Safety First: A Two-Stage Algorithm for LTL Games

Saqib Sohail    Fabio Somenzi

Department of Electrical and Computer Engineering
University of Colorado at Boulder

FMCAD 2009
Recently, significant algorithmic advances in the game-theoretic approach to synthesis of reactive systems has renewed interest. Piterman 06, Piterman et al 06, Kupferman et al 06, Chatterjee et al 07, Bloem et al 07 are a few examples.

Despite challenges in scalability, there is increasing hope that synthesis algorithms may be applied to the design and diagnosis of intricate, safety critical protocols.

The focus will be on how to avoid some of these challenges without any compromises.
Motivation

- Recently, significant algorithmic advances in the game-theoretic approach to synthesis of reactive systems has renewed interest. Piterman 06, Piterman et al 06, Kupferman et al 06, Chatterjee et al 07, Bloem et al 07 are a few examples.

- Despite challenges in scalability, there is increasing hope that synthesis algorithms may be applied to the design and diagnosis of intricate, safety critical protocols.

- The focus will be on how to avoid some of these challenges without any compromises.
Recently, significant algorithmic advances in the game-theoretic approach to synthesis of reactive systems has renewed interest. Piterman 06, Piterman et al 06, Kupferman et al 06, Chatterjee et al 07, Bloem et al 07 are a few examples.

Despite challenges in scalability, there is increasing hope that synthesis algorithms may be applied to the design and diagnosis of intricate, safety critical protocols.

The focus will be on how to avoid some of these challenges without any compromises.
Outline

1. Introduction

2. Games

3. Two Stage Synthesis
   - The Challenge
   - Algorithm
   - Optimizations
   - Implementation
   - Caveats

4. Results

5. Conclusions
LTL Synthesis - Pnueli and Rosner (POPL’89)

- **Automatically build design from specification**
  - Input:
    - Set of LTL formulae, e.g. $G(req \rightarrow F \text{ack})$, $G(\neg req \rightarrow X(\neg \text{ack}))$
    - Partition of the atomic propositions (input/output signals)
    - Environment controls inputs and system controls outputs
  - The set of LTL formulae are converted to a non-terminating game with system as **protagonist** and environment as **antagonist**.
  - Output: Automatically created functionally correct finite-state machine from the **winning strategy** of the system.
    - If such strategy doesn’t exist then the specification is **unrealizable**.

Safety First: A Two-Stage Algorithm for LTL Games
**LTL Synthesis - Pnueli and Rosner (POPL’89)**

- Automatically build design from specification
- **Input:**
  - Set of LTL formulae, e.g. $G(req \rightarrow F\, ack)$, $G(\neg req \rightarrow X(\neg ack))$
  - Partition of the atomic propositions (input/output signals)
  - Environment controls inputs and system controls outputs
- The set of LTL formulae are converted to a non-terminating game with system as **protagonist** and environment as **antagonist**.
- **Output:** Automatically created functionally correct finite-state machine from the **winning strategy** of the system.
- If such strategy doesn’t exist then the specification is **unrealizable**.
Automatically build design from specification

- Input:
  - Set of LTL formulae, e.g. $G(req \rightarrow F\ ack)$, $G(\neg req \rightarrow X(\neg ack))$
  - Partition of the atomic propositions (input/output signals)
  - Environment controls inputs and system controls outputs

- The set of LTL formulae are converted to a non-terminating game with system as protagonist and environment as antagonist.
  - If such strategy doesn’t exist then the specification is unrealizable.
LTL Synthesis - Pnueli and Rosner (POPL’89)

- Automatically build design from specification
- Input:
  - Set of LTL formulae, e.g. $G(req \rightarrow F\ text{ack})$, $G(\neg req \rightarrow X(\neg\ text{ack}))$
  - Partition of the atomic propositions (input/output signals)
  - Environment controls inputs and system controls outputs
- The set of LTL formulae are converted to a non-terminating game with system as protagonist and environment as antagonist.
- Output: Automatically created functionally correct finite-state machine from the winning strategy of the system.
  - If such strategy doesn’t exist then the specification is unrealizable.
The system’s intended behavior is described by combination of LTL formulae or as $\omega$-regular automata.

In a naive approach, all formulae and automata are reduced to one deterministic automaton, whose transition structure provides the game graph.

The acceptance condition is taken as the winning condition.

This approach suffers from the high cost of determinization, which is prohibitive for even moderate-sized automata.

How to avoid the high costs?
The system’s intended behavior is described by combination of LTL formulae or as $\omega$-regular automata.

In a naive approach, all formulae and automata are reduced to one deterministic automaton, whose transition structure provides the game graph.

The acceptance condition is taken as the winning condition.

This approach suffers from the high cost of determinization, which is prohibitive for even moderate-sized automata.

