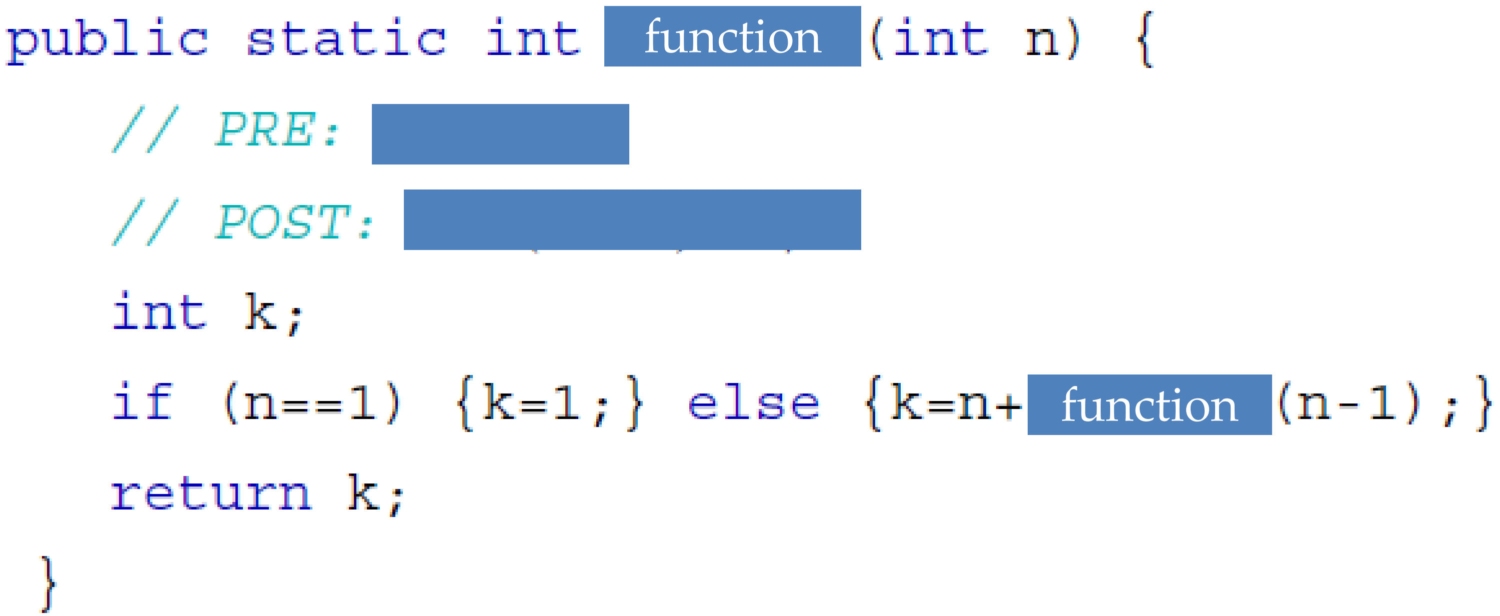
Reactive Synthesis

Nir Piterman
University of Gothenburg
Aalborg, June 13, 2022

UNIVERSITY OF
GOTHENBURG

European Research Council
Established by the European Commission

Programming



```
public static int function(int n) {
    // PRE: [redacted]
    // POST: [redacted]
    int k;
    if (n==1) {k=1;} else {k=n+function(n-1);}
    return k;
 }
```

A function defines a relation between inputs and outputs.

Doesn't quite work ...

Computation vs. Reactivity

Computational Programs: Run in order to produce a final result on termination.

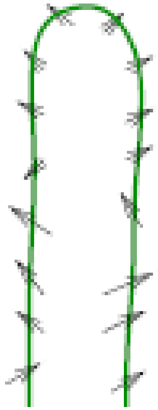
Can be modeled as a black box.

Specified in terms of Input/Output relations.

Reactive Programs

Programs whose role is to maintain an ongoing interaction with their environments.

Can be viewed as a green cactus (?)



Reactive Systems

- Systems whose main aim is to interact rather than compute (OS, driver, CPU, car controller).
- Main complexity is in maintaining communication with a user / another program / the environment.
- Reactive systems are notoriously hard to design.
- Major efforts are invested in development and validation of reactive systems.

The Requirement Language

- Correctness of computational programs is expressed as Hoare triples.

$$\{P\}C\{Q\}$$

- Correctness of reactive programs is expressed as behavioral specifications:
  - The behavior of a system is a sequence of system states.
  - Specification should tell us when a sequence is good/bad.
  - We use temporal logic: connect states through time.

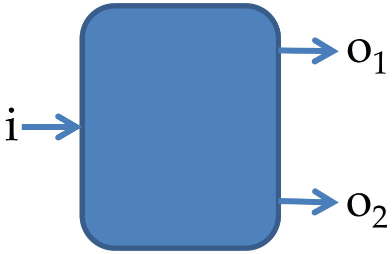
Validating Reactive Systems

- Simulations:
  - Run the system and check whether behavior satisfies specifications.
- Model checking:
  - Create a comprehensive model of the system and check whether all behaviors satisfy specifications.
- Model checking research:
  - Automatic construction of models.
    - Predicate extraction.
    - Heap analysis.
    - Counter-example guided abstraction refinement.
  - Techniques for model exploration.
    - Efficient enumerative graph exploration.
    - Symbolic representation of states.
    - Bounded model checking and SAT/SMT solving.
  - Specification.
    - Expressive specification languages.
    - Translation to model exploration.

Synthesis

- Developing systems is hard, expensive, and error prone.
- The common solution is extensive testing and verification.
- If we can verify, why not go directly from specification to correct-by-construction systems by synthesis?
- Church's synthesis problem:
  Given a circuit interface specification and a behavioral specification:
  - Determine if there is an automaton that realizes the specification.
  - If the specification is realizable, construct an implementing automaton.
- Circuit interface – partition to inputs and outputs.
- Behavioral specification – description in first order logic.

Synthesis from Temporal Specifications



$$\forall t.\neg o_1(t) \vee \neg o_2(t)$$
$$\forall t.i(t) \rightarrow \left(\exists t' > t.o_1(t) \vee o_2(t)\right)$$
$$\forall t.o_1(t) \rightarrow \left(\exists t' < t.\left(i(t') \wedge \forall t' < t'' < t.\left(\neg o_1(t'') \wedge \neg o_2(t'')\right)\right)\right)$$
$$\forall t.o_2(t) \rightarrow \left(\exists t' < t.\left(i(t') \wedge \forall t' < t'' < t.\left(\neg o_1(t'') \wedge \neg o_2(t'')\right)\right)\right)$$
$$\forall t.o_1(t) \rightarrow \left(\forall t' > t.\left(\neg\left(o_1(t') \vee \exists t < t'' < t'.o_2(t'')\right)\right)\right)$$
$$\forall t.o_1(t) \rightarrow \left(\forall t' > t.\left(\neg\left(o_2(t') \vee \exists t < t'' < t'.o_1(t'')\right)\right)\right)$$

- Is it possible to realize this specification?
- The formula defines a relation between $\mathrm{i}: \mathbb{N} \rightarrow \{0,1\}$ and $\mathrm{o}_1, \mathrm{o}_2: \mathbb{N} \rightarrow \{0,1\}$
- We want a function that is a subset.

Causal

$$o(0) \leftrightarrow (\exists t.\, i(t))$$

- The relation $R = \{(i, o) | i\colon \mathbb{N} \to \{0,1\}, o\colon \mathbb{N} \to \{0,1\}, o(0) \leftrightarrow (\exists t.\, i(t))\}$ is not empty.
- Find a function that implements it.
- The function cannot be clairvoyant.
- It needs to be causal: $\mathrm{o(n)} = \mathrm{f}\left(\mathrm{i}\, |_{\{0,\dots,n\}}\right)$

Adversarial

$$\forall t.\, i(t) \rightarrow \neg o(t)$$
$$\forall t.\, i(t) \rightarrow \exists t' > t.\, o(t')$$

- There are some input sequences for which this is possible.
- But not all!
- We want a function that can answer all input sequences.

$$f : \{\, i : \{0, \ldots, n\} \rightarrow \{0,1\} \mid n \in \mathbb{N} \,\} \rightarrow \{0,1\}$$

- Furthermore, for every $\mathrm{i} : \mathbb{N} \rightarrow \{0,1\}$ the unique $\mathrm{o} : \mathbb{N} \rightarrow \{0,1\}$ such that $\mathrm{o(n)} = \mathrm{f}(\mathrm{i}\,|_{\{0,\ldots,n\}})$ for every $n \in \mathbb{N}$ satisfies the specification.

Brief History

- Church's problem [1965].
- Rabin introduces automata on infinite trees. Effectively, generalizing Büchi's work on ω-automata to trees [1969].
- Büchi and Landweber define two-player games of infinite duration [1969].
- We now know that the two are effectively the same. These are still the techniques we use to solve the problem.

Modern Times

- Pnueli introduces linear temporal logic [1977].
- Emerson and Clarke and Quielle and Sifakis invent model checking [1981].
- Emerson and Clarke and Manna and Wolper ignore adversarial nature and propose reduction to satisfiability [1984].
- Pnueli and Rosner establish LTL realizability to be 2EXPTIME-complete.
  - This result established realizability and synthesis as highly intractable.

In these Lectures

- Synthesis as a game.
- Simple games (safety, reachability, Büchi).
- LTL Synthesis reduced to solution of parity games.
- Bypassing determinization:
  - Safraless approach.
  - Restricting the specification langauge.
  - Usage of synthesis in robotics.
- Current research direcions:
  - Distributed synthesis.
  - Safety of learned behaviour.
  - Strategic reasoning.

Lectures Outline

- Introduction
- Automata and Linear Temporal Logic
- Games and Synthesis
- General LTL Synthesis
- Bypassing Determinization
- Current Research Directions

A More Formal Context

- A specification in linear temporal logic over input and output propositions.
- A system will be an automaton with output.
- Input and output are combined to create a sequences of assignments to propositions.
- All possible infinite paths over the automaton should satisfy the specification.

Linear Temporal Logic

- A set of propositions ($\mathcal{Prop}$) denoting the basic facts about the world. Set $\mathcal{Prop}$ is partitioned to inputs $\mathcal{I}$ and outputs $\mathcal{O}$.
- Linear Temporal Logic formulae are constructed as follows:

$$\varphi ::= p \,\|\, \varphi \wedge \varphi \,\|\, \neg\varphi \,\|\, \bigcirc\varphi \,\|\, \ominus\varphi \,\|\, \varphi\mathcal{U}\varphi \,\|\, \varphi\mathcal{S}\varphi$$

- Other temporal formulae are derived:
  - $\Diamond\varphi \equiv \mathtt{T}\mathcal{U}\varphi$ – Eventually.
  - $\Box\varphi \equiv \neg\Diamond\neg\varphi$ – Always.
  - $\varphi\mathcal{W}\psi \equiv \varphi\mathcal{U}\psi \vee \Box\varphi$ – Weak Until.
  - $\diamondminus\varphi \equiv \mathtt{T}\mathcal{S}\varphi$ – Previously.
  - $\boxminus\varphi \equiv \neg\diamondminus\neg\varphi$ – Historically.
  - $\varphi\mathcal{B}\psi \equiv \varphi\mathcal{S}\psi \vee \boxminus\varphi$ – BackTo.

LTL Semantics

- A model for an LTL formula φ is an infinite sequence $\sigma = \sigma_0, \sigma_1, \ldots$ with a designated location $j \geq 0$.
- Each letter σ_i is a set of propositions true at time i.
- Formula φ holds over sequence σ in location $i \geq 0$, denoted $(\sigma, i) \vDash \varphi$, if:
  - If φ is a proposition $(\sigma, i) \vDash \varphi \Leftrightarrow \varphi \in \sigma_i$
  - $(\sigma, i) \vDash \neg\varphi \Leftrightarrow (\sigma, i) \nvDash \varphi$
  - $(\sigma, i) \vDash \varphi_1 \vee \varphi_2 \Leftrightarrow (\sigma, i) \vDash \varphi_1$ or $(\sigma, i) \vDash \varphi_2$
  - $(\sigma, i) \vDash \bigcirc\varphi \Leftrightarrow (\sigma, i+1) \vDash \varphi$
  - $(\sigma, i) \vDash \ominus\varphi \Leftrightarrow i > 0$ and $(\sigma, i-1) \vDash \varphi$
  - $(\sigma, i) \vDash \varphi_1 U \varphi_2 \Leftrightarrow \exists k \geq i. (\sigma, k) \vDash \varphi_2$ and $\forall i \leq j < k.\ (\sigma, j) \vDash \varphi_1$
  - $(\sigma, i) \vDash \varphi_1 S \varphi_2 \Leftrightarrow \exists k \leq i. (\sigma, k) \vDash \varphi_2$ and $\forall i \geq j > k.\ (\sigma, j) \vDash \varphi_1$
- Derived:
  - $(\sigma, i) \vDash \Diamond\varphi \Leftrightarrow \exists k \geq i\ (\sigma, k) \vDash \varphi$
  - $(\sigma, i) \vDash \Box\varphi \Leftrightarrow \forall k \geq i. (\sigma, k) \vDash \varphi$

LTL Exercises

$$\Box p$$

$$\Box\Diamond p$$

$$\Box(p \rightarrow \bigcirc(q\,\mathcal{U}\,r))$$

$$\Box(p \rightarrow p\,\mathcal{W}\,q) \overset{?}{\equiv} \Box(p \rightarrow (\bigcirc p \vee \bigcirc q))$$

$$p \overset{?}{\equiv} \Box(\ominus\mathtt{T} \vee p)$$

$$\Box(p \rightarrow \diamondminus q)$$

$$\Box(p \rightarrow \ominus(\neg p\,\mathcal{S}\,q))$$

$$\diamondminus(\neg\ominus\mathtt{T} \wedge p)$$

$$\Box(p \rightarrow \Diamond q) \overset{?}{\equiv} \Box\Diamond\neg(\neg q\,\mathcal{S}\,p)$$

$$(p\,\mathcal{U}\,(q\,\mathcal{U}\,r)) \not\equiv ((p\,\mathcal{U}\,q)\,\mathcal{U}\,r)$$

Automata

- Systems with discrete states.
- Formally, $A = \langle \Sigma, Q, \delta, q_0 \rangle$, where
  - Σ – a finite input alphabet.
  - Q – a finite set of states.
  - $\delta: Q \times \Sigma \rightarrow 2^Q$ – a transition function. Associates with state and an input letter a set of successor states.
  - q_0 – an initial state.
- An input word $\mathrm{w} = \sigma_0, \sigma_1, \ldots$ is a sequence of letters from Σ.
- A run $\mathrm{r} = \mathrm{q}_0, \mathrm{q}_1, \ldots$ over w is a sequence of states starting from q_0 such that for every $\mathrm{i} \geq 0$ we have $\mathrm{q}_{\mathrm{i}+1} \in \delta(q_i, \sigma_i)$.
- An automaton is deterministic if for every $q \in Q$ and $\sigma \in \Sigma$ we have $|\delta(q, \sigma)| \leq 1$.

Mealy Machines

- Systems with discrete states.
- Formally, $M = \langle \Sigma, \boxed{\Delta}, Q, \delta, q_0, \boxed{L} \rangle$, where
  - Σ – a finite input alphabet.
  - Δ – a finite output alphabet.
  - Q – a finite set of states.
  - $\delta : Q \times \Sigma \rightarrow 2^Q$ – a transition function. Associates with every state and an input letter a set of successor states.
  - q_0 – an initial state.
  - $L : Q \times \Sigma \rightarrow \Delta$ – an output function. Associates with every transition an output letter.
- A run $r = q_0, q_1, \ldots$ over w is a sequence of states starting from q_0 such that for every $i \geq 0$ we have $q_{i+1} \in \delta(q_i, \sigma_i)$.
- The computation correspoinding to $r = q_0, q_1, \ldots$ over w is $c = (\sigma_0, L(q_0, \sigma_0)), (\sigma_1, L(q_1, \sigma_1)), \ldots$.

Mealy Machines and LTL

- The set of computations of a machine $M = \langle \Sigma, \Delta, Q, \delta, q_0, L \rangle$ is denoted $\mathcal{L}(\mathrm{M})$.
- Assume $\Sigma = 2^{\mathcal{I}}$ and $\Delta = 2^{\mathcal{O}}$. So input letters are assignments to input propositions and outputs are assignments to output propositions.
- A machine M satisfies a formula φ, denoted $M \vDash \varphi$, if every computation in $\mathcal{L}(\mathrm{M})$ satisfies φ.
- Given an LTL formula φ over propositions $\mathcal{Prop} = \mathcal{I} \cup \mathcal{O}$ we say that φ is realizable if there is a Mealy machine that satisfies it.
- Our task is going to be to find such a Mealy machine or say that it does not exist.
- We will mostly be interested in deterministic machines.

Bibliography

1. Principles of Model Checking (C. Baier and J.-P. Katoen), *MIT Press*, 2008.
2. Model Checking (E. Clarke, O. Grumberg, and D. Peled), *MIT Press*, 1999.
3. Handbook of Model Checking (Eds., E. Clarke, T.A. Henzinger, H. Veith), *Springer-Verlag*.

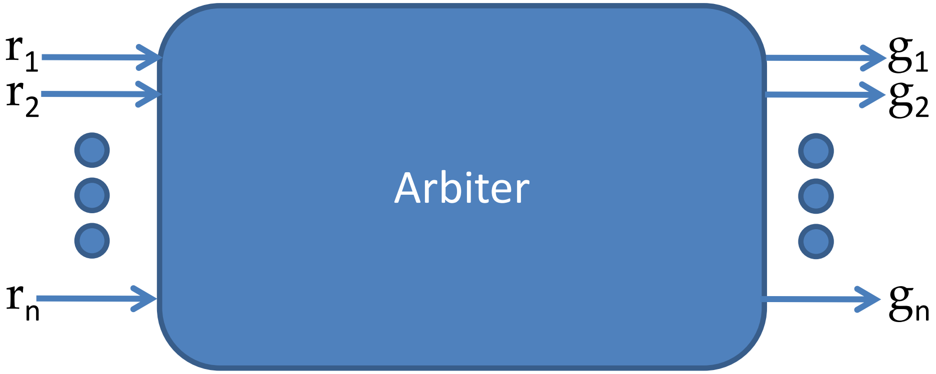
Lectures Outline

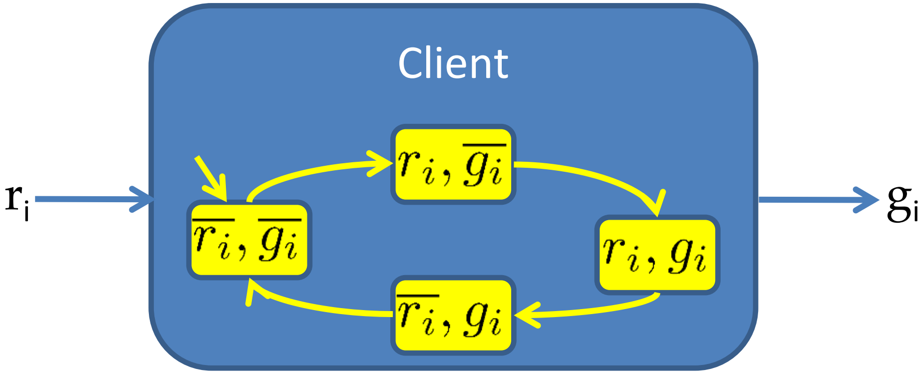
- Introduction
- Automata and Linear Temporal Logic
- Games and Synthesis
- General LTL Synthesis
- Bypassing Determinization
- Current Research Directions

Realizability

- So, given a property φ and a partition $\mathcal{P}rop = \mathcal{I} \cup \mathcal{O}$ find a system M such that $M \vDash \varphi$.
- For every possible input, decide on an output …
- All paths through the machine should satisfy the property.

Arbiter


r_1
r_2
r_n
Arbiter
g_1
g_2
g_n


Client
r_i
g_i
$\overline{r_i}, \overline{g_i}$
$r_i, \overline{g_i}$
r_i, g_i
$\overline{r_i}, g_i$

Arbiter_2

- Propositions $\mathcal{Prop} = \{r_1, r_2, g_1, g_2\}$, where $\mathcal{I} = \{r_1, r_2\}$ and $\mathcal{O} = \{g_1, g_2\}$.
- Requirements:
  - A_1: leave requests: $\Box(r_1 \wedge !\, g_1 \rightarrow \bigcirc r_1) \wedge \Box(r_2 \wedge !\, g_2 \rightarrow \bigcirc r_2)$
  - G_1: leave grants: $\Box(r_1 \wedge g_1 \rightarrow \bigcirc g_1) \wedge \Box(r_2 \wedge g_2 \rightarrow \bigcirc g_2)$
  - G_2: mutual exclusion: $\Box(!\, g_1 \vee !\, g_2)$
  - G_3: deliver and remove grants: $\Box\Diamond(g_1 \leftrightarrow r_1) \wedge \Box\Diamond(g_2 \leftrightarrow r_2)$
- Or together: $\text{A}_1 \rightarrow (G_1 \wedge G_2 \wedge G_3)$

What’s the idea?

- Think about control:
  - Some things are under our control.
  - Some things are not.
- We want to exercise our control so that to achieve certain goals.
- In some cases the environment is hostile.
- What we want:
  - Find a strategy that will guide our actions based on our view of the world.
- This leads to viewing the world as an opponent:
  - Exercise control so that uncontrollable events do not lead to damage.
- We model this as two-player games.

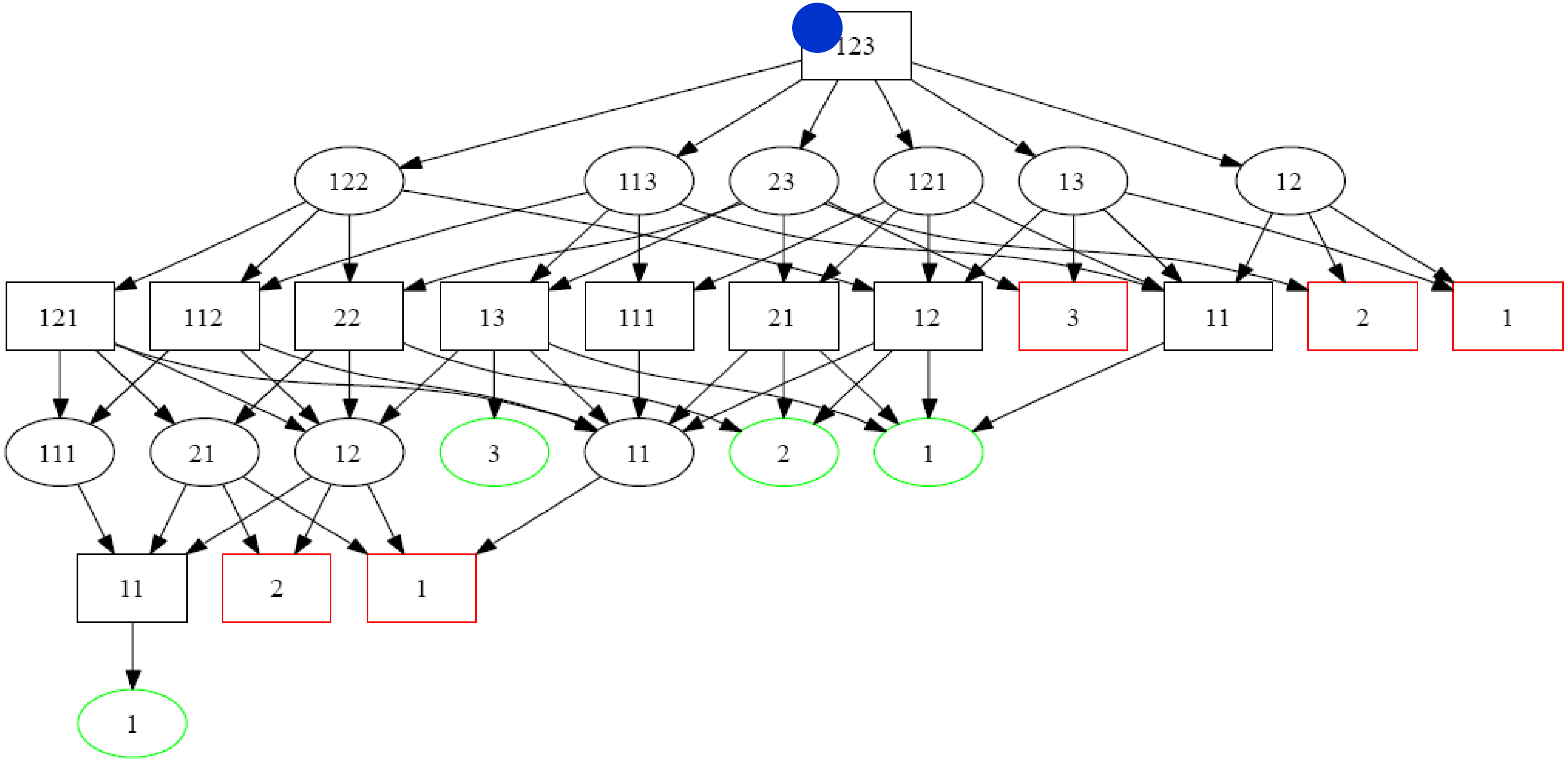
Example: Nim

- Some rows of matches.
- Every player removes in turn at least one match from one row.
- The one to remove last match wins.
- Can you win?

Whose in Control?

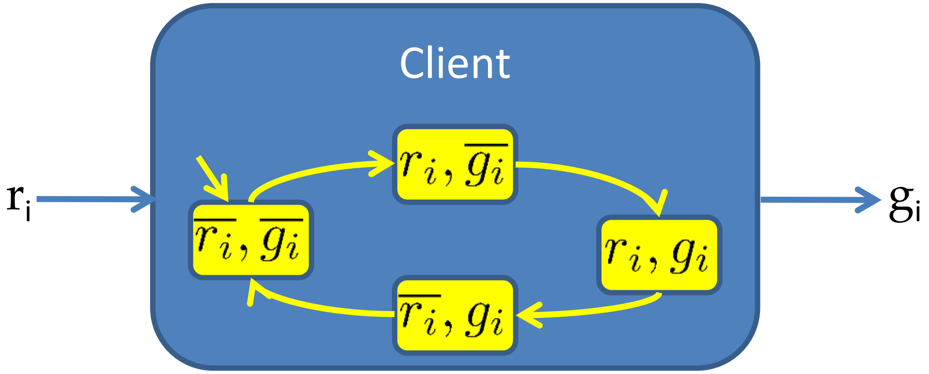
- We use graphs with vertices for states and edges for transitions.
- Ownership is by using two types of vertices.

