Change the specification to prevent uncooperative behavior → defeats the purpose of synthesis

Our approach

We modify the main generalized reactivity(1) synthesis approach to compute only *cooperative* controllers.

Making GR(1) synthesis cooperative

Synthesis objective

- All executions must be *correct*.
- From every state of the synthesized controller, there must always be an execution on which the assumptions are satisfied.

Making GR(1) synthesis cooperative

Synthesis objective

- All executions must be *correct*.
- From every state of the synthesized controller, there must always be an execution on which the assumptions are satisfied.

Implementation: Safety

- Prevent the system from issuing a next output that *forces* the environment to subsequently violate its specification

Making GR(1) synthesis cooperative

Synthesis objective

- All executions must be *correct*.
- From every state of the synthesized controller, there must always be an execution on which the assumptions are satisfied.

Implementation: Safety

- Prevent the system from issuing a next output that *forces* the environment to subsequently violate its specification

Implementation: Liveness

- Modify the GR(1) fixpoint computation

Standard GR(1) fixpoint equation

$$\nu Z. \bigwedge_{j \in \{1,\ldots,n\}} \mu Y. \bigvee_{i \in \{1,\ldots,m\}} \nu X. \mathrm{EnfPre}\Big((Z' \wedge \psi_j^g) \vee Y' \vee (\neg\psi_i^a \wedge X') \Big)$$

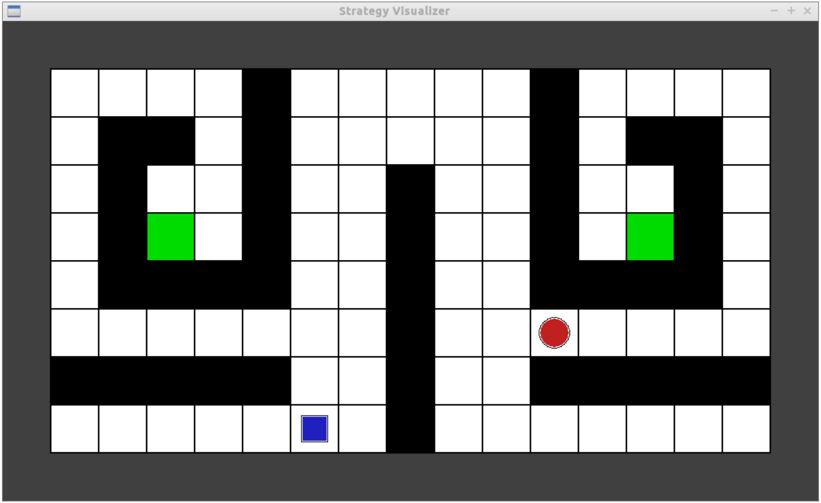
Cooperative GR(1) synthesis

$$\nu Z. \bigwedge_{j \in \{1,\ldots,n\}} \mu Y. \bigvee_{i \in \{1,\ldots,m\}} \nu X.\mathrm{EnfPre}\Big((Z' \wedge \psi_j^g) \vee Y' \vee (\neg \psi_i^a \wedge X')\Big)$$

$$\wedge\ \mu R.\mathrm{Reach}\Big((\psi_j^g \vee Y' \vee R') \wedge X\Big)$$

$$\wedge \bigwedge_{k \in \{1,\ldots,m\}} \mu R.\mathrm{Reach}\Big((\psi_k^a \vee R') \wedge Z\Big)$$

Demo



Optimal control in adversarial environments

Based on joint work with Gangyuan Jing and Hadas Kress-Gazit

(published at IROS 2013)

Adding an optimization criterion

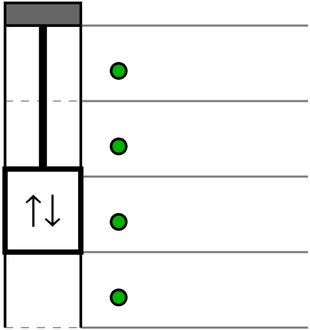
Basic idea

In addition to the specification, we introduce a *cost function*.

Adding an optimization criterion

Basic idea

In addition to the specification, we introduce a *cost function*.



Example due to Chatterjee and Henzinger (2006)

Adding an optimization criterion

Specification parts

- The door can be open or closed.
- $\forall i \in \{1, 2, 3, 4\}$: If button i is pressed, floor i is eventually visited with the door open.
- At every step, the current floor number is not increased or decreased by more than one.
- The current floor number can only be changed if the door is closed.
- The “door close” command can fail or succeed, while the “open door” command always succeeds.

Optimization criterion / cost function

- Every operation has a (fixed) cost.
- We want to minimize the average cost per execution step.

What is an optimal strategy?

Optimal strategy (mean-payoff)

- Service requests for 1 step, then wait for 1 step, then
- service requests for 1 step, then wait for 2 steps, then
- service requests for 1 step, then wait for 3 steps, then
- ...

What is an optimal strategy?

Optimal strategy (mean-payoff)

- Service requests for 1 step, then wait for 1 step, then
- service requests for 1 step, then wait for 2 steps, then
- service requests for 1 step, then wait for 3 steps, then
- ...

So what now?

We need to fix either:

- specification
- weigths/costs
- optimization objectives

What is an optimal strategy?

