Temporal Synthesis

Bernd Finkbeiner
Universität des Saarlandes

Jerusalem, December 2017
Realizability: Does there exist an implementation?
Synthesis: Construct an implementation (if there is one).
Focus

- Reactive systems
  - Continuous interaction with environment
  - Correctness depends on temporal properties (temporal logic, automata)
- Finite state space
- Focus on control, not data transformation
- Typical examples:
  - Reactive layer of cyberphysical systems
  - Hardware circuits
Fundamental premise of synthesis

It is easier... to say **what** a system should do

1. **Eventually** all garbage must be cleared from the table.
2. The robots may **never** push each other.

... than **how** it should be done.
Cyberphysical example: Autonomous driving

- Reactive traffic planner decides whether vehicle should stay in the travel lane or perform a passing maneuver, whether it should go or stop, whether it is allowed to reverse, etc.

- Specification consists of
  - traffic rules: for example “no collision”, “obey speed limits”
  - goals: for example “eventually the checkpoint should be reached”

[Murray et al, 2012]
Hierarchical control

Mission Planner

Path Planner and Follower

Traffic Planner

Actuation Interface

Vehicle

Planning

Reactive strategy

Continuous control

path planning problem → response

path planning problem → response

actuation commands → response

actuation commands → actuator state
Hardware example: AMBA AHB Bus

- High-performance on-chip bus
- Data, address, and control signals
- Up to 16 masters and 16 clients
- Specification consists of
  - 12 guarantees:
    for example “when a locked unspecified length burst starts, new access does not start until current master \( i \) releases bus by lowering HBUSREQi.”
  - 3 assumptions:
    for example “the clients indicate infinitely often that they have finished processing the data by lowering HREADY”

[Bloem et al, 2007]
The reactive synthesis problem

Alonzo Church (1957)

Given a requirement $\varphi$

on the input-output behavior of a boolean circuit,

compute a circuit $C$ that satisfies $\varphi$. 

Game theoretic formulation

Input $i_0i_1i_2\ldots$ $\rightarrow$ C $\rightarrow$ Output $o_0o_1o_2\ldots$

Synthesis game

- Player 0 produces outputs, Player 1 produces inputs.
- Game is played in infinitely many rounds.
- In each round, first Player 1 produces an input, then Player 0 produces an output.
- Player 0 wins if the resulting sequence of inputs and outputs satisfies $\varphi$. 
Example: Synthesis of an arbiter circuit

An arbiter circuit receives requests $r_1, r_2$ from two clients and produces grants $g_1, g_2$.

The specification $\varphi$ of the arbiter is the conjunction of the following properties:

1. **Mutual exclusion:** at no point in time should there be both $g_1$ and $g_2$ in the output.
2. **Response:** every request $r_i$ from the client $i$ (for $i \in \{1, 2\}$) should eventually be followed by grant $g_i$ for client $i$.

**Winning strategy:**

- Initial output is $\emptyset$.
- If input is $\emptyset$ (*no request*) respond with $\emptyset$.
- If input is $\{r_1\}$ (*only client 1 requests grant*) respond with $\{g_1\}$.
- If input is $\{r_2\}$ (*only client 2 requests grant*) respond with $\{g_2\}$.
- If input is $\{r_1, r_2\}$ (*both clients request grants*) ...
Winning strategy

\[
\begin{align*}
\emptyset & \rightarrow \{r_1\} \\
\{r_1\} & \rightarrow \{r_1, r_2\} \\
\{r_2\} & \rightarrow \emptyset \\
\{r_1, r_2\} & \rightarrow \emptyset \\
\{g_1\} & \rightarrow \emptyset \\
\{g_1, \{r_1, r_2\}\} & \rightarrow \emptyset \\
\{r_2\} & \rightarrow \emptyset \\
\{r_1, r_2\} & \rightarrow \emptyset \\
\{g_2\} & \rightarrow \emptyset \\
\{g_2, \{r_1, r_2\}\} & \rightarrow \emptyset 
\end{align*}
\]
Temporal synthesis

- The problem is hard.
  - Synthesis from LTL specifications is $2\text{EXPTIME}$ hard.
  - Synthesis of distributed systems (where the processes have incomplete information) is in general undecidable.

- There has been a lot of progress in the last 10 years.
  - Specification languages with lower complexity
  - Bounded synthesis
  - Applications, e.g., in hardware design and robotics

- Synthesis competition www.syntcomp.org
  - $\approx 3500$ benchmarks
Overview

Part I. **Infinite Games**
Fundamental algorithms to solve infinite games played over finite graphs.

Part II. **Synthesis from Logical Specifications**
Synthesis from specifications given as formulas of a temporal logic. The quest for an efficient and expressive specification language.

Part III. **Bounded Synthesis**
Finding simple solutions fast. The quest for structurally simple implementations.

Part IV. **Distributed Synthesis**
Synthesizing systems that consist of multiple distributed components.
Part I: Infinite Games

1. Definitions
2. Reachability games
3. Büchi games
4. Parity games
A **game arena** $\mathcal{A} = (V, V_0, V_1, E)$ consists of

- a finite set $V$ of states,
- a subset $V_0 \subseteq V$ of states owned by **Player 0** (circles),
- a subset $V_1 = V \setminus V_0$ of states owned by **Player 1** (boxes),
- an edge relation $E \subseteq V \times V$ such that every state $v \in V$ has at least one outgoing edge $(p, p') \in E$.

A **play** is an infinite path through $\mathcal{A}$. 
A strategy for Player $i$ in $A$ is a function $\sigma : \mathcal{V}^* \cdot \mathcal{V}_i \rightarrow \mathcal{V}$ such that $(\nu_n, \sigma(\nu_0 \nu_1 \ldots \nu_n)) \in E$ for every prefix $\nu_0 \nu_1 \ldots \nu_n$ of a play.

A play $\nu_0 \nu_1 \ldots$ is consistent with strategy $\sigma$, if $\nu_{n+1} = \sigma(\nu_0 \ldots \nu_n)$ whenever $\nu_n \in \mathcal{V}_i$.

Special types of strategies:

- **Positional strategies:** $\sigma(\nu_0 \nu_1 \ldots \nu_n) = \sigma(\nu_n)$
  
  strategy only depends on last state

- **Finite-state strategies:** implemented by some FSM
Winning conditions

- A **reachability game** $G = (A, R)$ consists of an arena $A$ and a set $S \subseteq V$ of states. **Player 0** wins a play $\pi$ if $\pi$ visits $R$ at least once, otherwise **Player 1** wins.

- A **Büchi game** $G = (A, F)$ consists of an arena $A$ and a set $F \subseteq V$ of states. **Player 0** wins a play $\pi$ if $\pi$ visits $F$ infinitely often, otherwise **Player 1** wins.

- A **parity game** $G = (A, \alpha)$ consists of an arena $A$ and a coloring function $\alpha : V \to \mathbb{N}$. **Player 0** wins a play $\pi$ if the highest color that is seen infinitely often is even, otherwise **Player 1** wins.
Winning regions

A strategy $\sigma$ is **winning** for Player $i$ from some state $v$ if all plays that are consistent with $\sigma$ and that start in $v$ are won by Player $i$.

The **winning region** $W_i(G)$ is the set of states from which Player $i$ has a winning strategy.

A game is **determined** if $V = W_0 \cup W_1$.

**Solving a game** means to determine the winning region and the winning strategies.
Part I: Infinite Games

1. Definitions
2. Reachability games
3. Büchi games
4. Parity games
Reachability games

Reachability game: Player 0 wins a play $\pi$ if $\pi$ visits $R$ at least once
Reachability games

Reachability game: Player 0 wins a play $\pi$ if $\pi$ visits $R$ at least once
Reachability games

Reachability game: Player $\text{0}$ wins a play $\pi$ if $\pi$ visits $R$ at least once
Reachability games

Reachability game: Player 0 wins a play $\pi$ if $\pi$ visits $R$ at least once.
Attractor construction

\[
\begin{align*}
Attr_i^0(R) &= R \\
Attr_i^{n+1}(R) &= Attr_i^n(R) \cup CPre_i(Attr_i^n(R)) \\
Attr_i(R) &= \bigcup_{n \in \mathbb{N}} Attr_i^n(R)
\end{align*}
\]

where

\[
CPre_i(R) = \{ v \in V_i \mid \exists v' \in V. (v,v') \in E \land v' \in R \} \cup \{ v \in V_{1-i} \mid \forall v' \in V. (v,v') \in E \Rightarrow v' \in R \}
\]

Winning regions of a reachability game

Winning region of **Player 0**: \( W_0(\mathcal{G}) = Attr_0(R) \)

Winning region of **Player 1**: \( W_1(\mathcal{G}) = V \setminus Attr_0(R) \)
Winning strategy of Player o (Attractor Strategy)

The winning strategy for Player o always moves to $Attr^n_o(R)$ for the smallest possible $n$. 
The winning strategy for Player 1 always avoids $\text{Attr}_o(R)$. 
Part I: Infinite Games

1. Definitions
2. Reachability games
3. Büchi games
4. Parity games
Büchi game: Player 0 wins a play $\pi$ if $\pi$ visits $F$ infinitely often, otherwise Player 1 wins.
Büchi games

Büchi game: Player 0 wins a play $\pi$ if $\pi$ visits $F$ infinitely often, otherwise Player 1 wins.
Recurrence construction

\[
\begin{align*}
\text{Recur}^0(F) & = F \\
W_1^n(F) & = V \setminus \text{Attr}_0(\text{Recur}^n(F)) \\
\text{Recur}^{n+1}(F) & = \text{Recur}^n(F) \setminus \text{CPre}_1(W_1^n(F))
\end{align*}
\]

\[
\text{Recur}(F) = \bigcap_{n \in \mathbb{N}} \text{Recur}^n(F)
\]

Winning regions of a Büchi game

Winning region of Player 0: \( W_0(\mathcal{G}) = \text{Attr}_0(\text{Recur}(F)) \)
Winning region of Player 1: \( W_1(\mathcal{G}) = V \setminus \text{Attr}_0(\text{Recur}(F)) \)
Winning strategy of Player 0 (Büchi Strategy)