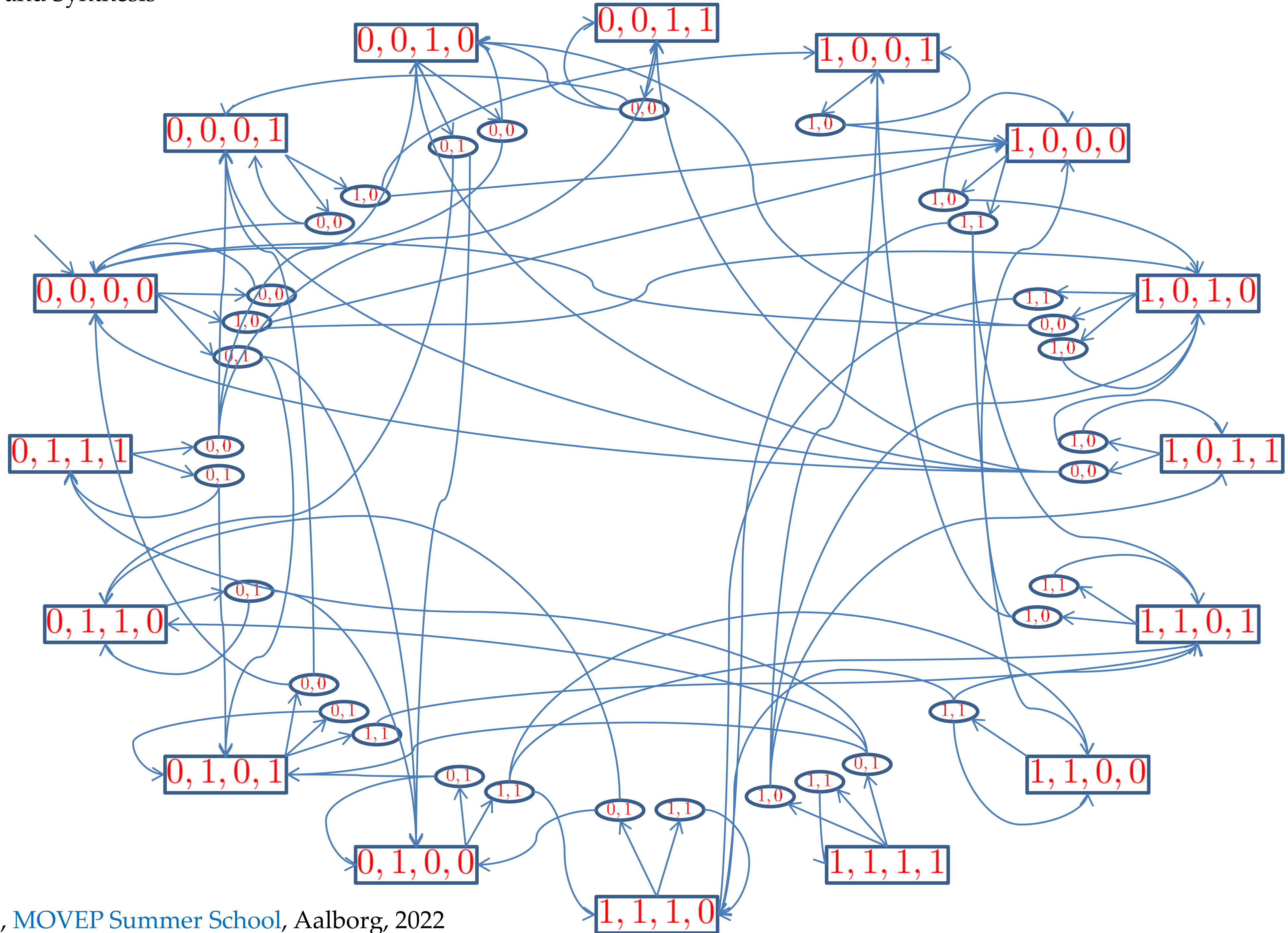
A Play


Environment
System
123
122
113
23
121
13
12
121
112
22
13
111
21
12
3
11
2
1
111
21
12
3
11
2
1
11
2
1
1

Arbiter


r_1
r_2
r_n
Arbiter
g_1
g_2
g_n


Client
r_i
$\overline{r_i}, \overline{g_i}$
$r_i, \overline{g_i}$
r_i, g_i
$\overline{r_i}, g_i$
g_i


0, 0, 1, 0
0, 0, 1, 1
1, 0, 0, 1
0, 0, 0, 1
1, 0, 0, 0
0, 0, 0, 0
1, 0, 1, 0
0, 1, 1, 1
1, 0, 1, 1
0, 1, 1, 0
1, 1, 0, 1
0, 1, 0, 1
1, 1, 0, 0
0, 1, 0, 0
1, 1, 1, 0
1, 1, 1, 1
0, 0
1, 0
0, 0
0, 1
1, 0
0, 0
1, 0
1, 1
0, 0
1, 0
0, 1
1, 1
0, 0
1, 0
0, 0
0, 1
1, 0
0, 0
0, 1
1, 1
1, 0
0, 0
0, 1
1, 1
1, 1
0, 1
0, 1
1, 1
1, 1
1, 0
0, 1
1, 1

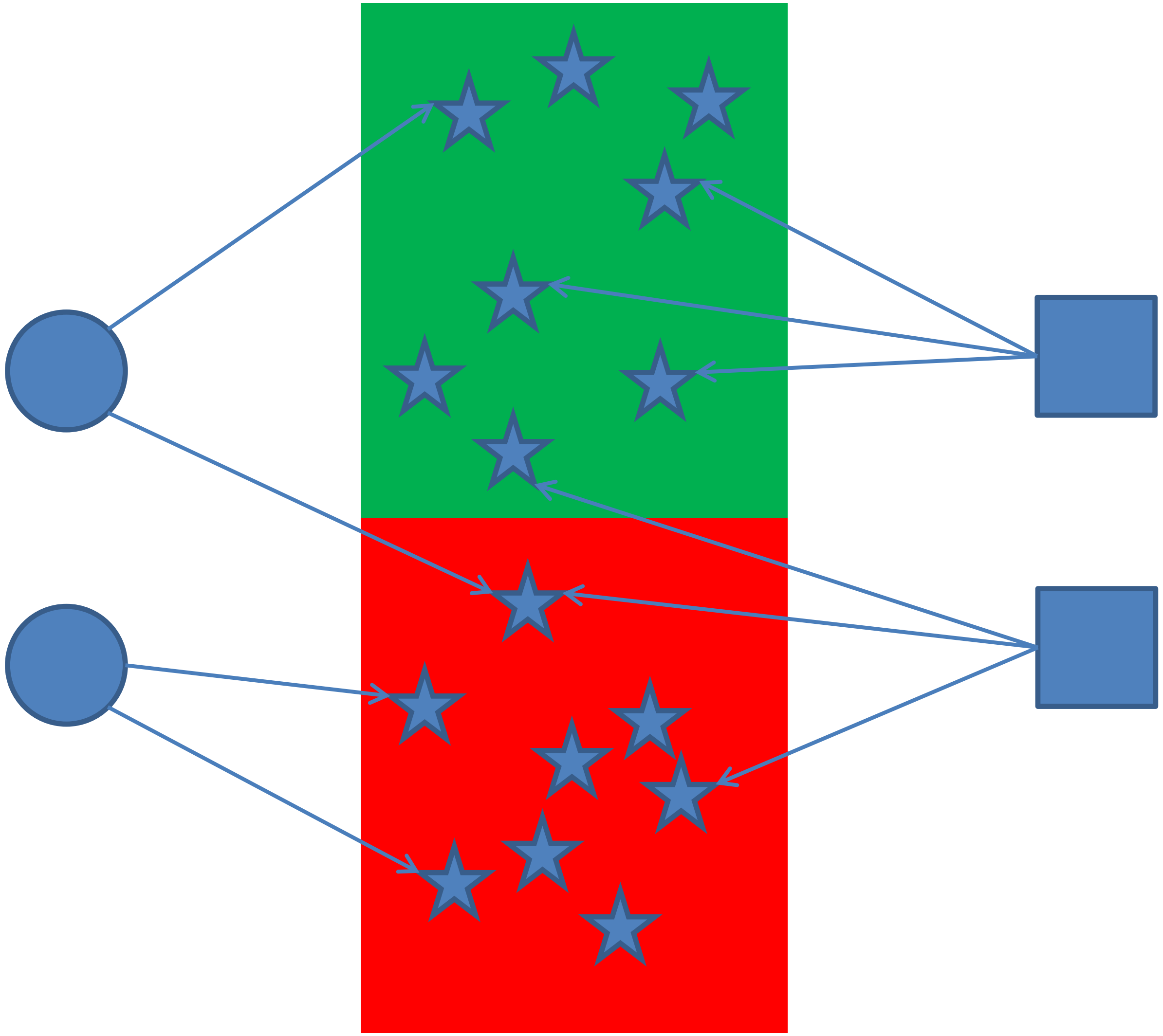
Games

- Formally, a game is $G = \langle V, V_0, V_1, E, \alpha \rangle$, where
  - V is a set of nodes.
  - V_0 and V_1 form a partition of V.
  - $E \subseteq V \times V$ is a set of edges.
- A play is $\pi = v_0, v_1, \dots$
  - α is a set of winning pl
- A strategy for playe $_i \to V$ such that $(\mathrm{v}, f_i(w \cdot v)) \in E$.
- A play $\pi = v_0$ f_i if for every $j \geq 0$ such that $v_j \in V_i$ we have $v_{j+1} = f_i(v_0$
- A strategy for ning if every play compatible with it is in α. A strategy for player 1 is w every play compatible with it is not in α.
- A node v is won player i if she has a winning strategy for all plays starting from v.

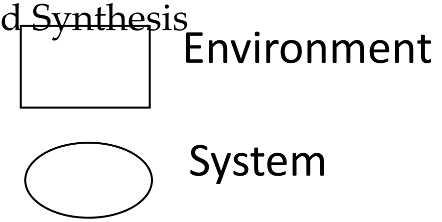
From now on we mostly care about player 0!

Control Predecessor

- In control it is easier to walk backwards.

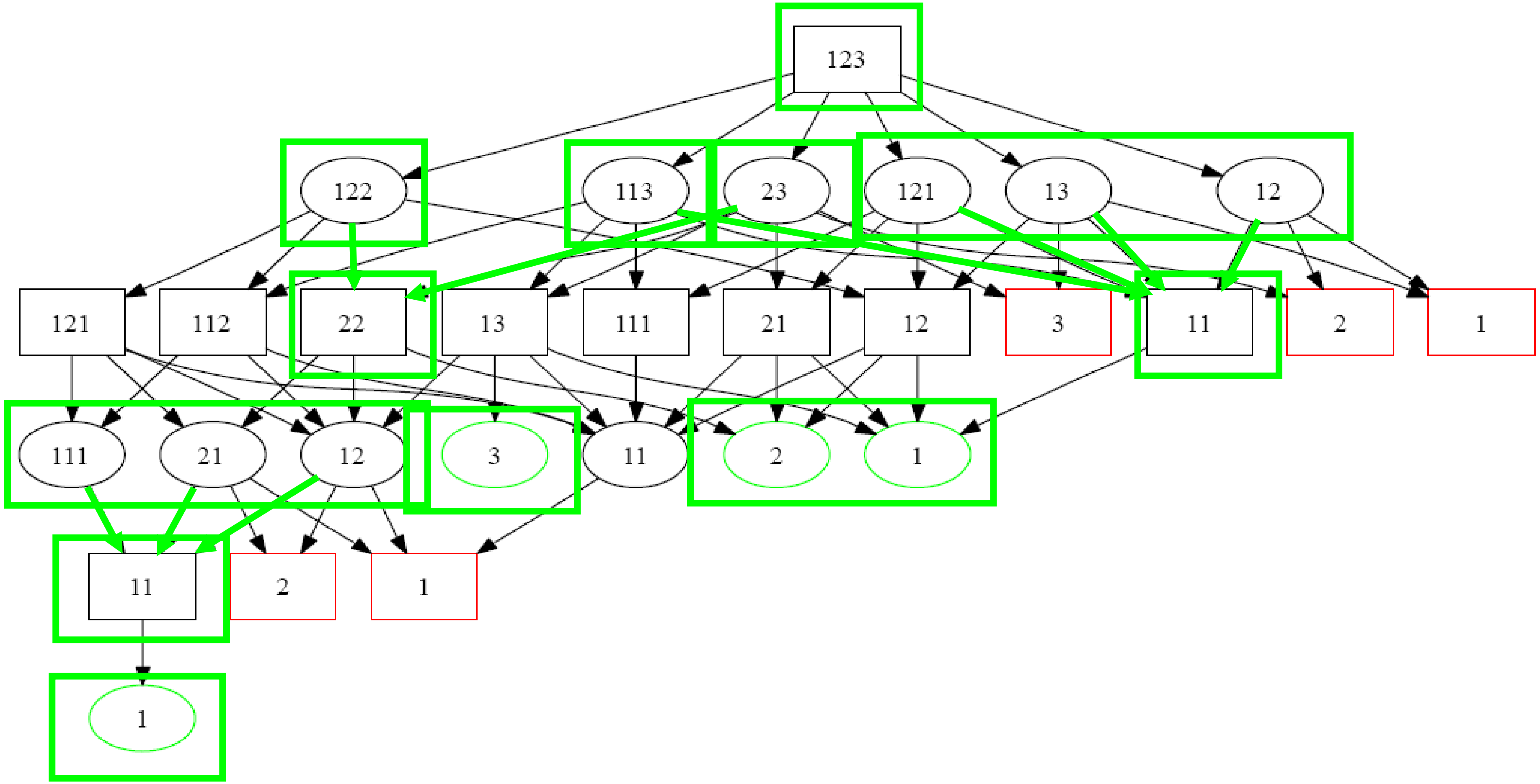


Game Analysis



Environment

System



Control Predecessor (for P0)

- Start from an set of nodes $W \subseteq V$.
- We want to say:
  - The system can force the environment to W in one move.
- That is:
  - Nodes $v \in V_0$ for which some successor is in W.
  - Nodes $v \in V_1$ for which all successors are in W.
- Formally:

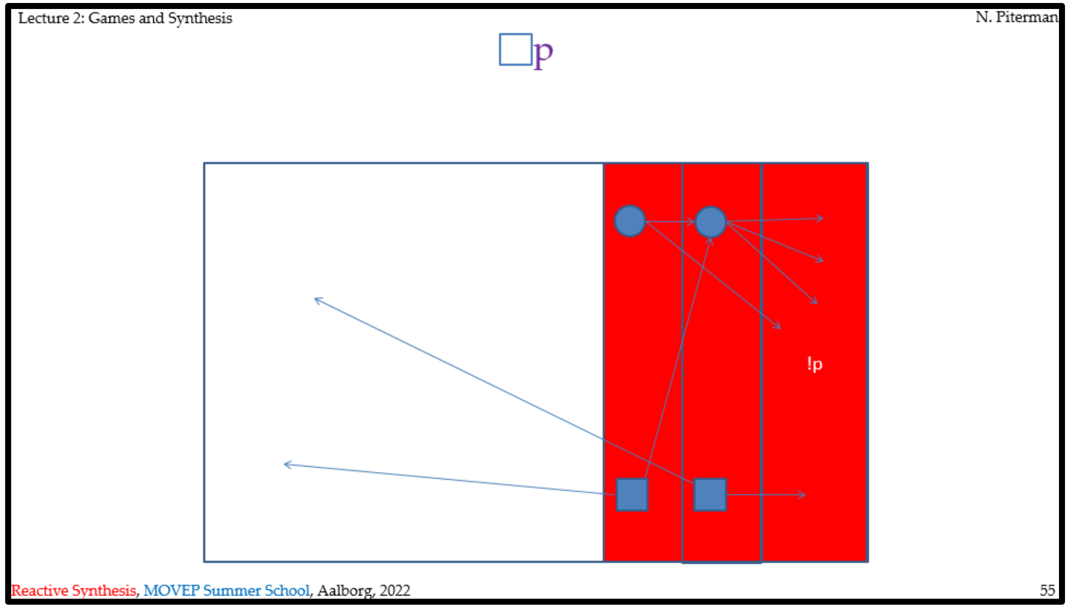
$$\begin{aligned} cpre(W) = {} & \{ v \in V_0 \mid \exists v' \in W. (v, v') \in E \} \cup \\ & \{ v \in V_1 \mid \forall v'. (v, v') \in E \rightarrow v' \in W \} \end{aligned}$$

Control Predecessor (for P1)

- Start from an set of nodes $W \subseteq V$.
- We want to say:
  - The environment can force the system to W in one move.
- That is:
  - Nodes $v \in V_1$ for which some successor is in W.
  - Nodes $v \in V_0$ for which all successors are in W.
- Formally:

$$\begin{aligned} cpre_1(W) = & \{ v \in V_1 \mid \exists v' \in W . (v, v') \in E \} \quad \cup \\ & \{ v \in V_0 \mid \forall v' . (v, v') \in E \rightarrow v' \in W \} \end{aligned}$$

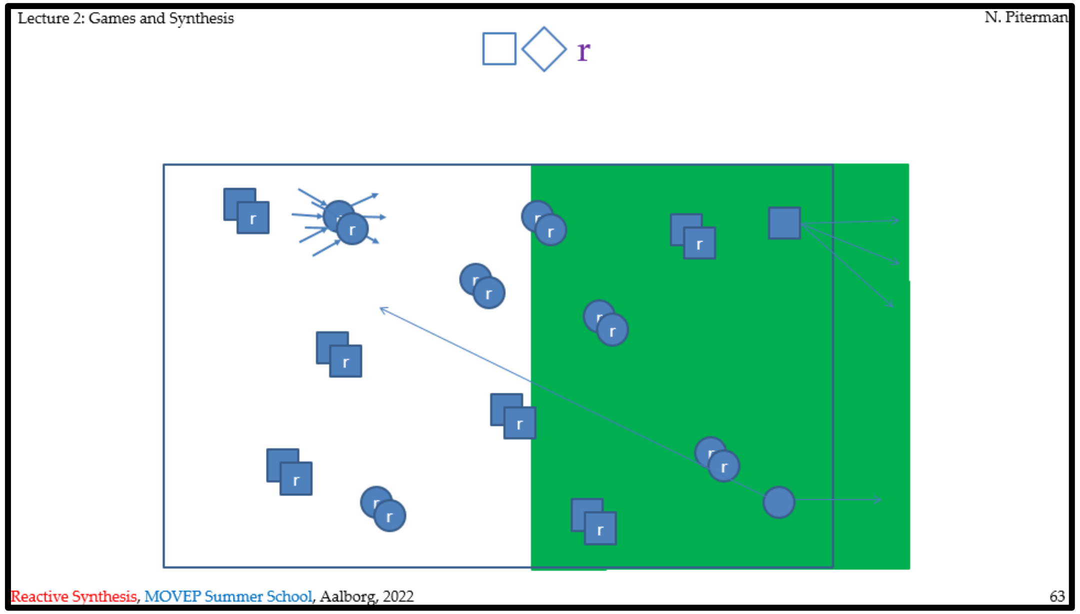
Let’s solve some games!



Lecture 2: Games and Synthesis N. Piterman

$\Box p$

Reactive Synthesis, MOVEP Summer School, Aalborg, 2022 55

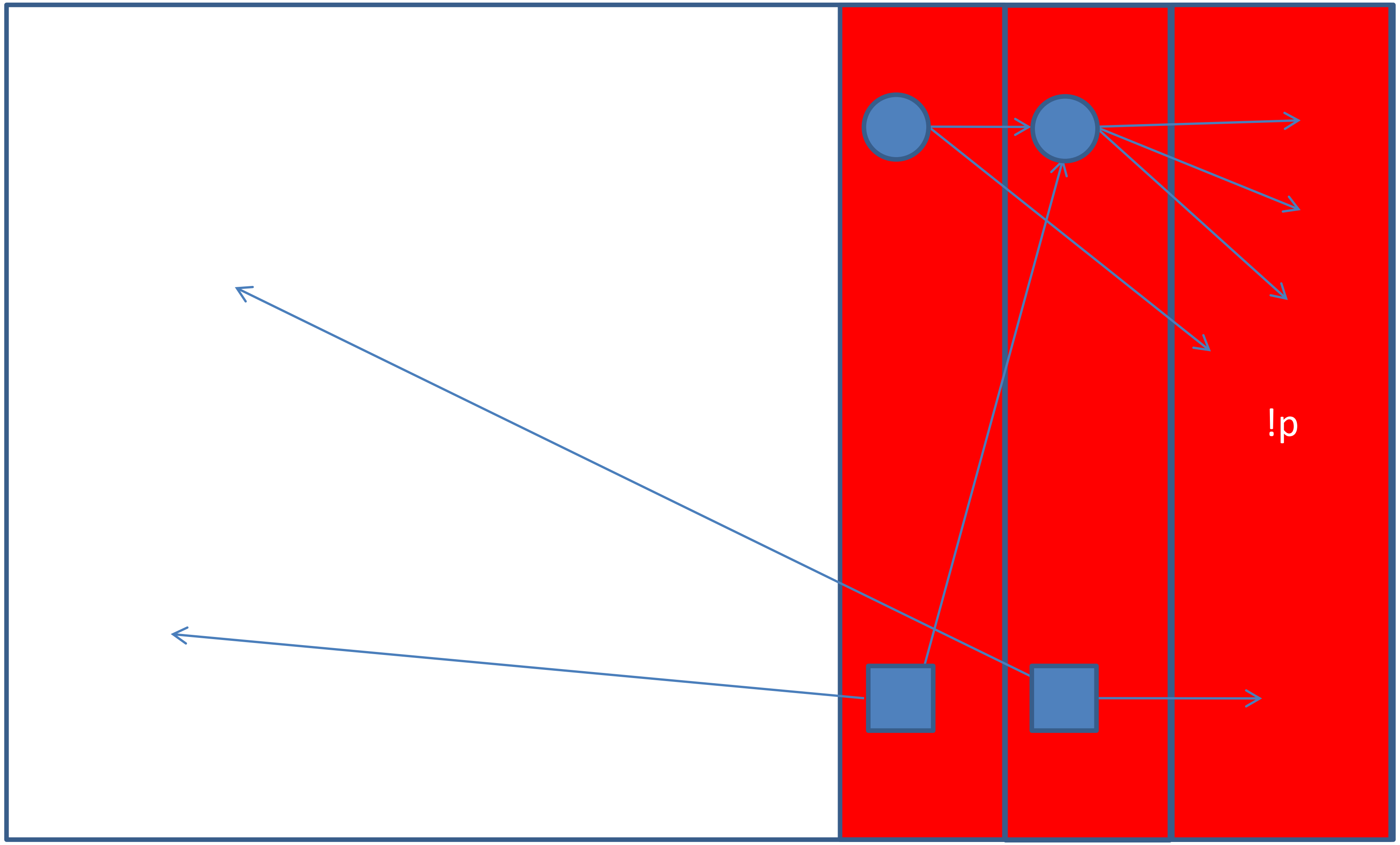


Lecture 2: Games and Synthesis N. Piterman

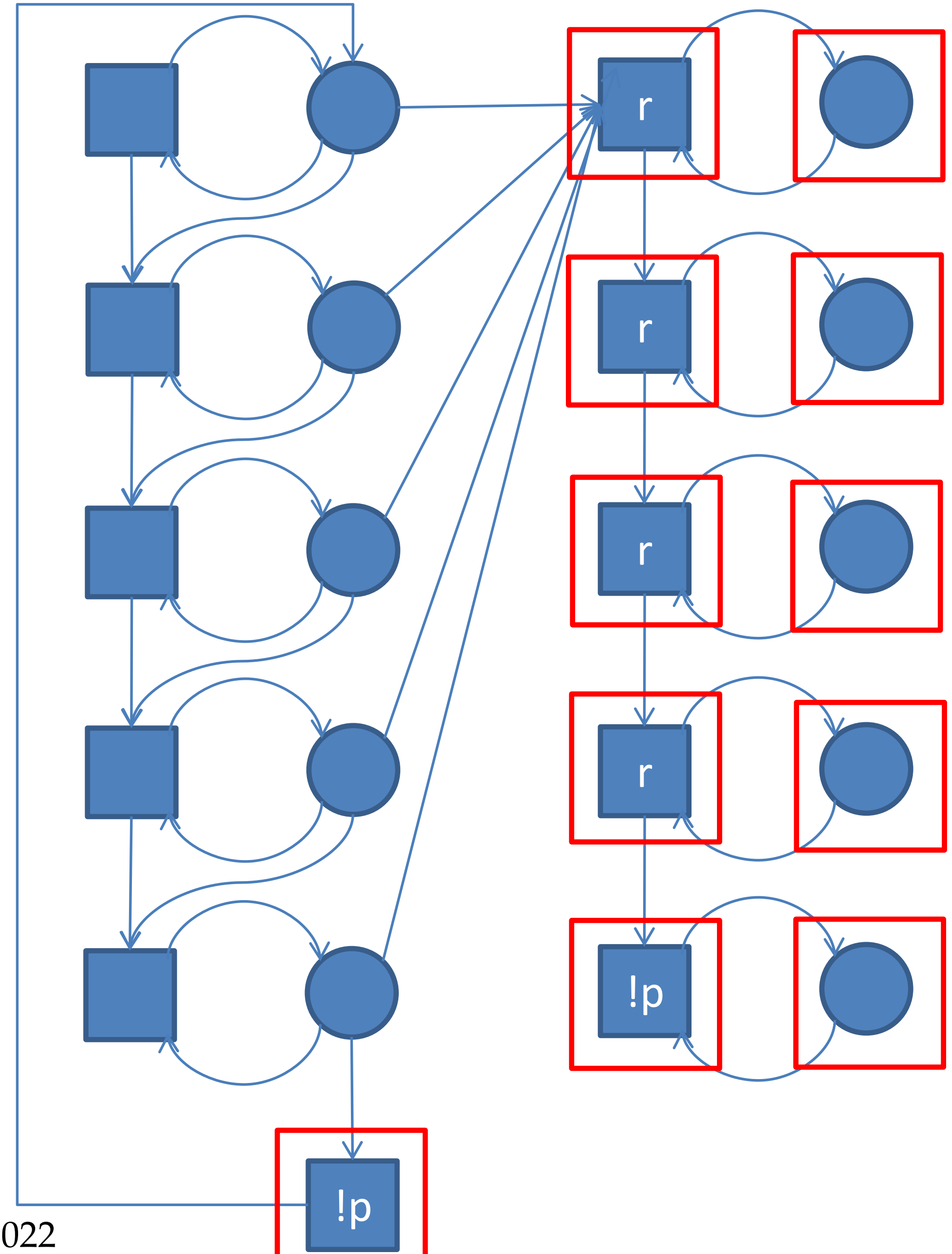
$\Box\Diamond r$

Reactive Synthesis, MOVEP Summer School, Aalborg, 2022 63

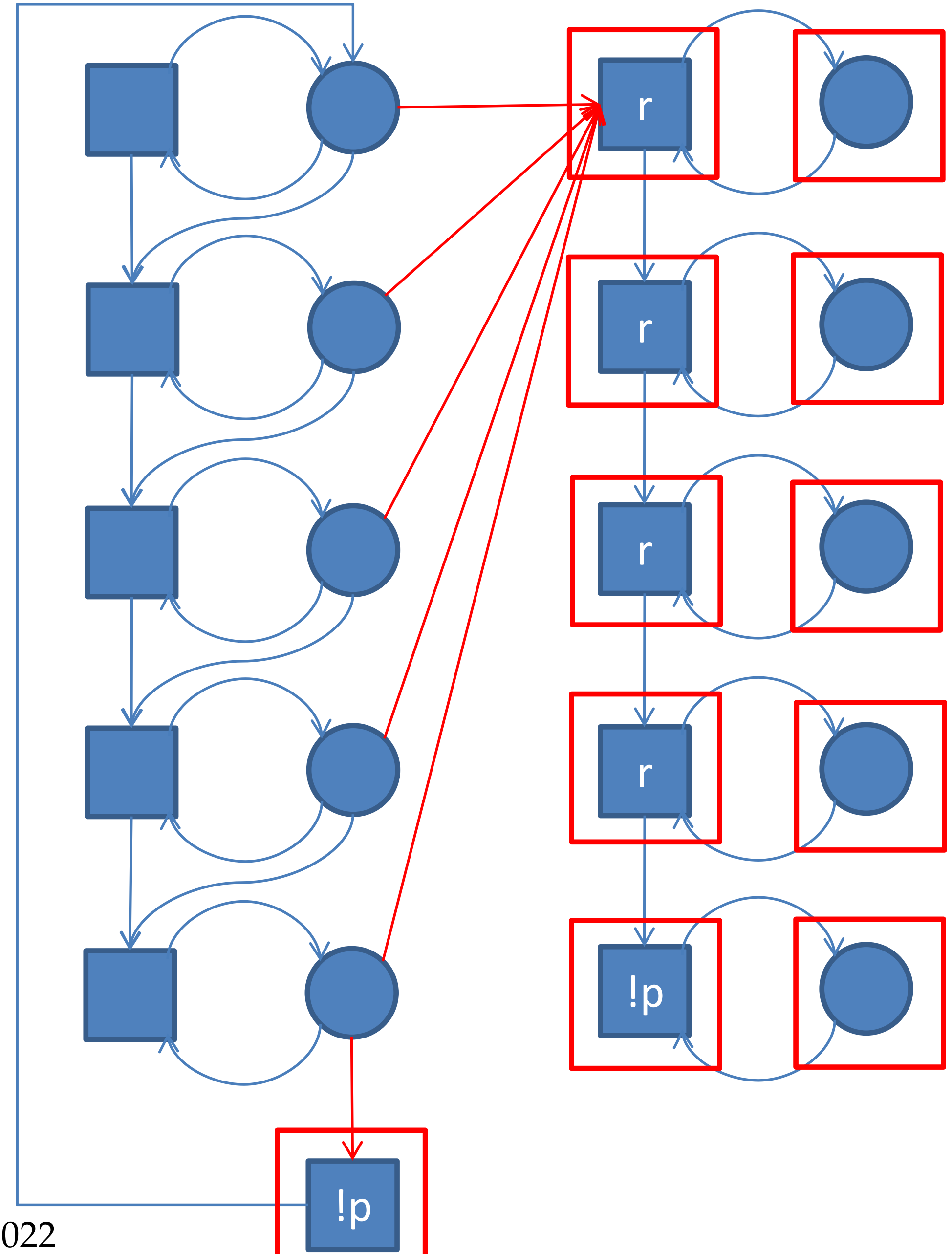
$\Box p$



$\Box p$



$\square p$



Safety Games

- Check that P0 can enforce $\square p$.

```
fix (new := p)
    new := new ∧ $cpre$(new)
end // fix
```

Lemma. The algorithm computes the set of states winning for P0 with objective $\square p$.