Optimal strategy (mean-payoff)

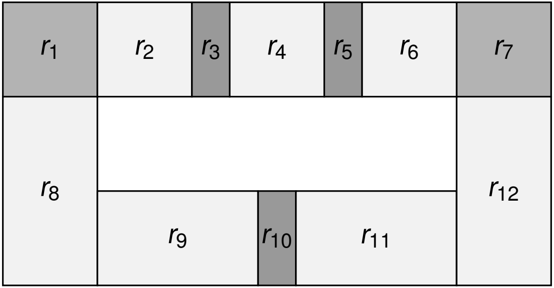
- Service requests for 1 step, then wait for 1 step, then
- service requests for 1 step, then wait for 2 steps, then
- service requests for 1 step, then wait for 3 steps, then
- ...

So what now?

We need to fix either:

- specification
- weigths/costs
- **optimization objectives**

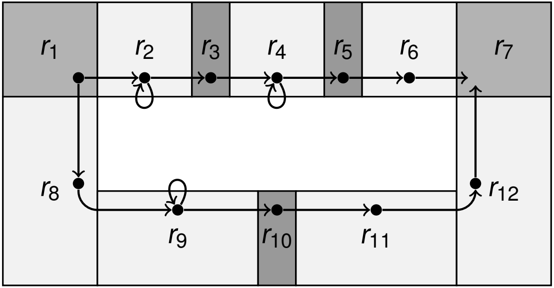
Example



Specification parts

Assumptions:	**Guarantees**	
$\mathbf{GF}(open_1)$	$\mathbf{G}(\neg open_1 \rightarrow \mathbf{X}(\neg r_3)))$	$\mathbf{F}r_7$
$\mathbf{GF}(open_2)$	$\mathbf{G}(\neg open_2 \rightarrow \mathbf{X}(\neg r_5)))$	
$\mathbf{GF}(open_3)$	$\mathbf{G}(\neg open_3 \rightarrow \mathbf{X}(\neg r_{10})))$	

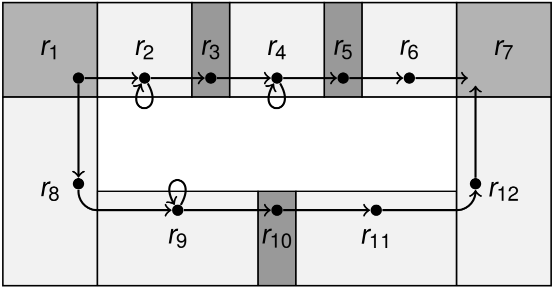
Example



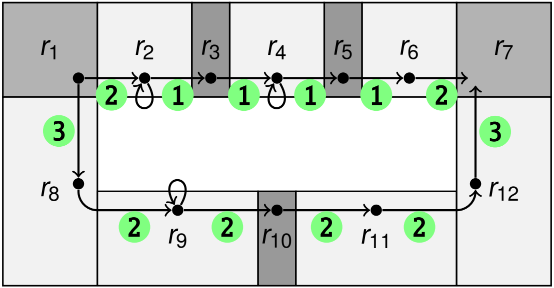
Specification parts

Assumptions:	**Guarantees**	
$\mathbf{GF}(open_1)$	$\mathbf{G}(\neg open_1 \rightarrow \mathbf{X}(\neg r_3)))$	
$\mathbf{GF}(open_2)$	$\mathbf{G}(\neg open_2 \rightarrow \mathbf{X}(\neg r_5)))$	$\mathbf{F} r_7$
$\mathbf{GF}(open_3)$	$\mathbf{G}(\neg open_3 \rightarrow \mathbf{X}(\neg r_{10})))$	

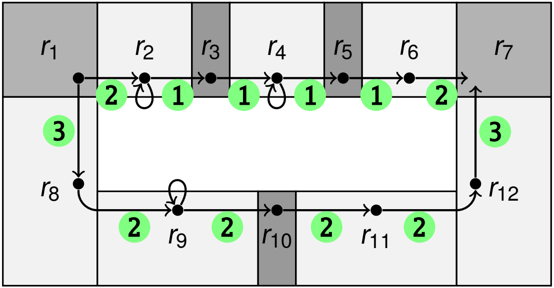
Introducing “action cost”



Introducing “action cost”



Introducing “action cost”



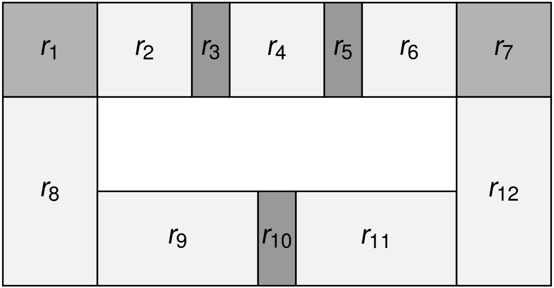
Computing *action cost* along a path

- Take the sum of costs until reaching the next goal

Characterizing waiting in strategies

Basic idea

Waiting in strategies can be detected by looking at the SCCs.



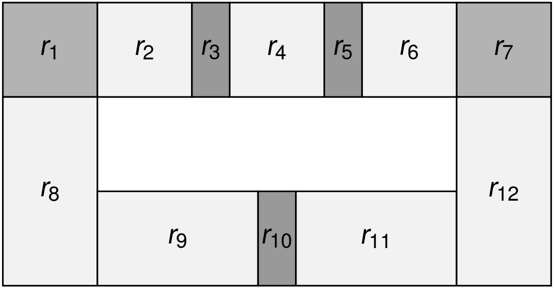
Characterizing waiting in strategies

Basic idea

Waiting in strategies can be detected by looking at the SCCs.