The winning strategy for Player 0 always moves to $\text{Attr}^n_0(\text{Recur}(F))$ for the smallest possible $n$. 
The winning strategy for Player 1 moves from a state in $W_1^n$ to $W_1^{n-1}$ whenever possible, and stays in $W_1^n$ otherwise.
Part I: Infinite Games

1. Definitions
2. Reachability games
3. Büchi games
4. Parity games
Parity games

Parity game: Player 0 wins a play $\pi$ if the highest color that is seen infinitely often is even.
Parity games

Parity game: Player 0 wins a play \( \pi \) if the highest color that is seen infinitely often is even.
McNaughton’s Algorithm

\textbf{McNaughton}(\mathcal{G})

1. \( c := \text{highest color in} \ \mathcal{G} \)
2. \text{if} \ c = 0 \ \text{or} \ V = \emptyset \ \text{then return} \ (V, \emptyset) 
3. set \( i \) to \( c \mod 2 \)
4. set \( W_{1-i} \) to \( \emptyset \)
5. \text{repeat}
   \begin{enumerate}
   \item \( \mathcal{G}' := \mathcal{G} \setminus \text{Attr}_i(\alpha^{-1}(c), \mathcal{G}) \)
   \item \( (W'_0, W'_1) := \text{McNaughton}(\mathcal{G}') \)
   \item \text{if} \ (W'_{1-i} = \emptyset) \ \text{then}
     \begin{enumerate}
     \item \( W_i := V \setminus W_{1-i} \)
     \item \text{return} \ (W_0, W_1) \end{enumerate}
   \item \( W_{1-i} := W_{1-i} \cup \text{Attr}_{1-i}(W'_{1-i}, \mathcal{G}) \)
   \item \( \mathcal{G} := \mathcal{G} \setminus \text{Attr}_{1-i}(W'_{1-i}, \mathcal{G}) \)
   \end{enumerate}
\text{end repeat}
McNaughton’s Algorithm

\[ \text{McNaughton}(\mathcal{G}) \]

1. \[ c := \text{highest color in } \mathcal{G} \]
2. \[ \text{if } c = 0 \text{ or } V = \emptyset \]
   \[ \text{then return } (V, \emptyset) \]
3. \[ \text{set } i \text{ to } c \mod 2 \]
4. \[ \text{set } W_{1-i} \text{ to } \emptyset \]
5. \[ \text{repeat} \]
5.1 \[ \mathcal{G}' := \mathcal{G} \setminus \text{Attr}_i(\alpha^{-1}(c), \mathcal{G}) \]
5.2 \[ (W'_0, W'_1) := \text{McNaughton}(\mathcal{G}') \]
5.3 \[ \text{if } (W'_{1-i} = \emptyset) \text{ then} \]
5.3.1 \[ W_i := V \setminus W_{1-i} \]
5.3.2 \[ \text{return } (W_0, W_1) \]
5.4 \[ W_{1-i} := W_{1-i} \cup \text{Attr}_{1-i}(W'_{1-i}, \mathcal{G}) \]
5.5 \[ \mathcal{G} := \mathcal{G} \setminus \text{Attr}_{1-i}(W'_{1-i}, \mathcal{G}) \]
McNaughton’s Algorithm

\textit{McNaughton}(G)

1. $c := \text{highest color in } G$

2. if $c = 0$ or $V = \emptyset$ then return $(V, \emptyset)$

3. set $i$ to $c \mod 2$

4. set $W_{1-i}$ to $\emptyset$

5. repeat

5.1 $G' := G \setminus \text{Attr}_i(\alpha^{-1}(c), G)$

5.2 $(W'_0, W'_1) := \text{McNaughton}(G')$

5.3 if $(W'_1 = \emptyset)$ then

5.3.1 $W_i := V \setminus W_{1-i}$

5.3.2 return $(W_0, W_1)$

5.4 $W_{1-i} := W_{1-i} \cup \text{Attr}_{1-i}(W'_1, G)$

5.5 $G := G \setminus \text{Attr}_{1-i}(W'_1, G)$


McNaughton’s Algorithm

\text{McNaughton}(G)

1. $c := \text{highest color in } G$
2. \textbf{if } $c = 0 \text{ or } V = \emptyset$ \textbf{then} \textbf{return } $(V, \emptyset)$
3. \text{set } $i$ to $c \mod 2$
4. \text{set } $W_{1-i}$ to $\emptyset$
5. \textbf{repeat}
   \hspace{1em}5.1 $G' := G \setminus \text{Attr}_i(\alpha^{-1}(c), G)$
   \hspace{1em}5.2 $(W'_0, W'_1) := \text{McNaughton}(G')$
   \hspace{1em}5.3 \textbf{if } $(W'_{1-i} = \emptyset)$ \textbf{then}
   \hspace{2em}5.3.1 $W_i := V \setminus W_{1-i}$
   \hspace{2em}5.3.2 \textbf{return } $(W_0, W_1)$
   \hspace{1em}5.4 $W_{1-i} := W_{1-i} \cup \text{Attr}_{1-i}(W'_{1-i}, G)$
   \hspace{1em}5.5 $G := G \setminus \text{Attr}_{1-i}(W'_{1-i}, G)$
Complexity

- McNaughton’s algorithm is exponential in the number of colors.

- **New result (2017):** Parity games can be solved in quasi-polynomial time.
  - Parity games can be solved in $O(n^{\log(d)+6})$ time and space for $n$ states and $d$ colors
    (Calude, Jain, Khoussainov, Li, Stephan, STOC 2017)
  - Parity games can be solved in quasi-polynomial time and polynomial space
    (Jurdziński, Lazić, LICS 2017)
    (Fearnley, Jain, Schewe, Stephan, Wojtczak, SPIN 2017)
Overview

Part I. Infinite Games
Fundamental algorithms to solve infinite games played over finite graphs.

Part II. Synthesis from Logical Specifications
Synthesis from specifications given as formulas of a temporal logic. The quest for an efficient and expressive specification language.

Part III. Bounded Synthesis
Finding simple solutions fast. The quest for structurally simple implementations.

Part IV. Distributed Synthesis
Synthesizing systems that consist of multiple distributed components.
Part II: Synthesis from Logical Specifications

1. Linear-time temporal logic (LTL)
2. Branching-time temporal logic (CTL/CTL*)
3. GR(1)
Linear-time temporal logic (LTL) (Pnueli, 1977)

- **propositional logic**
  - \( p \) for some atomic proposition \( p \)
  - \( \neg \varphi \)
  - \( \varphi \land \psi \)

- **temporal operators**
  - \( X \varphi \)
  - \( \varphi \mathbin{U} \psi \)

  derived operators:
  - \( F \varphi \equiv true \mathbin{U} \varphi \)
  - \( G \varphi \equiv \neg(F \neg \varphi) \)
Example: Synthesis of an arbiter circuit

An arbiter circuit receives requests $r_1, r_2$ from two clients and produces grants $g_1, g_2$.

The specification $\varphi$ of the arbiter is the conjunction of the following properties:

1. **Mutual exclusion**: at no point in time should there be both $g_1$ and $g_2$ in the output.

   $$G \neg (g_1 \land g_2)$$

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

   $$G r_1 \Rightarrow X F g_1$$
   $$\land$$
   $$G r_2 \Rightarrow X F g_2$$
LTL synthesis

LTL formula → translation → nondeterministic Büchi automaton → determinization → deterministic parity automaton → spreading into inputs and outputs → parity game

Player 0 wins → realizable
Player 1 wins → unrealizable
Why deterministic parity automata?

Why deterministic automata?

- Nondeterministic automata may have rejecting runs on accepted sequences.
- Player 0 may lose a play just because the wrong run was chosen.
- **Example:** \((F \, G \,(i \wedge Xo)) \lor (G \, F \,(\neg i \wedge X\neg o))\)
  - Player 0 has a winning strategy (copy input \(i\) to output \(o\))
  - Player 0 cannot choose between the two disjuncts.

Why not deterministic Büchi automata?

- Not every LTL formula can be translated into an equivalent deterministic Büchi automaton.
- **Example:** \(F \, G \, p\)
Simple response property:

\[ G(r \Rightarrow Fg) \]
Simple response property:

\( G (r \Rightarrow F g) \)

Simplified example


"no open request"

"waiting for grant"
Simplified example

Simple response property:

\[ G(r \Rightarrow Fg) \]
From strategies to implementations

A winning strategy for the Player 0 in the parity game can be represented as an FSM.
LTL synthesis is 2EXPTIME complete.
Part II: Synthesis from Logical Specifications

1. Linear-time temporal logic (LTL)
2. Branching-time temporal logic (CTL/CTL*)
3. GR(1)
Branching-time temporal logics

**CTL** (Emerson/Halpern 1985)

- **CTL* state formulas:**

  \[ \Phi ::= p \mid \Phi_1 \land \Phi_2 \mid \neg \Phi \mid E \varphi \mid A \varphi \]

- **CTL* path formulas:**

  \[ \varphi ::= \Phi \mid \varphi_1 \land \varphi_2 \mid \neg \varphi \mid X \varphi \mid \varphi_1 \mathcal{U} \varphi_2 \]
Strategy trees

- For a given set $\mathcal{Y}$ of directions, the (full infinite) tree is the set $\mathcal{Y}^*$ of finite sequences over $\mathcal{Y}$.
- A $\Sigma$-labeled $\mathcal{Y}$-tree is a function $\mathcal{Y}^* \rightarrow \Sigma$.