How to avoid the high costs?
The system’s intended behavior is described by combination of LTL formulae or as $\omega$-regular automata.

In a naive approach, all formulae and automata are reduced to one deterministic automaton, whose transition structure provides the game graph.

The acceptance condition is taken as the winning condition.

This approach suffers from the high cost of determinization, which is prohibitive for even moderate-sized automata.

How to avoid the high costs?
LTL Synthesis - Pnueli and Rosner (POPL’89)

- The system’s intended behavior is described by combination of LTL formulae or as \( \omega \)-regular automata.
- In a naive approach, all formulae and automata are reduced to one deterministic automaton, whose transition structure provides the game graph.
- The acceptance condition is taken as the winning condition.
- This approach suffers from the high cost of determinization, which is prohibitive for even moderate-sized automata.
- How to avoid the high costs?
LTL Synthesis - Pnueli and Rosner (POPL’89)

- The system’s intended behavior is described by combination of LTL formulae or as $\omega$-regular automata.
- In a naive approach, all formulae and automata are reduced to one deterministic automaton, whose transition structure provides the game graph.
- The acceptance condition is taken as the winning condition.
- This approach suffers from the high cost of determinization, which is prohibitive for even moderate-sized automata.
- How to avoid the high costs?
Example: Game graph with a parity acceptance condition

player 0 → □
wins if largest integer occurring infinitely often is even

player 1 → ○
wins if largest integer occurring infinitely often is odd
Example: Game graph with a parity acceptance condition

player0 $\rightarrow \square$
wins if largest integer occuring infinitely often is even

player1 $\rightarrow \bigcirc$
wins if largest integer occuring infinitely often is odd
A game graph $G = ((S, E), S_0, S_1)$ is a directed graph $(S, E)$ with a finite state space $S$, a set of edges $E$ and a partition $(S_0, S_1)$ of the state space belonging to player 0 and 1 respectively. We assume that every state has an outgoing edge.

The game is started by placing a token in one of the $S_{init}$ and then this token is moved along the edges, when the token is in a state $s \in S_1$, player 1 selects one of its outgoing edges and vice-versa. The result is an infinite path in the game graph termed as a play.

A strategy for a player is a recipe that specifies how to extend finite path. Formally strategy for player $i$ is a function $\sigma : S^* \cdot S_i \rightarrow S$. 

Safety First: A Two-Stage Algorithm for LTL Games
For a game graph $G = (Q, E)$ and a parity function $\pi : Q \rightarrow [k]$, a parity acceptance condition requires that the maximal $\pi(s)$ occurring infinitely often is odd (even) for player 1 (0).

A generalized parity game for a game graph $G = (Q, E)$ and a set of parity functions $\{\pi_i | \pi_i : Q \rightarrow [k_i] \}$ is played between the conjunctive and disjunctive player. The conjunctive player wins if it has a strategy to win all the parity acceptance conditions while the disjunctive player wins if it has a strategy for some parity acceptance condition.
Parity Game

- For a game graph $G = (Q, E)$ and a parity function $\pi : Q \rightarrow [k]$, a parity acceptance condition requires that the maximal $\pi(s)$ occurring infinitely often is odd (even) for player 1 (0).

- A generalized parity game for a game graph $G = (Q, E)$ and a set of parity functions $\{\pi_i | \pi_i : Q \rightarrow [k_i]\}$ is played between the conjunctive and disjunctive player. The conjunctive player wins if it has a strategy to win all the parity acceptance conditions while the disjunctive player wins if it has a strategy for some parity acceptance condition.
Two Game Theoretic Approaches

- The standard approach which is the focus of this talk, requires the determinization of word automata.

\[ LTL \rightarrow NBW \rightarrow DRW \]

- The Safraless Approach avoids determinization by working with Tree Automata.

\[ LTL \rightarrow NGBW \rightarrow UGCW \rightarrow \text{realizability} \rightarrow \text{lang-empt} \rightarrow \text{optimistic-reduction} \rightarrow NBT \]
Two Game Theoretic Approaches

- The standard approach which is the focus of this talk, requires the determinization of word automata.

  $$LTL \rightarrow NBW \rightarrow DRW$$

- The Safraless Approach avoids determinization by working with Tree Automata.

  $$LTL \rightarrow NGBW \rightarrow UGCW \rightarrow UGCT \rightarrow NBT$$
Specification of a simple 2-Client Arbiter

- Initially there are no acknowledgments.
  \[ \neg ack_0 \land \neg ack_1 \]
- The acknowledgments are mutually exclusive.
  \[ \mathcal{G}(\neg ack_0 \lor \neg ack_1) \]
- There are no spurious acknowledgments.
  \[ \forall i . \mathcal{G}(\neg req_i \rightarrow X(\neg ack_i)) \]
- Every request will eventually be acknowledged
  \[ \forall i . \mathcal{G}(req_i \rightarrow F ack_i) \]
Example: Game Graph and Synthesized Strategy