But what about repetitive tasks?

Here, we count SCCs up to the point of *reaching the next goal*.



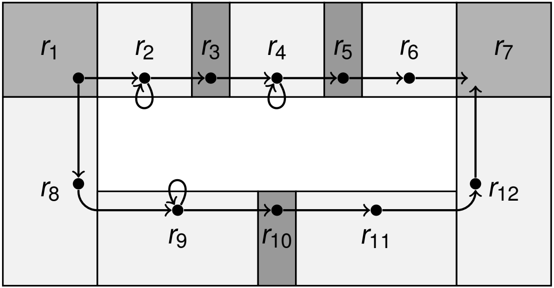
Characterizing waiting in strategies

Basic idea

Waiting in strategies can be detected by looking at the SCCs.

But what about repetitive tasks?

Here, we count SCCs up to the point of *reaching the next goal*.



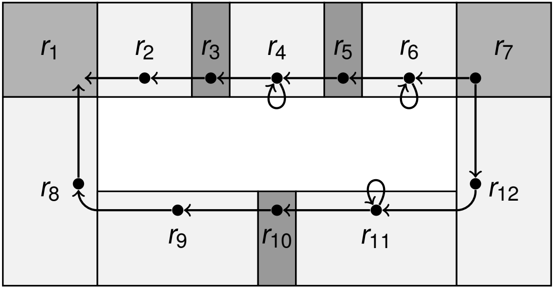
Characterizing waiting in strategies

Basic idea

Waiting in strategies can be detected by looking at the SCCs.

But what about repetitive tasks?

Here, we count SCCs up to the point of *reaching the next goal*.



Conditions on the specification

Idea

- If the specification has a set of *goals* for the system, we can count the number of waiting cycles for reaching the respective *next* goal.
- After reaching the next goal, the counter resets.
- It is the aim of the system to reach the *next* goal cheaply.

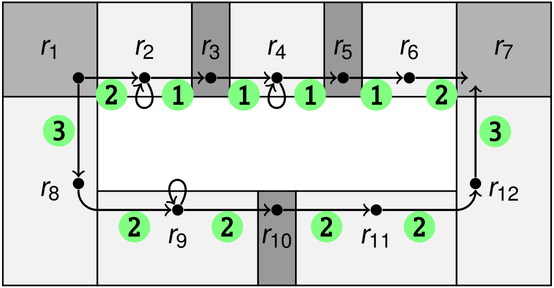
Using the idea for GR(1) specifications

- Still singly-exponential complexity
- Strategy shape is the same as for standard GR(1) synthesis: positional-per-goal

Combining action and waiting cost

Two-dimensional cost notion

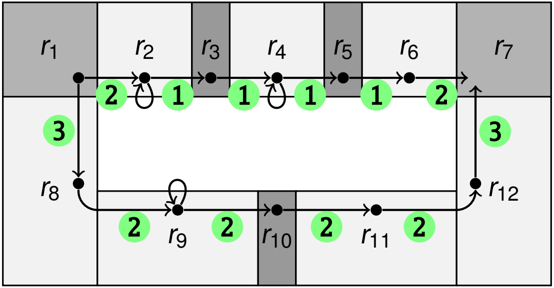
From every state of a strategy, a strategy has a *cost tuple* that describes the cost to reach the next goal.



Combining action and waiting cost

Two-dimensional cost notion

From every state of a strategy, a strategy has a *cost tuple* that describes the cost to reach the next goal.



Example

Cost of path 1 from r_1: $(c_w, c_a) = (2, 8)$, cost of path 2: $(c_w, c_a) = (1, 14)$
Overall cost: depends on the **preference**

Preference relations

Preference relations

- We use a preference relation $\leq_P$ to choose which executions the strategy should prefer (e.g., $(1, 14) \leq_P (2, 8)$).
- The value of a strategy from some state is the **highest** combined cost (w.r.t. $\leq_P$) of all executions originating from the state.

Preference relations

Preference relations

- We use a preference relation $\leq_P$ to choose which executions the strategy should prefer (e.g., $(1, 14) \leq_P (2, 8)$).
- The value of a strategy from some state is the **highest** combined cost (w.r.t. $\leq_P$) of all executions originating from the state.

Using the idea for GR(1) specifications

For almost-linear preference relations, we can still compute optimal strategies in exponential time. They track:

- The current atomic proposition values
- The “current” liveness assumption **and** liveness guarantee
- Whether ∞ action cost can still be avoided

Risk-Averse Control of Markov Decision Processes with ω-regular Objectives

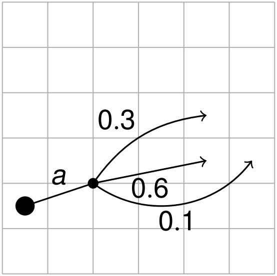
Based on joint work with Salar Moarref and Ufuk Topcu

(published at CDC 2016)

Basic problem setup

Basic Problem

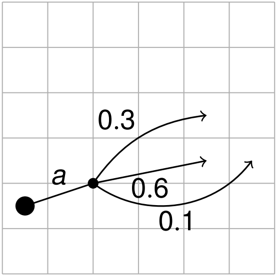
- We want to control a Markov decision process (MDP) such that an ω-regular specification is satisfied...