A strategy can be seen as a Output-labeled Input-tree:

```
... f(\emptyset) ...
  \phi \\
  \emptyset ...
  \{\emptyset\} \\
  f(\emptyset) ...
    \phi \\
    \emptyset ...
    \{\emptyset\} \\
    f(\emptyset) ...
      \phi \\
      \emptyset ...
      \{\emptyset\} \\
      f(\emptyset) ...
        \phi \\
        \emptyset ...
        \{\emptyset\} \\
        f(\emptyset) ...
          \phi \\
          \emptyset ...
          \{\emptyset\} \\
          f(\emptyset) ...
            \phi \\
            \emptyset ...
            \{\emptyset\} \\
            f(\emptyset) ...
              \phi \\
              \emptyset ...
              \{\emptyset\} \\
              f(\emptyset) ...
                \phi \\
                \emptyset ...
                \{\emptyset\} \\
                f(\emptyset) ...
                  \phi \\
                  \emptyset ...
                  \{\emptyset\} \\
                  f(\emptyset) ...
                    \phi \\
                    \emptyset ...
                    \{\emptyset\} \\
                    f(\emptyset) ...
                      \phi \\
                      \emptyset ...
                      \{\emptyset\} \\
                      f(\emptyset) ...
                        \phi \\
                        \emptyset ...
                        \{\emptyset\} \\
                        f(\emptyset) ...
                          \phi \\
                          \emptyset ...
                          \{\emptyset\} \\
                          f(\emptyset) ...
                            \phi \\
                            \emptyset ...
                            \{\emptyset\} \\
                            f(\emptyset) ...
                              \phi \\
                              \emptyset ...
                              \{\emptyset\} \\
                              f(\emptyset) ...
                                \phi \\
                                \emptyset ...
                                \{\emptyset\} \\
                                f(\emptyset) ...
                                  \phi \\
                                  \emptyset ...
                                  \{\emptyset\} \\
                                  f(\emptyset) ...
                                    \phi \\
                                    \emptyset ...
                                    \{\emptyset\} \\
                                    f(\emptyset) ...
                                      \phi \\
                                      \emptyset ...
                                      \{\emptyset\} \\
                                      f(\emptyset) ...
                                        \phi \\
                                        \emptyset ...
                                        \{\emptyset\} \\
                                        f(\emptyset) ...
                                          \phi \\
                                          \emptyset ...
                                          \{\emptyset\} \\
                                          f(\emptyset) ...
                                            \phi \\
                                            \emptyset ...
                                            \{\emptyset\} \\
                                            f(\emptyset) ...
                                              \phi \\
                                              \emptyset ...
                                              \{\emptyset\} \\
                                              f(\emptyset) ...
                                                \phi \\
                                                \emptyset ...
                                                \{\emptyset\} \\
                                                f(\emptyset) ...
                                                  \phi \\
                                                  \emptyset ...
                                                  \{\emptyset\} \\
                                                  f(\emptyset) ...
                                                    \phi \\
                                                    \emptyset ...
                                                    \{\emptyset\} \\
                                                    f(\emptyset) ...
                                                      \phi \\
                                                      \emptyset ...
                                                      \{\emptyset\} \\
                                                      f(\emptyset) ...
                                                        \phi \\
                                                        \emptyset ...
                                                        \{\emptyset\} \\
                                                        f(\emptyset) ...
                                                          \phi \\
                                                          \emptyset ...
                                                          \{\emptyset\} \\
                                                          f(\emptyset) ...
                                                            \phi \\
                                                            \emptyset ...
                                                            \{\emptyset\} \\
                                                            f(\emptyset) ...
                                                              \phi \\
                                                              \emptyset ...
                                                              \{\emptyset\} \\
                                                              f(\emptyset) ...
                                                                \phi \\
                                                                \emptyset ...
                                                                \{\emptyset\} \\
                                                                f(\emptyset) ...
                                                                  \phi \\
                                                                  \emptyset ...
                                                                  \{\emptyset\} \\
                                                                  f(\emptyset) ...
                                                                    \phi \\
                                                                    \emptyset ...
                                                                    \{\emptyset\} \\
                                                                    f(\emptyset) ...
                                                                      \phi \\
                                                                      \emptyset ...
                                                                      \{\emptyset\} \\
                                                                      f(\emptyset) ...
                                                                        \phi \\
                                                                        \emptyset ...
                                                                        \{\emptyset\} \\
                                                                        f(\emptyset) ...
                                                                          \phi \\
                                                                          \emptyset ...
                                                                          \{\emptyset\} \\
                                                                          f(\emptyset) ...
                                                                            \phi \\
                                                                            \emptyset ...
                                                                            \{\emptyset\} \\
                                                                            f(\emptyset) ...
                                                                              \phi \\
                                                                              \emptyset ...
                                                                              \{\emptyset\} \\
                                                                              f(\emptyset) ...
                                                                                \phi \\
                                                                                \emptyset ...
                                                                                \{\emptyset\} \\
                                                                                f(\emptyset) ...
                                                                                  \phi \\
                                                                                  \emptyset ...
                                                                                  \{\emptyset\} \\
                                                                                  f(\emptyset) ...
                                                                                    \phi \\
                                                                                    \emptyset ...
                                                                                    \{\emptyset\} \\
                                                                                    f(\emptyset) ...
                                                                                      \phi \\
                                                                                      \emptyset ...
                                                                                      \{\emptyset\} \\
                                                                                      f(\emptyset) ...
                                                                                        \phi \\
                                                                                        \emptyset ...
                                                                                        \{\emptyset\} \\
                                                                                        f(\emptyset) ...
                                                                                          \phi \\
                                                                                          \emptyset ...
                                                                                          \{\emptyset\} \\
                                                                                          f(\emptyset) ...
                                                                                           \phi \\
                                                                                           \emptyset ...
                                                                                           \{\emptyset\} \\
                                                                                           f(\emptyset) ...
                                                                                            \phi \\
                                                                                            \emptyset ...
                                                                                            \{\emptyset\} \\
                                                                                            f(\emptyset) ...
                                                                                              \phi \\
                                                                                              \emptyset ...
                                                                                              \{\emptyset\} \\
                                                                                              f(\emptyset) ...
                                                                                               \phi \\
                                                                                               \emptyset ...
                                                                                               \{\emptyset\} \\
                                                                                               f(\emptyset) ...
                                                                                                \phi \\
                                                                                                \emptyset ...
                                                                                                \{\emptyset\} \\
                                                                                                f(\emptyset) ...
                                                                                                  \phi \\
                                                                                                  \emptyset ...
                                                                                                  \{\emptyset\} \\
                                                                                                  f(\emptyset) ...
                                                                                                 
```
CTL* synthesis

- CTL* formula
  - translation
  - parity tree automaton
    - emptiness game
      - parity game
        - Player 0 wins
        - Player 1 wins
          - realizable
          - unrealizable
Tree automata

We use tree automata to represent sets of strategies.

A **parity tree automaton** over $\Sigma$-labeled $\Upsilon$-trees is a tuple 
$A = (Q, q_0, T, \alpha)$, where

- $Q$ is a finite set of **states**, 
- $q_0 \in Q$ is an **initial state**, 
- $T \subseteq Q \times \Sigma \times (\Upsilon \to Q)$ is a set of **transitions**, and 
- $\alpha : Q \to \mathbb{N}$ is a coloring function.

A run of the tree automaton on a given tree annotates each node of the tree with a state of the tree automaton such that each node satisfies, together with its children, some transition. The tree is accepted, if, on every path, the highest color that is visited infinitely often is even.
Example

Existential response property:

\[ A G (r \Rightarrow E F g) \]

\[ Q = \{p, q\} \]

„no open request“  „waiting for grant“
Example

Existential response property:

\[ A G (r \Rightarrow E F g) \]

\[ Q = \{p, q\} \]
Example

Existential response property:

\[ \text{A } G \ (r \Rightarrow E F g) \]

\( Q = \{p, q\} \)

\( q_0 = p \)
Example

Existential response property:

\[ (p \implies q) \]

\[ \alpha(p) = 2, \alpha(q) = 1 \]

\[ Q = \{p, q\} \]

\[ q_0 = p \]

\[ T = \{(p, \emptyset, \emptyset \implies p, \{r\} \implies q), \]
\[ (p, \{g\}, \emptyset \implies p, \{r\} \implies p), \]
\[ (q, \emptyset, \emptyset \implies p, \{r\} \implies q), \]
\[ (q, \emptyset, \emptyset \implies q, \{r\} \implies q), \]
\[ (q, \{g\}, \emptyset \implies p, \{r\} \implies p)\} \]
Example

Existential response property:

$\forall G (r \Rightarrow EF g)$

$Q = \{p, q\}$
$q_0 = p$
$T = \{(p, \emptyset, \emptyset \mapsto p, \{r\} \mapsto q),$
$(p, \{g\}, \emptyset \mapsto p, \{r\} \mapsto p),$
$(q, \emptyset, \emptyset \mapsto p, \{r\} \mapsto q),$
$(q, \emptyset, \emptyset \mapsto q, \{r\} \mapsto q),$
$(q, \emptyset, \emptyset \mapsto p, \{r\} \mapsto p)\}$

$\alpha(p) = 2, \alpha(q) = 1$
Example

Existential response property:

\[ A \ G (r \Rightarrow E \ F g) \]

\[ Q = \{ p, q \} \]
\[ q_0 = p \]
\[ T = \{(p, \emptyset, \emptyset \mapsto p, \{r\} \mapsto q), \]
\[ (p, \{g\}, \emptyset \mapsto p, \{r\} \mapsto p), \]
\[ (q, \emptyset, \emptyset \mapsto p, \{r\} \mapsto q), \]
\[ (q, \emptyset, \emptyset \mapsto q, \{r\} \mapsto q), \]
\[ (q, \{g\}, \emptyset \mapsto p, \{r\} \mapsto p)\} \]
\[ \alpha(p) = 2, \alpha(q) = 1 \]
Example

Existential response property:

\[ AG (r \Rightarrow EF g) \]

\[ Q = \{p, q\} \]
\[ q_0 = p \]
\[ T = \{(p, \emptyset, \emptyset \mapsto p, \{r\} \mapsto q), (p, \{g\}, \emptyset \mapsto p, \{r\} \mapsto p), (q, \emptyset, \emptyset \mapsto p, \{r\} \mapsto q), (q, \emptyset, \emptyset \mapsto q, \{r\} \mapsto q), (q, \{g\}, \emptyset \mapsto p, \{r\} \mapsto p)\} \]
\[ \alpha(p) = 2, \alpha(q) = 1 \]
Example

Existential response property:

$A G (r \Rightarrow E F g)$

$Q = \{p, q\}$
$q_0 = p$
$T = \{(p, \emptyset, \emptyset \Rightarrow p, \{r\} \Rightarrow q),$
(p, \{g\}, \emptyset \Rightarrow p, \{r\} \Rightarrow p),$
(q, \emptyset, \emptyset \Rightarrow p, \{r\} \Rightarrow q),$
(q, \emptyset, \emptyset \Rightarrow q, \{r\} \Rightarrow q),$
(q, \{g\}, \emptyset \Rightarrow p, \{r\} \Rightarrow p)\}$