Proof. Later.

Reachability Games

- Check that P1 can enforce $\Diamond \neg p$.

```
fix (new := ¬p)
    new := new ∨ $cpre_1$(new)
end // fix
```

Lemma. The algorithm computes the set of states winning for P1 with objective $\Diamond p$.
Proof. Later.

$Attr_i(W)$ the set of nodes from which player i can force reaching W.

Safety vs Reachability Games

- Goals $\Box p$ for P0 and $\Diamond \neg p$ for P1 are complementary.

```
fix (new := p)
    new := new ∧ $cpre$(new)
end // fix
```

```
fix (new := ¬p)
    new := new ∨ $cpre_1$(new)
end // fix
```

Safety Games

- Check that P0 can enforce $\square p$.

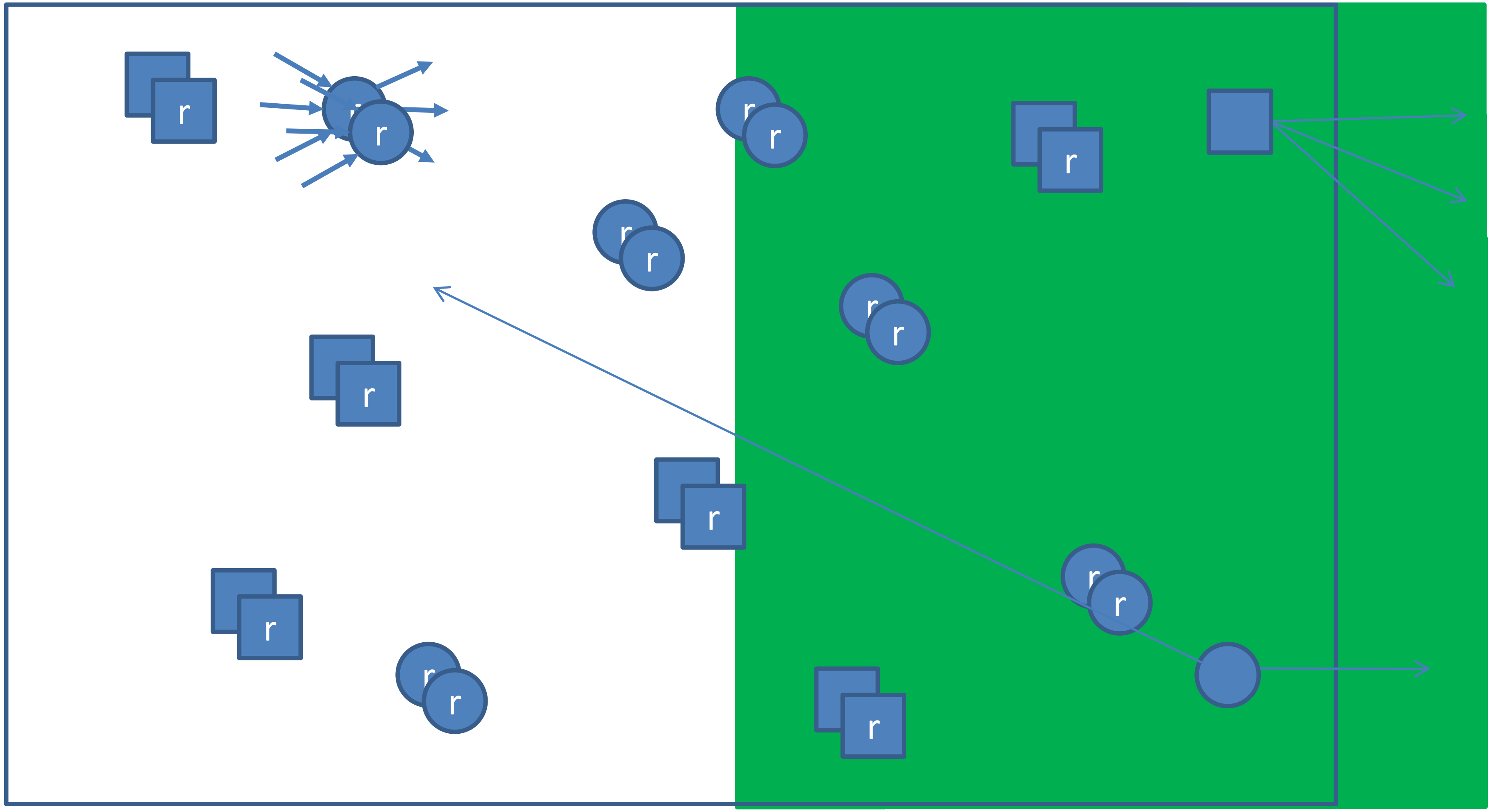
```
fix (new := p)
    new := new ∧ $cpre$(new)
end // fix
```

Proof

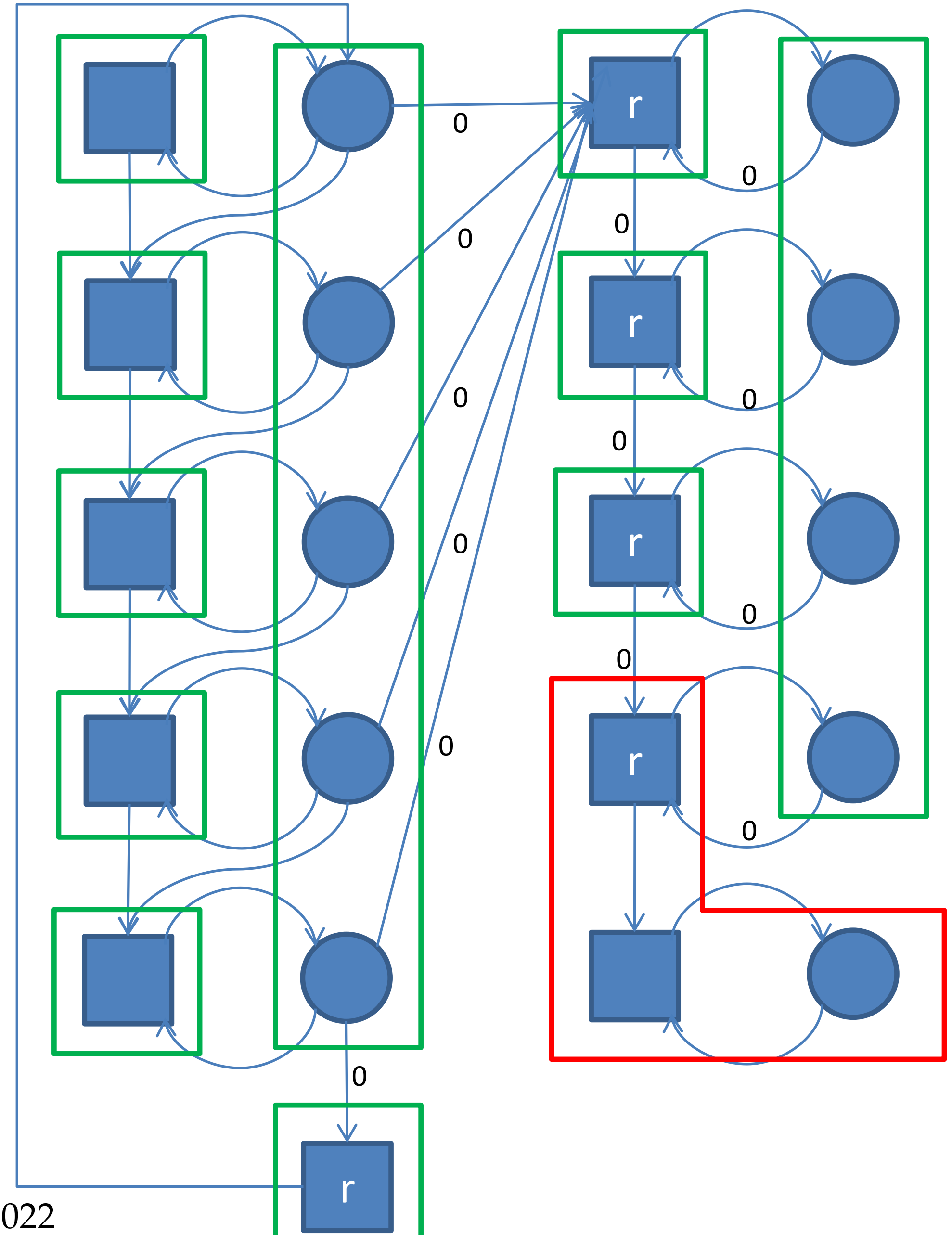
```
fix (new := p)
    new := new ∧ cpre(new)
end // fix
```

- Suppose that new is not empty.
  Consider $v \in$ new. Clearly, $v \in p$. But also $v \in cpre$(new).
  If $v \in V_0$, then v has a successor w such that $w \in$ new.
  If $v \in V_1$, then for every successor w of v we know $w \in$ new.
- If there is a strategy s.t. every play compliant with it wins $\Box p$.
  Let new_0, new_1, new_2, … be the series of approximations of new. We prove by induction that for every v winning for P0, $v \in \text{new}_i$ for every i.
  Clearly, $v \in p$ implies $v \in \text{new}_0$.
  Assume every v winning for P0 is in new_i for some i. Consider $v \in V_0$ winning for P0. Then, there is w such that $(v, w) \in E$ and w winning for P0. Then, w in new_i and v in new_{i+1}. Consider $v \in V_1$ winning for P0. Then, for every w such that $(v, w) \in E$ we have w winning for P0. Then, every w such that $(v, w) \in E$ is in new_i. So v in new_{i+1}.

