Synthesis of Reactive Systems

Bernd Finkbeiner
Universität des Saarlandes

MOVEP, July 2018
Synthesis

specification

synthesis

implementation

unrealizable
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
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

Traffic Planner

Path Planner and Follower

Actuation Interface

Vehicle

Path planning problem

Response

Path planning problem

Response

Actuation commands

Response

Actuation commands

Actuator state

Planning

Reactive strategy

Continuous control
Hardware example: AMBA AHB Bus

- High-performance on-chip bus
- Data, address, and control signals
- Up to 16 masters and 16 clients
- Specification consists of
  - 12 guarantees:
    for example “when a locked unspecified length burst starts, new access does not start until 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

Given a requirement \( \phi \) on the input-output behavior of a boolean circuit, compute a circuit \( C \) that satisfies \( \phi \).

Alonzo Church (1957)
Game theoretic formulation

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

The diagram illustrates a winning strategy for a game. The states are represented by circles, and the transitions are labeled with sets of elements. The strategy involves choosing a move that results in a state where the opponent has no winning moves. The states are labeled with \( g_1 \) and \( g_2 \), and the moves are labeled with elements from \( \{r_1\} \) and \( \{r_2\} \). The diagram shows the structure of the winning strategy, with arrows indicating the transitions that lead to victory.
Reactive 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. **Game-based Synthesis**
The „classic“ approach via a reduction to infinite games played over finite graphs.

Part II. **Bounded Synthesis**
Finding simple solutions fast via constraint solving.

Part III. **Distributed Synthesis**
Synthesizing systems that consist of multiple distributed components.
Part I: Game-based Synthesis

1. Infinite games over finite graphs
2. Synthesis from temporal logic
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 $\sigma : V^* \cdot V_i \rightarrow V$ such that $(v_n, \sigma(v_0v_1\ldots v_n)) \in E$ for every prefix $v_0v_1\ldots v_n$ of a play.

A play $v_0v_1\ldots$ is consistent with strategy $\sigma$, if $v_{n+1} = \sigma(v_0\ldots v_n)$ whenever $v_n \in V_i$.

Special types of strategies:

- **Positional strategies:** $\sigma(v_0v_1\ldots v_n) = \sigma(v_n)$
  strategy only depends on last state

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

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

- A **Büchi game** $G = (\mathcal{A}, F)$ consists of an arena $\mathcal{A}$ 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** $G = (\mathcal{A}, \alpha)$ consists of an arena $\mathcal{A}$ and a coloring function $\alpha : V \rightarrow \mathbb{N}$. **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.
Reachability games

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

More on game solving in Marcin Jurdzinski’s talk tomorrow.
Part I: Game-based Synthesis

1. Infinite games over finite graphs
2. Synthesis from temporal logic
Linear-time temporal logic

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:
  - $F \varphi \equiv \text{true} \mathbin{U} \varphi$
  - $G \varphi \equiv \neg (F \neg \varphi)$
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 $i$.

   \[ G r_1 \Rightarrow XF g_1 \]
   \[ \land \]
   \[ G r_2 \Rightarrow XF g_2 \]
LTL synthesis

LTL formula

- translation

nondeterministic Büchi automaton

- determinization

deterministic parity automaton

- spreading into inputs and outputs

parity game

- Player 0 wins
- Player 1 wins

realizable

unrealizable
Simplified example

Simple response property:

\[ G (r \Rightarrow F g) \]
Simplified example

Simple response property:

\[ G(r \Rightarrow Fg) \]
Simplified example

Simple response property:

\[ G(r \Rightarrow Fg) \]
A winning strategy for the Player $o$ in the parity game can be represented as an FSM.
Why deterministic parity automata?

Why deterministic automata?

- Nondeterministic automata may have rejecting runs on accepted sequences.
- Player o may lose a play just because the wrong run was chosen.
- **Example:** \((F \land Xo) \lor (G \land X\neg o)\)
  - Player o has a winning strategy (copy input i to output o)
  - Player o 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: \(F \land p\)
**Complexity**

- **LTL formula**
  - #states: exponential
- **nondeterministic Büchi automaton**
  - #states: doubly exponential, #colors: exponential
- **deterministic parity automaton**
  - #states: doubly exponential, #colors: exponential
- **parity game**
  - Game solving: polynomial in states, < exponential in colors