$\alpha(p) = 2, \alpha(q) = 1$
Example

Existential response property:

\[ A \, G \, (r \Rightarrow E \, F \, g) \]

\[ Q = \{p, q\} \]
\[ q_0 = p \]
\[ T = \{(p, \emptyset, \emptyset \mapsto p, \{r\} \mapsto q), (p, \{g\}, \emptyset \mapsto p, \{r\} \mapsto p), (q, \emptyset, \emptyset \mapsto p, \{r\} \mapsto q), (q, \emptyset, \emptyset \mapsto q, \{r\} \mapsto q), (q, \emptyset, \emptyset \mapsto p, \{r\} \mapsto p)\} \]

\[ \alpha(p) = 2, \alpha(q) = 1 \]
Example

Existential response property:

$$\text{AG} (r \Rightarrow E\ F\ g)$$

$$Q = \{p, q\}$$
$$q_0 = p$$
$$T = \{(p, \emptyset, \emptyset \mapsto p, \{r\} \mapsto q),\ (p, \{g\}, \emptyset \mapsto p, \{r\} \mapsto p),\ (q, \emptyset, \emptyset \mapsto p, \{r\} \mapsto q),\ (q, \emptyset, \emptyset \mapsto q, \{r\} \mapsto q),\ (q, \{g\}, \emptyset \mapsto p, \{r\} \mapsto p)\}$$
$$\alpha(p) = 2, \alpha(q) = 1$$
Emptiness game

Language emptiness of a tree automaton can be checked by solving the emptiness game. The language is non-empty iff Player 0 wins.

- Player 0 picks transitions
- Player 1 picks successor state
**Complexity**

- **CTL* formula**
  - #states: doubly exponential, #colors: exponential

- **Parity tree automaton**
  - #states: doubly exponential, #colors: exponential

- **Parity game**
  - Game solving: polynomial in states, < exponential in colors

**CTL* synthesis**

CTL* synthesis is 2EXPTIME complete.
Branching-time temporal logics

**CTL (Clarke/Emerson 1982)**

CTL* with the restriction that every temporal operator is immediately preceded by a path quantifier.

**CTL synthesis is EXPTIME-complete.**

translation CTL $\rightarrow$ tree automaton *single exponential*
Problem with CTL

It is difficult to specify environment assumptions in CTL.

Example:

- “Every request is followed by a grant” can be expressed:
  \[ A \, G \, (r \implies A \, F \, g) \]

- “If there are infinitely many requests then there are infinitely many grants” cannot be expressed.
Part II: Synthesis from Logical Specifications

1. Linear-time temporal logic (LTL)
2. Branching-time temporal logic (CTL/CTL*)
3. GR(1)
General Reactivity (1)

GR(1) specifications have the following form:

\[ A_1 \land A_2 \land \ldots \land A_m \rightarrow G_1 \land G_2 \land \ldots \land G_n, \]

**Assumptions** \( A_i \) and **guarantees** \( G_i \) are restricted to the following types of formulas:

- **initialization properties**: state formulas
- **safety properties** of the form \( G(\phi \rightarrow X \psi) \),
- **liveness properties** of the form \( G F \phi \).

**Example:** \((G F r) \rightarrow (G F g)\)
General Reactivity (1)

GR(1) games are (comparatively) small:

- safety properties of the form $G (\varphi \rightarrow X \psi)$
  - lead to an exponential state space (keep track of the “active” $X$-formulas)
  - states can be stored efficiently using BDDs and other symbolic data structures
- liveness properties of the form $G F \varphi$ lead to parity games with 3 colors (see next slide)

GR (1) realizability can be checked in exponential time.

(Piterman/Pnueli/Sa’ar, 2006)
General Reactivity (1)

\[(G F a_1) \land (G F a_2 \land) \land \ldots \land (G F a_m) \rightarrow (G F g_1) \land (G F g_2) \land \ldots \land (G F g_n)\]
Pre-synthesis

- Specifications often cannot directly be expressed in GR(1). For example:

\[ G ((START \land PRE) \rightarrow X (\neg START \lor \neg HBUSREQi)) \]

(from the AMBA specification) requires an Until operator.

- **Pre-synthesis** sidesteps this problem: synthesize a deterministic monitor automaton such that the property can be expressed in terms of the states of the monitor. The monitor itself is specified in GR(1).

- The construction of the monitor is double exponential...
Pre-synthesis

Example: $G ( (\text{START} \land \text{PRE}) \Rightarrow X (\neg\text{START} \lor \neg \text{HBUSREQ}i) )$

General Reactivity (1) games are (comparatively) small:
▸ safety properties of the form $G (\phi \rightarrow X \psi)$ lead to an exponential state space (keep track of the "active" $X$-formulas)
▸ lead to states can be stored efficiently using BDDs and other symbolic data structures
▸ liveness properties properties of the form $G F \phi$ lead to parity games with 3 colors (see next slide)

$GR (1)$ realizability can be checked in exponential time.

(Piterman/Pnueli/Sa’ar, 2006)
Overview

Part I. Infinite Games
Fundamental algorithms to solve infinite games played over finite graphs.

Part II. Synthesis from Logical Specifications
Synthesis from specifications given as formulas of a temporal logic. The quest for an efficient and expressive specification language.

Part III. Bounded Synthesis
Finding simple solutions fast. The quest for structurally simple implementations.

Part IV. Distributed Synthesis
Synthesizing systems that consist of multiple distributed components.
TBURST4 component (AMBA)

Depends: on the solution format: Mealy Automata on the specification'

Need: general metrics for structurally simple solutions

Bounded Synthesis (Finkbeiner & Schewe, 2007)

Bounded Cycle Synthesis

CAV 2016, Toronto

Solution Structure 5 / 11
Standard synthesis

(Acacia+ v2.3)
### Synthesis

- Is there an implementation that satisfies the specification?

### Bounded Synthesis

- Is there an implementation with no more than $N$ states?
From input to output complexity

Synthesis
- $2\text{EXPTIME}$ in length of LTL formula (input)

Bounded Synthesis
- NP-complete in size of implementation (output)
Bounded LTL synthesis

LTL formula

- Translation: exponential (in formula)

universal co-Büchi automaton

- Translation: polynomial

constraint system

- Constraint solving: NP (in bound)

Output complexity

LTL synthesis is NP-complete in the size of implementation.
Part III: Bounded Synthesis

1. **Universal co-Büchi automata**
2. Constraint systems
3. Towards structurally simple implementations
Universal co-Büchi automata

A universal co-Büchi automaton accepts a word iff all runs satisfy the co-Büchi condition.

Example: Arbiter

\[ \varphi = G (r_1 \rightarrow XF g_1) \land G (r_2 \rightarrow XFg_2) \land G \neg (g_1 \land g_2) \]
Acceptance of an FSM

A FSM satisfies an LTL formula iff every trace is accepted by the universal co-Büchi automaton corresponding to the formula.

This is the case iff every path in the product satisfies the co-Büchi condition.
Acceptance of an FSM
Acceptance of an FSM
Acceptance of an FSM
Acceptance of an FSM
Acceptance of an FSM
Acceptance of an FSM
Acceptance of an FSM

Diagram showing a transition system with nodes B, G, and R connected by transitions r₁ and r₂, with transitions on edges labeled g₁g₂ and g₂g₁.
Acceptance of a Transition System
Part III: Bounded Synthesis

1. Universal co-Büchi automata
2. **Constraint systems**
3. Towards structurally simple implementations
Annotated FSM

**Annotation**

- for each automaton state, indicates whether state visited on some path, and if so, max number of visits to rejecting states

**Theorem – Completeness**

An FSM is accepted by a universal co-Büchi automaton ⇔ it has a valid annotation.
## Constraint system

The constraint system specifies the existence of an annotated FSM.

### Representation of the FSM

<table>
<thead>
<tr>
<th></th>
<th><strong>states</strong>: $\mathbb{N}_N$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>labeling</strong>: functions $\nu : \mathbb{N}_N \rightarrow \mathbb{B}$</td>
</tr>
<tr>
<td></td>
<td><strong>transitions</strong>: functions $\tau_{in} : \mathbb{N}_N \rightarrow \mathbb{N}_N$</td>
</tr>
</tbody>
</table>

### Representation of annotation

<table>
<thead>
<tr>
<th></th>
<th><strong>state occurrence</strong>: functions $\lambda^\mathbb{B}_q : \mathbb{N}_N \rightarrow \mathbb{B}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>rejecting bound</strong>: functions $\lambda^#_q : \mathbb{N}_N \rightarrow \mathbb{N}$</td>
</tr>
</tbody>
</table>
Constraints

- $\lambda^B_G(o)$

- $\forall t. \lambda^B_G(t) \rightarrow \lambda^B_G(\tau_{\text{r}_1\text{r}_2}(t)) \land \lambda^B_G(\tau_{\text{r}_1\text{r}_2}(t)) \geq \lambda^B_G(t)$
  $\land \lambda^B_G(\tau_{\text{r}_1r_2}(t)) \land \lambda^B_G(\tau_{\text{r}_1r_2}(t)) \geq \lambda^B_G(t)$
  $\land \lambda^B_G(\tau_{\text{r}_1\text{r}_2}(t)) \land \lambda^B_G(\tau_{\text{r}_1\text{r}_2}(t)) \geq \lambda^B_G(t)$

- $\forall t. \lambda^B_G(t) \rightarrow \neg g_1(t) \lor \neg g_2(t)$

- $\forall t. \lambda^B_G(t) \land r_1(t) \rightarrow \lambda^B_B(\tau_{\text{r}_1\text{r}_2}(t)) \land \lambda^B_B(\tau_{\text{r}_1\text{r}_2}(t)) > \lambda^B_G(t)$
  $\land \lambda^B_B(\tau_{\text{r}_1r_2}(t)) \land \lambda^B_B(\tau_{\text{r}_1r_2}(t)) > \lambda^B_G(t)$
  $\land \lambda^B_B(\tau_{\text{r}_1\text{r}_2}(t)) \land \lambda^B_B(\tau_{\text{r}_1\text{r}_2}(t)) > \lambda^B_G(t)$
  $\land \lambda^B_B(\tau_{\text{r}_1r_2}(t)) \land \lambda^B_B(\tau_{\text{r}_1r_2}(t)) > \lambda^B_G(t)$

- $\forall t. \lambda^B_B(t) \land \neg g_1(t) \rightarrow \lambda^B_B(\tau_{\text{r}_1\text{r}_2}(t)) \land \lambda^B_B(\tau_{\text{r}_1\text{r}_2}(t)) > \lambda^B_B(t)$
  $\land \lambda^B_B(\tau_{\text{r}_1r_2}(t)) \land \lambda^B_B(\tau_{\text{r}_1r_2}(t)) > \lambda^B_B(t)$
  $\land \lambda^B_B(\tau_{\text{r}_1\text{r}_2}(t)) \land \lambda^B_B(\tau_{\text{r}_1\text{r}_2}(t)) > \lambda^B_B(t)$
  $\land \lambda^B_B(\tau_{\text{r}_1r_2}(t)) \land \lambda^B_B(\tau_{\text{r}_1r_2}(t)) > \lambda^B_B(t)$
Propositional encodings

SAT

- Only existentially quantified boolean variables permitted.
- **No** symbolic encoding of functions.