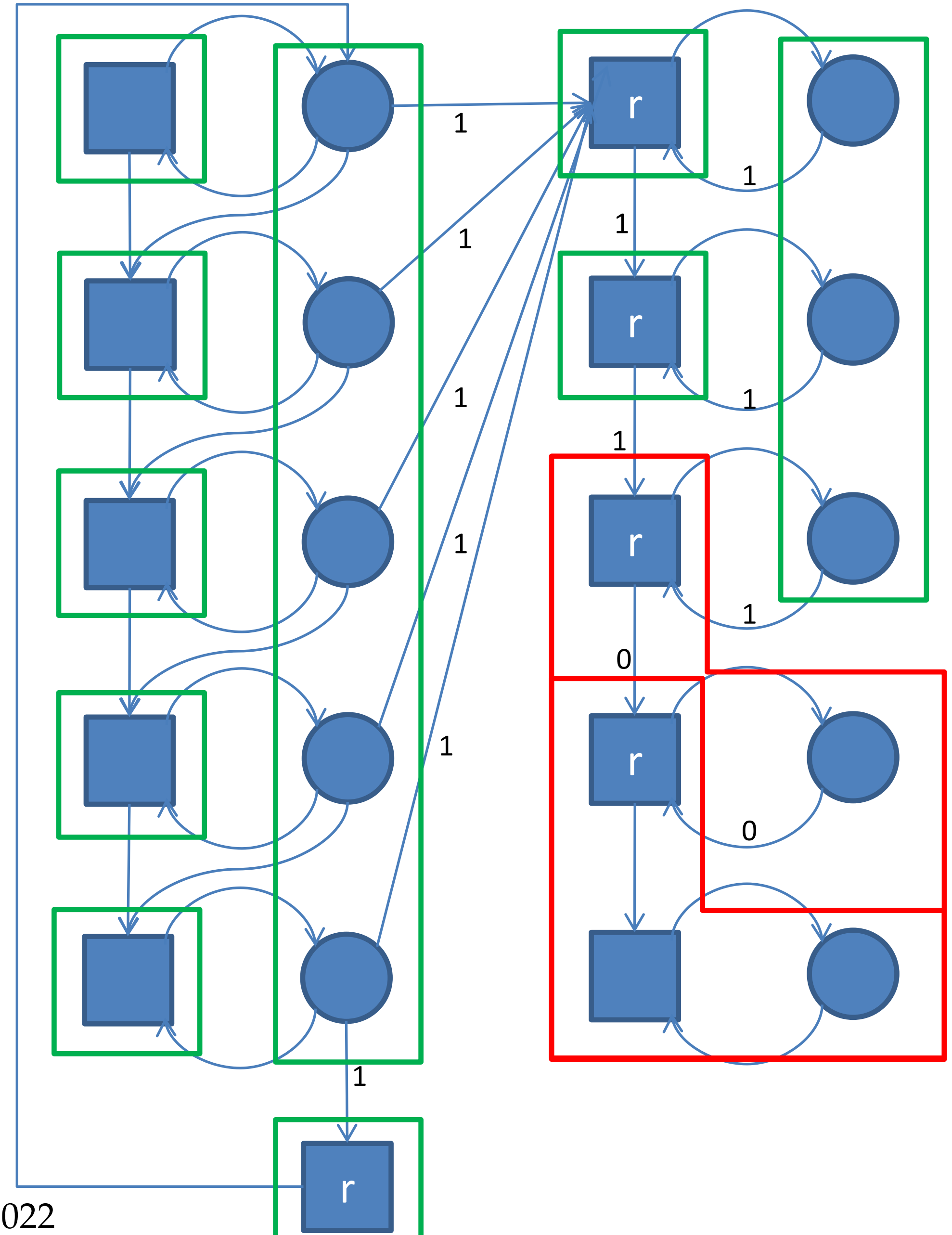
$\Box\Diamond r$



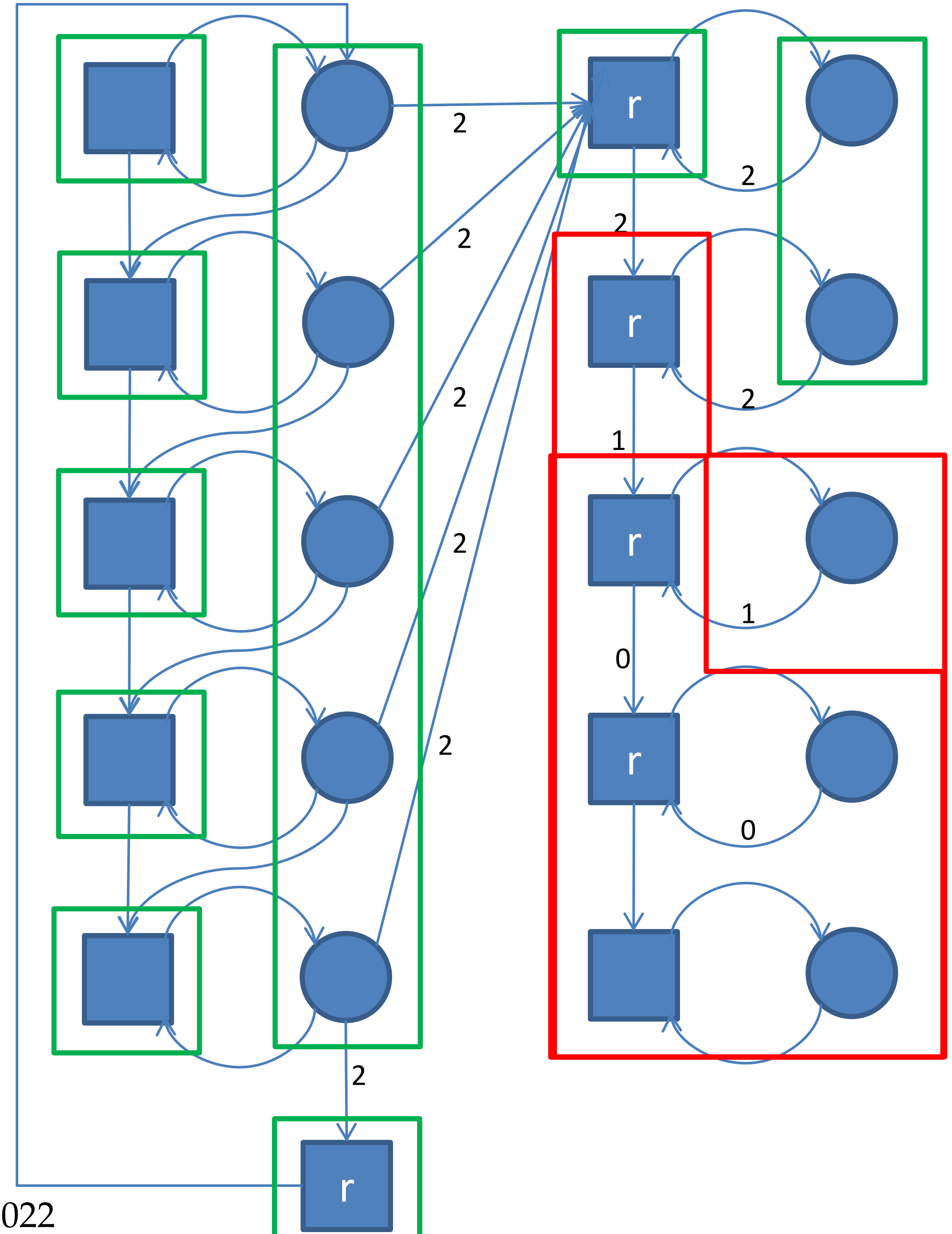
$\Box\Diamond r$



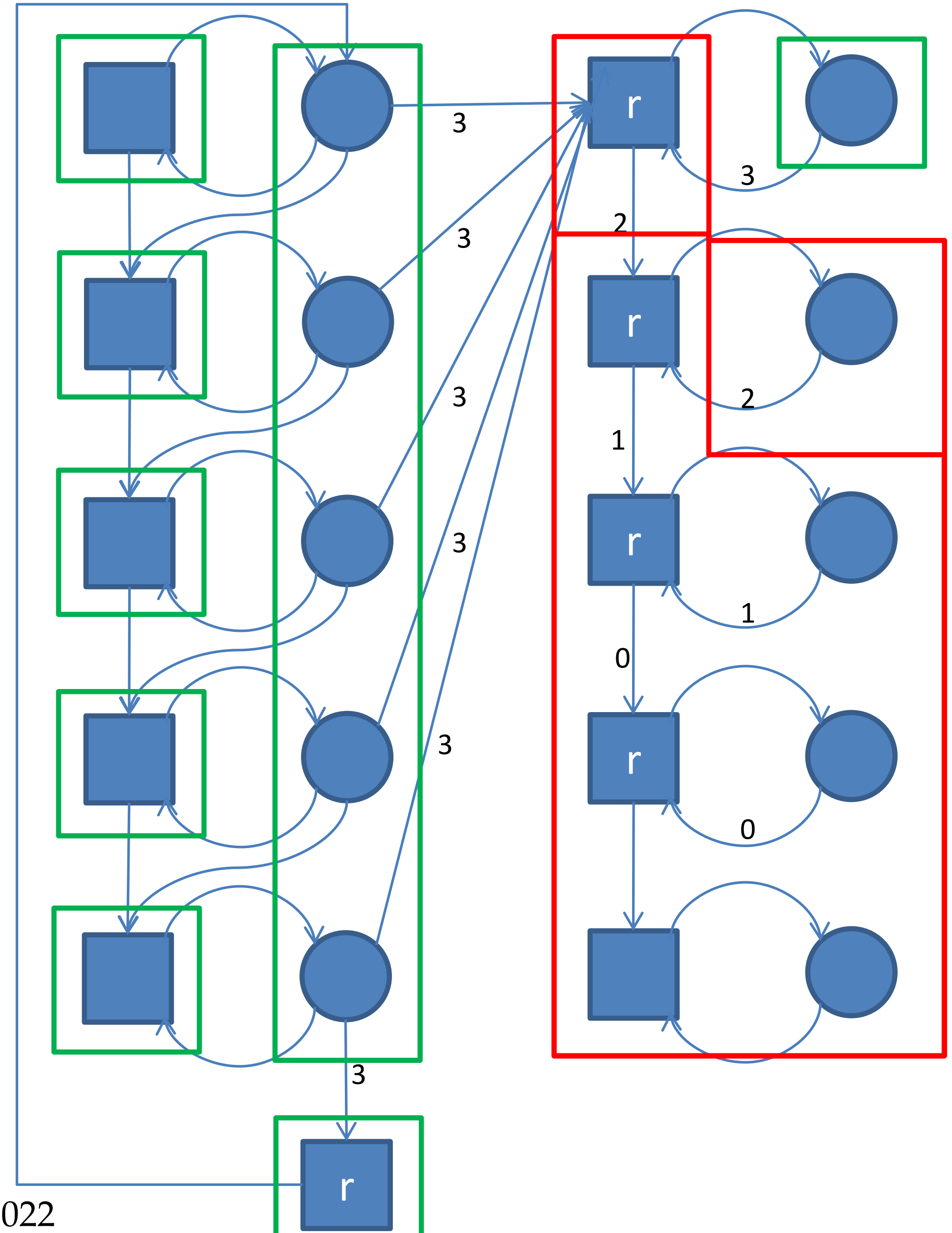
$\Box\Diamond r$



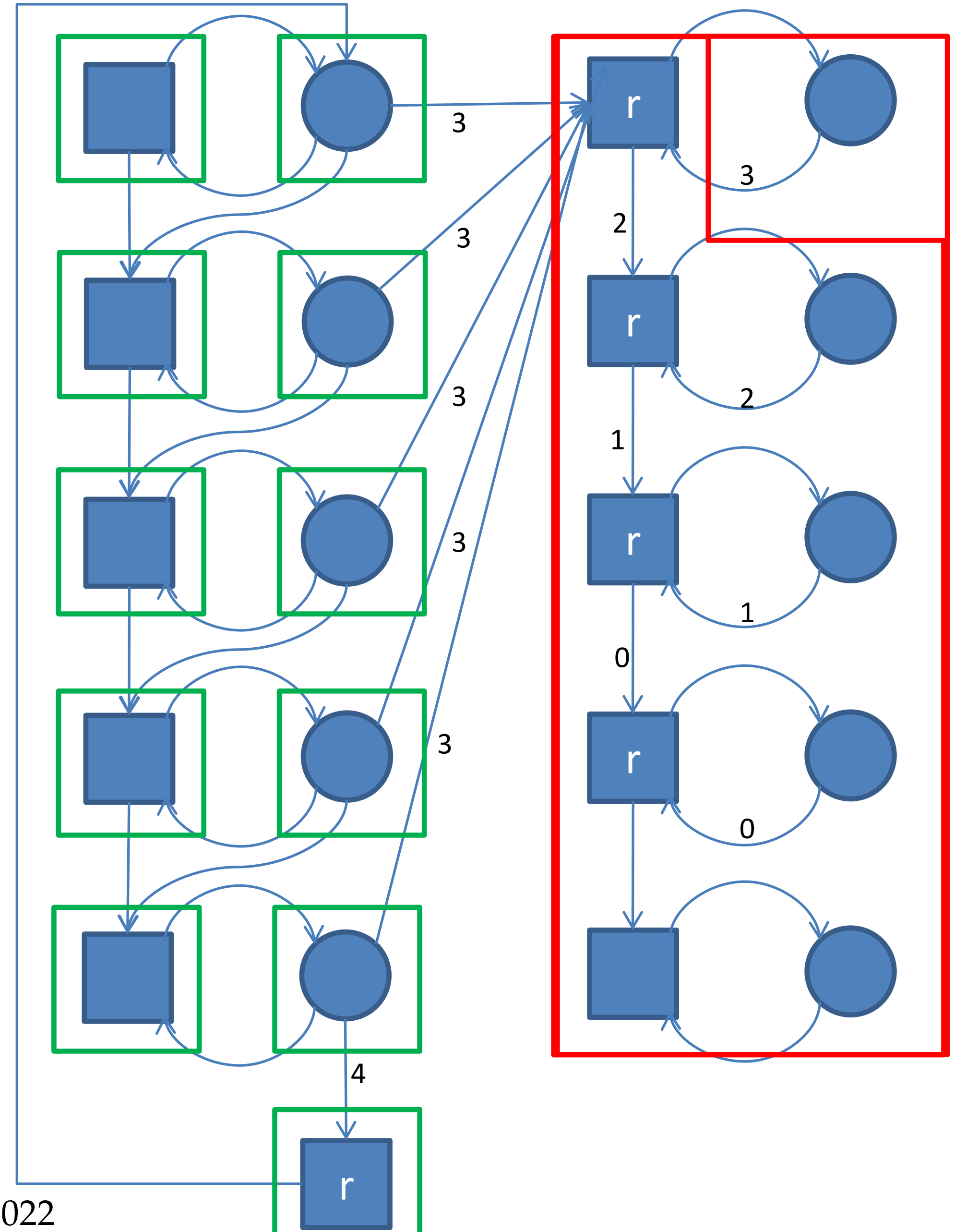
$\Box\Diamond r$



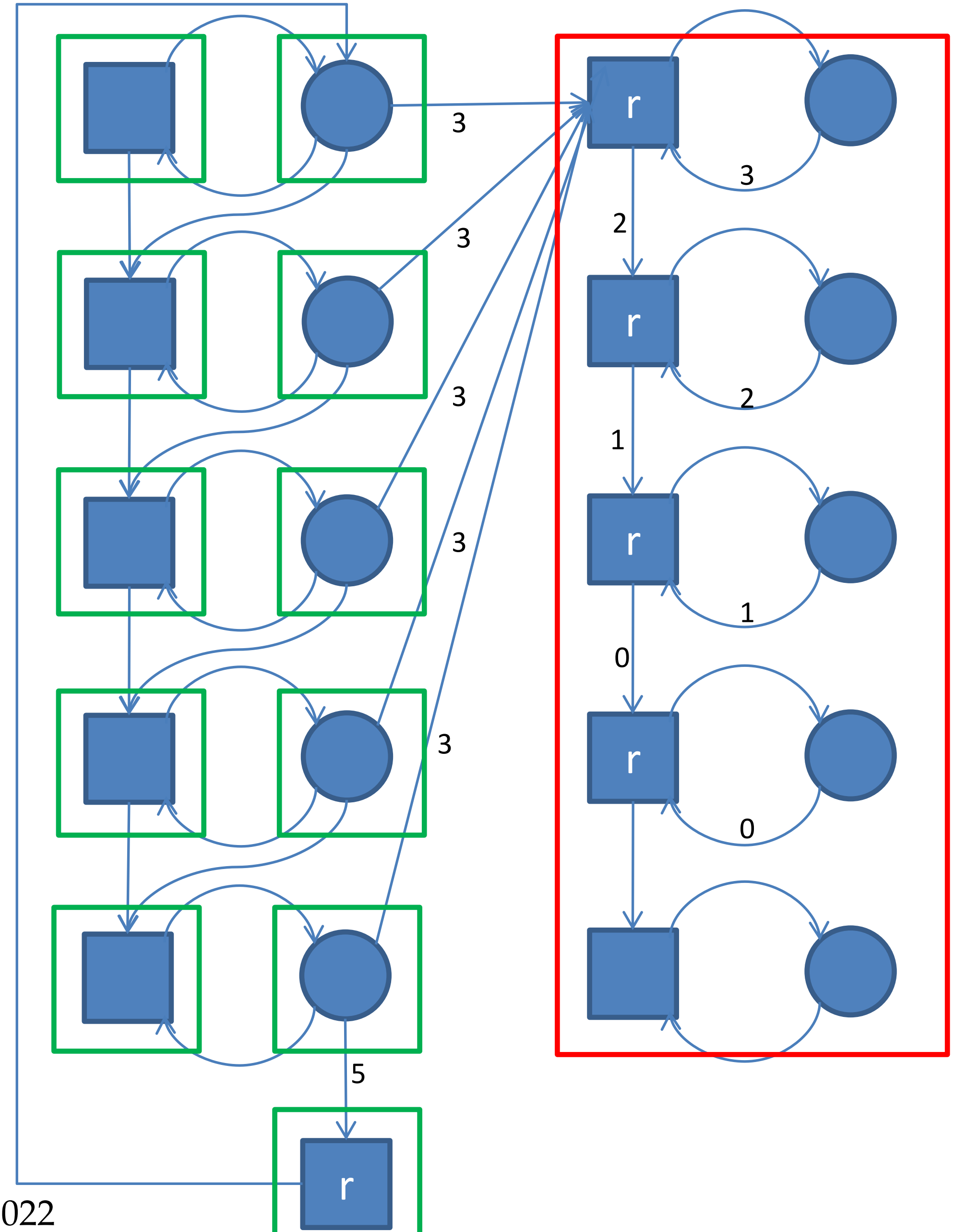
$\Box\Diamond r$



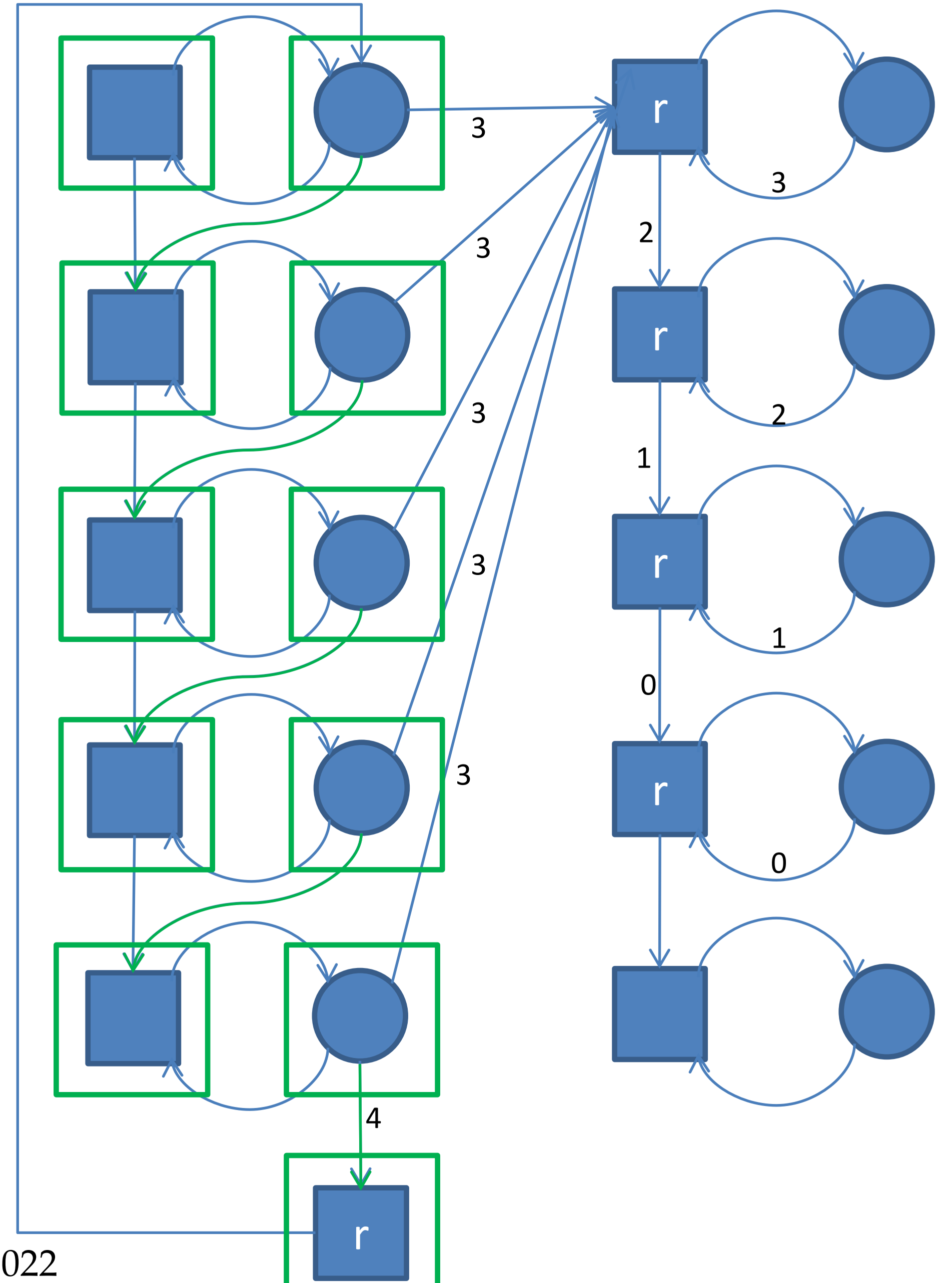
$\Box\Diamond r$



$\Box\Diamond r$



$\Box\Diamond r$



Büchi Games

- Check that P0 can enforce $\Box\Diamond p$.

```
fix (greatest := $V$)
    fix (least := $p \wedge cpre$(greatest)
        least := least $\vee$ $cpre$(least);
    end // fix least
    greatest := least;
end // fix greatest
```

Lemma. The algorithm computes the set of nodes winning for P0 with objective $\Box\Diamond p$.

Büchi Games

- Check that P0 can enforce $\Box\Diamond p$.

```
fix (greatest := $V$)
    fix (least := $p \wedge cpre$(greatest)
        least := least $\vee$ $cpre$(least);
    end // fix least
    greatest := least;
end // fix greatest
```

Proof (Control of Büchi –Soundness)

```
fix (greatest := $V$)
    fix (least := $p \wedge cpre$(greatest)
        least := least ∨ $cpre$(least);
    end // fix least
    greatest := least;
end // fix greatest
```

- Suppose that greatest is not empty. For the fixpoint to terminate, the inner fixpoint starting from this value recomputes it.
- Let least_0, least_1, least_2, … be the sequence of values that least has through the computation of this last iteration.
- Consider $v \in$ greatest. Let i_0 be the index such that $v \in \text{least}_{i_0}$. By definition of $cpre(\cdot)$, P0 can force a successor w of v. But then, $w \in \text{least}_{i_1}$ for some $i_1 < i_0$.
- This shows that P0 can ensure to reach $\text{least}_0 = p \wedge cpre(\text{greatest})$. So it ensures a visit p.
- But now $\text{least}_0 = p \wedge cpre(\text{greatest})$. So in the next step P0 forces least_j for some j and repeat this process.
- P0 can enforce $\Box\Diamond p$.

Proof (Control of Büchi - completeness)

```
fix (greatest := $V$)
    fix (least := $p \wedge cpre$(greatest)
        least := least ∨ $cpre$(least);
    end // fix least
    greatest := least;
end // fix greatest
```

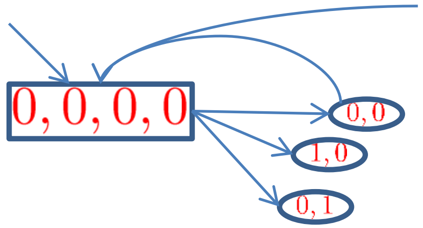
- If there is a strategy f s.t. every play compliant with it wins $\square\Diamond p$.
- Every node v from which f is winning remains in every approximation of the fixpoint greatest:
  - From v there is a maximum on the length of paths to reach p (König's lemma).
  - Prove by induction on the number of iterations in the first fixpoint that win$\subseteq$ greatest.
  - For greatest$_0 = V$ this is clear.
  - Assume win$\subseteq$ greatest$_i$. Then for every node $v \in$ win it must be that $v \in$ least$_j$ for the distance to reach $p \wedge$ win.

Strategy

- A strategy is the way of enforcing the goal.
- Let D be some memory domain and let d_0 be an initial memory value. Elements in the memory domain recall facts about the history of play so far.
- A strategy for player i is a function $f_i\colon V^* \cdot V_0 \rightarrow V$ such that $\left(\mathrm{v}, f_i(w \cdot v)\right) \in E$.
- We look to replace V^* by some (finite) domain D. Then, instead of considering V we could consider $D \times V$.
- The strategy is replaced by two functions:
  - Move function: $f_i^m\colon D \times V_i \rightarrow V$ s.t. $\left(v, f(d, v)\right) \in E$.
  - Update function: $f_i^u\colon D \times V \rightarrow D$.

What about Synthesis?

- Our goal is to construct a Mealy machine that realizes the specification.
  - A Mealy machine from every state reads input and answers with output.
- A node in the game corresponding to choice of input will be followed by node corresponding to choice of output.



- We can define a specialized game with nodes in $2^{\mathcal{I} \cup \mathcal{O}}$.
- We can define the winning condition with an LTL formula over $\mathcal{I} \cup \mathcal{O}$. A play naturally corresponds to a possible model.
- For a set of nodes W, define

$$cpre(W) = \{v \mid \forall x \in 2^{\mathcal{I}}.\ \exists y \in 2^{\mathcal{O}}.\ (x \cup y) \in W\}$$

- When computing the set of winning states, check that for every $x \in 2^{\mathcal{I}}$ there is $y \in 2^{\mathcal{O}}$ such that $x \cup y$ is winning.

Further Specialize Strategy

- Let D be some memory domain and let d_0 be an initial memory value. Elements in the memory domain recall facts about the history of play so far.
- A strategy for player i is a function $f_i: V^* \cdot V_0 \rightarrow V$ such that $\left(\mathrm{v}, f_i(w \cdot v)\right) \in E$.
- We look to replace V^* by some (finite) domain D. Then, instead of considering V we could consider $D \times V$.
- The strategy is replaced by two functions:
  - Move function: $f_i^m: D \times V_i \rightarrow V$ s.t. $\left(v, f(d, v)\right) \in E$.
  - Update function: $f_i^u: D \times V \rightarrow D$.

Further Specialize Strategy

- Let D be some memory domain and let d_0 be an initial memory value. Elements in the memory domain recall facts about the history of play so far.
- A strategy for player i is a function $f_i\colon \left(2^{\mathcal{I} \cup \mathcal{O}}\right)^* \cdot 2^{\mathcal{I}} \to 2^{\mathcal{O}}$.
- We look to replace V^* by some (finite) domain D. Then, instead of considering V we could consider $D \times V$.
- The strategy is replaced by two functions:
  - Move function: $f_i^m\colon D \times V_i \to V$ s.t. $\left(v, f(d, v)\right) \in E$.
  - Update function: $f_i^u\colon D \times V \to D$.

Further Specialize Strategy

- Let D be some memory domain and let d_0 be an initial memory value. Elements in the memory domain recall facts about the history of play so far.
- A strategy for player i is a function $f_i\colon \left(2^{\mathcal{I}\cup\mathcal{O}}\right)^* \cdot 2^{\mathcal{I}} \rightarrow 2^{\mathcal{O}}$.
- We look to replace $\left(2^{\mathcal{I}\cup\mathcal{O}}\right)^*$ by some (finite) domain D. Then, instead of considering $\left(2^{\mathcal{I}\cup\mathcal{O}}\right)^*$ we could consider $D \times 2^{\mathcal{I}\cup\mathcal{O}}$.
- The strategy is replaced by two functions:
  - Move function: $f_i^m\colon D \times V_i \rightarrow V$ s.t. $\left(v, f(d,v)\right) \in E$.
  - Update function: $f_i^u\colon D \times V \rightarrow D$.

Further Specialize Strategy

- Let D be some memory domain and let d_0 be an initial memory value. Elements in the memory domain recall facts about the history of play so far.
- A strategy for player i is a function $f_i: \left(2^{\mathcal{I} \cup \mathcal{O}}\right)^* \cdot 2^{\mathcal{I}} \rightarrow 2^{\mathcal{O}}$.
- We look to replace $\left(2^{\mathcal{I} \cup \mathcal{O}}\right)^*$ by some (finite) domain D. Then, instead of considering $\left(2^{\mathcal{I} \cup \mathcal{O}}\right)^*$ we could consider $D \times 2^{\mathcal{I} \cup \mathcal{O}}$.
- The strategy becomes $f_i: D \times 2^{\mathcal{I}} \rightarrow D \times 2^{\mathcal{O}}$.

From Strategy to System

Consider a strategy $f_0 : D \times 2^{\mathcal{I}} \rightarrow D \times 2^{\mathcal{O}}$ and let $d_0 \in D$ be the initial memory value.

Construct the machine $M = \langle \Sigma, \Delta, D, \delta, d_0, L \rangle$ with:

$\Sigma = 2^{\mathcal{I}}$

$\Delta = 2^{\mathcal{O}}$

$\delta(d, i\,) = f_0(d, i) \Downarrow_1$

$\mathrm{L}(d, i\,) = f_0(d, i) \Downarrow_2$

What's the memory domain in the cases we've seen?

Winning → Realizability

Consider a run $\mathrm{r} = \mathrm{q}_0, q_1, \ldots$ over $\mathrm{w} = \sigma_0, \sigma_1, \ldots$ and the corresponding computation $\mathrm{c} = \big(\sigma_0, L(q_0, \sigma_0)\big), \big(\sigma_1, L(q_1, \sigma_1)\big), \ldots$ of M.

i. For every $i \in 2^{\mathcal{I}}$ there is $\mathrm{o} \in 2^{\mathcal{O}}$ s.t. (i, o) is winning.
ii. By f winning c satisfies the formula.

Realizability → Winning

Take a machine M and use it to construct the winning strategy.
A play in the game is a computation of the machine.

Memorize Intermediate Values

```
fix (greatest := $V$)
    fix (least := $p \wedge cpre$(greatest)
        least := least $\vee$ $cpre$(least)
    end // fix least
    greatest := least
end // fix greatest
```

```
fix (greatest := $V$)
    $cY \coloneqq 0$;
    fix (least := $p \wedge cpre$(greatest)
        y[$cY$]:= least;
        least := least $\vee$ $cpre$(least)
        $cY \coloneqq cY + 1$;
    end // fix least
    greatest := least
end // fix greatest
```

Construct the Realizing Machine

- Given $G = \langle 2^{\mathcal{I}\cup\mathcal{O}} \cup \left(2^{\mathcal{I}\cup\mathcal{O}} \times 2^{\mathcal{I}}\right), 2^{\mathcal{I}\cup\mathcal{O}} \times 2^{\mathcal{I}}, 2^{\mathcal{I}\cup\mathcal{O}}, E, \square\Diamond p\rangle$.

$$E = \{\left((i,o),(i,o,i')\right), \left((i,o,i'),(i',o')\right)\}$$

- Construct a $M = \langle 2^{\mathcal{I}}, 2^{\mathcal{O}}, 2^{\mathcal{I}\cup\mathcal{O}}, \delta, s_0, L\rangle$:

$$\delta\left((i,o),i'\right) = \begin{cases} \{(i',o') \mid (i',o') \text{ is winning}\} & (i,o) \in p \\ \{(i',o') \mid (i',o') \in \mathrm{y}[\leq j]\} & (i,o) \in \mathrm{y}[j+1] \end{cases}$$

Summary

- Starting from an LTL formula φ, construct the game $G = \langle 2^{\mathcal{I}\cup\mathcal{O}} \cup \left(2^{\mathcal{I}\cup\mathcal{O}} \times 2^{\mathcal{I}}\right), 2^{\mathcal{I}\cup\mathcal{O}} \times 2^{\mathcal{I}}, 2^{\mathcal{I}\cup\mathcal{O}}, E, \varphi\rangle$.
- Compute the set win.
- If for every $i \in 2^{\mathcal{I}}$ there is $o \in 2^{\mathcal{O}}$ such that $(i, o) \in$win then declare φ realizable.
- Extract from the winning strategy a realizing Machine.

- But we only know to solve reachability/safety and Büchi games.
- What about general LTL?

Bibliography

1. Infinite Games (R. Mazala), in Automata, Logic, and Infinite-Games (Eds., E. Grädel, W. Thomas, and T. Wilke), *Springer-Verlag*, 2002.
2. Supervisory control of a class of discrete event processes (P. J. Ramadge and W. M. Wonham), *SIAM J. Control and Optimization*, Vol. 25, No. 1, pp. 206-230, 1987.
3. On the Synthesis of Discrete Controllers for Timed Systems (O. Maler, A. Pnueli, and J. Sifakis), *STACS 1995*: 229-242.
4. An $O(n^2)$ time algorithm for alternating Büchi games (K. Chatterjee and M. Henzinger), *SODA 2012*: 1386-1399.

Lectures Outline

- Introduction
- Automata and Linear Temporal Logic
- Games and Synthesis
- General LTL Synthesis
- Bypassing Determinization
- Current Research Directions

From Logic to Graphs?

How to embed the logical winning condition into the graph notation?

Automata as Acceptors

Nondeterministic Büchi Automata

- Systems with discrete states.
- Formally, $A = \langle \Sigma, Q, \delta, q_0, \boxed{\alpha} \rangle$, where
  - Σ – a finite input alphabet.
  - Q – a finite set of states.
  - $\delta: Q \times \Sigma \to 2^Q$ – a transition function. Associates with state and input letter a set of successor states.
  - q_0 – an initial state.
  - $\boxed{\alpha \subseteq Q \text{– a set of accepting states.}}$
- An input word $w = \sigma_0, \sigma_1, \ldots$ is a sequence of letters from Σ.
- A run $r = q_0, q_1, \ldots$ over w is a sequence of states starting from q_0 such that for every $i \geq 0$ we have $q_{i+1} \in \delta(q_i, \sigma_i)$.
- A run is accepting if for infinitely many $i \in \mathbb{N}$ we have $q_i \in \alpha$.
- A word is accepted if some run over it is accepting.
- The language of A, denoted $\mathcal{L}(A)$, is the set of words accepted by A.

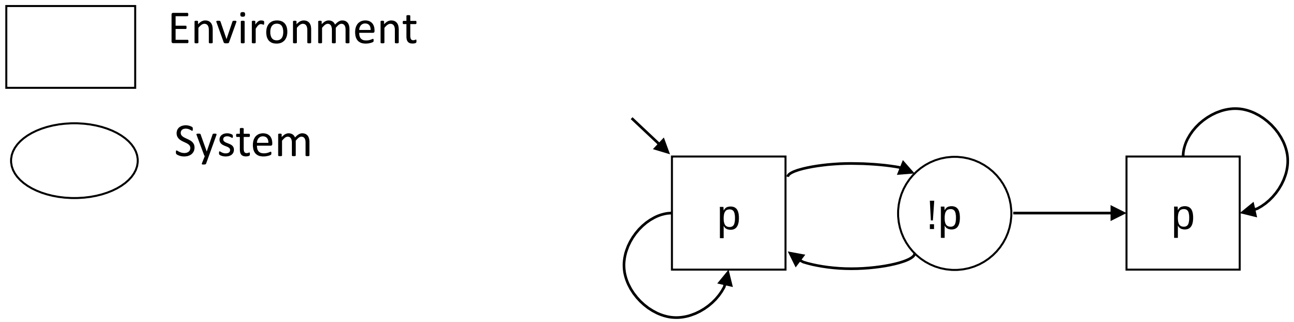
From LTL to Büchi Automata

Theorem. Given an LTL formula φ we can construct a nondeterministic Büchi automaton N_φ such that $\mathcal{L}(N_\varphi) = \mathcal{L}(\varphi)$.
The size of N_φ is exponential in the length of φ.

Intuitively, if $\mathrm{sub}(\varphi)$ is the set of subformulas of φ, a state of N_φ corresponds to a set of subformulas that are true (in an accepting run).

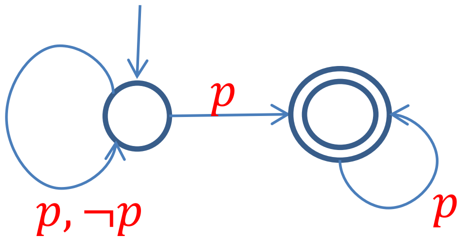
Control with Automaton Observer

Visit finitely many not-p's $\Diamond\Box p$

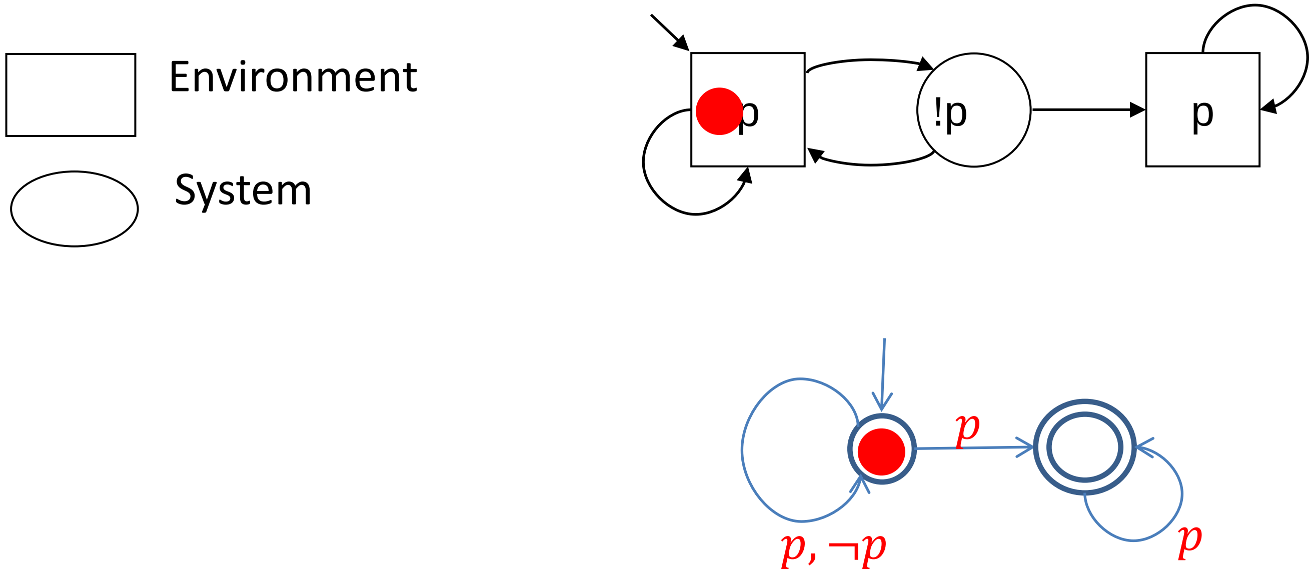


NBW for $\Diamond\Box p$

- NBW for $\varphi = \Diamond\Box p$:



Nondeterminism is bad



What went wrong?

- The automaton is nondeterministic.
- It makes predictions regarding the future and aborts runs that do not match these predictions.
- In the context of games nondeterminism is added as choice of one side:
  - If the system resolves nondeterminism, it has to find a solution that matches all possible futures.
  - If the environment resolves nondeterminism, the system must force all runs to be accepting.

Solution: Determinism

- If the automaton were deterministic, there would be no added choice!
- We create a synchronous parallel composition of the automaton with the game.
- Solve the resulting game.
- Extract system from winning strategy.

Automata as Acceptors

Nondeterministic parity Automata

- Systems with discrete states.
- Formally, $A = \langle \Sigma, Q, \delta, q_0, \boxed{\alpha} \rangle$, where
  - Σ – a finite input alphabet.
  - Q– a finite set of states.
  - $\delta: Q \times \Sigma \rightarrow 2^Q$– a transition function. Associates with state and input letter a set of successor states.
  - q_0– an initial state.
  - $\alpha: Q \rightarrow \mathbb{N}$– a ranking of states.
- An input word $w = \sigma_0, \sigma_1, \ldots$ is a sequence of letters from Σ.
- A run $r = q_0, q_1, \ldots$ over w is a sequence of states starting from q_0 such that for every $i \geq 0$ we have $q_{i+1} \in \delta(q_i, \sigma_i)$.
- A run is accepting if for the minimum rank to occur infinitely often is even.
- The language of A, denoted $\mathcal{L}(M)$, is the set of words accepted by A.

Synchronous Composition of Games

- Consider a game $G = \langle V, V_0, V_1, E, \varphi \rangle$ and a deterministic (with respect to entire alphabet Σ) automaton $A_\varphi = \langle \Sigma, D, \delta, d_0, \beta \rangle$.
- Their synchronous parallel composition ($\mathrm{G} \parallel A_\varphi$) is the game, $\hat{G} = \langle \hat{V}, \hat{V}_0, \hat{V}_1, \hat{E}, \gamma \rangle$ where:
  - $\hat{V} = D \times V$ – a new node holds a game node and an automaton state..
  - $\hat{E} = \{(d, v), (d', v') \mid (v, v') \in E$ and $d' = \delta(d, L(v))\}$ – the transitions of the automaton are updated.
  - $\gamma(d, v) = \beta(d)$ – acceptance only considers the acceptance of the automaton.
- The results is a parity game.

Deterministic Automata Work!

Theorem. P0 wins G with winning condition φ iff P0 wins G ∥ A_φ, where A_φ is a deterministic automaton for φ.

⇒ If P0 wins G all she has to do in G ∥ A_φ is to use the same strategy. Every play in G ∥ A_φ corresponds to a play in G and the unique run of A_φ that reads this play. But the play satisfies φ, so the run must be accepting. So the play in G ∥ A_φ is winning for P0 as well.

⇐ If P0 wins G ∥ A_φ she can use the states of A_φ as (part of) the memory in G. She will then be able to use the winning strategy from G ∥ A_φ. Now, a play in G corresponds to an accepting run of A_φ. But then the play satisfies φ, which means that P0 wins.

Two tiny issues …

- How do we get a deterministic parity automata for LTL?
- How do we solve a parity games?

Deterministic Automata

- Well, the answer is simple: construct a nondeterministic automaton and determinize it!
- Starting from an automaton with n states:
  - Create an automaton with $O((n!)^2)$ states and $2n$ rank.
- Subset construction augmented with a tree structure. Will not be shown.

Solving parity Games

```
Func main()
    Return even_parity(0, $\emptyset$);
End // Func main
```

```
Func even_parity(i, win)
    fix (greatest := $V$)
        greateast := win $\vee$ ($\{v|\alpha(v) = i\}$ $\wedge$ $cpre$(greatest))
            if (i!=max)
                greatest := odd_parity(i+1, greatest)
    end // fix greatest
    Return greatest;
End // Func even_parity
```

```
Func odd_parity(i, win)
    fix (least := $\emptyset$)
        least:= win $\vee$ ($\{v|\alpha(v) \geq i\}$ $\wedge$ $cpre$(least) )
            if (i!=max)
                least := even_parity(i+1, least)
    end // fix least
    Return least;
End // Func odd_parity
```

Proof (Soundness)

- Suppose that win is not empty. Have the intermediate least fixpoint approximations: least_0^p, least_1^p, least_2^p, … for an odd parity p.
- Consider $v \in$ win. Let $i_1, i_3, \ldots, i_m$ be the indices such that $v \in \text{least}_{i_j}^j$. By definition of $cpre(\cdot)$, P0 can force a successor w of v. But then, either (a) for some even j we have $v \in \alpha(j)$ and w has $i'_1, i'_3, \ldots, i'_m$ such that for $j' < j$ we have $i'_{j'} \leq i'_{j'}$ or (b) there is some j such that w has $i'_1, i'_3, \ldots, i'_m$, for $j' < j$ we have $i'_{j'} = i'_{j'}$, and for j we have $i'_j < i'_j$.
- Consider an infinite path and what happens to these numbers. There must be an even priority that is "reset" infinitely often, showing that P0 wins.

```
Func odd_parity(i, win)
    fix (least := ∅)
        least:= win ∨ ($\{v|\alpha(v) \geq i\}$ ∧ $cpre$(least) )
            if (i!=max)
                least := even_parity(i+1, least)
    end // fix least
    Return least;
End // Func odd_parity
```

```
Func even_parity(i, win)
    fix (greatest := $V$)
        greateast := win ∨ ($\{v|\alpha(v) = i\}$ ∧ $cpre$(greatest))
            if (i!=max)
                greatest := odd_parity(i+1, greatest)
    end // fix greatest
    Return greatest;
End // Func even_parity
```

To Summarize

$|\varphi| = n$

- Start with a game structure G with winning condition φ.
- Construct a deterministic automaton A_φ for φ.
- Construct the product $G \parallel A_\varphi$.
- Solve the game $G \parallel A_\varphi$.
- Construct a winning strategy for $G \parallel A_\varphi$.
- Construct from the winning strategy a Mealy machine realizing φ.