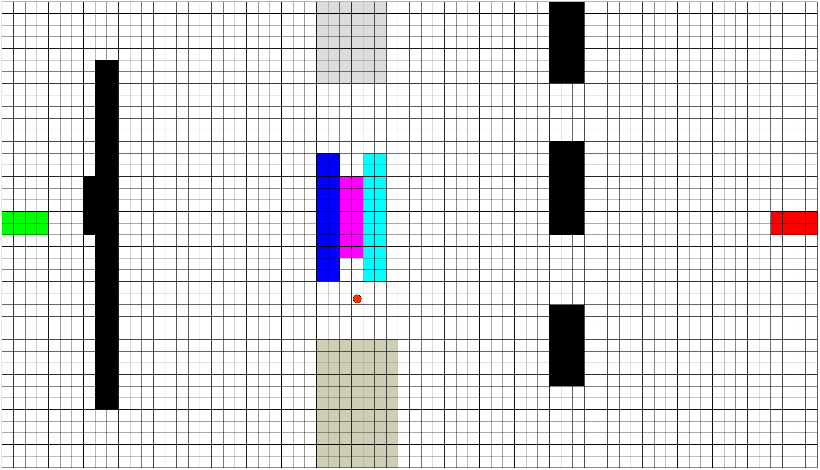
Basic problem setup

Basic Problem

- We want to control a Markov decision process (MDP) such that an ω-regular specification is satisfied...
- ...but we want to do this in MDPs in which all policies have a probability of 0 for satisfying the specification.



Example problem



So how can we control the system?

Idea

We compute a controller that maximizes the probability to reach the next *goal*.

From every goal (or initially), A *p-risk averse controller* reaches the next goal with probability at least *p*.

So how can we control the system?

Idea

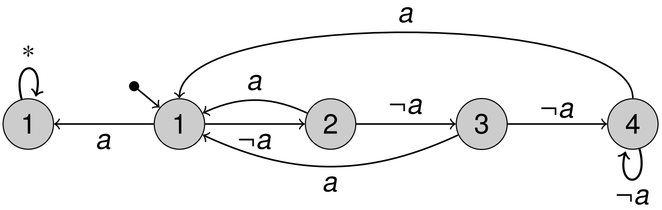
We compute a controller that maximizes the probability to reach the next *goal*.

From every goal (or initially), A *p-risk averse controller* reaches the next goal with probability at least *p*.

But what *is* the next goal?

When working with general ω-regular specifications, this is not so easy to tell!

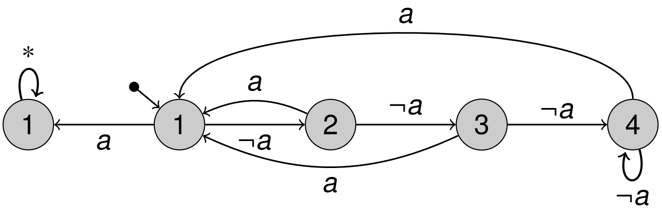
Deterministic Parity Automata



Acceptance

A deterministic parity automaton accepts all words that have a run on which the highest color occurring infinitely often is even.

Deterministic Parity Automata



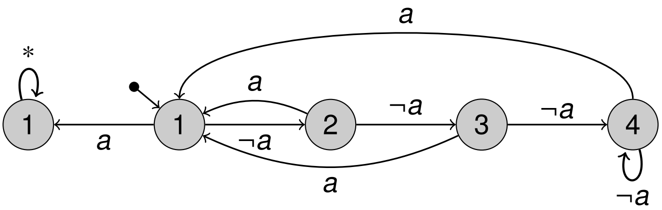
Acceptance

A deterministic parity automaton accepts all words that have a run on which the highest color occurring infinitely often is even.

Example (1)

$$\begin{array}{lcccccc} \vec{c} & = & 1 & 2 & 1 & 2 & \dots \\ w & = & \overline{a} & a & \overline{a} & a & \dots \end{array}$$

Deterministic Parity Automata



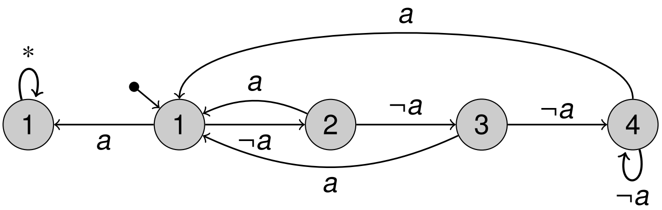
Acceptance

A deterministic parity automaton accepts all words that have a run on which the highest color occurring infinitely often is even.

Example (2)

$$\begin{array}{lllllll} \vec{c} & = & 1 & 2 & 1 & 1 & \ldots \\ w & = & \overline{a} & a & a & \overline{a} & \ldots \end{array}$$

Deterministic Parity Automata



Acceptance

A deterministic parity automaton accepts all words that have a run on which the highest color occurring infinitely often is even.

Example (3)

$$\begin{array}{ccccccccc} \vec{c} & = & 1 & 2 & 3 & 4 & 1 & 2 & \ldots \\ w & = & \overline{a} & \overline{a} & \overline{a} & a & \overline{a} & \overline{a} & \ldots \end{array}$$

Connecting Deterministic Parity Automata and MDP Control

Basic idea

We let the *controller* always tell the *current goal color* and when it just reached a goal.

Connecting Deterministic Parity Automata and MDP Control

Basic idea

We let the *controller* always tell the *current goal color* and when it just reached a goal.

The controller may always *increase* the goal color, but *decrease* it only finitely a fixed number of times.

Connecting Deterministic Parity Automata and MDP Control

Basic idea

We let the *controller* always tell the *current goal color* and when it just reached a goal.