QBF

- Quantified boolean variables in **total** order.
- Symbolic encoding of functions with **single** applications:

\[ \ldots f(x) \ldots \approx \forall x \exists y \ldots y \ldots \]

DQBF

- Quantified boolean variables in **partial** order.
- Symbolic encoding of functions with multiple applications:

\[ \ldots f(x_1) \ldots f(x_2) \ldots \approx \forall x_1 \exists y_1 \forall x_2 \exists y_2 (x_1 = x_2 \rightarrow y_1 = y_2) \land \ldots y_1 \ldots y_2 \ldots \]
Constraints

- $\lambda^G_B(o)$
- $\forall t. \lambda^G_B(t) \rightarrow \lambda_B^G(\tau_{\neg r_1 r_2}(t)) \land \lambda^G_B(\tau_{\neg r_1 r_2}(t)) \geq \lambda^G_B(t) \land \lambda^G_B(\tau_{r_1 r_2}(t)) \land \lambda^G_B(\tau_{r_1 r_2}(t)) \geq \lambda^G_B(t) \land \lambda_B^G(\tau_{r_1 r_2}(t)) \land \lambda^G_B(\tau_{r_1 r_2}(t)) \geq \lambda^G_B(t)$

- $\forall t. \lambda^G_B(t) \rightarrow \neg g_1(t) \lor \neg g_2(t)$

- $\forall t. \lambda^G_B(t) \land r_1(t) \rightarrow \lambda_B^B(\tau_{\neg r_1 r_2}(t)) \land \lambda_B^B(\tau_{\neg r_1 r_2}(t)) > \lambda^G_B(t) \land \lambda_B^B(\tau_{r_1 r_2}(t)) \land \lambda_B^G(\tau_{r_1 r_2}(t)) > \lambda^G_B(t) \land \lambda_B^B(\tau_{r_1 r_2}(t)) \land \lambda_B^G(\tau_{r_1 r_2}(t)) > \lambda^G_B(t) \land \lambda_B^B(\tau_{r_1 r_2}(t)) \land \lambda_B^G(\tau_{r_1 r_2}(t)) > \lambda^G_B(t) \land \lambda_B^B(\tau_{r_1 r_2}(t)) \land \lambda_B^G(\tau_{r_1 r_2}(t)) > \lambda^G_B(t)$

- $\forall t. \lambda^B_B(t) \land \neg g_1(t) \rightarrow \lambda_B^B(\tau_{\neg r_1 r_2}(t)) \land \lambda_B^B(\tau_{\neg r_1 r_2}(t)) > \lambda_B^B(t) \land \lambda_B^B(\tau_{r_1 r_2}(t)) \land \lambda_B^B(\tau_{r_1 r_2}(t)) > \lambda_B^B(t) \land \lambda_B^B(\tau_{r_1 r_2}(t)) \land \lambda_B^B(\tau_{r_1 r_2}(t)) > \lambda_B^B(t) \land \lambda_B^B(\tau_{r_1 r_2}(t)) \land \lambda_B^B(\tau_{r_1 r_2}(t)) > \lambda_B^B(t)$

Inputs occur only once (per conjunct)

Diagram:

- $B$ (left)
- $G$ (top)
- $R$ (right)

- $r_1$ (right)
- $r_2$ (right)
- $g_1 g_2$ (left)
- $g_2$ (left)
Constraints

- $\lambda^B_G(o)$

- $\forall t. \lambda^B_G(t) \rightarrow \lambda^B_G(\tau_{\overline{r}_1\overline{r}_2}(t)) \land \lambda^#_G(\tau_{\overline{r}_1\overline{r}_2}(t)) \geq \lambda^#_G(t)$
  $\land \lambda^B_G(\tau_{\overline{r}_1r_2}(t)) \land \lambda^#_G(\tau_{\overline{r}_1r_2}(t)) \geq \lambda^#_G(t)$
  $\land \lambda^B_G(\tau_{r_1\overline{r}_2}(t)) \land \lambda^#_G(\tau_{r_1\overline{r}_2}(t)) \geq \lambda^#_G(t)$
  $\land \lambda^B_G(\tau_{r_1r_2}(t)) \land \lambda^#_G(\tau_{r_1r_2}(t)) \geq \lambda^#_G(t)$

- $\forall t. \lambda^B_G(t) \rightarrow \neg g_1(t) \lor \neg g_2(t)$

- $\forall t. \lambda^B_G(t) \land r_1(t) \rightarrow \lambda^B_B(\tau_{\overline{r}_1\overline{r}_2}(t)) \land \lambda^#_B(\tau_{\overline{r}_1\overline{r}_2}(t)) > \lambda^#_G(t)$
  $\land \lambda^B_B(\tau_{\overline{r}_1r_2}(t)) \land \lambda^#_B(\tau_{\overline{r}_1r_2}(t)) > \lambda^#_G(t)$
  $\land \lambda^B_B(\tau_{r_1\overline{r}_2}(t)) \land \lambda^#_B(\tau_{r_1\overline{r}_2}(t)) > \lambda^#_G(t)$
  $\land \lambda^B_B(\tau_{r_1r_2}(t)) \land \lambda^#_B(\tau_{r_1r_2}(t)) > \lambda^#_G(t)$

- $\forall t. \lambda^B_B(t) \land \neg g_1(t) \rightarrow \lambda^B_B(\tau_{\overline{r}_1\overline{r}_2}(t)) \land \lambda^#_B(\tau_{\overline{r}_1\overline{r}_2}(t)) > \lambda^#_B(t)$
  $\land \lambda^B_B(\tau_{\overline{r}_1r_2}(t)) \land \lambda^#_B(\tau_{\overline{r}_1r_2}(t)) > \lambda^#_B(t)$
  $\land \lambda^B_B(\tau_{r_1\overline{r}_2}(t)) \land \lambda^#_B(\tau_{r_1\overline{r}_2}(t)) > \lambda^#_B(t)$
  $\land \lambda^B_B(\tau_{r_1r_2}(t)) \land \lambda^#_B(\tau_{r_1r_2}(t)) > \lambda^#_B(t)$

states of the FSM occur twice
Constraints

- \( \lambda_B^G(o) \)

- For all states, the automaton states at \( t \):
  \[
  \lambda_B^G(t) \rightarrow \lambda_B^G(\tau_{\overline{r}_1\overline{r}_2}(t)) \land \lambda_B^G(\tau_{\overline{r}_1\overline{r}_2}(t)) \geq \lambda_B^G(t) \\
  \land \lambda_B^G(\tau_1r_2(t)) \land \lambda_B^G(\tau_1r_2(t)) \geq \lambda_B^G(t) \\
  \land \lambda_B^G(\tau_1\overline{r}_2(t)) \land \lambda_B^G(\tau_1\overline{r}_2(t)) \geq \lambda_B^G(t) \\
  \land \lambda_B^G(\tau_1r_2(t)) \land \lambda_B^G(\tau_1r_2(t)) \geq \lambda_B^G(t)
  \]

- For all states, the automaton states at \( t \):
  \[
  \lambda_B^G(t) \rightarrow \neg g_1(t) \lor \neg g_2(t)
  \]

- For all states, the automaton states at \( t \) and \( r_1(t) \):
  \[
  \lambda_B^G(t) \land r_1(t) \rightarrow \lambda_B^G(\tau_{\overline{r}_1\overline{r}_2}(t)) \land \lambda_B^G(\tau_{\overline{r}_1\overline{r}_2}(t)) > \lambda_B^G(t) \\
  \land \lambda_B^G(\tau_1r_2(t)) \land \lambda_B^G(\tau_1r_2(t)) > \lambda_B^G(t) \\
  \land \lambda_B^G(\tau_1\overline{r}_2(t)) \land \lambda_B^G(\tau_1\overline{r}_2(t)) > \lambda_B^G(t) \\
  \land \lambda_B^G(\tau_1r_2(t)) \land \lambda_B^G(\tau_1r_2(t)) > \lambda_B^G(t)
  \]

- For all states, the automaton states at \( t \) and \( \neg g_1(t) \):
  \[
  \lambda_B^G(t) \land \neg g_1(t) \rightarrow \lambda_B^G(\tau_{\overline{r}_1\overline{r}_2}(t)) \land \lambda_B^G(\tau_{\overline{r}_1\overline{r}_2}(t)) > \lambda_B^G(t) \\
  \land \lambda_B^G(\tau_1r_2(t)) \land \lambda_B^G(\tau_1r_2(t)) > \lambda_B^G(t) \\
  \land \lambda_B^G(\tau_1\overline{r}_2(t)) \land \lambda_B^G(\tau_1\overline{r}_2(t)) > \lambda_B^G(t) \\
  \land \lambda_B^G(\tau_1r_2(t)) \land \lambda_B^G(\tau_1r_2(t)) > \lambda_B^G(t)
  \]

states of the automaton occur twice
Propositional Encodings

basic SAT encoding
  ▸ explicit inputs, explicit states of FSM, explicit states of automaton

input-symbolic QBF encoding
  ▸ **symbolic inputs**, explicit states of FSM, explicit states of automaton

state-symbolic DQBF encoding
  ▸ **symbolic inputs**, **symbolic states of FSM**, explicit states of automaton

fully-symbolic DQBF encoding
  ▸ **symbolic inputs**, **symbolic states of FSM**, **symbolic states of automaton**
Experiments (SYNTCOMP 2016 benchmarks)

- fully-symbolic
- state-symbolic
- Party elli rally
- SMT
- basic
- input-symbolic
- Acacia

![Graph showing the performance of different benchmarks with respect to the number of instances and time in seconds.](image)
Implementation size (#AND gates)
Bounded synthesis as a decision procedure

\textbf{Increase bound} until

\begin{itemize}
\item implementation found, or
\item counterstrategy found (by searching for implementation in dual problem)
\end{itemize}

\textbf{Termination:} There is an implementation or a counterstrategy with at most doubly exponentially many states

\begin{itemize}
\item the parity game has doubly exponentially many states
\item one of the players has a winning positional strategy, i.e., a winning strategy with at most doubly exponentially many states.
\end{itemize}
https://www.react.uni-saarland.de/tools/online/BoSy/
Part III: Bounded Synthesis

1. Universal co-Büchi automata
2. Constraint systems
3. Towards structurally simple implementations
Standard synthesis

TBURST4 component (AMBA) Acacia+ v2.3
Bounded synthesis

TBURST4 component (AMBA) BoSy

Depends: on the solution format: Mealy Automata on the specification!