$|A_\varphi| = 2^{2^{O(n \log n)}}$

$|\alpha| = 2^n$

$2^{2^{O(n^2 \log n)}}$

The problem is 2EXPTIME-complete.
- Determinization is an issue.
- Practical solutions of parity games.

Bibliography

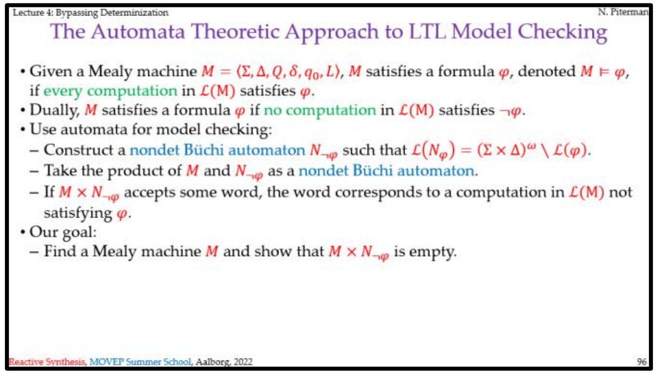
1. Reasoning About Infinite Computations (M.Y. Vardi and P. Wolper), *Information and Computation*, Vol. 115, No. 1, pp. 1-37, 1994.
2. Simple On-The-Fly Automatic Verification of Linear Temporal Logic (R. Gerth, D. Peled, M.Y. Vardi, and P. Wolper), *Protocol Specification, Testing, and Verification 1995*: 3-18.
3. On the Synthesis of a Reactive Module (A. Pnueli and R. Rosner), *POPL 1989*: 179-190.
4. Determinization of Büchi-Automata (M. Roggenbach), in Automata, Logic, and Infinite-Games (Eds., E. Grädel, W. Thomas, and T. Wilke), *Springer-Verlag*, 2002.
5. On the Complexity of ω-Automata (S. Safra), *FOCS 1998*, 319-327.
6. From Nondeterminstic Büchi and Streett Automata to Deterministic Parity Automata (N. Piterman), *Logical Methods in Computer Science*, Vol. 3, No. 3, pp. e5, 2007.
7. Algorithms for Parity Games (H. Klauck), in Automata, Logic, and Infinite-Games (Eds., E. Grädel, W. Thomas, and T. Wilke), *Springer-Verlag*, 2002.

Lectures Outline

- Introduction
- Automata and Linear Temporal Logic
- Games and Synthesis
- General LTL Synthesis
- Bypassing Determinization
- Current Research Directions

Two Ways to Avoid Determinization

- Replace by counting:
  - Search for bounded strategy.
  - Express winning through safety games.
  - Limited determinization through counting.
  - Translate to an SMT problem.
- Concentrate on simpler specifications:
  - Both system and environment are Büchi automata.
  - Enforce "deterministic" specification.
  - State-space exponential. Exponent linear.

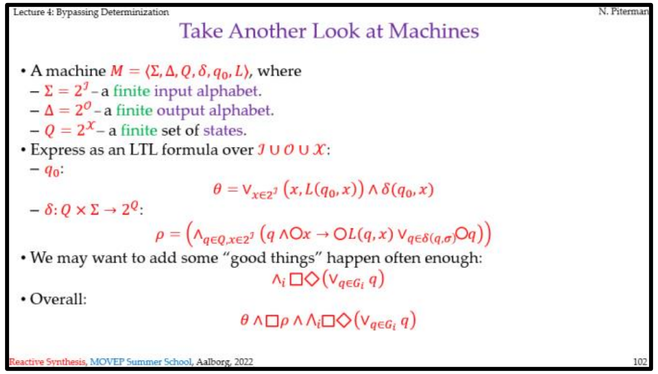


Lecture 4: Bypassing Determinization N. Piterman

The Automata Theoretic Approach to LTL Model Checking

- Given a Mealy machine $M = \langle \Sigma, \Delta, Q, \delta, q_0, L \rangle$, M satisfies a formula φ, denoted $M \vDash \varphi$, if every computation in $\mathcal{L}(M)$ satisfies φ.
- Dually, M satisfies a formula φ if no computation in $\mathcal{L}(M)$ satisfies $\neg\varphi$.
- Use automata for model checking:
  - Construct a nondet Büchi automaton $N_{\neg\varphi}$ such that $\mathcal{L}(N_\varphi) = (\Sigma \times \Delta)^\omega \setminus \mathcal{L}(\varphi)$.
  - Take the product of M and $N_{\neg\varphi}$ as a nondet Büchi automaton.
  - If $M \times N_{\neg\varphi}$ accepts some word, the word corresponds to a computation in $\mathcal{L}(M)$ not satisfying φ.
- Our goal:
  - Find a Mealy machine M and show that $M \times N_{\neg\varphi}$ is empty.

Reactive Synthesis, MOVEP Summer School, Aalborg, 2022 96



Lecture 4: Bypassing Determinization N. Piterman

Take Another Look at Machines

- A machine $M = \langle \Sigma, \Delta, Q, \delta, q_0, L \rangle$, where
  - $\Sigma = 2^{\mathcal{I}}$ – a finite input alphabet.
  - $\Delta = 2^{\mathcal{O}}$ – a finite output alphabet.
  - $Q = 2^{\mathcal{X}}$ – a finite set of states.
- Express as an LTL formula over $\mathcal{I} \cup \mathcal{O} \cup \mathcal{X}$:
  - q_0:

$$\theta = \vee_{x \in 2^{\mathcal{I}}} \left(x, L(q_0, x)\right) \wedge \delta(q_0, x)$$

  - $\delta : Q \times \Sigma \rightarrow 2^Q$:

$$\rho = \left(\wedge_{q \in Q, x \in 2^{\mathcal{I}}} \left(q \wedge \bigcirc x \rightarrow \bigcirc L(q,x) \vee_{q \in \delta(q,\sigma)} \bigcirc q\right)\right)$$

- We may want to add some "good things" happen often enough:

$$\wedge_i \Box \Diamond \left(\vee_{q \in G_i} q\right)$$

- Overall:

$$\theta \wedge \Box \rho \wedge \wedge_i \Box \Diamond \left(\vee_{q \in G_i} q\right)$$

Reactive Synthesis, MOVEP Summer School, Aalborg, 2022 102

The Automata Theoretic Approach to LTL Model Checking

- Given a Mealy machine $M = \langle \Sigma, \Delta, Q, \delta, q_0, L \rangle$, M satisfies a formula φ, denoted $M \vDash \varphi$, if every computation in $\mathcal{L}(\mathrm{M})$ satisfies φ.
- Dually, M satisfies a formula φ if no computation in $\mathcal{L}(\mathrm{M})$ satisfies $\neg\varphi$.
- Use automata for model checking:
  - Construct a nondet Büchi automaton $N_{\neg\varphi}$ such that $\mathcal{L}(N_\varphi) = (\Sigma \times \Delta)^\omega \setminus \mathcal{L}(\varphi)$.
  - Take the product of M and $N_{\neg\varphi}$ as a nondet Büchi automaton.
  - If $M \times N_{\neg\varphi}$ accepts some word, the word corresponds to a computation in $\mathcal{L}(\mathrm{M})$ not satisfying φ.
- Our goal:
  - Find a Mealy machine M and show that $M \times N_{\neg\varphi}$ is empty.

Nondeterministic Büchi Automata

- Systems with discrete states.
- Formally, $A = \langle \Sigma, Q, \delta, q_0, \alpha \rangle$, where
  - Σ – a finite input alphabet.
  - Q – a finite set of states.
  - $\delta : Q \times \Sigma \rightarrow 2^Q$ – a transition function. Associates with state and an input letter a set of successor states.
  - q_0 – an initial state.
  - $\alpha \subseteq Q$ – a set of accepting states.
- An input word $\mathrm{w} = \sigma_0, \sigma_1, \ldots$ is a sequence of letters from Σ.
- A run $\mathrm{r} = \mathrm{q}_0, \mathrm{q}_1, \ldots$ over w is a sequence of states starting from q_0 such that for every $\mathrm{i} \geq 0$ we have $\mathrm{q}_{\mathrm{i}+1} \in \delta(q_i, \sigma_i)$.
- A run is accepting if for infinitely many $\mathrm{i} \in \mathbb{N}$ we have $\mathrm{q}_\mathrm{i} \in \alpha$.
- A word is accepted if some run over it is accepting.
- The language of A, denoted $\mathcal{L}(A)$, is the set of words accepted by A.

LTL Model Checking

Theorem. Given an LTL formula φ over propositions $\mathcal{I} \cup \mathcal{O}$ we can construct a nondet Büchi automaton $N_{\neg\varphi}$ over alphabet $2^{\mathcal{I}\cup\mathcal{O}}$such that $\mathcal{L}\left(N_{\neg\varphi}\right) = (2^{\mathcal{I}\cup\mathcal{O}})^{\omega} \setminus \mathcal{L}(\varphi)$.

- We have:
  - Mealy machine $M = \langle 2^{\mathcal{I}}, 2^{\mathcal{I}}, Q, \delta, q_0, L\rangle$
  - Büchi automaton $N_{\neg\varphi} = \langle 2^{\mathcal{I}\cup\mathcal{O}}, S, \rho, s_0, \alpha\rangle$
- Construct:
  - $M \times N_{\neg\varphi} = \left\langle 2^{\mathcal{I}\cup\mathcal{O}}, Q \times S, \delta', (q_0, s_0), Q \times \alpha\right\rangle$, where

$$\delta'\left((\mathrm{q}, \mathrm{s}), (i, o)\right) = \{(q', s') | \delta(s, i) = s', L(s, i) = o, \text{and } q' \in \rho(q, (i, o))\}$$

- An accepting run $\mathrm{r} = (\mathrm{q}_0, s_0), (\mathrm{q}_1, s_1), \ldots$ on word $\mathrm{w} = \sigma_0, \sigma_1, \ldots$ is exactly a computation of M accepted by $N_{\neg\varphi}$.
- But we are interested in the case that $M \vDash \varphi$ …

Analyze the Graph

- Assume that $M \times N_{\neg\varphi} = \langle 2^{\mathcal{I}\cup\mathcal{O}}, Q \times S, \delta', (q_0, s_0), Q \times \alpha \rangle$ is empty ($M \vDash \varphi$).
- Every run of $M \times N_{\neg\varphi}$ contains finitely many accepting states in $Q \times \alpha$.
- But how many?
  - Think about $M \times N_{\neg\varphi}$ as a graph.
  - If there are more than $|\alpha| \cdot |S|$ accepting states on a path then this is an accepting loop.
  - Create a proof that $M \times N_{\neg\varphi}$ is empty by adding a function $f : Q \times S \rightarrow \mathbb{N}$ such that:
    - $f(q_0, s_0) = |\alpha| \cdot |S|$
    - If for some (i, o) we have $(q', s') \in \delta'\big((\mathrm{q}, \mathrm{s}), (i, o)\big)$ then:
      - If $\mathrm{s} \in \alpha$ then $f(q, s) > f(q', s')$.
      - If $\mathrm{s} \notin \alpha$ then $f(q, s) \geq f(q', s')$.

Bounded Synthesis

- Remember, given φ (and $N_{\neg\varphi} = \langle 2^{\mathcal{I}\cup\mathcal{O}}, S, \rho, s_0, \alpha\rangle$) we want a machine M s.t. $M \vDash \varphi$.
- What if we search for a machine with at most m states?
- We can just "nondeterministically guess" its structure along with the proof that it satisfies φ.
- Create an SMT instance Γ:
  - Variables encoding transitions:
    For $\mathrm{j} \in \{1, \dots, m\}$ and $\sigma \in 2^{\mathcal{I}}$ have $tr_{\mathrm{j},\sigma} \in \{1, \dots, m\}$.
  - Variables encoding outputs:
    For $\mathrm{j} \in \{1, \dots, m\}$ and $\sigma \in 2^{\mathcal{I}}$ have $l_{j,\sigma} \in 2^{\mathcal{O}}$.
  - Variables encoding Büchi proof:
    For $\mathrm{j} \in \{1, \dots, m\}$ and $s \in S$ have $f_{j,s} \in \{0, \dots, m \cdot |S|, \top\}$ ($\top > \top$ and for all k,$\top > \mathrm{k}$).
  - Add constraints:
    $f_{0,s_0} \neq \top$
    If $s' \in \rho(s, \sigma, l_{j,\sigma})$ and $s \in \alpha$ then $f_{j,s} > f_{tr_{j,\sigma},s'}$.
    If $s' \in \rho(s, \sigma, l_{j,\sigma})$ and $s \notin \alpha$ then $f_{j,s} \geq f_{tr_{j,\sigma},s'}$.
- If Γ is satisfiable there exists a machine of size at most m realizing φ and it can be extracted from the satisfying assignment.

Advantages

- Simple structure of states.
  - Replace the tree structure over sets of states by a function from states to ranks.
  - Determinization is a challenge for implementation.
- Safety games compared with parity games.
  - Solution of safety games is much simpler.
  - Exact complexity and practical solving of parity games are interesting open problems.
- Search for small machines first.
  - By increasing the bound gradually we can ensure to find small implementations first (and compute less).
  - Information from failed search for small sizes can be reused for searching for larger sizes.
  - Worst case complexity is as the general technique.
- Add additional quality constraints.
  - Low number of loops …

Take Another Look at Machines

- A machine $M = \langle \Sigma, \Delta, Q, \delta, q_0, L \rangle$, where
  - $\Sigma = 2^{\mathcal{I}}$ – a finite input alphabet.
  - $\Delta = 2^{\mathcal{O}}$ – a finite output alphabet.
  - $Q = 2^{\mathcal{X}}$ – a finite set of states.
- Express as an LTL formula over $\mathcal{I} \cup \mathcal{O} \cup \mathcal{X}$:
  - q_0:

$$\theta = \bigvee_{x \in 2^{\mathcal{I}}} \big(x, L(q_0, x)\big) \wedge \delta(q_0, x)$$

  - $\delta : Q \times \Sigma \rightarrow 2^{Q}$:

$$\rho = \left(\bigwedge_{q \in Q, x \in 2^{\mathcal{I}}} \left(q \wedge \bigcirc x \rightarrow \bigcirc L(q, x) \bigvee_{q \in \delta(q,\sigma)} \bigcirc q \right) \right)$$

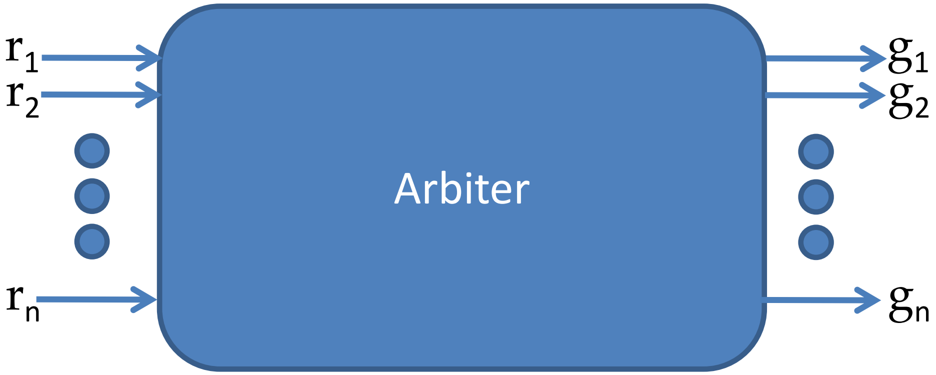
- We may want to add some "good things" happen often enough:

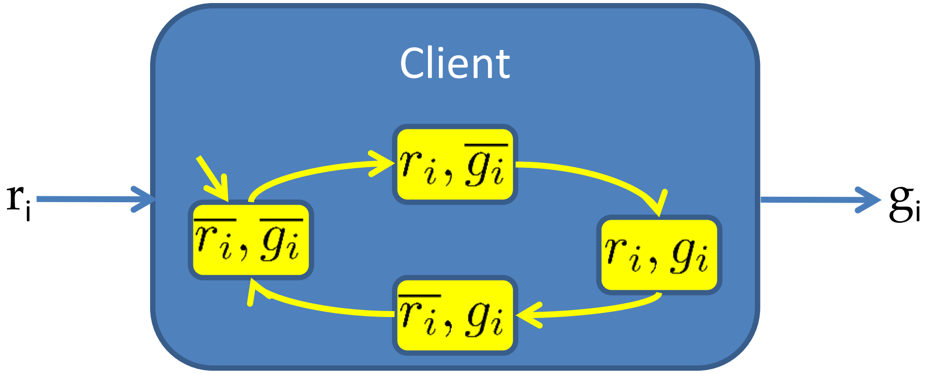
$$\bigwedge_i \Box \Diamond \left(\bigvee_{q \in G_i} q \right)$$

- Overall:

$$\theta \wedge \Box \rho \wedge \bigwedge_i \Box \Diamond \left(\bigvee_{q \in G_i} q \right)$$

Arbiter


r_1
r_2
r_n
Arbiter
g_1
g_2
g_n


r_i
Client
$\overline{r_i}, \overline{g_i}$
$r_i, \overline{g_i}$
r_i, g_i
$\overline{r_i}, g_i$
g_i

Translate to LTL

- Variables:

  $\mathcal{I} = \{r_1, r_2\}$

  $\mathcal{O} = \{g_1, g_2\}$

- Initially:

  $\neg r_1 \land \neg r_2 \land \neg g_1 \land \neg g_2$

- Transition:

  $((r_1 \land \neg g_1) \rightarrow \bigcirc r_1)$

  $((\neg r_1 \land g_1) \rightarrow \bigcirc \neg r_1)$

  If requesting, stay until granted

  Don’t reuse grants

  $((r_2 \land \neg g_2) \rightarrow \bigcirc r_2)$

  $((\neg r_2 \land g_2) \rightarrow \bigcirc \neg r_2)$

  $(\neg g_1 \lor \neg g_2)$

  Mutual exclusion

  $((g_1 \leftrightarrow \bigcirc r_1) \rightarrow (g_1 \leftrightarrow \bigcirc g_1))$

  $((g_2 \leftrightarrow \bigcirc r_2) \rightarrow (g_2 \leftrightarrow \bigcirc g_2))$

  Don’t grant w.o. request

  Don’t take away used grants

- Good things:

  $\square\Diamond(g_1 = r_1) \land \square\Diamond(g_2 = r_2)$

Separate to Assumptions and Guarantees

Environment:

- Initially:

  $\neg r_1 \wedge \neg r_2$
- Transition:

  $\big((r_1 \wedge \neg g_1) \rightarrow \bigcirc r_1\big) \wedge$

  $\big((\neg r_1 \wedge g_1) \rightarrow \bigcirc \neg r_1\big) \wedge$

  $\big((r_2 \wedge \neg g_2) \rightarrow \bigcirc r_2\big) \wedge$

  $\big((\neg r_2 \wedge g_2) \rightarrow \bigcirc \neg r_2\big)$

System:

- Initially:

  $\neg g_1 \wedge \neg g_2$
- Transition:

  $(\neg g_1 \vee \neg g_2) \wedge$

  $\big((g_1 \leftrightarrow \bigcirc r_1) \rightarrow (g_1 \leftrightarrow \bigcirc g_1)\big) \wedge$

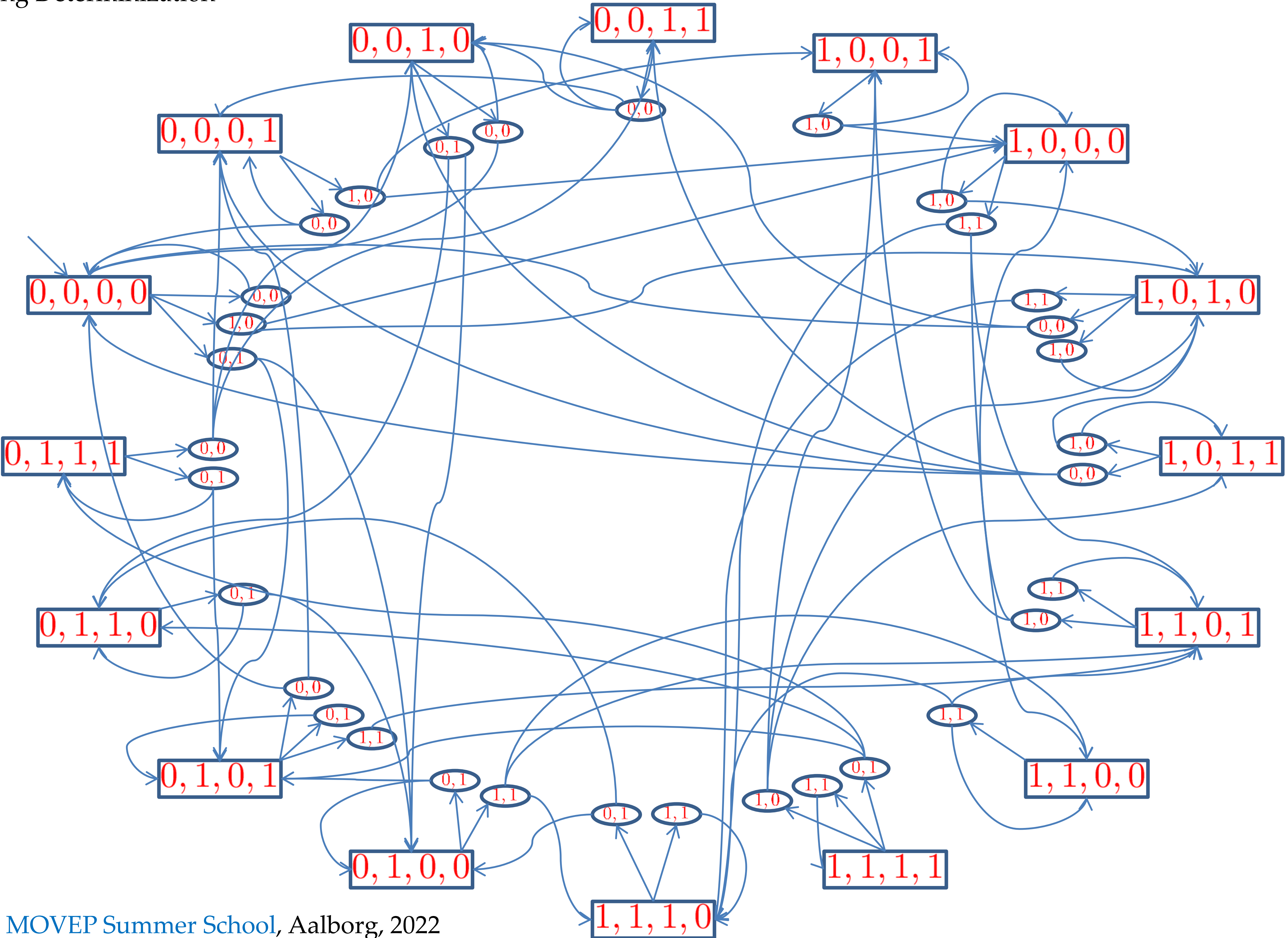
  $\big((g_2 \leftrightarrow \bigcirc r_2) \rightarrow (g_2 \leftrightarrow \bigcirc g_2)\big)$
- Good things:

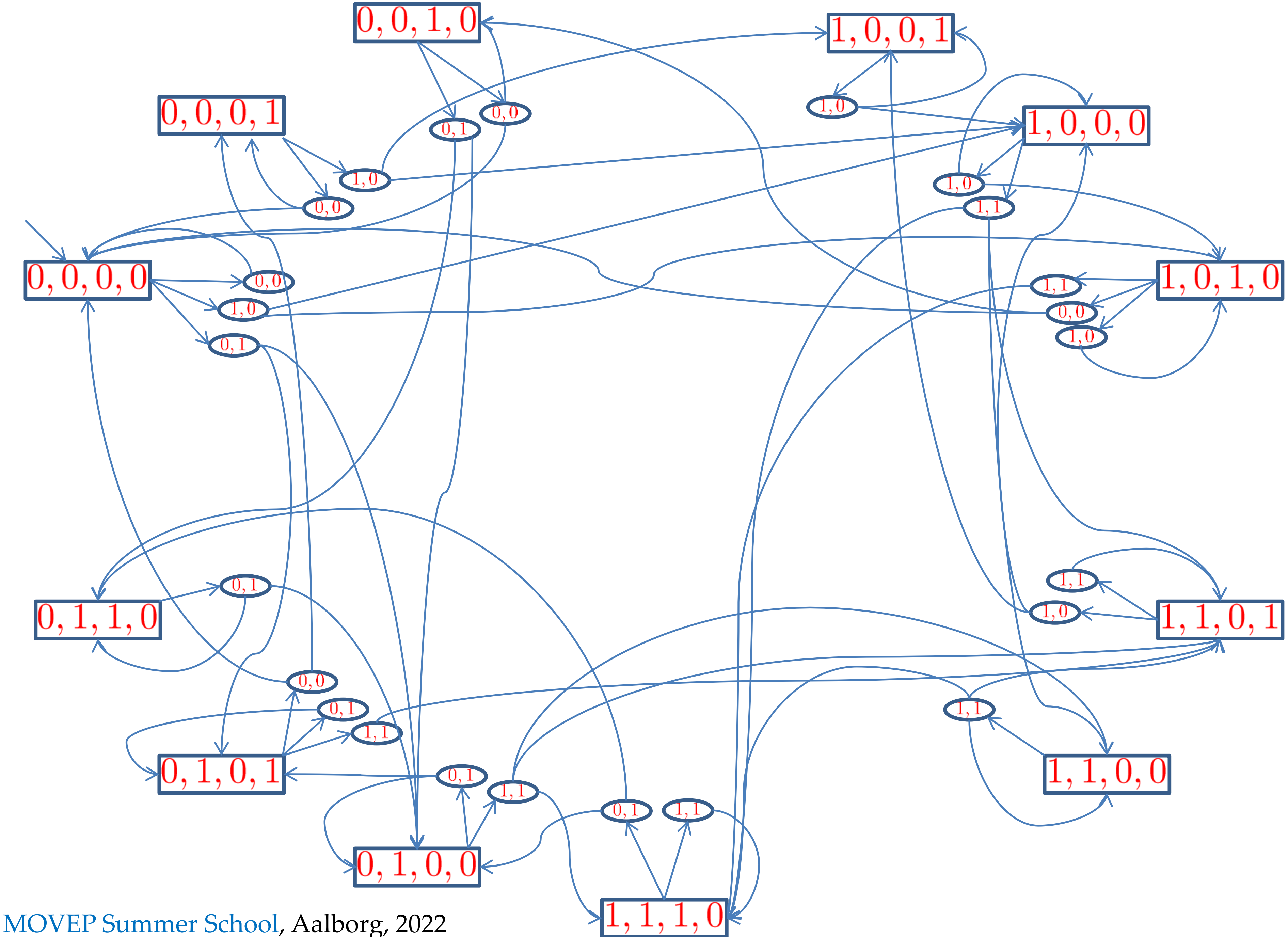
  $\square\Diamond(g_1 = r_1) \wedge \square\Diamond(g_2 = r_2)$

The Goal for Synthesis

$$(\theta_e \wedge \Box \rho_e) \rightarrow (\theta_s \wedge \Box \rho_s \wedge (\wedge_i \Box \Diamond G_i))$$

- This still does not look very simple …
- Can we do anything with the bits θ_e, θ_s, $\Box \rho_e$, and $\Box \rho_s$?
  - θ_s can be used to restrict the initial moves of P0:
    For every initial input there is initial output satisfying θ_s …
  - $\Box \rho_s$ can be used to restrict the transitions of P0.
  - What if we use θ_e and $\Box \rho_e$ to restrict the moves of P1?