**LTL synthesis**

LTL synthesis is 2EXPTIME complete.
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:**

  \[\varphi ::= \Phi \mid \varphi_1 \land \varphi_2 \mid \neg \varphi \mid X\varphi \mid \varphi_1 U \varphi_2\]
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:
**CTL* synthesis**

- **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 → 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: $\text{AG} (r \Rightarrow \text{AF} g)$
- “If there are infinitely many requests then there are infinitely many grants” cannot be expressed.
**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 \( G \left( \phi \rightarrow X \psi \right) \),
- **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 $\text{G} (\varphi \rightarrow X \psi)$
  - 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 $\text{G} F \varphi$ 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)

\[(GF a_1) \land (GF a_2) \land \ldots \land (GF a_m) \rightarrow (GF g_1) \land (GF g_2) \land \ldots \land (GF g_n)\]
Part I.  Game-based Synthesis

The "classic" approach via a reduction to infinite games played over finite graphs.

Part II.  Bounded Synthesis

Finding simple solutions fast via constraint solving.

Part III.  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

- the size / number of states of the solution
- the number of simple cycles of the solution

Bounded Synthesis (Finkbeiner & Schewe, 2007)
Standard synthesis

(Acacia+ v2.3)
Bounded synthesis

**Synthesis**
- Is there an implementation that satisfies the specification?

**Bounded Synthesis**
- Is there an implementation with no more than $N$ states?

[Schewe/F. 2007]
From input to output complexity

Synthesis

- 2EXPTIME 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 II: Bounded Synthesis

1. Universal co-Büchi automata
2. Constraint systems
Universal co-Büchi automata

Universal co-Büchi tree automaton for an arbiter specification:

\[ G \neg (g_1 \land g_2) \land \]
\[ G r_1 \Rightarrow X F g_1 \land \]
\[ G r_2 \Rightarrow X F g_2 \]
Universal co-Büchi automata

Universal co-Büchi tree automaton for an arbiter specification:

Candidate implementation:
Universal co-Büchi automata

Universal co-Büchi tree automaton for an arbiter specification:

Run graph:

Candidate implementation:
Universal co-Büchi automata

Universal co-Büchi tree automaton for an arbiter specification:

Candidate implementation:

Run graph:
Universal co-Büchi automata

Universal co-Büchi tree automaton for an arbiter specification:

Run graph:

Candidate implementation:
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.
Part II: Bounded Synthesis

1. Universal co-Büchi automata
2. Constraint systems
Constraint system

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 $\lambda^\#_q : \mathbb{N}_N \to \mathbb{N}$
Constraints

- $\lambda^B_G(o)$

- $\forall t. \lambda^B_G(t) \rightarrow \lambda^B_G(\tau_{\bar{1} \bar{2}}(t)) \land \lambda^#_G(\tau_{\bar{1} \bar{2}}(t)) \geq \lambda^#_G(t)$
  $\land \lambda^B_G(\tau_{\bar{1} \bar{2}}(t)) \land \lambda^#_G(\tau_{\bar{1} \bar{2}}(t)) \geq \lambda^#_G(t)$
  $\land \lambda^B_G(\tau_{\bar{1} \bar{2}}(t)) \land \lambda^#_G(\tau_{\bar{1} \bar{2}}(t)) \geq \lambda^#_G(t)$
  $\land \lambda^B_G(\tau_{\bar{1} \bar{2}}(t)) \land \lambda^#_G(\tau_{\bar{1} \bar{2}}(t)) \geq \lambda^#_G(t)$

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

- $\forall t. \lambda^B_G(t) \land r_1(t) \rightarrow \lambda^B_B(\tau_{\bar{1} \bar{2}}(t)) \land \lambda^#_B(\tau_{\bar{1} \bar{2}}(t)) > \lambda^#_G(t)$
  $\land \lambda^B_B(\tau_{\bar{1} \bar{2}}(t)) \land \lambda^#_B(\tau_{\bar{1} \bar{2}}(t)) > \lambda^#_G(t)$
  $\land \lambda^B_B(\tau_{\bar{1} \bar{2}}(t)) \land \lambda^#_B(\tau_{\bar{1} \bar{2}}(t)) > \lambda^#_G(t)$
  $\land \lambda^B_B(\tau_{\bar{1} \bar{2}}(t)) \land \lambda^#_B(\tau_{\bar{1} \bar{2}}(t)) > \lambda^#_G(t)$