Need: general metrics for structurally simple solutions

the size / number of states of the solution

Bounded Synthesis (Finkbeiner & Schewe, 2007)

the number of simple cycles of the solution

Bounded Cycle Synthesis

CAV 2016, Toronto

Solution Structure 5 / 11
Bounded cycle synthesis

TBURST4 component (AMBA) BoCy
Towards structurally simple implementations

- **Bounded synthesis** minimizes the number of states.
- FSMs with a minimal number of states may still have a complicated control structure.
- Additional parameters are needed.
- **Bounded cycle synthesis** additionally minimizes the number of simple cycles.
- The number of cycles is an explosive parameter:
  - The number of cycles of an FSM is exponentially bounded in the size of the FSM.
  - There is a realizable LTL formula $\varphi$ such that every implementation has at least triply-exponentially many cycles in the size of $\varphi$. 
Experiments

# Experimental Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>(A_\varphi)</th>
<th>Aca+</th>
<th>BoSy/BoCy</th>
<th>Cycles</th>
<th>Time (s)</th>
<th>Aca+</th>
<th>BoSy</th>
<th>BoCy</th>
<th>Aca+</th>
<th>SAT</th>
<th>UNSAT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARBITER[2]</td>
<td>6</td>
<td>26</td>
<td>2</td>
<td>5439901</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>0.261</td>
<td>0.847</td>
<td>0.868</td>
<td>0.300</td>
</tr>
<tr>
<td>LOCK[2]</td>
<td>12</td>
<td>4</td>
<td>3</td>
<td>12</td>
<td>6</td>
<td>5</td>
<td>0.459</td>
<td>0.395</td>
<td>0.522</td>
<td>0.165</td>
<td>0.487</td>
</tr>
<tr>
<td>LOCK[3]</td>
<td>20</td>
<td>4</td>
<td>3</td>
<td>12</td>
<td>5</td>
<td>5</td>
<td>55.917</td>
<td>1.037</td>
<td>1.245</td>
<td>0.433</td>
<td>1.107</td>
</tr>
<tr>
<td>ENCODE[2]</td>
<td>3</td>
<td>6</td>
<td>2</td>
<td>41</td>
<td>3</td>
<td>3</td>
<td>0.473</td>
<td>0.071</td>
<td>0.089</td>
<td>0.048</td>
<td>0.084</td>
</tr>
<tr>
<td>ENCODE[3]</td>
<td>5</td>
<td>16</td>
<td>3</td>
<td>90428</td>
<td>8</td>
<td>8</td>
<td>1.871</td>
<td>0.292</td>
<td>0.561</td>
<td>0.200</td>
<td>0.503</td>
</tr>
<tr>
<td>ENCODE[4]</td>
<td>5</td>
<td>20</td>
<td>4</td>
<td>(&gt;9999999)</td>
<td>24</td>
<td>24</td>
<td>4.780</td>
<td>1.007</td>
<td>16.166</td>
<td>0.579</td>
<td>999</td>
</tr>
<tr>
<td>DECODE</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>0.328</td>
<td>0.055</td>
<td>0.051</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>SHIFT</td>
<td>3</td>
<td>6</td>
<td>2</td>
<td>31</td>
<td>3</td>
<td>3</td>
<td>0.387</td>
<td>0.060</td>
<td>0.072</td>
<td>0.041</td>
<td>0.071</td>
</tr>
<tr>
<td>TBURST4</td>
<td>103</td>
<td>14</td>
<td>7</td>
<td>61</td>
<td>19</td>
<td>7</td>
<td>0.634</td>
<td>8.294</td>
<td>206.604</td>
<td>6.261</td>
<td>999</td>
</tr>
<tr>
<td>TINCR</td>
<td>43</td>
<td>5</td>
<td>3</td>
<td>7</td>
<td>5</td>
<td>2</td>
<td>0.396</td>
<td>2.262</td>
<td>2.279</td>
<td>0.845</td>
<td>2.221</td>
</tr>
<tr>
<td>TSINGLE</td>
<td>22</td>
<td>8</td>
<td>4</td>
<td>12</td>
<td>5</td>
<td>4</td>
<td>0.372</td>
<td>1.863</td>
<td>2.143</td>
<td>1.165</td>
<td>2.067</td>
</tr>
</tbody>
</table>
Overview

Part I. Infinite Games
Fundamental algorithms to solve infinite games played over finite graphs.

Part II. Synthesis from Logical Specifications
Synthesis from specifications given as formulas of a temporal logic. The quest for an efficient and expressive specification language.

Part III. Bounded Synthesis
Finding simple solutions fast. The quest for structurally simple implementations.

Part IV. Distributed Synthesis
Synthesizing systems that consist of multiple distributed components.
Distributed synthesis specification

Implementation

Alonzo Church, 1957
Applications of recursive arithmetic to the problem of circuit synthesis

distributed synthesis implementation unrealizable

specification

implementation unrealizable
Part IV: Distributed Synthesis

1. Synthesis in the Pnueli/Rosner model
2. Bounded synthesis of distributed systems
3. Synthesis in the causal memory model
Pnueli/Rosner model

Architectures

Nodes:
- system processes -- unknown implementation
- environment -- unconstrained behavior

Edges:
- communication structure
- variables

[PNueli/Rosner, 1989]
Pnueli/Rosner model

Implementation

The implementation defines for each process $p$ with input variables $I_p$ and output variables $O_p$ a strategy tree with directions $\mathcal{2}^{I_p}$ and labels $\mathcal{2}^{O_p}$. 
Specification

The combination of the process strategies defines the computation tree with directions $2^I$ and labels $2^V$, where $I$ are the global input variables and $V$ is the set of variables.

The implementation is correct iff the computation tree satisfies the given specification, e.g., some formula in a temporal logic.
process 2 does not know $x$ decisions of process 2 must not depend on $x$. 
process 2 does not know $x$
decisions of process 2 must not depend on $x$. 

consistent
Pnueli/Rosner model: Decidability

- Closed systems (1981 Manna/Wolper, Clarke/Emerson)
- Open systems (1969 Rabin, Büchi, Landweber)
- Pipelines (1990, Pnueli/Rosner)
- Rings (2001, Kupferman/Vardi)
- Weakly-ordered architectures (2005, F./Schewe)

decidable

undecidable

- Independent processes (1990, Pnueli/Rosner)
- Architectures with information forks (2005, F./Schewe)
Synthesis in the Pnueli/Rosner model

1-process architectures --- 2EXPTIME

Pipeline architectures --- non-elementary

2-process arbiter architecture --- undecidable
Part IV: Distributed Synthesis

1. Synthesis in the Pnueli/Rosner model
2. Bounded synthesis of distributed systems
3. Synthesis in the causal memory model
Bounded synthesis of distributed systems

1-process architectures --- NP

Pipeline architectures --- NP

2-process arbiter architecture --- NP
The constraint system specifies the existence of an annotated FSM.

### Representation of the FSM

- **states**: $\mathbb{N}_N$
- **labeling**: functions $\nu : \mathbb{N}_N \to \mathbb{B}$
- **transitions**: functions $\tau_{in} : \mathbb{N}_N \to \mathbb{N}_N$

### Representation of annotation

- **state occurrence**: functions $\lambda^B_q : \mathbb{N}_N \to \mathbb{B}$
- **rejecting bound**: functions $\lambda^\#_q : \mathbb{N}_N \to \mathbb{N}$
Extended constraint system

Local transition system

- **projection** from global states to local states for process $p_i$:
  \[ \text{proj}_i : \mathbb{N}_N \rightarrow \mathbb{N}_{N_i} \]

- **local transition function** for local input and local state
  \[ \tau_{i;\text{inp}} : \mathbb{N}_{N_i} \rightarrow \mathbb{N}_{N_i} \]
Consistency constraint

\[ \forall t. \tau_1; r_1, g_2(\text{proj}_2(t)) \left(\text{proj}_1(t)\right) = \text{proj}_1(\tau_{r_1 r_2}(t)) = \text{proj}_1(\tau_{\bar{r}_1 \bar{r}_2}(t)) \]
\[ \land \tau_1; \bar{r}_1, g_2(\text{proj}_2(t)) \left(\text{proj}_1(t)\right) = \text{proj}_1(\tau_{\bar{r}_1 r_2}(t)) = \text{proj}_1(\tau_{\bar{r}_1 \bar{r}_2}(t)) \]

\[ \land \tau_2; r_2, g_1(\text{proj}_1(t)) \left(\text{proj}_2(t)\right) = \text{proj}_2(\tau_{r_1 r_2}(t)) = \text{proj}_2(\tau_{\bar{r}_1 r_2}(t)) \]
\[ \land \tau_2; \bar{r}_2, g_1(\text{proj}_1(t)) \left(\text{proj}_2(t)\right) = \text{proj}_2(\tau_{\bar{r}_1 \bar{r}_2}(t)) = \text{proj}_2(\tau_{\bar{r}_1 \bar{r}_2}(t)) \]
Part IV: Distributed Synthesis

1. Synthesis in the Pnueli/Rosner model
2. Bounded synthesis of distributed systems
3. **Synthesis in the causal memory model**
The causal memory model

- processes memorize their causal history
- processes communicate causal history to each other during each synchronization
- this abstracts from the content of a communication (it is part of the synthesis problem to determine what should be communicated during the synchronization)

Hope: Since the communication is less restricted than under partial observation, the undecidability results do not carry over.