0, 0, 1, 0
0, 0, 1, 1
1, 0, 0, 1
0, 0, 0, 1
1, 0, 0, 0
0, 0, 0, 0
1, 0, 1, 0
0, 1, 1, 1
1, 0, 1, 1
0, 1, 1, 0
1, 1, 0, 1
0, 1, 0, 1
1, 1, 0, 0
0, 1, 0, 0
1, 1, 1, 0
1, 1, 1, 1


0, 0, 1, 0
1, 0, 0, 1
0, 0, 0, 1
0, 1
0, 0
1, 0
1, 0, 0, 0
1, 0
0, 0
1, 0
1, 1
0, 0, 0, 0
0, 0
1, 0
0, 1
1, 0, 1, 0
1, 1
0, 0
1, 0
0, 1, 1, 0
0, 1
1, 1
1, 0
1, 1, 0, 1
0, 0
0, 1
1, 1
1, 1
0, 1, 0, 1
0, 1
1, 1
1, 1, 0, 0
0, 1
1, 1
0, 1, 0, 0
1, 1, 1, 0

What's left?

~~$(\theta_e \wedge \Box \rho_e) \rightarrow (\theta_s \wedge \Box \rho_s \wedge$~~ $(\bigwedge_i \Box \Diamond G_i))$

- This is slightly more complicated than response. We call it generalized Büchi.

Büchi:

```
fix (greatest := $V$)
    fix (least := $G \wedge cpre$(greatest)
        least := least $\vee$ $cpre$(least);
    end // fix least
 greatest := least;
end // fix greatest
```

Generalized Büchi:

```
fix (greatest := $V$)
    foreach ($G_i$)
        fix (least := $G_i \wedge cpre$(greatest)
            least := least $\vee$ $cpre$(least);
        end // fix least
        greatest := least;
    end // foreach
end // fix greatest
```

Proof (Generalized Büchi–Soundness)

Generalized Büchi:

```
fix (greatest := $V$)
   foreach ($G_i$)
      fix (least := $G_i \wedge cpre$(greatest)
         least := least $\vee$ $cpre$(least);
      end // fix least
      greatest := least;
   end // foreach
end // fix greatest
```

- Suppose that greatest is not empty. For the fixpoint to terminate, for each G_i the inner fixpoint starting from this value recomputes it.
- Let least_0^i, least_1^i, least_2^i, … be the sequence of values that least has through the computation of this last iteration for G_i.
- Consider $v \in$ greatest. Let j_0 be the index such that $v \in \text{least}_{j_0}^i$.

  By definition of $cpre(\cdot)$, P0 can force a successor w of v. But then, $w \in \text{least}_{j_1}^i$ for some $j_1 < j_0$. This shows that P0 can ensure to reach $\text{least}_0^i = G_0 \wedge cpre(\text{greatest})$.

  So it ensures a visit G_i.
- But now $\text{least}_0^i = G_i \wedge cpre(\text{greatest})$.

  So next P0 forces least_k^{i+1}, for some k and repeat this process.
- By induction, P0 can enforce $\wedge_i \Box\Diamond G_i$.

Proof (Control of Büchi - completeness)

If there is a strategy f s.t. every play compliant with it wins $\wedge_i \Box\Diamond G_i$.

Every node v from which f is winning remains in every approximation of the fixpoint greatest:

Consider some G_i. From v there is a maximum on the length of paths to reach $G_i \wedge cpre$(greatest) (König's lemma).

Prove by induction on the number of iterations in the first fixpoint that win⊆ greatest.

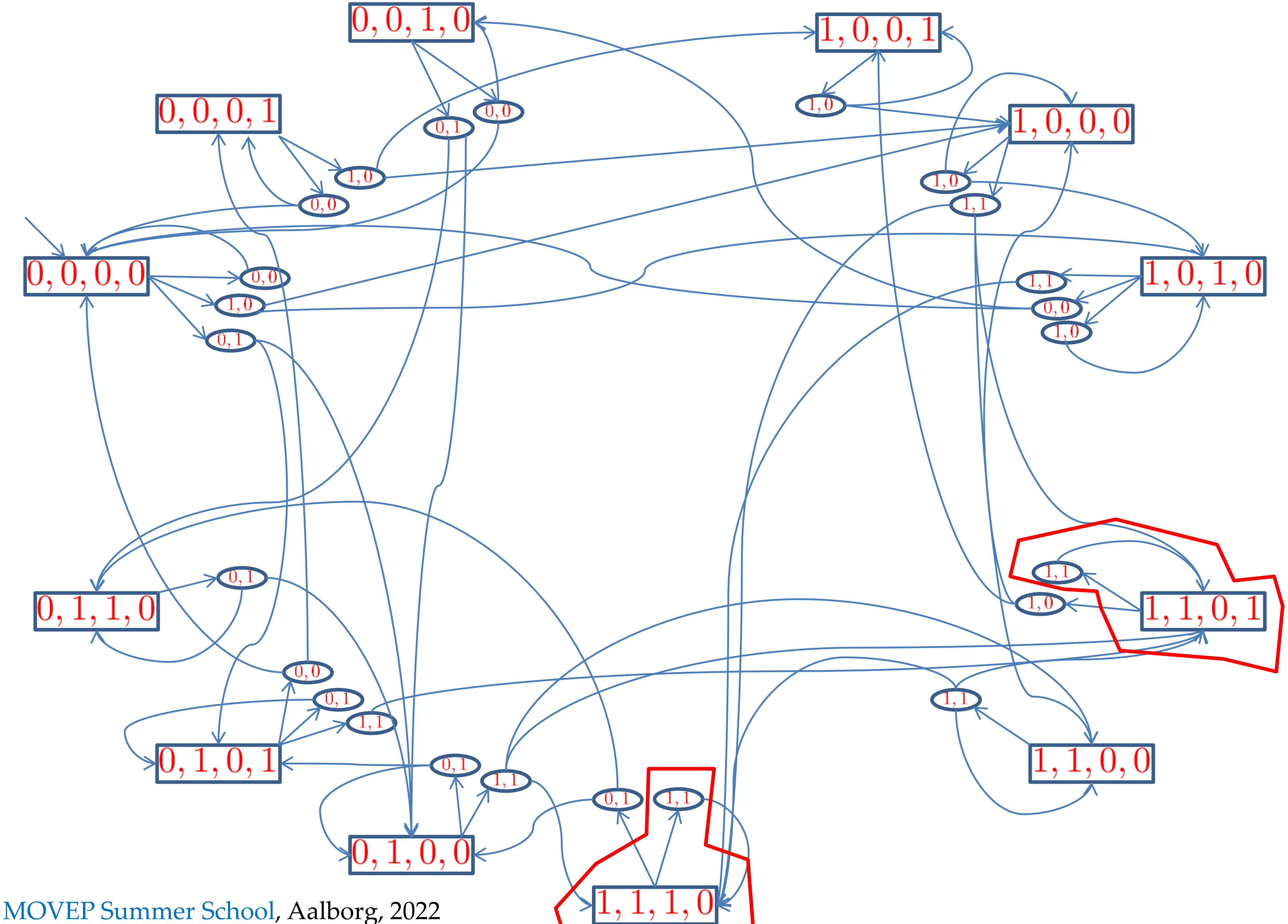
For $\text{greatest}_0 = V$ this is clear. Assu

every node $v \in$ win it must be that v

reach $G_i \wedge cpre$(win).

Generalized Büchi:

```
fix (greatest := $V$)
    foreach ($G_i$)
        fix (least := $G_i \wedge cpre$(greatest))
            least := least $\vee$ $cpre$(least);
        end // fix least
        greatest := least;
    end // foreach
end // fix greatest
```


0, 0, 1, 0
1, 0, 0, 1
0, 0, 0, 1
1, 0, 0, 0
0, 0, 0, 0
1, 0, 1, 0
0, 1, 1, 0
1, 1, 0, 1
0, 1, 0, 1
1, 1, 0, 0
0, 1, 0, 0
1, 1, 1, 0
0, 1
0, 0
1, 0
1, 1

Oops ...

- The clients do not release the bus!
- It's not only the system that has to do good things.
- The environment has to do good things as well!
- We need: $\left(\bigwedge_j \Box\Diamond A_j\right) \rightarrow \left(\bigwedge_i \Box\Diamond G_i\right)$
- We call this Generalized Reactivity (1) or GR(1).

Solving GR(1) Games

Generalized Reactivity (1):

```
fix (greatestZ := $V$)
  foreach ($G_i$)
    fix (leastY := $G_i \wedge cpre$(greatestZ))
      leastY := leastY $\vee$ $cpre$(leastY);
      foreach ($A_j$)
        fix (greatestX := $V$)
          greatestX := least $\vee$ ($\neg A_j \wedge cpre$(greatestX))
        end // fix greatestX
        leastY := leastY $\vee$ greatestX;
      end // foreach $A$
    end // fix leastY
    greatestZ :=leastY;
  end // foreach $G$
end // fix greatestZ
```

Proof (Control of GR(1) –Soundness)

Suppose that greatestZ is not empty. For each G_i the inner fixpoint starting from greatestZ recomputes greatestZ.

Let leastY_0^i, leastY_1^i, leastY_2^i, … be the sequence of values that leastY has during the last iteration. Each leastY_k^i is equal to the union of $\text{greatestX}_k^{i,1}$, $\text{greatestX}_k^{i,2}$, …, $\text{greatestX}_k^{i,m}$.

Consider $v \in$ greatestZ. Let k_0 be the minimal index such that $v \in \text{leastY}_{k_0}^i$ and let j_0 be the minimal such that $v \in \text{greatestX}_{k_0}^{i,j_0}$.

By definition of $cpre$, P0 can control to reach in one move $\text{greatestX}_{k_1}^{i,j_1}$ such that either (A) $k_1 < k_0$ or (B) $k_1 = k_0$ and $j_1 = j_0$.

In case (B), we know that $v \vDash \neg A_{j_0}$. So by playing this strategy, P0 can ensure that either some A is visited finitely often, or reach $\text{leastY}_0^i \wedge cpre(\text{greatestZ})$.

By repeating the same for all G_i P0 can enforce

$$(\wedge_j \Box \Diamond A_j) \rightarrow (\wedge_i \Box \Diamond G_i)$$

```
fix (greatestZ := $V$)
  foreach ($G_i$)
    fix (leastY := $G_i \wedge cpre$(greatestZ))
      leastY := leastY $\vee$ $cpre$(leastY);
      foreach ($A_j$)
        fix (greatestX := $V$)
          greatestX := least $\vee$ ($\neg A_j \wedge cpre$(greatestX))
        end // fix greatestX
        leastY := leastY $\vee$ greatestX;
      end // foreach $A$
    end // fix leastY
    greatestZ :=leastY;
  end // foreach $G$
end // fix greatestZ
```

Proof (Control of GR(1) – completeness sketch)

If there is a strategy f s.t. every play compliant with it wins

$$(\wedge_j \Box\Diamond A_j) \rightarrow (\wedge_i \Box\Diamond G_i)$$

Every v from which f is winning remains in every approximation of the fixpoint greatestZ:

As before, consider some G_i. From v there is a maximum on the number of visits to A_j before arriving to $G_i \wedge cpre$(win) (König's lemma).

Prove by induction on the number of iterations in the first fixpoint that win $\subseteq$ greatestZ.

For $\text{greatestZ}_0 = V$ this is clear. Assume win $\subseteq$ greatestZ_l. Then for every $v \in$ win it must be that $v \in \text{leastY}_k^i$ for some k.

```
fix (greatestZ := $V$)
  foreach ($G_i$)
    fix (leastY := $G_i \wedge cpre$(greatestZ))
        leastY := leastY $\vee$ $cpre$(leastY);
        foreach ($A_j$)
          fix (greatestX := $V$)
            greatestX := least $\vee$ ($\neg A_j \wedge cpre$(greatestX))
          end // fix greatestX
          leastY := leastY $\vee$ greatestX;
        end // foreach $A$
    end // fix leastY
    greatestZ :=leastY;
  end // foreach $G$
end // fix greatestZ
```

Memorizing Intermediate Values

Generalized Reactivity (1):

```
fix (greatestZ := $V$)
  foreach ($G_i$)
    $cY$ := $0$;
    fix (leastY := $G_i \wedge cpre$(greatestZ))
      leastY := leastY $\vee$ $cpre$(leastY);
      foreach ($A_j$)
        fix (greatestX := $V$)
          greatestX := least $\vee$ $(\neg A_j \wedge cpre$(greatestX)$)$
        end // fix greatestX
        $x[G_i][cY][A_j]$ := greatestX;
        leastY := leastY $\vee$ greatestX;
      end // foreach $A$
      $y[G_i][cY]$ := leastY;
      $cY$ := $cY + 1$;
    end // fix leastY
    greatestZ :=leastY;
  end // foreach $G$
end // fix greatestZ
```

Construct the Realizing Machine

$$\left(\theta_e \wedge \square\rho_e \wedge (\wedge_j \ \square\diamond A_j)\right) \rightarrow (\theta_s \wedge \square\,\rho_s \wedge (\wedge_i \ \square\diamond G_i))$$

- Embed θ_e, ρ_e, θ_s, and ρ_s into $G = \langle V, V_0, V_1, E, \varphi\rangle$, where

$$\varphi = \left(\wedge_j \ \square\diamond A_j\right) \rightarrow (\wedge_i \ \square\diamond G_i)$$

- Set let $\mathrm{m} = |\{G_i\}|$ and $\mathrm{n} = |\{A_i\}|$.
- Construct a machine M realizing φ:

$$M = \langle 2^{\mathcal{J}}, 2^{\mathcal{O}}, 2^{\mathcal{J}\cup\mathcal{O}} \times [1..m] \cup \{s_0\}, \rho, s_0, L\rangle:$$

$$\rho(s_0, i) = \begin{cases} \theta_s & i \vDash \theta_e \\ T & i \vDash \neg\theta_e \end{cases}$$

$$\rho\left((i,o,l), i'\right) = \begin{cases} (i', o', l \oplus 1) & (i,o) \vDash G_l \wedge (i',o') \in \text{win} \\ (i',o',l) & (i,o) \in y[G_l][cY] \wedge (i',o') \in y[G_l][<cY] \\ (i',o',l) & \begin{array}{c}(i,o) \vDash \neg A_j \wedge (i,o) \in \mathrm{x}[G_l][cY][A_j] \wedge \\ (i',o') \in y[G_l][\leq cY][\leq A_j]\end{array} \end{cases}$$

Optimizing Symbolic Runtime

Generalized Reactivity (1):

```
fix (greatestZ := $V$)
    foreach ($G_i$)
        $cY$ := $0$;
        fix (leastY := $G_i \wedge cpre$(greatestZ))
            leastY := leastY $\vee$ $cpre$(leastY);
            foreach ($A_j$)
                fix (greatestX :=$y[G_i][maxprev]$)
                    greatestX := least $\vee$ ($\neg A_j \wedge cpre$(greatestX))
                end // fix greatestX
                $x[G_i][cY][A_j]$ := greatestX;
                leastY := leastY $\vee$ greatestX;
            end // foreach $A$
            $y[G_i][cY]$ := leastY;
            $cY := cY + 1$;
        end // fix leastY
        greatestZ :=leastY;
    end // foreach $G$
end // fix greatestZ
```

Back to the Arbiter

Environment:

- Initially:

  $\neg r_1 \wedge \neg r_2$
- Transition:

  $((r_1 \wedge \neg g_1) \rightarrow \bigcirc r_1) \wedge$

  $((\neg r_1 \wedge g_1) \rightarrow \bigcirc \neg r_1) \wedge$

  $((r_2 \wedge \neg g_2) \rightarrow \bigcirc r_2) \wedge$

  $((\neg r_2 \wedge g_2) \rightarrow \bigcirc \neg r_2)$
- Good things:

$\Box\Diamond(\neg r_1 \vee \neg g_1) \wedge \Box\Diamond(\neg r_2 \vee \neg g_2)$

System:

- Initially:

  $\neg g_1 \wedge \neg g_2$
- Transition:

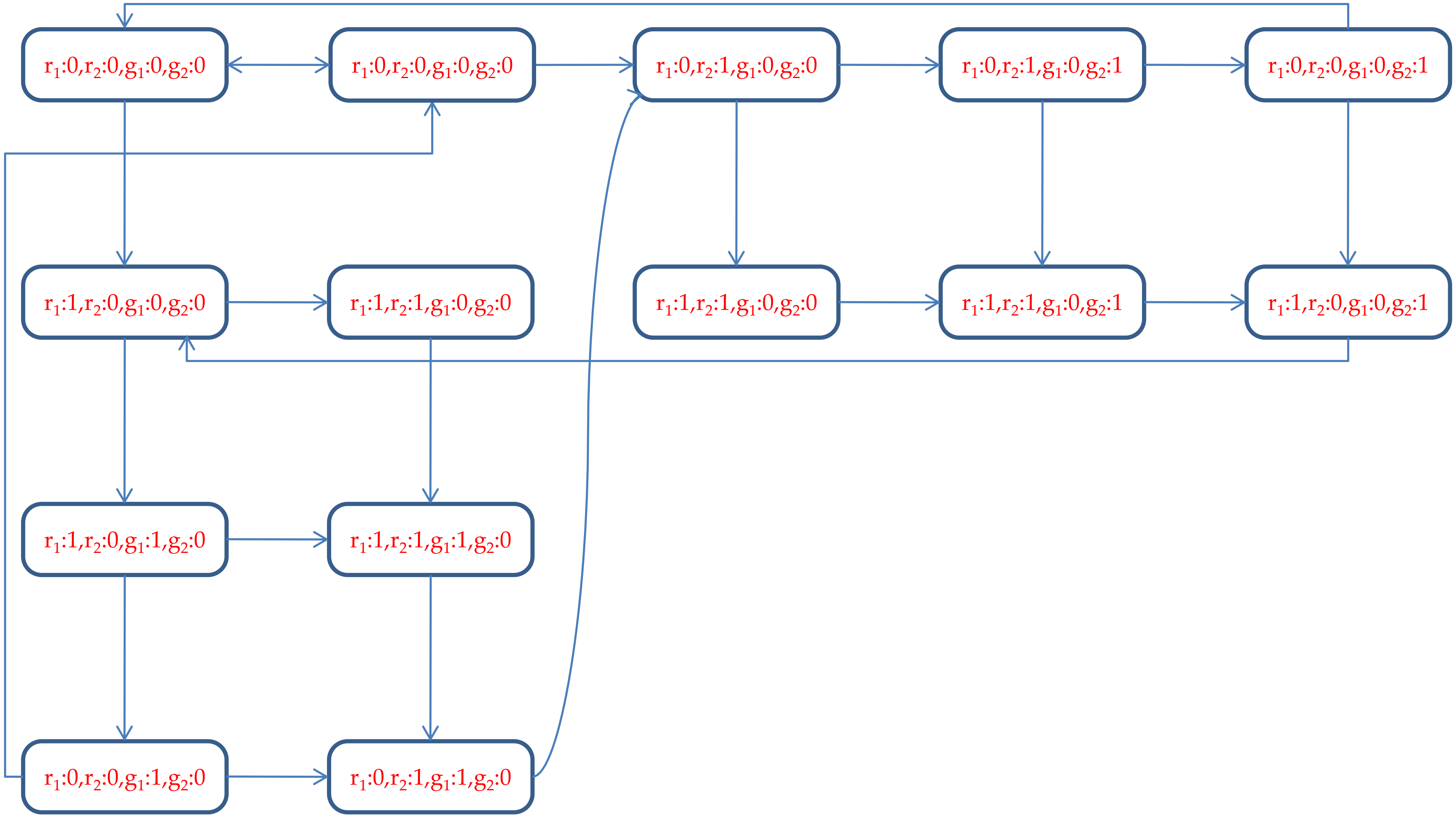
  $(\neg g_1 \vee \neg g_2) \wedge$

  $((g_1 \leftrightarrow \bigcirc r_1) \rightarrow (g_1 \leftrightarrow \bigcirc g_1)) \wedge$

  $((g_2 \leftrightarrow \bigcirc r_2) \rightarrow (g_2 \leftrightarrow \bigcirc g_2))$
- Good things:

  $\Box\Diamond(g_1 = r_1) \wedge \Box\Diamond(g_2 = r_2)$

Result of Synthesis



r_1:0,r_2:0,g_1:0,g_2:0

r_1:0,r_2:0,g_1:0,g_2:0

r_1:0,r_2:1,g_1:0,g_2:0

r_1:0,r_2:1,g_1:0,g_2:1

r_1:0,r_2:0,g_1:0,g_2:1

r_1:1,r_2:0,g_1:0,g_2:0

r_1:1,r_2:1,g_1:0,g_2:0

r_1:1,r_2:1,g_1:0,g_2:0

r_1:1,r_2:1,g_1:0,g_2:1

r_1:1,r_2:0,g_1:0,g_2:1

r_1:1,r_2:0,g_1:1,g_2:0

r_1:1,r_2:1,g_1:1,g_2:0

r_1:0,r_2:0,g_1:1,g_2:0

r_1:0,r_2:1,g_1:1,g_2:0

But why do you embed safety?

- We started from:

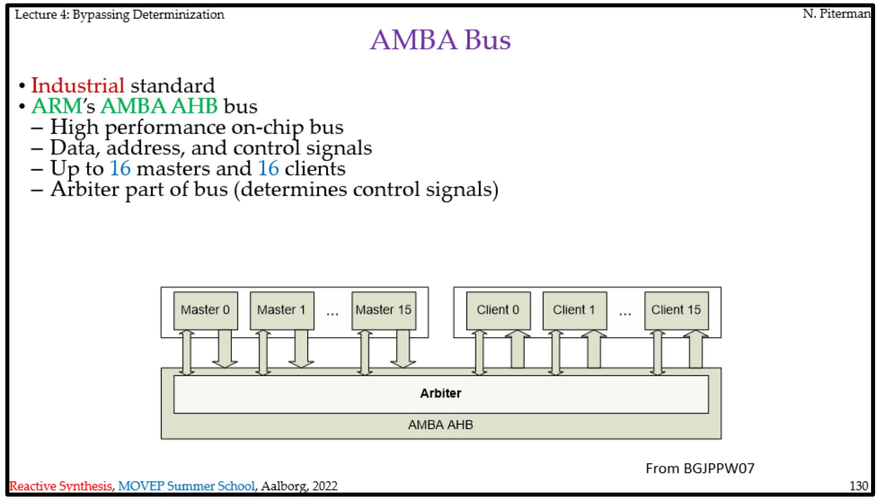
$$\left(\theta_e \wedge \Box\rho_e \wedge (\wedge_j \Box\Diamond A_j)\right) \rightarrow (\theta_s \wedge \Box\rho_s \wedge (\wedge_i \Box\Diamond G_i))$$

- And ended up with:

$$\left(\wedge_j \Box\Diamond A_j\right) \rightarrow (\wedge_i \Box\Diamond G_i)$$

  with some modifications to permitted moves in $2^{\mathcal{I}\cup\mathcal{O}}$.
- Are the two the same?
- No!
- What's the difference?
  - Realizability in our game implies realizability of the general formula.
  - Other direction is not true.

Some applications



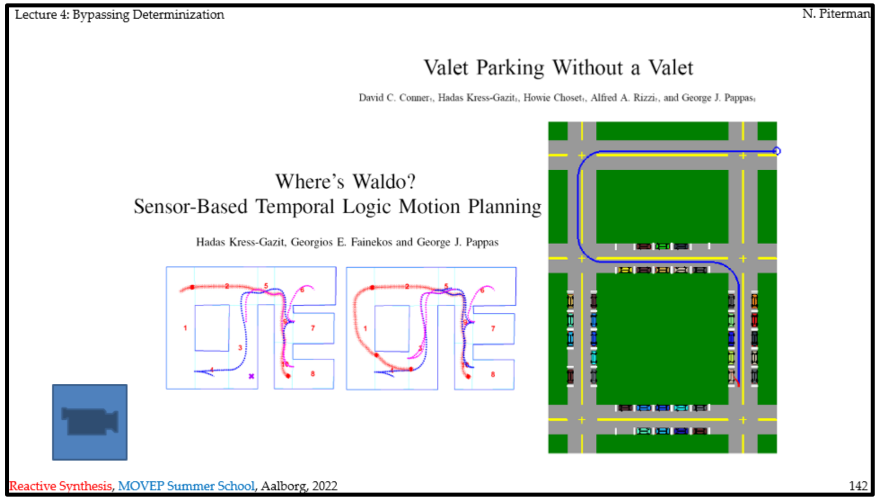
Lecture 4: Bypassing Determinization N. Piterman

AMBA Bus

- Industrial standard
- ARM's AMBA AHB bus
  - High performance on-chip bus
  - Data, address, and control signals
  - Up to 16 masters and 16 clients
  - Arbiter part of bus (determines control signals)

From BGJPPW07

Reactive Synthesis, MOVEP Summer School, Aalborg, 2022 130



Lecture 4: Bypassing Determinization N. Piterman

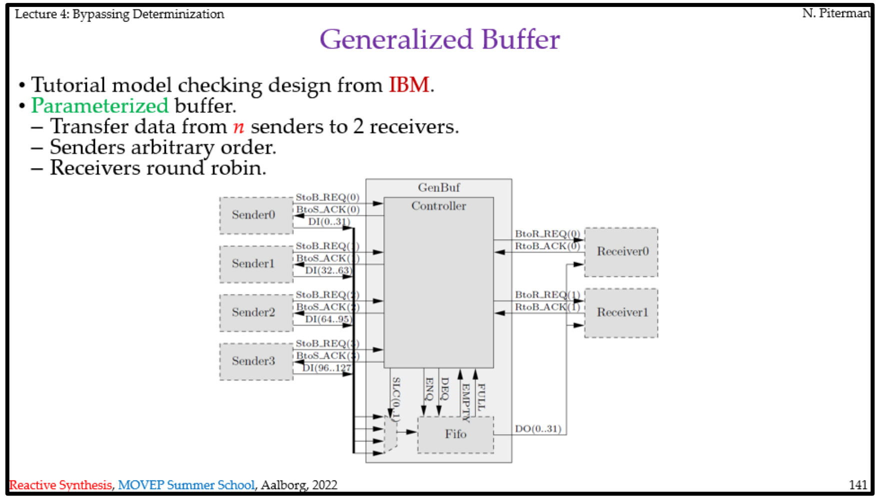
Valet Parking Without a Valet

David C. Conner, Hadas Kress-Gazit, Howie Choset, Alfred A. Rizzi, and George J. Pappas

Where's Waldo?
Sensor-Based Temporal Logic Motion Planning

Hadas Kress-Gazit, Georgios E. Fainekos and George J. Pappas

Reactive Synthesis, MOVEP Summer School, Aalborg, 2022 142

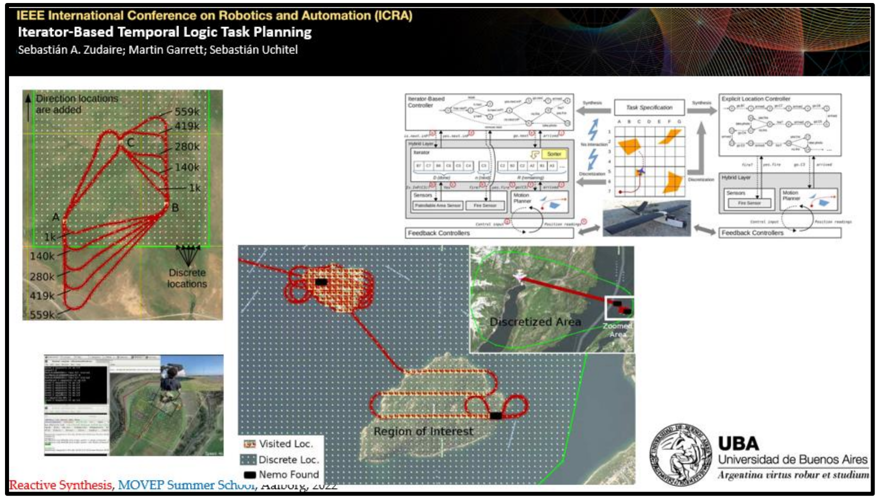


Lecture 4: Bypassing Determinization N. Piterman

Generalized Buffer

- Tutorial model checking design from IBM.
- Parameterized buffer.
  - Transfer data from n senders to 2 receivers.
  - Senders arbitrary order.
  - Receivers round robin.