- $\forall t. \lambda^B_B(t) \land \neg g_1(t) \rightarrow \lambda^B_B(\tau_{\bar{1} \bar{2}}(t)) \land \lambda^#_B(\tau_{\bar{1} \bar{2}}(t)) > \lambda^#_G(t)$
  $\land \lambda^B_B(\tau_{\bar{1} \bar{2}}(t)) \land \lambda^#_B(\tau_{\bar{1} \bar{2}}(t)) > \lambda^#_G(t)$
  $\land \lambda^B_B(\tau_{\bar{1} \bar{2}}(t)) \land \lambda^#_B(\tau_{\bar{1} \bar{2}}(t)) > \lambda^#_G(t)$
  $\land \lambda^B_B(\tau_{\bar{1} \bar{2}}(t)) \land \lambda^#_B(\tau_{\bar{1} \bar{2}}(t)) > \lambda^#_G(t)$

- $\forall t. \lambda^B_B(t) \land r_2(t) \rightarrow \lambda^B_B(\tau_{\bar{1} \bar{2}}(t)) \land \lambda^#_B(\tau_{\bar{1} \bar{2}}(t)) > \lambda^#_G(t)$
  $\land \lambda^B_B(\tau_{\bar{1} \bar{2}}(t)) \land \lambda^#_B(\tau_{\bar{1} \bar{2}}(t)) > \lambda^#_G(t)$
  $\land \lambda^B_B(\tau_{\bar{1} \bar{2}}(t)) \land \lambda^#_B(\tau_{\bar{1} \bar{2}}(t)) > \lambda^#_G(t)$
  $\land \lambda^B_B(\tau_{\bar{1} \bar{2}}(t)) \land \lambda^#_B(\tau_{\bar{1} \bar{2}}(t)) > \lambda^#_G(t)$
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:

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

[Faymonville, F., Tentrup, Rabe, 2017]
**Constraints**

- $\lambda^B_G(o)$

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

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

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

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

![Diagram](https://via.placeholder.com/150)
**Constraints**

- $\lambda_B^G(o)$
- $\forall t. \lambda_B^G(t) \rightarrow \lambda_B^G(\tau_1, \tau_2(t)) \land \lambda_B^G(\tau_1, \tau_2(t)) \geq \lambda_B^G(t)$
- $\forall t. \lambda_B^G(t) \land r_1(t) \rightarrow \lambda_B^G(\tau_1, \tau_2(t)) \land \lambda_B^G(\tau_1, \tau_2(t)) > \lambda_B^G(t)$
- $\forall t. \lambda_B^G(t) \land \neg g_1(t) \rightarrow \lambda_B^G(\tau_1, \tau_2(t)) \land \lambda_B^G(\tau_1, \tau_2(t)) > \lambda_B^G(t)$

---

different states of the FSM
Constraints

- $\lambda^B_G(o)$

- $\forall t. \lambda^B_G(t) \rightarrow \lambda^B_G(\tau_{\overline{r}_1 \overline{r}_2}(t)) \land \lambda^G_G(\tau_{\overline{r}_1 \overline{r}_2}(t)) \geq \lambda^G_G(t)$
  $\land \lambda^B_G(\tau_{\overline{r}_1 r_2}(t)) \land \lambda^G_G(\tau_{\overline{r}_1 r_2}(t)) \geq \lambda^G_G(t)$
  $\land \lambda^B_G(\tau_{r_1 \overline{r}_2}(t)) \land \lambda^G_G(\tau_{r_1 \overline{r}_2}(t)) \geq \lambda^G_G(t)$
  $\land \lambda^B_G(\tau_{r_1 r_2}(t)) \land \lambda^G_G(\tau_{r_1 r_2}(t)) \geq \lambda^G_G(t)$

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

- $\forall t. \lambda^B_G(t) \land r_1(t) \rightarrow \lambda^B_B(\tau_{\overline{r}_1 \overline{r}_2}(t)) \land \lambda^B_B(\tau_{\overline{r}_1 \overline{r}_2}(t)) > \lambda^B_G(t)$
  $\land \lambda^B_B(\tau_{\overline{r}_1 r_2}(t)) \land \lambda^B_B(\tau_{\overline{r}_1 r_2}(t)) > \lambda^B_G(t)$
  $\land \lambda^B_B(\tau_{r_1 \overline{r}_2}(t)) \land \lambda^B_B(\tau_{r_1 \overline{r}_2}(t)) > \lambda^B_G(t)$
  $\land \lambda^B_B(\tau_{r_1 r_2}(t)) \land \lambda^B_B(\tau_{r_1 r_2}(t)) > \lambda^B_G(t)$