[Gastin/Lerman/Zeitoun, 2004]
Causal memory synthesis

- Control of series-parallel asynchronous automata decidable [Gastin/Lerman/Zeitoun, 2004]
- Control of connectedly communicating asynchronous automata decidable [Madhusudan/Thiagarajan/Yang, 2004]
- Control of asynchronous automata with tree-shaped communication graph decidable [Genest/Gimbert/Muscholl/Walukiewicz, 2013]
- Petri games with multiple system processes, single environment process EXPTIME-complete [F./Olderog, 2014]
- Petri games with single system process, multiple environment processes EXPTIME-complete [F./Gölz, 2017]
- Four player games decidable [Gimbert, 2017]

- Decidability of causal memory synthesis in general is open.
Consider the development of a distributed security alarm system. If a burglar rashes into our building either at location A or B, the tokens that initially reside on place Env report the location where the alarm is triggered. In the Petri game, the transitions to the bad place represent the environment, which is, in our example, the burglar, who can decide to break into the building at any point, the players of the Petri game may have a different level of knowledge about the global state of the system players when a communication can occur (e.g., when a device may be communicating with the base station, etc.). This is useful at a design stage before the details of the interface have been decided and one is more interested in restricting what communication can occur. Alternatively, by the winning strategies, one location, the alarm should go off everywhere, and all locations should be decided and one is more interested in restricting the synchronizations chosen by the players.

Figure 1: Introductory example of a Petri game modeling a distributed security alarm system. If a burglar breaks into our building either at location A or B, the tokens that initially reside on place Env report the location where the alarm is triggered. In the Petri game, the transitions to the bad place represent the environment, which is, in our example, the burglar, who can decide to break into the building at any point, the players of the Petri game may have a different level of knowledge about the global state of the system players when a communication can occur (e.g., when a device may be communicating with the base station, etc.). This is useful at a design stage before the details of the interface have been decided and one is more interested in restricting what communication can occur. Alternatively, by the winning strategies, one location, the alarm should go off everywhere, and all locations should be decided and one is more interested in restricting the synchronizations chosen by the players.
Consider the development of a distributed security alarm system. If a burglar enters our building either at location A or B, the tokens that initially reside on place $pA$ and $pB$ are shown with dotted lines.

Places belonging to the distributed controller consisting of two processes, the one on the left for location A and the one on the right for location B are shown in gray. In the Petri game, the transitions to the bad place $q_{bad}$ represent the environment, which is, in our example, the burglar, who can decide to break into our building if the alarm goes off at location A. This situation is depicted as a Petri net in Fig. 1. The final interface is then determined by the information actually used by the players.

The level of informedness changes dynamically as a result of any point, the players of the Petri game may have a different level of knowledge about the global state of the system. The synchronizations chosen by the winning strategies, which is typically only a small fraction of the causal history. Note that even though the alarm should go off everywhere, and all locations should report the location where the alarm occurred. This situation is depicted as a Petri net in Fig. 1.
Consider the development of a distributed security alarm system. If a burglar enters our building either at location A or B, the tokens that initially reside on place Env represent the environment, which is, in our example, the burglar, who can decide to break into our building either at location A or B. The token that initially resides on place Env triggers the alarm at location A.

 lugares

When the alarm goes off at location A, the alarm should go off everywhere, and all locations should report the location where the alarm occurred. This situation is depicted as a Petri net in Fig. 1. The synchronizations chosen by the players may be communicated, and when a device is connected to its base station, while a network connection is active, etc.) than we assume the players to communicate everything they know, the flow of information in a Petri game is far from trivial. At any point, the players of the Petri game may have a different level of knowledge about the global state of the system players.

The final interface is then determined by the information actually used by the Winning strategies, and the level of informedness changes dynamically as a result of the players' decisions and the synchronizations chosen. Note that even though the distributed controller consisting of two processes, the one on the left represents the players and the one on the right represents the environment, which is, in our example, the burglar, who can decide to break into our building either at location A or B. The tokens that initially reside on place Env.

Local transition

This situation is depicted as a Petri net in Fig. 1. TheToken that initially resides on place Env triggers the alarm at location A.

When the alarm goes off at location A, the alarm should go off everywhere, and all locations should report the location where the alarm occurred. This situation is depicted as a Petri net in Fig. 1. The synchronizations chosen by the players may be communicated, and when a device is connected to its base station, while a network connection is active, etc.) than we assume the players to communicate everything they know, the flow of information in a Petri game is far from trivial. At any point, the players of the Petri game may have a different level of knowledge about the global state of the system players.

The final interface is then determined by the information actually used by the Winning strategies, and the level of informedness changes dynamically as a result of the players' decisions and the synchronizations chosen. Note that even though the distributed controller consisting of two processes, the one on the left represents the players and the one on the right represents the environment, which is, in our example, the burglar, who can decide to break into our building either at location A or B. The tokens that initially reside on place Env.

Local transition

This situation is depicted as a Petri net in Fig. 1. TheToken that initially resides on place Env triggers the alarm at location A.

When the alarm goes off at location A, the alarm should go off everywhere, and all locations should report the location where the alarm occurred. This situation is depicted as a Petri net in Fig. 1. The synchronizations chosen by the players may be communicated, and when a device is connected to its base station, while a network connection is active, etc.) than we assume the players to communicate everything they know, the flow of information in a Petri game is far from trivial. At any point, the players of the Petri game may have a different level of knowledge about the global state of the system players.

The final interface is then determined by the information actually used by the Winning strategies, and the level of informedness changes dynamically as a result of the players' decisions and the synchronizations chosen. Note that even though the distributed controller consisting of two processes, the one on the left represents the players and the one on the right represents the environment, which is, in our example, the burglar, who can decide to break into our building either at location A or B. The tokens that initially reside on place Env.

Local transition

This situation is depicted as a Petri net in Fig. 1. TheToken that initially resides on place Env triggers the alarm at location A.

When the alarm goes off at location A, the alarm should go off everywhere, and all locations should report the location where the alarm occurred. This situation is depicted as a Petri net in Fig. 1. The synchronizations chosen by the players may be communicated, and when a device is connected to its base station, while a network connection is active, etc.) than we assume the players to communicate everything they know, the flow of information in a Petri game is far from trivial. At any point, the players of the Petri game may have a different level of knowledge about the global state of the system players.

The final interface is then determined by the information actually used by the Winning strategies, and the level of informedness changes dynamically as a result of the players' decisions and the synchronizations chosen. Note that even though the distributed controller consisting of two processes, the one on the left represents the players and the one on the right represents the environment, which is, in our example, the burglar, who can decide to break into our building either at location A or B. The tokens that initially reside on place Env.

Local transition

This situation is depicted as a Petri net in Fig. 1. TheToken that initially resides on place Env triggers the alarm at location A.

When the alarm goes off at location A, the alarm should go off everywhere, and all locations should report the location where the alarm occurred. This situation is depicted as a Petri net in Fig. 1. The synchronizations chosen by the players may be communicated, and when a device is connected to its base station, while a network connection is active, etc.) than we assume the players to communicate everything they know, the flow of information in a Petri game is far from trivial. At any point, the players of the Petri game may have a different level of knowledge about the global state of the system players.

The final interface is then determined by the information actually used by the Winning strategies, and the level of informedness changes dynamically as a result of the players' decisions and the synchronizations chosen. Note that even though the distributed controller consisting of two processes, the one on the left represents the players and the one on the right represents the environment, which is, in our example, the burglar, who can decide to break into our building either at location A or B. The tokens that initially reside on place Env.

Local transition

This situation is depicted as a Petri net in Fig. 1. TheToken that initially resides on place Env triggers the alarm at location A.

When the alarm goes off at location A, the alarm should go off everywhere, and all locations should report the location where the alarm occurred. This situation is depicted as a Petri net in Fig. 1. The synchronizations chosen by the players may be communicated, and when a device is connected to its base station, while a network connection is active, etc.) than we assume the players to communicate everything they know, the flow of information in a Petri game is far from trivial. At any point, the players of the Petri game may have a different level of knowledge about the global state of the system players.

The final interface is then determined by the information actually used by the Winning strategies, and the level of informedness changes dynamically as a result of the players' decisions and the synchronizations chosen. Note that even though the distributed controller consisting of two processes, the one on the left represents the players and the one on the right represents the environment, which is, in our example, the burglar, who can decide to break into our building either at location A or B. The tokens that initially reside on place Env.

Local transition

This situation is depicted as a Petri net in Fig. 1. TheToken that initially resides on place Env triggers the alarm at location A.

When the alarm goes off at location A, the alarm should go off everywhere, and all locations should report the location where the alarm occurred. This situation is depicted as a Petri net in Fig. 1. The synchronizations chosen by the players may be communicated, and when a device is connected to its base station, while a network connection is active, etc.) than we assume the players to communicate everything they know, the flow of information in a Petri game is far from trivial. At any point, the players of the Petri game may have a different level of knowledge about the global state of the system players.

The final interface is then determined by the information actually used by the Winning strategies, and the level of informedness changes dynamically as a result of the players' decisions and the synchronizations chosen. Note that even though the distributed controller consisting of two processes, the one on the left represents the players and the one on the right represents the environment, which is, in our example, the burglar, who can decide to break into our building either at location A or B. The tokens that initially reside on place Env.

Local transition

This situation is depicted as a Petri net in Fig. 1. TheToken that initially resides on place Env triggers the alarm at location A.

When the alarm goes off at location A, the alarm should go off everywhere, and all locations should report the location where the alarm occurred. This situation is depicted as a Petri net in Fig. 1. The synchronizations chosen by the players may be communicated, and when a device is connected to its base station, while a network connection is active, etc.) than we assume the players to communicate everything they know, the flow of information in a Petri game is far from trivial. At any point, the players of the Petri game may have a different level of knowledge about the global state of the system players.