The controller may always *increase* the goal color, but *decrease* it only finitely a fixed number of times.

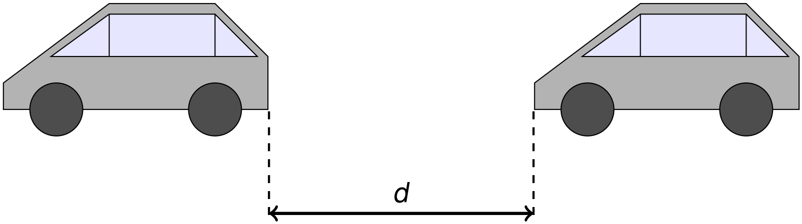
Finding p-risk averse policies

- For every $p \in [0, \ldots, 1]$, a p-risk averse control policy has a finite number of states
- Optimal strategies can be computed by solving a series of optimal reachability policy computations in MDPs.

Estimator-based synthesis

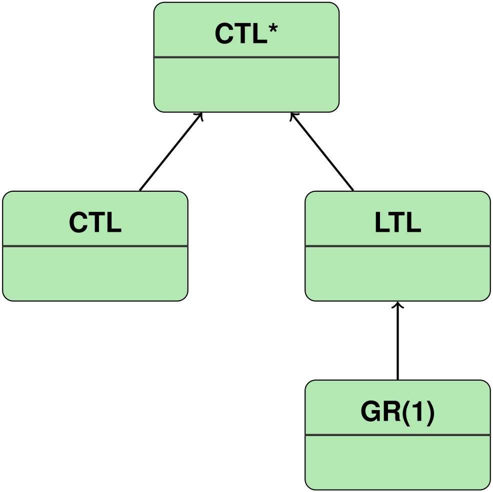
Based on joint work with Ufuk Topcu, HSCC 2015

Example application: Distance keeping assistant

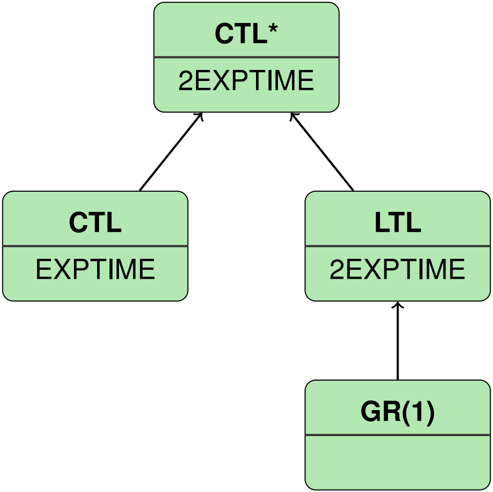


Observable:	Speed of the follower car
Noisily Measured:	Distance between cars
Unobserved:	Speed of the leader car
Controlled:	Acceleration (follower)

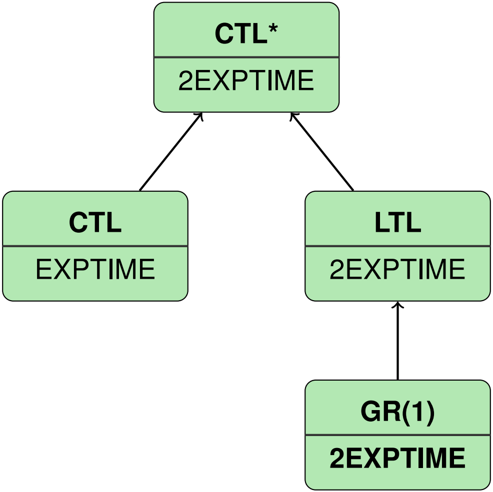
Synthesis – complexity vs. expressivity (incomplete inf.)



Synthesis – complexity vs. expressivity (incomplete inf.)



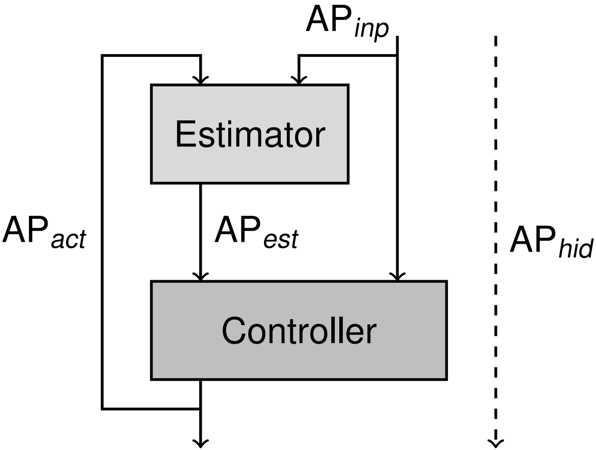
Synthesis – complexity vs. expressivity (incomplete inf.)



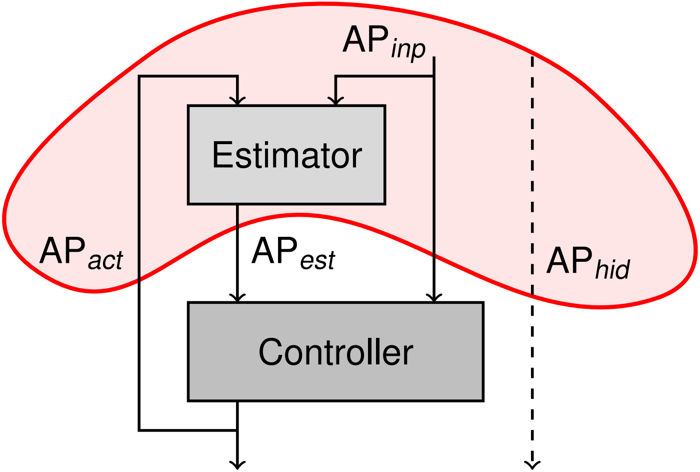
Synthesis with estimators (1/2)

Central question:
How to retain the
singly-exponential complexity of
GR(1) synthesis
under incomplete information?