Reactive Synthesis, MOVEP Summer School, Aalborg, 2022 141



IEEE International Conference on Robotics and Automation (ICRA)

Iterator-Based Temporal Logic Task Planning

Sebastián A. Zudaire; Martin Garrett; Sebastián Uchitel

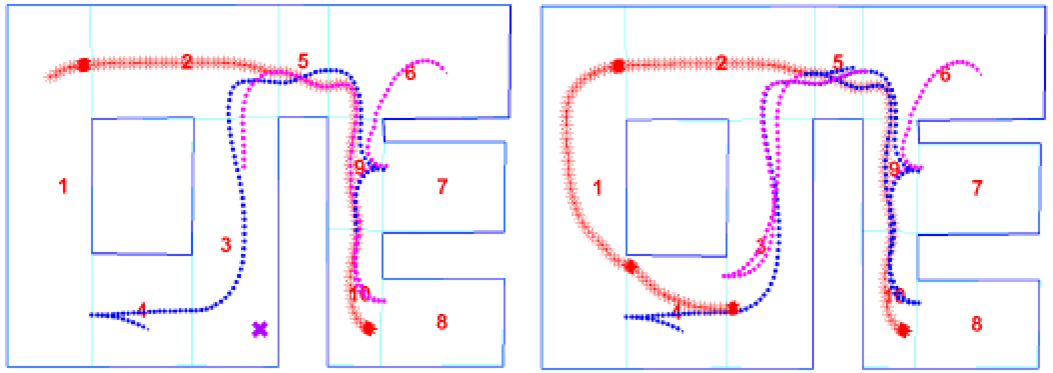
Reactive Synthesis, MOVEP Summer School, Aalborg, 2022

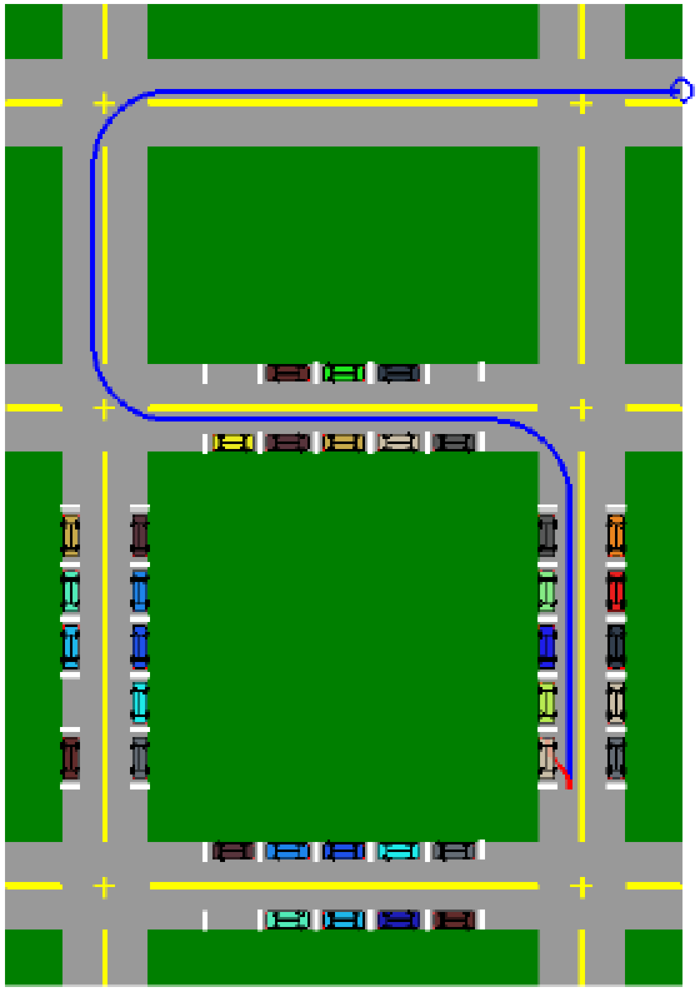
Valet Parking Without a Valet

David C. Conner†, Hadas Kress-Gazit‡, Howie Choset†, Alfred A. Rizzi†, and George J. Pappas‡

Where's Waldo? Sensor-Based Temporal Logic Motion Planning

Hadas Kress-Gazit, Georgios E. Fainekos and George J. Pappas





Automatic Synthesis of Robust Embedded Control Software

Tichakorn Wongpiromsarn, Ufuk Topcu and **Richard M. Murray**
California Institute of Technology
Pasadena, California 91125

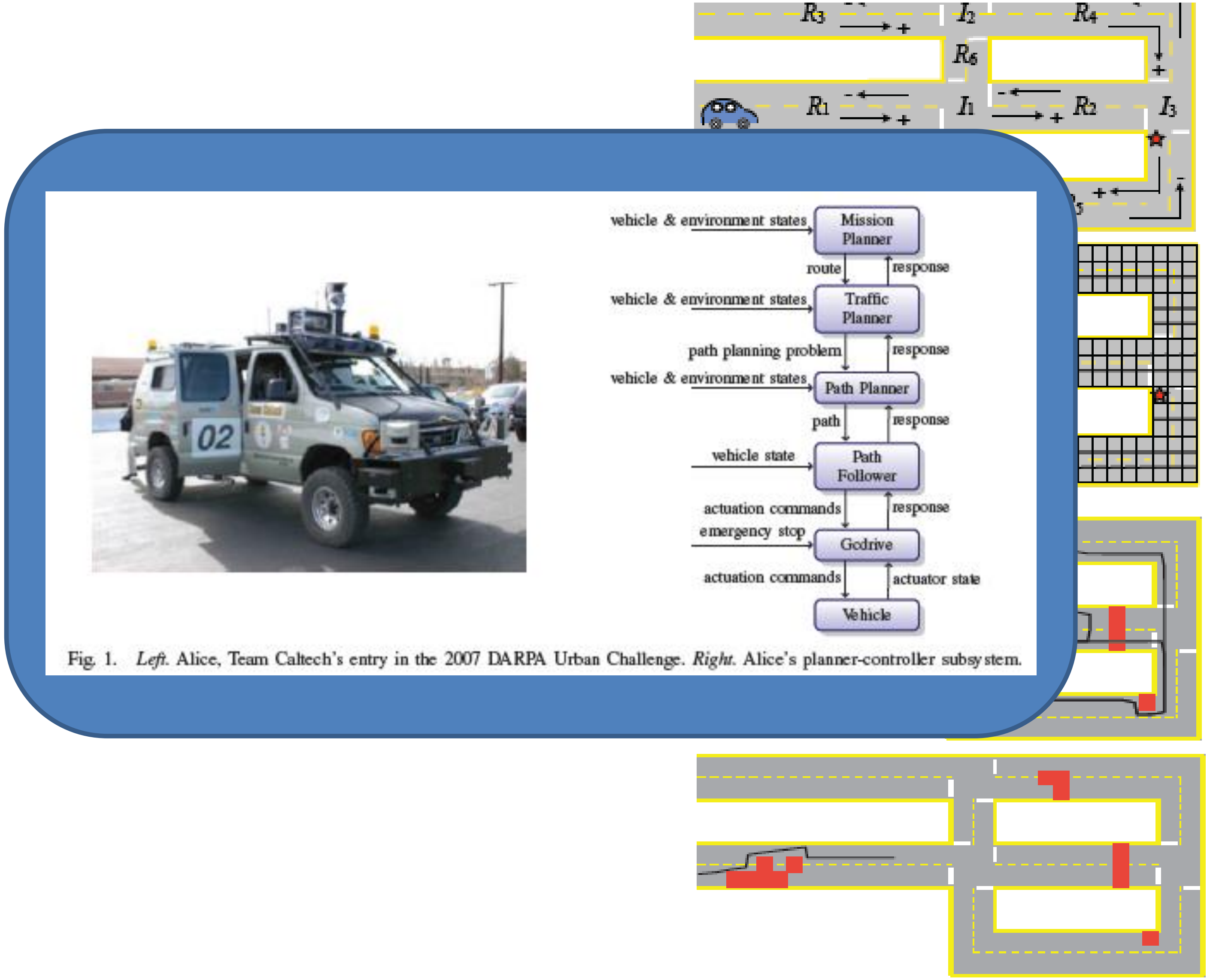
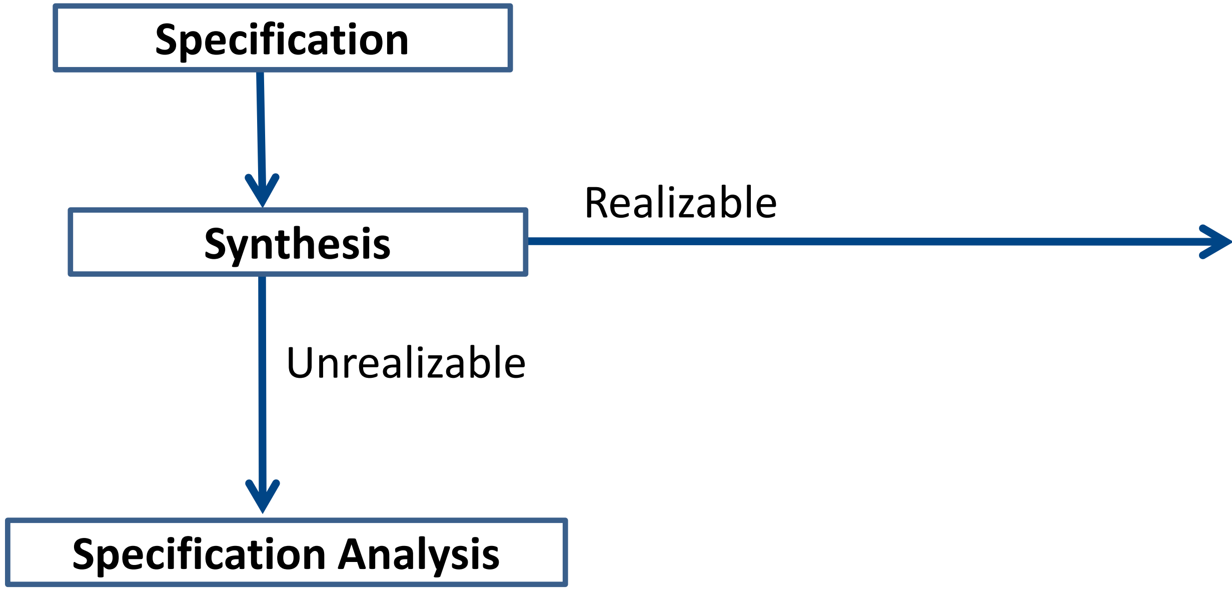


Fig. 1. *Left.* Alice, Team Caltech's entry in the 2007 DARPA Urban Challenge. *Right.* Alice's planner-controller subsystem.

Robotics Approach Overview



Specification

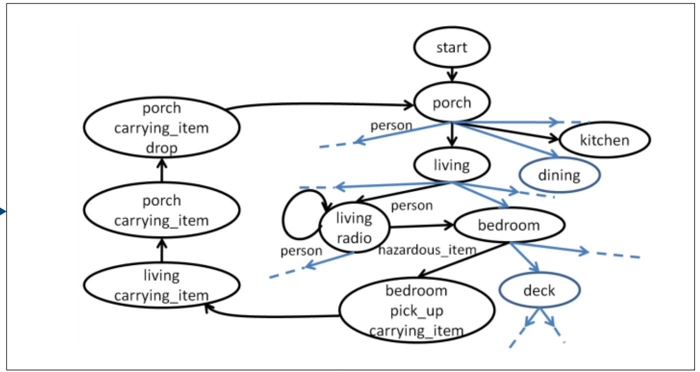
Synthesis

Realizable

Unrealizable

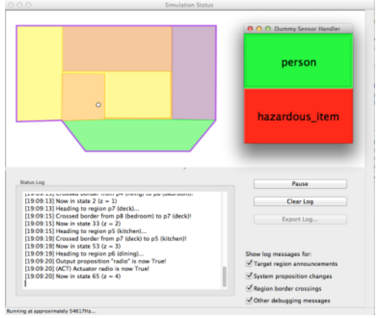
Specification Analysis

FSM



Hybrid Controller

Simulation



```
[19:09:13] Now in state 2 (z = 1)
[19:09:13] Heading to region p7 (deck)...
[19:09:15] Crossed border from p8 (bedroom) to p7 (deck)!
[19:09:15] Now in state 33 (z = 2)
[19:09:15] Heading to region p5 (kitchen)...
[19:09:19] Crossed border from p7 (deck) to p5 (kitchen)!
[19:09:19] Now in state 53 (z = 3)
[19:09:19] Heading to region p6 (dining)...
[19:09:20] Output proposition "radio" is now True!
[19:09:20] (ACT) Actuator radio is now True!
[19:09:20] Now in state 65 (z = 4)
```

Show log messages for:
☑ Target region announcements
☑ System proposition changes
☑ Region border crossings
☑ Other debugging messages

Running at approximately 54617Hz...

Physical Robot

The International Symposium of Robotics Research

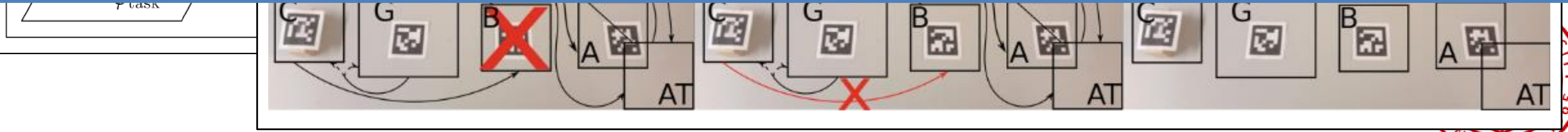
↳ ISRR 2019: **Robotics Research** pp 509–525

Automatic Encoding and Repair of Reactive High-Level

$$\varphi^{s}_{\text{t,pre}} = \bigwedge_{a\in A} \Box \left[\neg\left(\bigvee_{\sigma_p\in\sigma_{\text{pre}(a)}}\left(\bigwedge_{\sigma\in\sigma_p}\sigma\right)\right)\rightarrow\neg a\right]$$

$$\varphi^{e}_{\text{t,eff}} = \bigwedge_{a\in A}\Box\left[a\rightarrow\bigvee_{j\in\{1,\dots,k(a)\}}\left(\left(\bigwedge_{\sigma\in\sigma^{\top}_{\text{eff}^j(a)}}\bigcirc\sigma\right)\bigwedge\left(\bigwedge_{\sigma\in\sigma^{\bot}_{\text{eff}^j(a)}}\bigcirc\neg\sigma\right)\bigwedge\left(\bigwedge_{\sigma\in\sigma^{\text{stay}}_{\text{eff}^j(a)}}(\sigma\leftrightarrow\bigcirc\sigma)\right)\right)\right]$$

$$\varphi^{e}_{\text{t,no\_act}} = \Box\left[\left(\bigwedge_{a\in A}\neg a\right)\rightarrow\left(\bigwedge_{\sigma\in\Sigma}(\sigma\leftrightarrow\bigcirc\sigma)\right)\right]$$



IEEE Robotics and Automation Letters

David Gundana ; Hadas Kress-Gazit

01:23 01:00 04:58

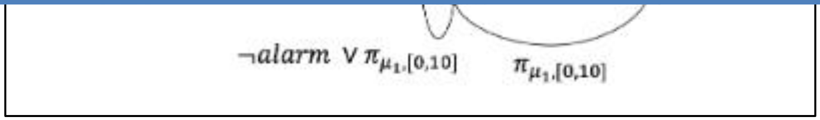
Event-Based Signal Temporal Logic Synthesis for Single and Multi-Robot Tasks

$$\Psi_{host} = G\left(lead \Rightarrow \left(\begin{array}{ll} (F_{[0,25]}(\| \mathbf{x}_{1,t} - \mathbf{x}_{cstmr,1,t} \| < 1) & \wedge \\ (\| \mathbf{x}_{1,t} - \mathbf{x}_{cstmr,1,t} \| < 1) U_{[25,60]} (\| \mathbf{x}_{1,t} - [1.75, -1] \| < 1) & \end{array} \right)\right)$$

$$\Psi_{request_k} = G\left(request_k \Rightarrow F_{[0,20]} \left(\begin{array}{ll} ((\| \mathbf{x}_{2,t} - \mathbf{x}_{cstmr,k,t} \| < 1) \vee (\| \mathbf{x}_{3,t} - \mathbf{x}_{cstmr,k,t} \| < 1)) & \wedge \\ ((\| \mathbf{x}_{4,t} - \mathbf{x}_{cstmr,k,t} \| < 1) \vee (\| \mathbf{x}_{5,t} - \mathbf{x}_{cstmr,k,t} \| < 1)) & \end{array} \right)\right)$$

$$\Psi_{collision} = G_{[0,\infty]}(\| \mathbf{x}_{i,t} - \mathbf{x}_{j,t} \| > 0.05) \ , \forall i \neq j$$

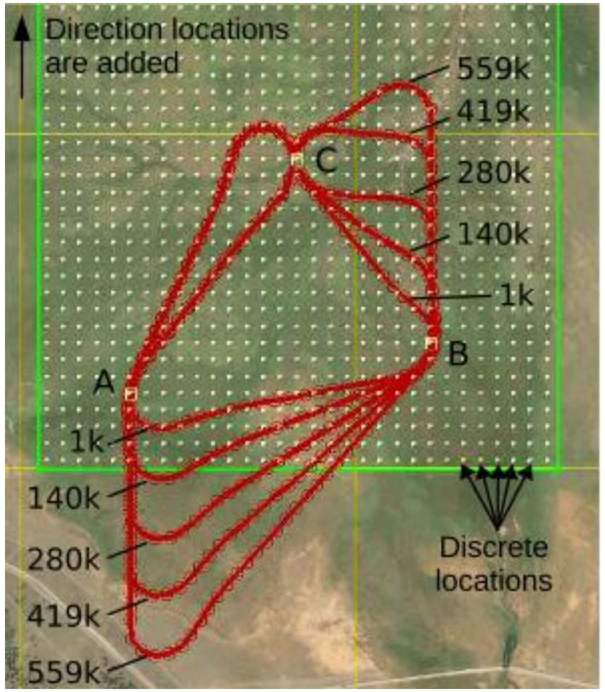
$$\Psi_{wallAvoid_i} = G_{[0,\infty]}(min(\| \mathbf{x}_{i,t} - M \|) > 0.1) \ , i = (1, 2, \ldots, 5)$$

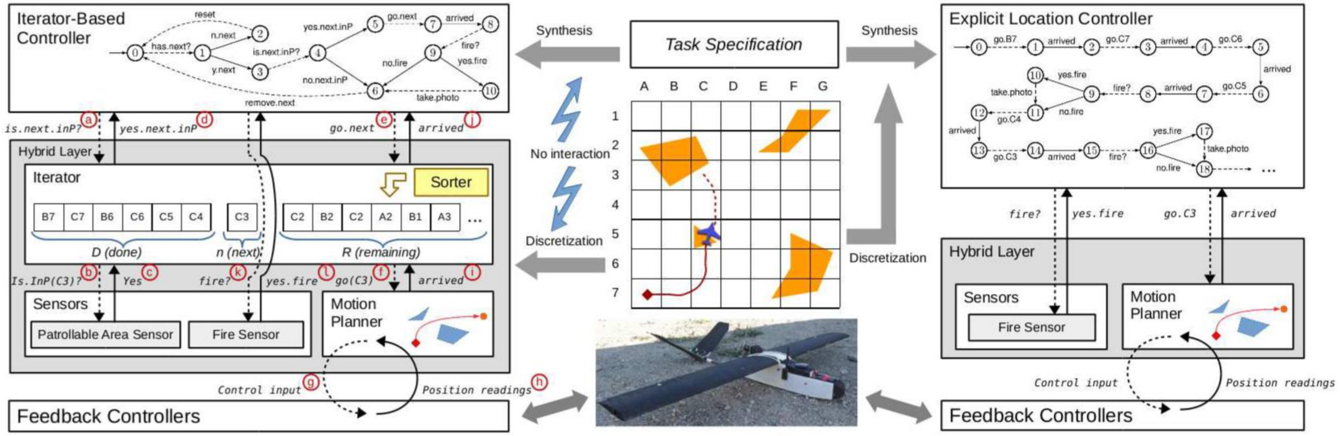


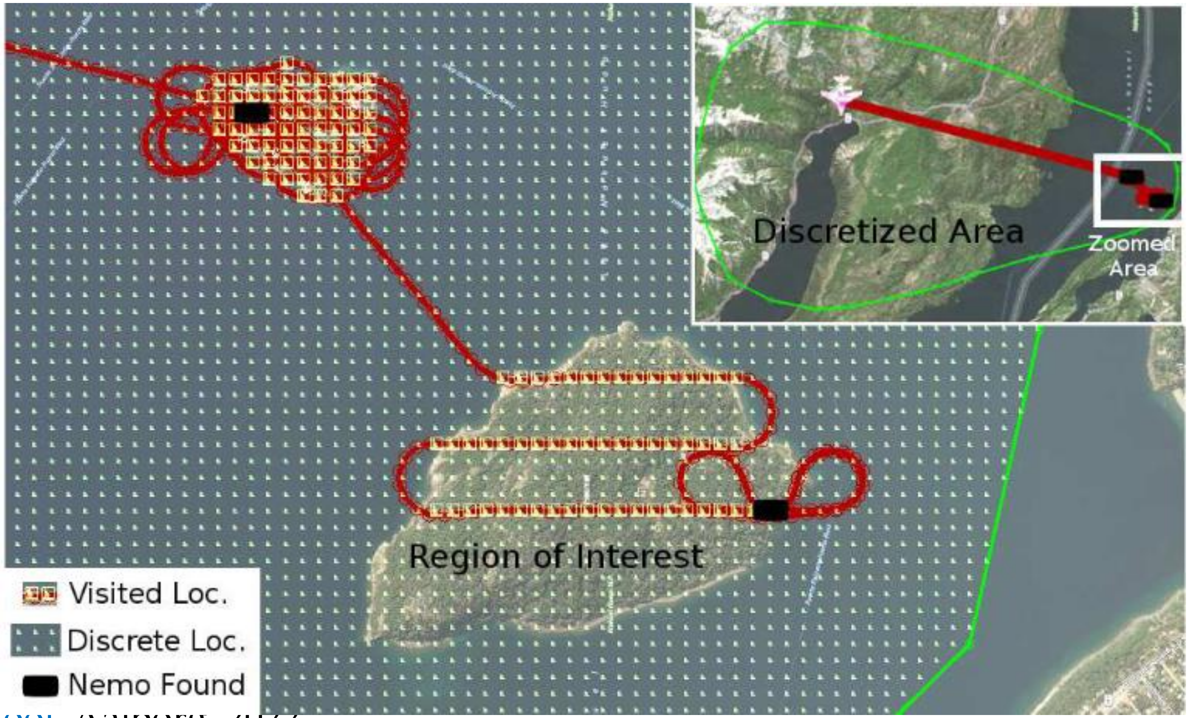
IEEE International Conference on Robotics and Automation (ICRA)

Iterator-Based Temporal Logic Task Planning

Sebastián A. Zudaire; Martin Garrett; Sebastián Uchitel

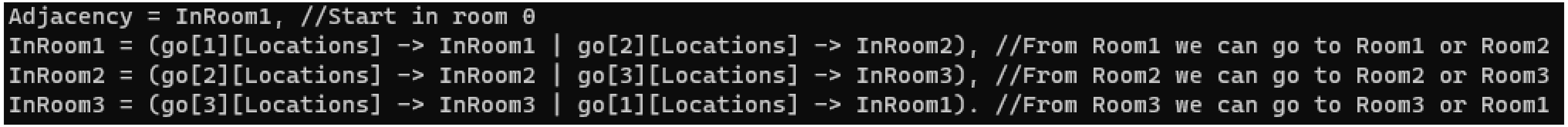




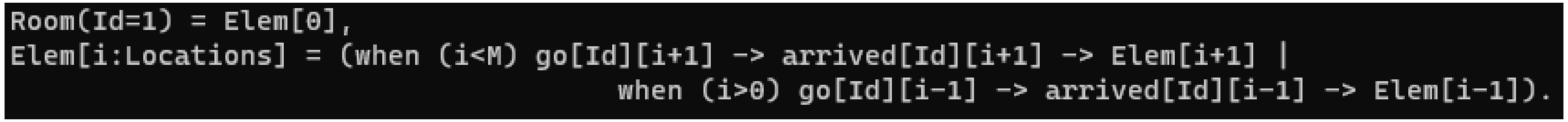


UBA
Universidad de Buenos Aires
Argentina virtus robur et studium

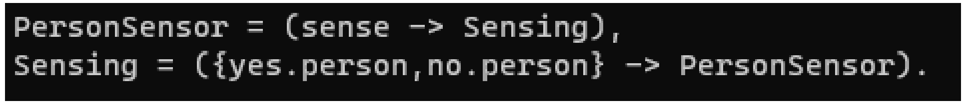
```
MovementModel = (go[Rooms][Locations] -> GoModel),
GoModel = (arrived[Rooms][Locations] -> MovementModel).
```



```
Adjacency = InRoom1, //Start in room 0
InRoom1 = (go[1][Locations] -> InRoom1 | go[2][Locations] -> InRoom2), //From Room1 we can go to Room1 or Room2
InRoom2 = (go[2][Locations] -> InRoom2 | go[3][Locations] -> InRoom3), //From Room2 we can go to Room2 or Room3
InRoom3 = (go[3][Locations] -> InRoom3 | go[1][Locations] -> InRoom1). //From Room3 we can go to Room3 or Room1
```



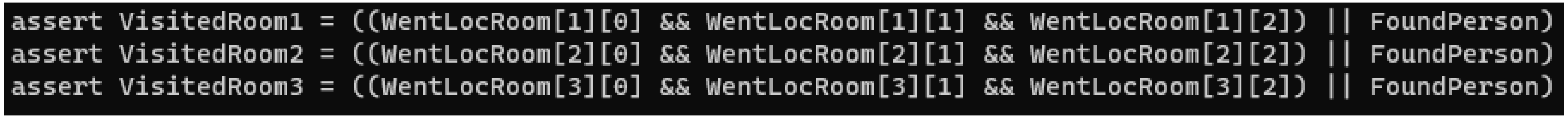
```
Room(Id=1) = Elem[0],
Elem[i:Locations] = (when (i<M) go[Id][i+1] -> arrived[Id][i+1] -> Elem[i+1] |
                                     when (i>0) go[Id][i-1] -> arrived[Id][i-1] -> Elem[i-1]).
```



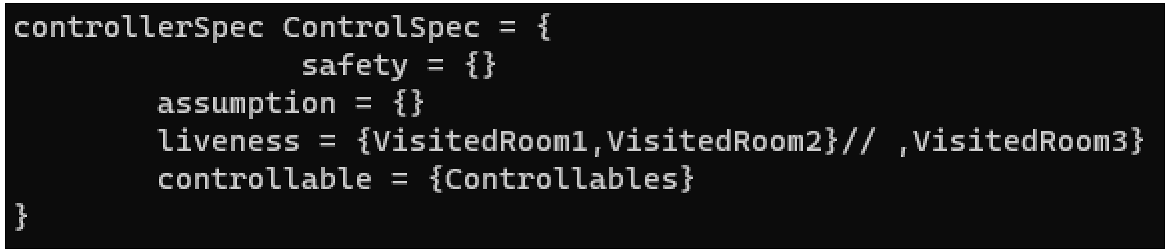
```
PersonSensor = (sense -> Sensing),
Sensing = ({yes.person,no.person} -> PersonSensor).
```

```
ltl_property SenseAtEachLoc = [](arrived[Rooms][Locations] -> (!go[Rooms][Locations] W {yes.person,no.person}))
```

```
fluent WentLocRoom[j:1..N][i:0..M] = <arrived[j][i],yes.person>
fluent FoundPerson = <yes.person,Alphabet\{yes.person}>
```



```
assert VisitedRoom1 = ((WentLocRoom[1][0] && WentLocRoom[1][1] && WentLocRoom[1][2]) || FoundPerson)
assert VisitedRoom2 = ((WentLocRoom[2][0] && WentLocRoom[2][1] && WentLocRoom[2][2]) || FoundPerson)
assert VisitedRoom3 = ((WentLocRoom[3][0] && WentLocRoom[3][1] && WentLocRoom[3][2]) || FoundPerson)
```



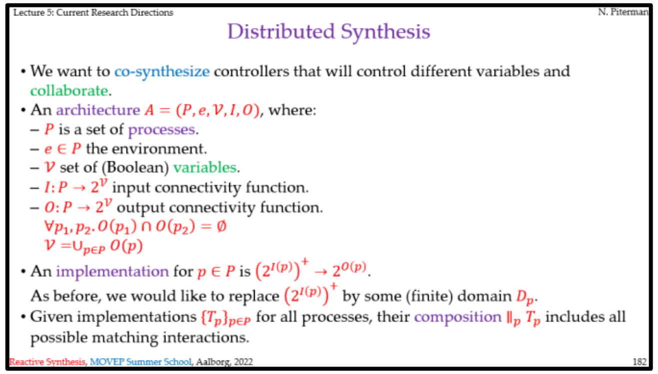
```
controllerSpec ControlSpec = {
                safety = {}
        assumption = {}
        liveness = {VisitedRoom1,VisitedRoom2}// ,VisitedRoom3}
        controllable = {Controllables}
}
```

Bibliography

1. Safraless Decision Procedures (O. Kupferman and M.Y. Vardi), *FOCS 2005*, 531-542.
2. Bounded Synthesis (B. Finkbeiner and S. Schewe), *STTT*, Vol. 15, No. 5-6, pp. 519-539, 2013.
3. Unbeast: Symbolic Bounded Synthesis (R. Ehlers), *TACAS 2011*, 272-275.
4. Synthesis of Reactive(1) Designs (R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar), *Journal of Computer and System Sciences*, Vol. 78, No. 3, 911-938, 2012.
5. Valet Parking Without a Valet (D.C. Conner, H. Kress-Gazit, H. Choset, A. Rizzi, and G.J. Pappas), *Conference on Intelligent Robots and Systems 2007*, 572-577.