The final interface is then determined by the information actually used by the Winning strategies, and the level of informedness changes dynamically as a result of the players' decisions and the synchronizations chosen. Note that even though the distributed controller consisting of two processes, the one on the left represents the players and the one on the right represents the environment, which is, in our example, the burglar, who can decide to break into our building either at location A or B. The tokens that initially reside on place Env.

Local transition

This situation is depicted as a Petri net in Fig. 1. TheToken that initially resides on place Env triggers the alarm at location A.

When the alarm goes off at location A, the alarm should go off everywhere, and all locations should report the location where the alarm occurred. This situation is depicted as a Petri net in Fig. 1. The synchronizations chosen by the players may be communicated, and when a device is connected to its base station, while a network connection is active, etc.) than we assume the players to communicate everything they know, the flow of information in a Petri game is far from trivial. At any point, the players of the Petri game may have a different level of knowledge about the global state of the system players.

The final interface is then determined by the information actually used by the Winning strategies, and the level of informedness changes dynamically as a result of the players' decisions and the synchronizations chosen. Note that even though the distributed controller consisting of two processes, the one on the left represents the players and the one on the right represents the environment, which is, in our example, the burglar, who can decide to break into our building either at location A or B. The tokens that initially reside on place Env.

Local transition

This situation is depicted as a Petri net in Fig. 1. TheToken that initially resides on place En...
Consider the development of a distributed security alarm system. If a burglar gets inside our building either at location A or B, the tokens that initially reside on place $pA$ and $pB$ represent the environment, which is, in our example, the burglar, who can decide to break into the building. Places belonging to location A and the one on the left are shown with dotted lines.

When $t_A$ triggers the alarm at location A, all agents must be informed of the alarm report the location where the alarm occurred. Places belonging to location B and the one on the right are shown with dotted lines. From place $pA$ tokens are added to $A$ and $A_2$, which is typically only a small fraction of the causal history. Note that even though we assume the players of the Petri game may have a different level of knowledge about the global state of the system, they are not supposed to communicate everything they know, which is far from trivial. At any point, the players of the Petri game may have a different level of knowledge, and one is more interested in restricting the information actually used by the system players to communicate. 

The final interface is then determined by the information actually used by the distributed controller consisting of two processes, the one on the left for location A and the one on the right for location B. The tokens that initially reside on place $pA$ and $pB$ are shown in gray. In the Petri game, the transitions to the bad place $q$ and $EA$ are not represented.
Consider the development of a distributed security alarm system. If a burglar attempts to enter our building either at location A or B, the tokens that initially reside on place $p_A$, which is typically only a small fraction of the causal history. Note that even though the system players decide to break into the building at location A and the one on the right at location B, the synchronizations chosen may be communicated everything they know, the flow of information in a Petri game is to the system players. The final interface is then determined by the information actually used by the players.

Figure 1: Introductory example of a Petri game modeling a distributed security alarm system.
Petri games: Unfolding

Figure 3: Unfolding of the Petri game in Fig. ?? To aid visibility, the transitions leading to \( q_{\text{bad}} \) are omitted from the unfolding. If the transitions shown with dashed lines are removed from the unfolding, the resulting net is a winning strategy for the system players.

Our definition of strategies is based on the unfolding of the net, which is shown for our example in Fig. ?? By eliminating all joins in the net, net unfoldings [?, ?, ?] separate places that are reached via multiple causal histories into separate copies. In the example, place \( p_B \) has been unfolded into four separate copies, corresponding to the four different ways to reach \( p_B \) via the transition arcs \( B_1 \) through \( B_4 \). Each copy represents different knowledge: in \( B_1 \), only \( B \) knows that there has been a burglary at location \( B \); in \( B_2 \), \( B \) knows nothing; in \( B_3 \), \( B \) knows that \( A \) knows that there has been a burglary at position \( B \); in \( B_4 \), \( B \) knows that there has been a burglary at location \( A \). (Symmetric statements hold for \( p_A \) and the transition arcs \( A_1 \)–\( A_4 \).) In the unfolding, it becomes clear that taking transition \( B_2 \) is a bad move, because reaching the bad marking containing \( Env \) and either \( BA \) or \( BB \) now has become unavoidable. A strategy is a subprocess of the unfolding that preserves the local nondeterminism of the environment token. Fig. ?? shows a winning strategy for the system players: by omitting the dashed arrows, they can make all bad markings unreachable and therefore win the game.

We show that for a single environment token and an arbitrary (but bounded) number of system tokens, deciding the existence of a safety strategy for the system players is EXPTIME-complete. This means that as long as there is a single source of information, such as the input of an algorithm or the sender in a communication protocol, solving Petri games is no more difficult than solving standard combinatorial games under complete information [?]. The case of Petri games with two or more environment tokens, i.e., situations with two or more independent information sources, remains open.

The remainder of the paper is structured as follows. In Section ?? we introduce the notion of Petri games and define strategies based on net unfoldings. In Section ?? we show that for concurrency preserving games every strategy can be distributed over local controllers. In Section ?? we introduce the new notion of mcuts on net unfoldings. In Section ?? we show that the problem of deciding the winner...
Petri games: Global strategy

B. Finkbeiner and E.-R. Olderog

A global strategy is now obtained from the unfolding by deleting some of the branches that are under control of the system players. We call this a “global” strategy because it looks at all players simultaneously. Note that nevertheless a strategy describes for each place which transitions the player in that place can take. Formally, this is expressed by the net-theoretic notion of subprocess.
Petri games: Play

Figure 5: Unfolding of the Petri game in Fig. ?? To aid visibility, the transitions leading to $q_{\text{bad}}$ are omitted from the unfolding. If the transitions shown with dashed lines are removed from the unfolding, the resulting net is a winning strategy for the system players.

Example 2.1 Fig. ?? shows the underlying P/T net $N$ of a small Petri game for two system players in place Sys and one environment player in place Env. Environment places are white and system places are gray. The environment chooses A or B by executing one of the transitions $t_1$ or $t_2$.

The goal of the system players is to achieve the same decisions than Env, i.e., both system players should choose $A'$ if Env chooses A, and $B'$ if Env chooses B. Without communication, the system players do not know which decision the environment has taken. However, when both system players and the environment communicate by synchronizing via the transitions $\text{test}_1$ or $\text{test}_2$, the system players learn about the decision taken by the environment and can mimic it. If $\text{test}_1$ was successful, they choose $A'$ via transition $t'_1$, and if $\text{test}_2$ was successful, they choose $B'$ via transition $t'_2$. □

Unfolded global strategy

An unfolded (global) strategy for the system players in $G$ is a subprocess $\sigma=\left(N_\sigma, \lambda_\sigma\right)$ of the unfolding $\beta_U=\left(N_U, \lambda_U\right)$ of $N$ subject to the following conditions for all $p \in P_\sigma$:

(S1) if $p \in P_{\text{S}}$ then $\sigma$ is deterministic at $p$,

(S2) if $p \in P_{\text{E}}$ then $\forall t \in T_U: (p, t) \in F_U \land |\text{pre}_U(t)| = 1 \Rightarrow (p, t) \in F_\sigma$, i.e., at an environment place the strategy does not restrict any local transitions.

Due to the unfolding, a decision taken by $\sigma$ in a place $p$ depends on the causal past of $p$, which may be arbitrarily large. The adjective "global" indicates that $\sigma$ looks at all players simultaneously. Local controllers are discussed in Section ??.
Strategy distribution

- A global strategy $\sigma$ is distributable if $\sigma$ can be represented as the parallel composition of local controllers for the environment and the system players in the sense that the reachable part of the parallel composition is isomorphic to $\sigma$.

Distribution Lemma

Every global strategy for a concurrency-preserving Petri game is distributable.
Petri games: Distributed strategy

\[ \mathcal{N}_1 \parallel \mathcal{N}_2 = (\mathcal{P}_1 \cup \mathcal{P}_2, \mathcal{T}_1 \cup \mathcal{T}_2, \mathcal{F}_1 \cup \mathcal{F}_2, \text{In}_1 \cup \text{In}_2) \]
Solving Petri games

EXPTIME-completeness

For Petri games with
- one environment player,
- a bounded number of system players,
- a safety objective (given as a set of safe places)

the question whether the system players have a deadlock-avoiding winning strategy is EXPTIME-complete. If a winning strategy for the system players exists, it can be constructed in exponential time.

Proof via reduction to Büchi games

[F./Olderog, 2014]
• Tool for the automated synthesis of distributed systems based on Petri games
• BDD-based symbolic representation of sets of cuts/mcuts
• Various optimizations (e.g., net invariants)
• Case studies from manufacturing with many (>10) independent agents

[F./Gieseking/Olderog, 2015]
ADAM

The diagram shows the benchmarks for Concurrent Machines (CM), Self-reconfiguring robots (SR), Job processing (JP), Document workflow (DW), and Document workflow simple (DWs) as a function of the number of processes. The y-axis represents time in minutes, and the x-axis represents the number of processes. The graph illustrates the performance and efficiency of these benchmarks under different process loads.
Conclusions
Temporal synthesis is a hard problem

1-process architectures --- 2EXPTIME

Pipeline architectures --- non-elementary

2-process arbiter architecture --- undecidable
Success stories

Figure 1: Left. Alice, Team Caltech’s entry in the 2007 DARPA Urban Challenge. Right. Alice’s navigation protocol stack that reactively determines the motion of the vehicle based on the current state of its environment (as perceived by the sensing and estimation subsystems).
Bounded synthesis

1-process architectures --- NP

Pipeline architectures --- NP

2-process arbiter architecture --- NP
Overview papers

**Synthesis of Reactive Systems**
Bernd Finkbeiner  
DOI 10.3233/978-1-61499-627-9-72  
https://www.react.uni-saarland.de/publications/F16.html

**Reactive Synthesis: Towards Output-Sensitive Algorithms**
Bernd Finkbeiner and Felix Klein  
DOI 10.3233/978-1-61499-810-5-25  
https://www.react.uni-saarland.de/publications/FKl17.html

**Automata, Games, and Verification**
Bernd Finkbeiner, Felix Klein, Tobias Salzmann  
Lecture Notes  