- $\forall t. \lambda^B_B(t) \land \neg g_1(t) \rightarrow \lambda^B_B(\tau_{\overline{r}_1 \overline{r}_2}(t)) \land \lambda^B_B(\tau_{\overline{r}_1 \overline{r}_2}(t)) > \lambda^B_B(t)$
  $\land \lambda^B_B(\tau_{\overline{r}_1 r_2}(t)) \land \lambda^B_B(\tau_{\overline{r}_1 r_2}(t)) > \lambda^B_B(t)$
  $\land \lambda^B_B(\tau_{r_1 \overline{r}_2}(t)) \land \lambda^B_B(\tau_{r_1 \overline{r}_2}(t)) > \lambda^B_B(t)$
  $\land \lambda^B_B(\tau_{r_1 r_2}(t)) \land \lambda^B_B(\tau_{r_1 r_2}(t)) > \lambda^B_B(t)$

Different states of the automaton
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)
Implementation size (#AND gates)
Implementation size (#AND gates)
**Bounded synthesis as a decision procedure**

**Increase bound** until

- implementation found, or
- counterstrategy found (by searching for implementation in dual problem)

**Termination:** There is an implementation or a counterstrategy with at most doubly exponentially many states

- the parity game has doubly exponentially many states
- one of the players has a winning positional strategy, i.e., a winning strategy with at most doubly exponentially many states.
BoSy

https://www.react.uni-saarland.de/tools/online/BoSy/
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

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
- the number of simple cycles of the solution
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 $\varphi$.

[F./Klein 2016]
## Experiments

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Size</th>
<th>Cycles</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$A_\varphi$</td>
<td>Aca+</td>
<td>BoSy/BoCy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Aca+</td>
<td>BoSy</td>
</tr>
<tr>
<td>ARBITER[2]</td>
<td>6</td>
<td>26</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>111</td>
<td>3</td>
</tr>
<tr>
<td>LOCK[2]</td>
<td>12</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>LOCK[3]</td>
<td>20</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>ENCODE[2]</td>
<td>3</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>ENCODE[3]</td>
<td>5</td>
<td>16</td>
<td>3</td>
</tr>
<tr>
<td>ENCODE[4]</td>
<td>5</td>
<td>20</td>
<td>4</td>
</tr>
<tr>
<td>DECODE</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>SHIFT</td>
<td>3</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>TBURST4</td>
<td>103</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>TINCR</td>
<td>43</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>TSINGLE</td>
<td>22</td>
<td>8</td>
<td>4</td>
</tr>
</tbody>
</table>
Overview

Part I. **Game-based Synthesis**

The „classic“ approach via a reduction to infinite games played over finite graphs.

Part II. **Bounded Synthesis**

Finding simple solutions fast via constraint solving.

Part III. **Distributed Synthesis**

Synthesizing systems that consist of multiple distributed components.
Distributed synthesis

specification

implementation

unrealizable

Alonzo Church, 1957

Applications of recursive arithmetic to the problem of circuit synthesis

distributed synthesis

implementation

unrealizable

Alonzo Church, 1957

Applications of recursive arithmetic to the problem of circuit synthesis
Part III: Distributed Synthesis

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

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

Edges:
- communication structure
- variables

[Pnueli/Rosner,1989]
Pnueli/Rosner model

The implementation defines for each process $p$ with input variables $I_p$ and output variables $O_p$ a strategy tree with directions $2^{I_p}$ and labels $2^{O_p}$.
Pnueli/Rosner model

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 \( x \).
Partial observation

process 2 does not know $x$

decisions of process 2 must not depend on $x$. 
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

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

Undecidable
Transformation-based synthesis

- $\mathcal{A}_\varphi$ --- accepts implementations for super-process.
- $\mathcal{B}_\varphi$ --- accepts an implementation for 2 iff there is an implementation for process 1 such that their composition is accepted by $\mathcal{A}_\varphi$
- If $\mathcal{B}_\varphi$ is nonempty it accepts an implementation 2
- $\mathcal{A}_\varphi'$ --- accepts proper implementations for 1.
- If $\mathcal{A}_\varphi'$ is non-empty, it accepts an implementation 1