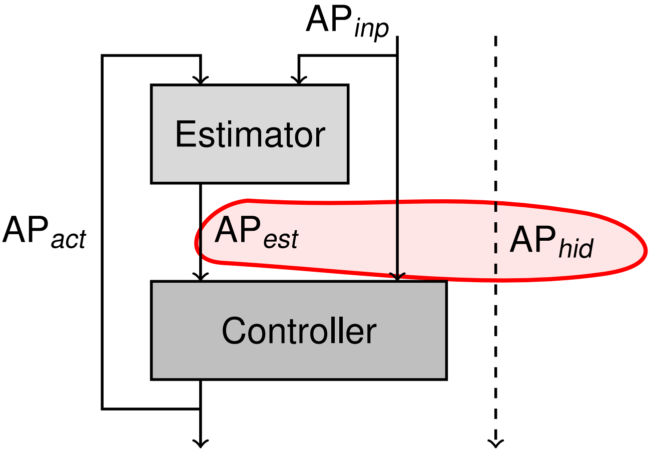
Synthesis with estimators (2/2)



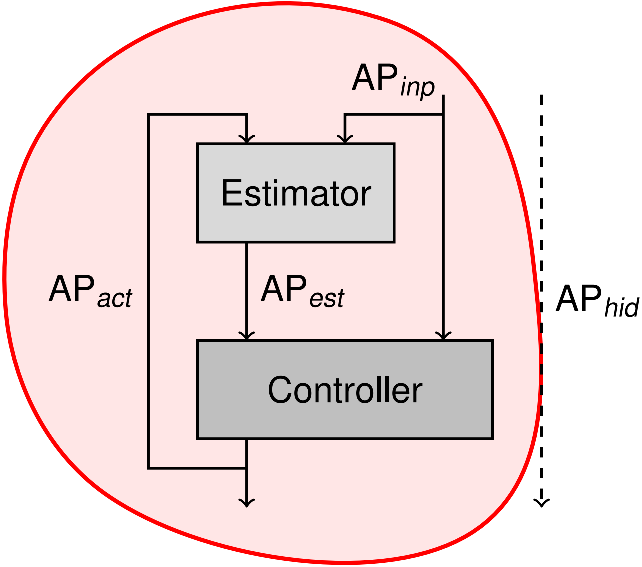
Synthesis with estimators (2/2)



Synthesis with estimators (2/2)



Synthesis with estimators (2/2)



Decoupling estimator computation from synthesis

Main ideas

- We decouple the **estimation** of the physical values and **reactive synthesis** using an **estimator specification** as glue.
- We modify the controller specification to only talk about observable variables.

Decoupling estimator computation from synthesis

Main ideas

- We decouple the **estimation** of the physical values and **reactive synthesis** using an **estimator specification** as glue.
- We modify the controller specification to only talk about observable variables.

Example specification parts

Estimator specification	Controller Specification
$\mathbf{G}(minDistance \leq distance)$ $\mathbf{G}(maxDistance \geq distance)$	$\mathbf{G}(minDistance \geq 5)$

Main correctness proposition

We can soundly reduce the synthesis problem for φ under incomplete information to one over φ' over complete information if

$$(\varphi' \;\wedge\; \mathbf{G}\,\rho_e \;\wedge\; \mathbf{G}\,\rho_s) \;\rightarrow\; \varphi$$

holds.

Modified System Specification

Main correctness proposition

We can soundly reduce the synthesis problem for φ under incomplete information to one over φ' over complete information if

$$(\varphi' \;\wedge\; \mathbf{G}\,\rho_e \;\wedge\; \mathbf{G}\,\rho_s) \;\rightarrow\; \varphi$$

holds.

Environment Assumptions

Modified System Specification

Main correctness proposition

We can soundly reduce the synthesis problem for φ under incomplete information to one over φ' over complete information if

$$(\varphi' \wedge \mathbf{G}\,\rho_e \wedge \mathbf{G}\,\rho_s) \rightarrow \varphi$$

holds.

Environment Assumptions

Modified System Specification

Estimator Specification

Main correctness proposition

We can soundly reduce the synthesis problem for φ under incomplete information to one over φ' over complete information if

$$(\varphi' \;\wedge\; \mathbf{G}\,\rho_e \;\wedge\; \mathbf{G}\,\rho_s) \;\rightarrow\; \varphi$$

holds.