Lectures Outline

- Introduction
- Automata and Linear Temporal Logic
- Games and Synthesis
- General LTL Synthesis
- Bypassing Determinization
- Current Research Directions

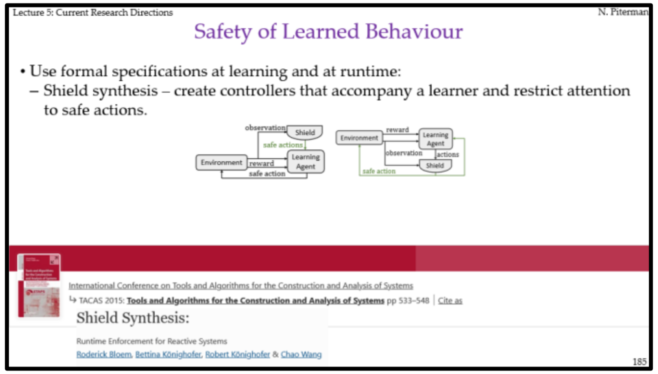


Lecture 5: Current Research Directions N. Piterman

Distributed Synthesis

- We want to co-synthesize controllers that will control different variables and collaborate.
- An architecture $A = (P, e, \mathcal{V}, I, O)$, where:
  - P is a set of processes.
  - $e \in P$ the environment.
  - $\mathcal{V}$ set of (Boolean) variables.
  - $I: P \rightarrow 2^{\mathcal{V}}$ input connectivity function.
  - $O: P \rightarrow 2^{\mathcal{V}}$ output connectivity function.
    $\forall p_1, p_2 . O(p_1) \cap O(p_2) = \emptyset$
    $\mathcal{V} = \cup_{p \in P} O(p)$
- An implementation for $p \in P$ is $\left(2^{I(p)}\right)^+ \rightarrow 2^{O(p)}$.
  As before, we would like to replace $\left(2^{I(p)}\right)^+$ by some (finite) domain D_p.
- Given implementations $\{T_p\}_{p \in P}$ for all processes, their composition $\|_p \, T_p$ includes all possible matching interactions.

Reactive Synthesis, MOVEP Summer School, Aalborg, 2022 182



Lecture 5: Current Research Directions N. Piterman

Safety of Learned Behaviour

- Use formal specifications at learning and at runtime:
  - Shield synthesis – create controllers that accompany a learner and restrict attention to safe actions.

International Conference on Tools and Algorithms for the Construction and Analysis of Systems

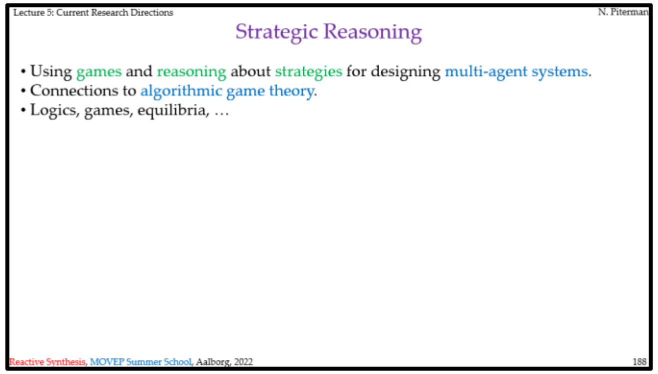
↳ TACAS 2015: **Tools and Algorithms for the Construction and Analysis of Systems** pp 533–548 | Cite as

Shield Synthesis:

Runtime Enforcement for Reactive Systems

Roderick Bloem, Bettina Könighofer, Robert Könighofer & Chao Wang

185

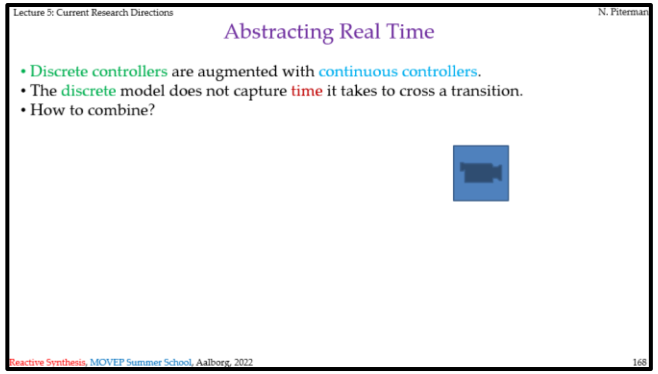


Lecture 5: Current Research Directions N. Piterman

Strategic Reasoning

- Using games and reasoning about strategies for designing multi-agent systems.
- Connections to algorithmic game theory.
- Logics, games, equilibria, …

Reactive Synthesis, MOVEP Summer School, Aalborg, 2022 188

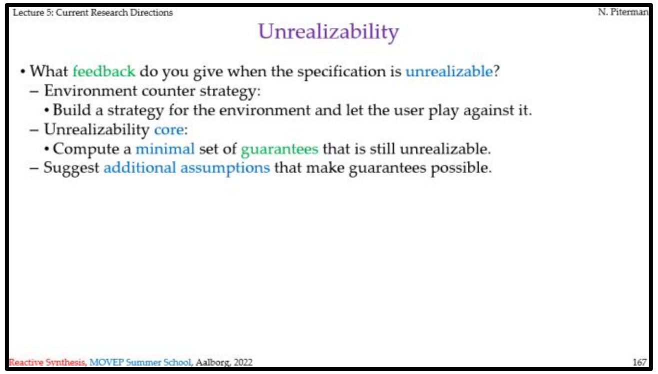


Lecture 5: Current Research Directions N. Piterman

Abstracting Real Time

- Discrete controllers are augmented with continuous controllers.
- The discrete model does not capture time it takes to cross a transition.
- How to combine?

Reactive Synthesis, MOVEP Summer School, Aalborg, 2022 168

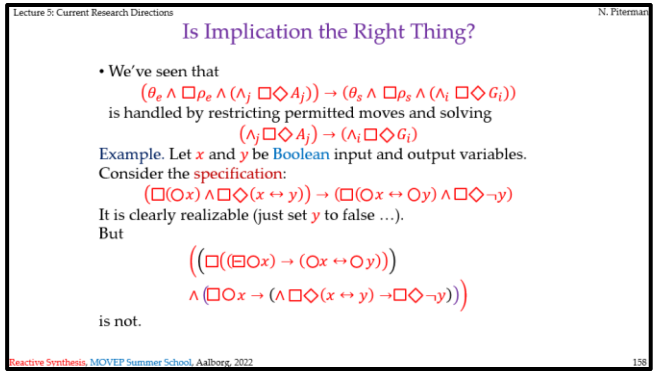


Lecture 5: Current Research Directions N. Piterman

Unrealizability

- What feedback do you give when the specification is unrealizable?
  - Environment counter strategy:
    - Build a strategy for the environment and let the user play against it.
  - Unrealizability core:
    - Compute a minimal set of guarantees that is still unrealizable.
  - Suggest additional assumptions that make guarantees possible.

Reactive Synthesis, MOVEP Summer School, Aalborg, 2022 167



Lecture 5: Current Research Directions N. Piterman

Is Implication the Right Thing?

- We've seen that

$$(\theta_e \wedge \Box\rho_e \wedge (\wedge_j \Box\Diamond A_j)) \rightarrow (\theta_s \wedge \Box\rho_s \wedge (\wedge_i \Box\Diamond G_i))$$

is handled by restricting permitted moves and solving

$$(\wedge_j \Box\Diamond A_j) \rightarrow (\wedge_i \Box\Diamond G_i)$$

Example. Let x and y be Boolean input and output variables.
Consider the specification:

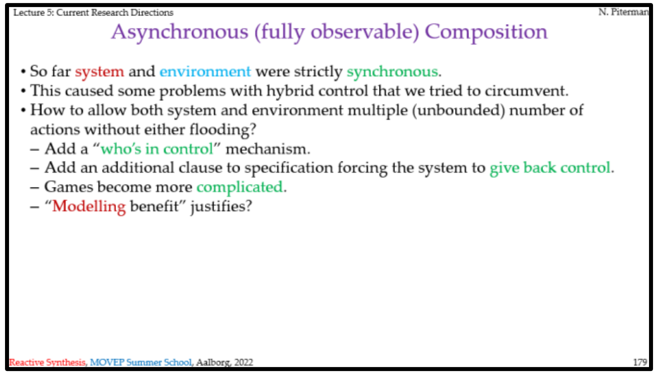
$$(\Box(\bigcirc x) \wedge \Box\Diamond(x \leftrightarrow y)) \rightarrow (\Box(\bigcirc x \leftrightarrow \bigcirc y) \wedge \Box\Diamond\neg y)$$

It is clearly realizable (just set y to false …).
But

$$\Big(\big(\Box((\boxminus \bigcirc x) \rightarrow (\bigcirc x \leftrightarrow \bigcirc y))\big)$$
$$\wedge \big(\Box\bigcirc x \rightarrow (\wedge \Box\Diamond(x \leftrightarrow y) \rightarrow \Box\Diamond\neg y)\big)\Big)$$

is not.

Reactive Synthesis, MOVEP Summer School, Aalborg, 2022 158



Lecture 5: Current Research Directions N. Piterman

Asynchronous (fully observable) Composition

- So far system and environment were strictly synchronous.
- This caused some problems with hybrid control that we tried to circumvent.
- How to allow both system and environment multiple (unbounded) number of actions without either flooding?
  - Add a "who's in control" mechanism.
  - Add an additional clause to specification forcing the system to give back control.
  - Games become more complicated.
  - "Modelling benefit" justifies?

Reactive Synthesis, MOVEP Summer School, Aalborg, 2022 179

Distributed Synthesis

- We want to co-synthesize controllers that will control different variables and collaborate.
- An architecture $A = (P, e, \mathcal{V}, I, O)$, where:
  - P is a set of processes.
  - $e \in P$ the environment.
  - $\mathcal{V}$ set of (Boolean) variables.
  - $I: P \rightarrow 2^{\mathcal{V}}$ input connectivity function.
  - $O: P \rightarrow 2^{\mathcal{V}}$ output connectivity function.

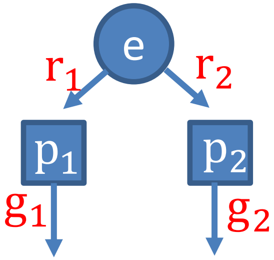
    $\forall p_1, p_2 . O(p_1) \cap O(p_2) = \emptyset$

    $\mathcal{V} = \cup_{p \in P} O(p)$
- An implementation for $p \in P$ is $\left(2^{I(p)}\right)^+ \rightarrow 2^{O(p)}$.

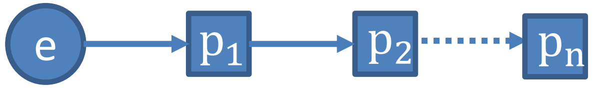
  As before, we would like to replace $\left(2^{I(p)}\right)^+$ by some (finite) domain D_p.
- Given implementations $\{T_p\}_{p \in P}$ for all processes, their composition $\|_p \, T_p$ includes all possible matching interactions.

The Synthesis Problem

- Given an architecture $A = (P, e, \mathcal{V}, I, O)$ and a specification φ over $\mathcal{V}$, do there exists implementations $\{T_p\}_{p\in P}$ such that $\|_p\ T_p$ satisfies φ?
- In general the problem is undecidable.
  - It is enough to have an architecture with two processes with separate inputs.



  - If the architecture contains an information fork, synthesis for it is undecidable.
- Some architectures are possible:



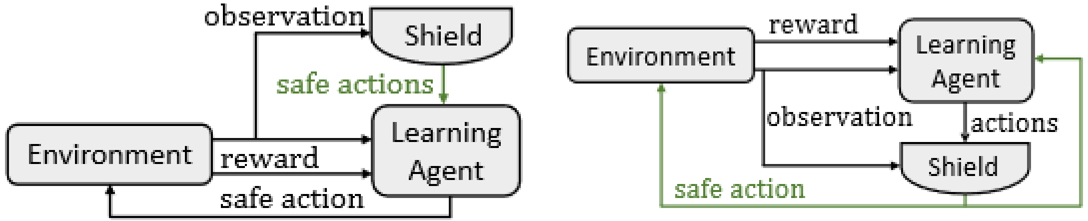
But complexity is non-elementary.

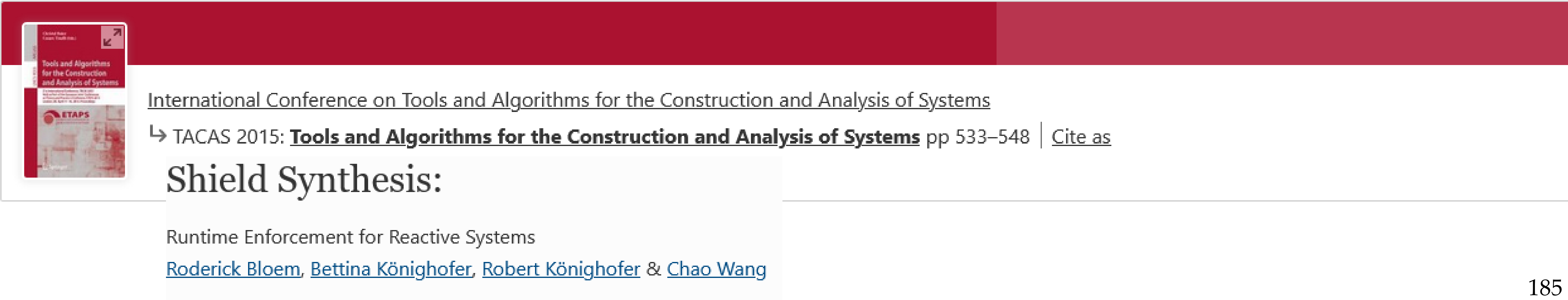
What are my options?

- Bounded synthesis:
  - Use the bounded synthesis for each process separately.
  - Synthesize all the processes together.
- Construct dominant strategies inductively:
  - For a process construct a dominant strategy for the full specification.
  - Extract from the dominant strategy the assumptions for other processes.
  - Synthesize a dominant strategy for specification and new assumptions for all others.
- Use Zielonka/Asynchronous Automata.
  - Communication by synchronous message passing (blocking multicast).
  - More architectures are decidable.
  - Sending of Full information leads to algorithmic distribution.

Safety of Learned Behaviour

- Use formal specifications at learning and at runtime:
  - Shield synthesis – create controllers that accompany a learner and restrict attention to safe actions.





International Conference on Tools and Algorithms for the Construction and Analysis of Systems

↳ TACAS 2015: **Tools and Algorithms for the Construction and Analysis of Systems** pp 533–548 | Cite as

Shield Synthesis:

Runtime Enforcement for Reactive Systems

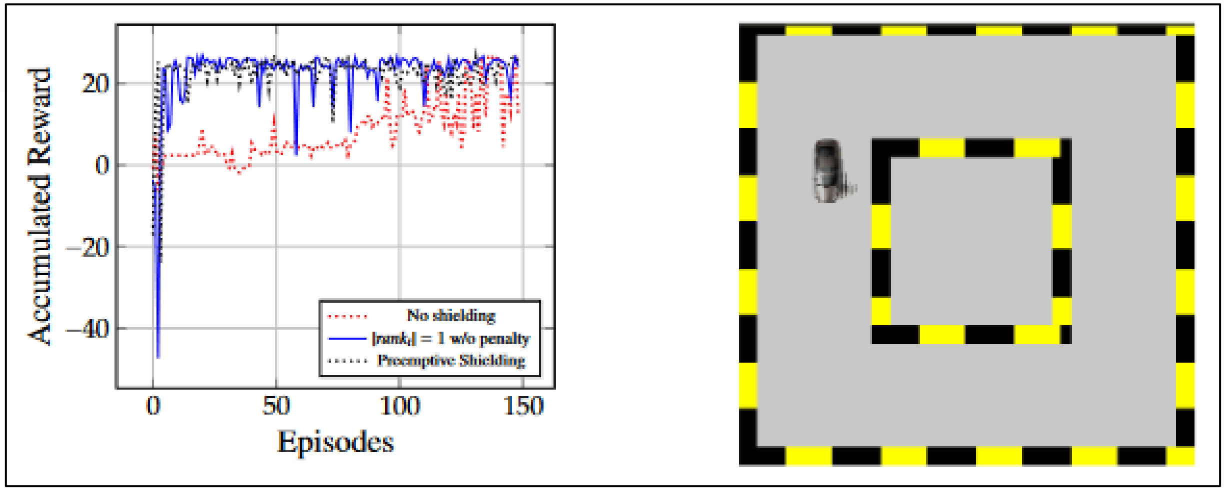
Roderick Bloem, Bettina Könighofer, Robert Könighofer & Chao Wang

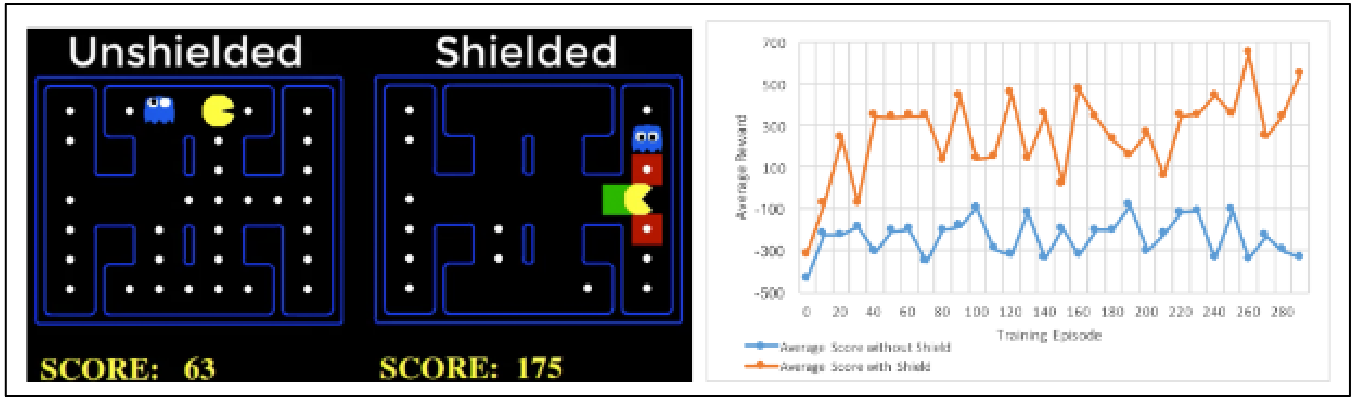
International Symposium on Leveraging Applications of Formal Methods

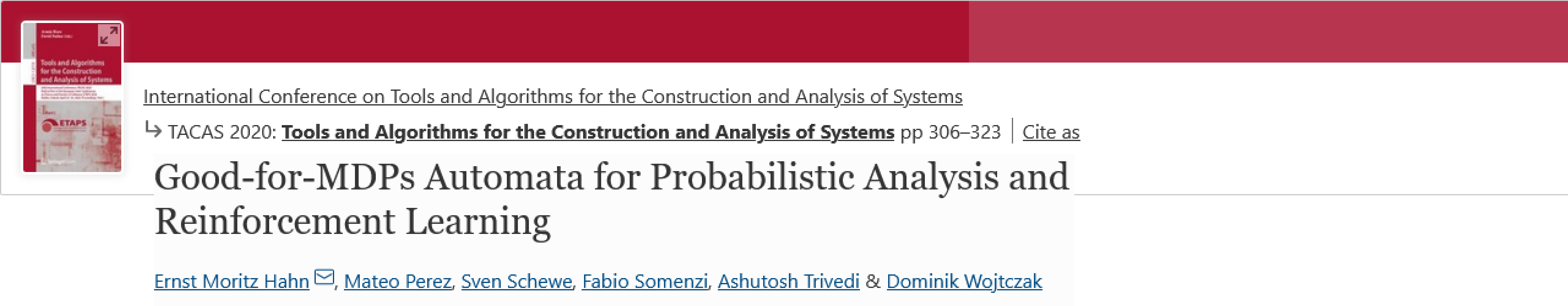
↳ ISoLA 2020: **Leveraging Applications of Formal Methods, Verification and Validation: Verification Principles** pp 290–306 | Cite as

Shield Synthesis for Reinforcement Learning

Bettina Könighofer ✉, Florian Lorber, Nils Jansen & Roderick Bloem





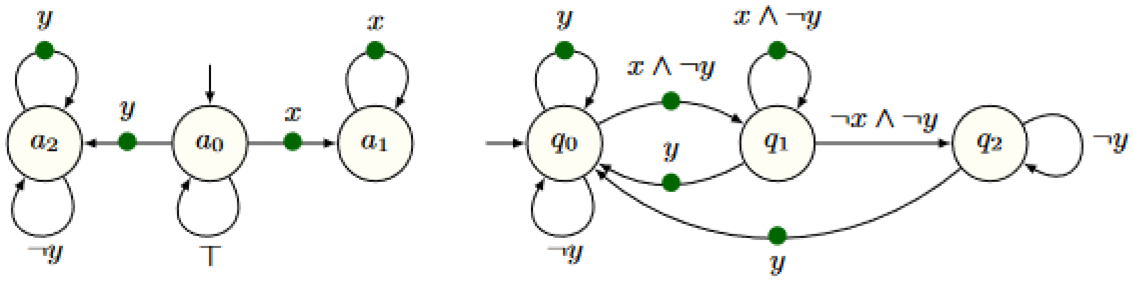


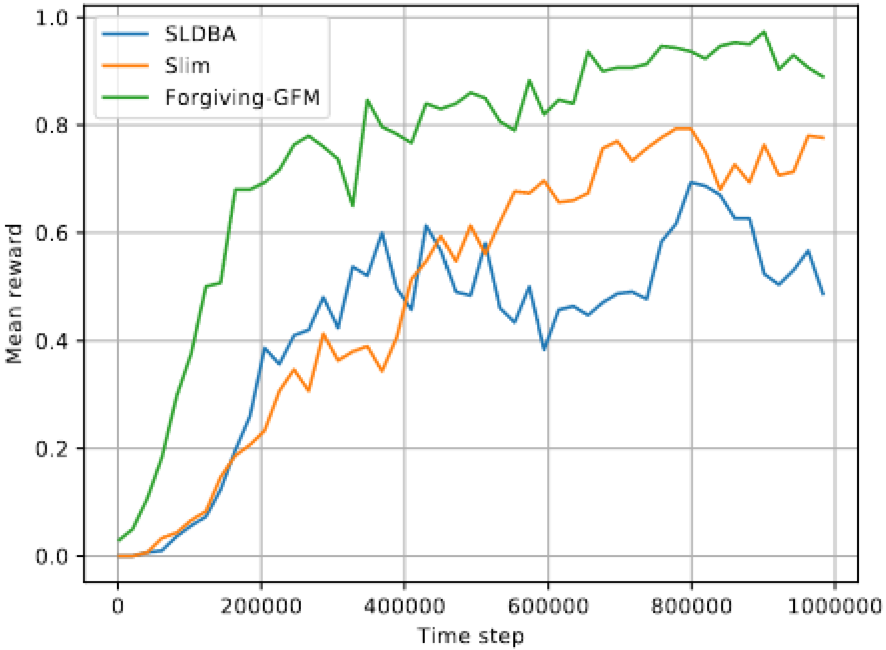
International Conference on Tools and Algorithms for the Construction and Analysis of Systems

↳ TACAS 2020: **Tools and Algorithms for the Construction and Analysis of Systems** pp 306–323 | Cite as

Good-for-MDPs Automata for Probabilistic Analysis and Reinforcement Learning

Ernst Moritz Hahn ✉, Mateo Perez, Sven Schewe, Fabio Somenzi, Ashutosh Trivedi & Dominik Wojtczak





Strategic Reasoning

- Using games and reasoning about strategies for designing multi-agent systems.
- Connections to algorithmic game theory.
- Logics, games, equilibria, …

Concurrent Game Structures

- A concurrent game structure $G = \langle AP, Ag, Ac, St, \lambda, \tau, s_0 \rangle$:
  - AP – atomic set of propositions.
  - Ag – set of agents.
  - Ac – set of actions.
  - St – set of states.
  - $\lambda: St \rightarrow 2^{AP}$ - labeling function.
  - $\tau: St \times Ac^{Ag} \rightarrow St$ – transition function.
- History / track: $\rho \in St^*$.
- Strategy: $f: St^* \rightarrow Ac$.
- Strategy profile: $\{f_{ag}\}_{ag \in Ag}$.
- A strategy profile defines exactly one infinite run.

Logics and Equilibria

- Alternating Temporal Logic – quantify existentially and universally about abilities of coalitions.

$$\langle\langle X \rangle\rangle \Diamond P$$

- Strategy logic – quantify existentially and universally about individual strategies.

$$\exists x_1, x_2 \forall x_3, x_4 \Diamond P(x_1, x_2, x_3, x_4)$$

$$\exists x_1, x_2 \forall x_3, x_4 \Diamond P(x_1, x_2) \wedge \Box Q_1 (x_1, x_4) \wedge \Box Q_2 (x_2, x_3)$$

- Nash equilibrium – a strategy profile such that if a player deviates, other players can join forces to punish them.
- Subgame perfect equilibrium – a strategy profile that is optimal from every location in the game.

Rationality

- What does it mean for an agent to be rational?
- Nash equilibrium in Boolean context?
- Rational synthesis …
- Dominant strategies …
- Good-enough synthesis …

Related Work / Open Problems

- Other determinization [Křetínský, Esparza, …].
- History Determinization (GFG) [HP06, Boker, Lehtinen, …]
- Partial information [Chatterjee, Doyen, Raskin, …].
- Stochastic elements [Chatterjee, Kucera, …].
- Real time [Alur, Maler, Larsen, …].
- Quantitative Objectives [Henzinger, Kupferman, Raskin, …].
- Distributed Synthesis [Muschol, Finkbeiner, Raskin, Walukiewicz, …].

Summary

- Theoretical solution well known since 1969/1989.
- Still provides motivation for a lot of theoretical and practical work.
- In theory, theory and practice are the same.
- Thank you.