[Kupferman/Vardi 2001]
Synthesis in the Pnueli/Rosner model

1-process architectures --- 2EXPTIME

Pipeline architectures --- non-elementary

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

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

1-process architectures --- NP

![Diagram](attachment://1-process_architectures.png)

Pipeline architectures --- NP

![Diagram](attachment://pipeline_architectures.png)

2-process arbiter architecture --- NP

![Diagram](attachment://2-process_arbiter_architecture.png)
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

\[ \forall t. \tau_{1;r_1,g_2(\text{proj}_2(t))}(\text{proj}_1(t)) = \text{proj}_1(\tau_{r_1r_2}(t)) = \text{proj}_1(\tau_{\bar{r}_1r_2}(t)) \]
\[ \land \tau_{1;\bar{r}_1,g_2(\text{proj}_2(t))}(\text{proj}_1(t)) = \text{proj}_1(\tau_{\bar{r}_1r_2}(t)) = \text{proj}_1(\tau_{\bar{r}_1\bar{r}_2}(t)) \]

\[ \forall t. \tau_{2;r_2,g_1(\text{proj}_1(t))}(\text{proj}_2(t)) = \text{proj}_2(\tau_{r_1r_2}(t)) = \text{proj}_2(\tau_{\bar{r}_1r_2}(t)) \]
\[ \land \tau_{2;\bar{r}_2,g_1(\text{proj}_1(t))}(\text{proj}_2(t)) = \text{proj}_2(\tau_{r_1\bar{r}_2}(t)) = \text{proj}_2(\tau_{\bar{r}_1\bar{r}_2}(t)) \]
3 generals, consisting of 1 commander and 2 lieutenants, need to agree on a plan of attack.

One of them is a traitor.

Specification:
- if the commander is the traitor, the lieutenants should reach consensus.
- if one of the lieutenants is the traitor, the loyal lieutenant should agree with the commander.

The specification is **unrealizable**.
A counterexample in model checking is one path that violates $\varphi$. 

$x \iff X z$
Counterexamples in synthesis

A counterexample to realizability is a set of paths such that for all implementations at least one path violates $\varphi$. 
Bounded counterexamples

A counterexample to realizability is a **set of paths** such that for **all** implementations **at least one** path violates $\varphi$.

- **Bounded-size counterexamples:** bounded number of paths in the set
- **Bounded-length counterexamples:** paths with bounded length

Bounded counterexamples are a sufficient criterion (but not in general necessary) for **unrealizability**.

[F./Tentrup, 2014]
QBF encoding

\[ \exists I_1^0, I_1^1, \ldots, I_1^k, \ldots, I_n^0, I_n^1, \ldots, I_n^k. \]
\[ \forall S_1^0, S_1^1, \ldots, S_1^k, \ldots, S_n^0, S_n^1, \ldots, S_n^k. \]
\[ \text{consistent}(S_1^0, \ldots, S_1^k, \ldots, S_n^0, \ldots, S_n^k) \]
\[ \rightarrow \bigvee_{1 \leq i \leq n} \text{counterexample}(\varphi_i) \]
Part III: Distributed Synthesis

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

Quantifiers with trace variables: $\forall \pi. \varphi$  $\exists \pi. \varphi$

Syntax: $\varphi ::= \forall \pi. \varphi \mid \exists \pi. \varphi \mid \psi$
$\psi ::= a_\pi \mid X \psi \mid \psi U \psi$

“All executions have the light on at the same time.”

$\forall \pi. \forall \pi'. G (on_\pi \leftrightarrow on_{\pi'} )$

[Clarkson/F./Koleini/Micinski/Rabe/Sánchez, 2014]
Satisfiability of HyperLTL

Does there exist a non-empty trace set $T$ that satisfies a given HyperLTL formula $\varphi$?

**Application:** Two versions of Observational Determinism:
- $\forall \pi. \forall \pi'. G(l_\pi = l_{\pi'}) \rightarrow G(O_\pi = O_{\pi'})$
- $\forall \pi. \forall \pi'. (O_\pi = O_{\pi'}) \not\forall (l_\pi \neq l_{\pi'})$

Which version is stronger?