Environment Assumptions

Original System Specification

Modified System Specification

Estimator Specification

Scalable estimator computation

Problem

In general, estimator computation is still a doubly exponential problem

Scalable estimator computation

Our solution

We only consider **positional estimators**. These may only base their next estimates on:

- the last sensor values and the last estimates
- the current sensor values
- the possible evolutions of the environment

Properties of our approach

- fixed size of the estimators
- no “strategic planning” possible by the estimators
- unique optimal estimators exist for most estimate preference relations

Example: Discretized car following controller (1)

Properties

- $\mathit{distance} \in \{0, \ldots, 84, 85\}$
- $\mathit{speedLeader} \in \{0, \ldots, 15\}$
- $\mathit{speedFollower} \in \{0, \ldots, 15\}$
- $\mathit{accelerationLeader} \in \{-2, -1, 0, 1, 2\}$
- $\mathit{accelerationFollower} \in \{-2, -1, 0, 1, 2\}$
- Noisy distance update
- Approximate (± 2) distance measurement
- …

Specification parts

- The *distance* must always be at least 5.
- $\mathbf{G}(\mathit{distance} < 85 \vee \mathit{speedFollower} = 15)$

Example: Discretized car following controller (2)

$$\cdots \longrightarrow \left(\begin{array}{c} 84 \\ 14 \\ \hline 4 \\ 13 \\ 15 \\ 83 \\ 85 \end{array}\right) \longrightarrow \left(\begin{array}{c} 78 \\ 15 \\ \hline 0 \\ 11 \\ 12 \\ \\ \\ \end{array}\right)$$

Meaning of the encoding : $\left(\begin{array}{c} \mathit{observedDistance} \\ \mathit{speedFollower} \\ \hline \mathit{accFollower} \\ \mathit{minSpeedLeader} \\ \mathit{maxSpeedLeader} \\ \mathit{minDist} \\ \mathit{maxDist} \end{array}\right)$

Example: Discretized car following controller (2)

$$\cdots \longrightarrow \left(\begin{array}{c}84\\14\\\hline 4\\13\\15\\83\\85\end{array}\right) \longrightarrow \left(\begin{array}{c}78\\15\\\hline 0\\11\\12\\\textcolor{red}{80}\\\textcolor{red}{80}\end{array}\right)$$

Meaning of the encoding : $\left(\begin{array}{c}\mathit{observedDistance}\\\mathit{speedFollower}\\\hline \mathit{accFollower}\\\mathit{minSpeedLeader}\\\mathit{maxSpeedLeader}\\\mathit{minDist}\\\mathit{maxDist}\end{array}\right)$

Example: Discretized car following controller (3)

$$\cdots \longrightarrow \begin{pmatrix} 84 \\ 15 \\ \hline 2 \\ 13 \\ 15 \\ 82 \\ 83 \end{pmatrix} \rightarrow \begin{pmatrix} 84 \\ 15 \\ \hline 2 \\ 12 \\ 15 \\ 82 \\ 84 \end{pmatrix} \rightarrow \begin{pmatrix} 85 \\ 15 \\ \hline 2 \\ 12 \\ 15 \\ 83 \\ 85 \end{pmatrix} \rightarrow \begin{pmatrix} 85 \\ 15 \\ \hline 2 \\ 10 \\ 15 \\ 83 \\ 85 \end{pmatrix} \rightarrow \begin{pmatrix} 85 \\ 15 \\ \hline 2 \\ 8 \\ 15 \\ 83 \\ 85 \end{pmatrix} \rightarrow \begin{pmatrix} 85 \\ 15 \\ \hline 2 \\ 6 \\ 15 \\ 83 \\ 85 \end{pmatrix} \rightarrow \begin{pmatrix} 85 \\ 15 \\ \hline 2 \\ 4 \\ 15 \\ 83 \\ 85 \end{pmatrix} \rightarrow \begin{pmatrix} 85 \\ 15 \\ \hline 2 \\ 2 \\ 15 \\ 83 \\ 85 \end{pmatrix} \longrightarrow \cdots$$

Meaning of the encoding : $\begin{pmatrix} \mathit{observedDistance} \\ \mathit{speedFollower} \\ \hline \mathit{accFollower} \\ \mathit{minSpeedLeader} \\ \mathit{maxSpeedLeader} \\ \mathit{minDist} \\ \mathit{maxDist} \end{pmatrix}$

Computing positional estimators

1. Compute the reachable states of *any* estimator

$$R = \mu X.(\{x_0\} \cup \{x' \subseteq AP_{obs} \cup AP_{hid} \cup AP_{est} \mid \exists x \in X.$$
$$(x \setminus AP_{est}, x' \setminus AP_{est}) \in \rho_e, (x, x') \in \rho_s\})$$

2. Compute which estimates are admissible

$$\rho_u = \{(x, x') \in (2^{AP_{obs} \cup AP_{est}})^2 \mid \forall y, y' \subseteq 2^{AP_{hid}} :$$
$$((x \cup y) \in R \wedge ((x \setminus AP_{est} \cup y),$$
$$(x' \setminus AP_{est} \cup y'))) \in \rho_e \rightarrow ((x \cup y), (x' \cup y')) \in \rho_s\}.$$

3. Restriction to optimal estimates

$$\hat{\rho}_u = \{(x, x') \in \rho_u : x'|_{AP_{est}} = \min\{x''|_{AP_{est}} : (x, x'') \in \rho_u,$$
$$x' \setminus AP_{est} = x'' \setminus AP_{est}\}\}$$

Discretized car following controller (cont'd)

Computation times with `slugs` (BDD-based)

- Basic scenario: 22+28 minutes
- *Cruise mode scenario:* 22+460 minutes (6 realizability checks)

Without estimator-based synthesis

Belief space: $2^{16 \cdot 86}$ states – **beyond tractability**

Conclusion

“Spicing up CPS controller synthesis”

In this talk...

...we discussed a few approaches to make the concept of *reactive synthesis* more applicable to CPS controller computation.