<table>
<thead>
<tr>
<th>$\exists^*$</th>
<th>$\forall^*$</th>
<th>$\exists^* \forall^*$</th>
<th>Bounded $\exists^* \forall^*$</th>
<th>$\forall \exists$</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSPACE-complete</td>
<td>PSPACE-complete</td>
<td>EXPSPACE-complete</td>
<td>PSPACE-complete</td>
<td>undecidable</td>
</tr>
</tbody>
</table>

[F./Hahn, 2016]
The power of $\forall \exists$

$\exists \pi'. a_{\pi'}$  \hspace{1cm} (1)

$\forall \pi. \ G(a_\pi \rightarrow XG\neg a_\pi)$  \hspace{1cm} (2)

$\forall \pi \exists \pi'. G(a_\pi \rightarrow Xa_{\pi'})$  \hspace{1cm} (3)

$t_1 : \{a\}(\{}\{}\omega$

$t_2 : \{}\{}\{a\}(\{}\{}\omega$

$t_3 : \{}\{}\{}\{a\}(\{}\{}\omega$

$\ldots$

$\rightarrow$ Model has infinitely many traces.
Distributed systems

› (In)dependence:

\[ D_{A \to C} := \forall \pi \forall \pi'. \left( \bigvee_{a \in A} (a_\pi \leftrightarrow a_{\pi'}) \right) \mathcal{R} \left( \bigwedge_{c \in C} (c_\pi \leftrightarrow c_{\pi'}) \right) \]

› Distributed synthesis:

\[ \forall \pi \forall \pi'. \phi \land \bigwedge_{p \in P^-} D_{I(p) \to O(p)} \]
HyperLTL synthesis

- With HyperLTL, architectural constraints can be expressed directly in the logic
- And more: fault tolerance, symmetry, secrecy, ...
- Undecidable in general, bounded synthesis provides efficient semi-algorithms

<p>| $\exists^<em>$ | $\exists^</em> \forall^1$ | $\exists^* \forall &gt;^1 \forall^<em>$ | linear $\forall^</em>$ |
| PSPACE | 3EXPTIME | undecidable | decidable |</p>
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Property</th>
<th>Result</th>
<th>Time[sec.]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dining Cryptographers</td>
<td>secrecy</td>
<td>realizable</td>
<td>82.4</td>
</tr>
<tr>
<td>Encoder/Decoder</td>
<td>1-2-hamming-2</td>
<td>realizable</td>
<td>1.3</td>
</tr>
<tr>
<td></td>
<td>1-2-fault-tolerant</td>
<td>unrealizable</td>
<td>54.9</td>
</tr>
<tr>
<td></td>
<td>1-3-fault-tolerant</td>
<td>realizable</td>
<td>1.7</td>
</tr>
<tr>
<td></td>
<td>2-2-hamming-2</td>
<td>unrealizable</td>
<td>10.6</td>
</tr>
<tr>
<td></td>
<td>2-3-hamming-2</td>
<td>realizable</td>
<td>1.5</td>
</tr>
<tr>
<td></td>
<td>2-3-hamming-3</td>
<td>unrealizable</td>
<td>126.7</td>
</tr>
<tr>
<td>Bus Protocol</td>
<td>noninterference</td>
<td>unrealizable</td>
<td>69.6</td>
</tr>
<tr>
<td></td>
<td>short-term-interference</td>
<td>realizable</td>
<td>33.9</td>
</tr>
<tr>
<td>Symmetric Mutex</td>
<td>non-sym</td>
<td>realizable</td>
<td>1.3</td>
</tr>
<tr>
<td></td>
<td>sym</td>
<td>unrealizable</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>tie</td>
<td>realizable</td>
<td>1.6</td>
</tr>
<tr>
<td>CAP Theorem</td>
<td>cap-2</td>
<td>unrealizable</td>
<td>1823.9</td>
</tr>
<tr>
<td></td>
<td>ca-2</td>
<td>realizable</td>
<td>4.4</td>
</tr>
<tr>
<td></td>
<td>ca-3</td>
<td>realizable</td>
<td>15.0</td>
</tr>
<tr>
<td></td>
<td>cp-2</td>
<td>realizable</td>
<td>1.6</td>
</tr>
<tr>
<td></td>
<td>cp-3</td>
<td>realizable</td>
<td>10.6</td>
</tr>
<tr>
<td></td>
<td>ap-2</td>
<td>realizable</td>
<td>2.0</td>
</tr>
<tr>
<td></td>
<td>ap-3</td>
<td>realizable</td>
<td>43.4</td>
</tr>
</tbody>
</table>
Conclusions
Synthesis of reactive systems is hard

1-process architectures --- 2EXPTIME

Pipeline architectures --- non-elementary

2-process arbiter architecture --- undecidable
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  