“Spicing up CPS controller synthesis”

In this talk...

...we discussed a few approaches to make the concept of *reactive synthesis* more applicable to CPS controller computation.

But can they be combined?

Apart from the *error-resilient synthesis* part, they all require modifications of the synthesis process. → **So no!**

“Spicing up CPS controller synthesis”

In this talk...

...we discussed a few approaches to make the concept of *reactive synthesis* more applicable to CPS controller computation.

But can they be combined?

Apart from the *error-resilient synthesis* part, they all require modifications of the synthesis process. → **So no!**

Ok, so what now?

We will need to research methods to combine the considerations presented in this talk in a larger framework that still allows for scalable synthesis!

References I

Massimo Benerecetti, Marco Faella, and Stefano Minopoli. Automatic synthesis of switching controllers for linear hybrid systems: Safety control. *Theor. Comput. Sci.*, 493:116–138, 2013. doi: 10.1016/j.tcs.2012.10.042. URL `http://dx.doi.org/10.1016/j.tcs.2012.10.042`.

Roderick Bloem, Stefan Galler, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Martin Weiglhofer. Interactive presentation: Automatic hardware synthesis from specifications: a case study. In *DATE*, pages 1188–1193, 2007a. doi: 10.1145/1266366.1266622.

Roderick Bloem, Stefan J. Galler, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Martin Weiglhofer. Specify, compile, run: Hardware from PSL. *Electr. Notes Theor. Comput. Sci.*, 190(4):3–16, 2007b. doi: 10.1016/j.entcs.2007.09.004. URL `http://dx.doi.org/10.1016/j.entcs.2007.09.004`.

Roderick Bloem, Karin Greimel, Thomas A. Henzinger, and Barbara Jobstmann. Synthesizing robust systems. In *Proceedings of 9th International Conference on Formal Methods in Computer-Aided Design, FMCAD 2009, 15-18 November 2009, Austin, Texas, USA*, pages 85–92, 2009. doi: 10.1109/FMCAD.2009.5351139. URL `http://dx.doi.org/10.1109/FMCAD.2009.5351139`.

Roderick Bloem, Krishnendu Chatterjee, Karin Greimel, Thomas A. Henzinger, and Barbara Jobstmann. Robustness in the presence of liveness. In *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, pages 410–424, 2010. doi: 10.1007/978-3-642-14295-6_36. URL `http://dx.doi.org/10.1007/978-3-642-14295-6_36`.

Krishnendu Chatterjee and Thomas A. Henzinger. Finitary winning in omega-regular games. In *Tools and Algorithms for the Construction and Analysis of Systems, 12th International Conference, TACAS 2006 Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 25 - April 2, 2006, Proceedings*, pages 257–271, 2006. doi: 10.1007/11691372_17. URL `http://dx.doi.org/10.1007/11691372_17`.

Yashdeep Godhal, Krishnendu Chatterjee, and Thomas A. Henzinger. Synthesis of AMBA AHB from formal specification: a case study. *International Journal on Software Tools for Technology Transfer*, 2011. doi: 10.1007/s10009-011-0207-9.

Chung-Hao Huang, Doron A. Peled, Sven Schewe, and Farn Wang. Rapid recovery for systems with scarce faults. In *Proceedings Third International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2012, Napoli, Italy, September 6-8, 2012.*, pages 15–28, 2012. doi: 10.4204/EPTCS.96.2. URL `http://dx.doi.org/10.4204/EPTCS.96.2`.

References II

Gangyuan Jing, Rüdiger Ehlers, and Hadas Kress-Gazit. Shortcut through an evil door: Optimality of correct-by-construction controllers in adversarial environments. In *IROS*, pages 4796–4802. IEEE, 2013.

Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25:1370–1381, 2009.

Petter Nilsson, Omar Hussien, Ayca Balkan, Yuxiao Chen, Aaron D. Ames, Jessy W. Grizzle, Necmiye Ozay, Huei Peng, and Paulo Tabuada. Correct-by-construction adaptive cruise control: Two approaches. *IEEE Trans. Contr. Sys. Techn.*, 24(4):1294–1307, 2016. doi: 10.1109/TCST.2015.2501351. URL http://dx.doi.org/10.1109/TCST.2015.2501351.

Necmiye Ozay, Ufuk Topcu, and Richard M. Murray. Distributed power allocation for vehicle management systems. In *Proceedings of the 50th IEEE Conference on Decision and Control and European Control Conference, CDC-ECC 2011, Orlando, FL, USA, December 12-15, 2011*, pages 4841–4848, 2011a. doi: 10.1109/CDC.2011.6161470. URL http://dx.doi.org/10.1109/CDC.2011.6161470.

Necmiye Ozay, Ufuk Topcu, Richard M. Murray, and Tichakorn Wongpiromsarn. Distributed synthesis of control protocols for smart camera networks. In *IEEE/ACM Second International Conference on Cyber-Physical Systems (ICCPS)*, pages 45–54, Washington, DC, USA, 2011b. IEEE Computer Society. doi: 10.1109/ICCPS.2011.22.

Ivan Papusha, Jie Fu, Ufuk Topcu, and Richard M. Murray. Automata theory meets approximate dynamic programming: Optimal control with temporal logic constraints. In *CDC*, 2016. to appear.

Vasumathi Raman, Nir Piterman, and Hadas Kress-Gazit. Provably correct continuous control for high-level robot behaviors with actions of arbitrary execution durations. In *ICRA*. IEEE, 2013.