Safety First: A Two-Stage Algorithm for LTL Games
Example: Game Graph and Synthesized Strategy
Example: Game play & Strategy Computation for Player 1

[Player 1] wins if the maximal $\pi(s)$ occurring infinitely often is odd.
Example: Game play & Strategy Computation for Player 1

[Player 1] wins if the maximal $\pi(s)$ occurring infinitely often is odd.
Example: Game play & Strategy Computation for Player 1

- [Player 1] wins if the maximal $\pi(s)$ occurring infinitely often is odd.
Example: Game play & Strategy Computation for Player 1

[Player 1] wins if the maximal $\pi(s)$ occurring infinitely often is odd.
Example: Game play & Strategy Computation for Player 1

○ [Player 1] wins if the maximal $\pi(s)$ occurring infinitely often is **odd**.
Example: Game play & Strategy Computation for Player 1

[Player 1] wins if the maximal $\pi(s)$ occurring infinitely often is odd.
Example: Game play & Strategy Computation for Player 1

[Player 1] wins if the maximal $\pi(s)$ occurring infinitely often is odd.
Example: Game play & Strategy Computation for Player 1

*Player 1* wins if the maximal $\pi(s)$ occurring infinitely often is odd.
Example: Game play & Strategy Computation for Player 1

[Player 1] wins if the maximal $\pi(s)$ occurring infinitely often is odd.
Example: Game play & Strategy Computation for Player 1

○ [Player 1] wins if the maximal $\pi(s)$ occurring infinitely often is odd.
Example: Game play & Strategy Computation for Player 1

[Player 1] wins if the maximal $\pi(s)$ occurring infinitely often is odd.
Example: Game play & Strategy Computation for Player 1

[Player 1] wins if the maximal $\pi(s)$ occurring infinitely often is odd.
Example: Game play & Strategy Computation for Player 1

[Player 1] wins if the maximal $\pi(s)$ occurring infinitely often is odd.
Example: Game play & Strategy Computation for Player 1

[Player 1] wins if the maximal $\pi(s)$ occurring infinitely often is odd.
Example: Game play & Strategy Computation for Player 1

○ [Player 1] wins if the maximal $\pi(s)$ occurring infinitely often is odd.
[Player 1] wins if the maximal $\pi(s)$ occurring infinitely often is odd.
Example: Game play & Strategy Computation for Player 1

[Player 1] wins if the maximal $\pi(s)$ occurring infinitely often is odd.
Example: Game play & Strategy Computation for Player 1

- [Player 1] wins if the maximal $\pi(s)$ occurring infinitely often is odd.
Example: Game play & Strategy Computation for Player 1

[Player 1] wins if the maximal $\pi(s)$ occurring infinitely often is odd.
Example: Generalized Parity Game

○ [Conjunctive Player] wins if it has a strategy to win all the parity functions

□ [Disjunctive Player] wins if it has a strategy to win according to some parity function
Example: Generalized Parity Game

- **Conjunctive Player** wins if it has a strategy to win all the parity functions

- **Disjunctive Player** wins if it has a strategy to win according to some parity function
Example: Generalized Parity Game

○ [Conjunctive Player] wins if it has a strategy to win all the parity functions

□ [Disjunctive Player] wins if it has a strategy to win according to some parity function
Example: Generalized Parity Game

- [Conjunctive Player] wins if it has a strategy to win all the parity functions
- [Disjunctive Player] wins if it has a strategy to win according to some parity function
Example: Generalized Parity Game

- [Conjunctive Player] wins if it has a strategy to win all the parity functions
- [Disjunctive Player] wins if it has a strategy to win according to some parity function
Example: Generalized Parity Game

○ [Conjunctive Player] wins if it has a strategy to win all the parity functions

□ [Disjunctive Player] wins if it has a strategy to win according to some parity function
Example: Generalized Parity Game

〇 [Conjunctive Player] wins if it has a strategy to win all the parity functions

□ [Disjunctive Player] wins if it has a strategy to win according to some parity function
Example: Generalized Parity Game

- [Conjunctive Player] wins if it has a strategy to win all the parity functions

- [Disjunctive Player] wins if it has a strategy to win according to some parity function
Example: Generalized Parity Game

- [Conjunctive Player] wins if it has a strategy to win all the parity functions

- [Disjunctive Player] wins if it has a strategy to win according to some parity function
1. Introduction
2. Games
3. Two Stage Synthesis
   - The Challenge
   - Algorithm
   - Optimizations
   - Implementation
   - Caveats
4. Results
5. Conclusions
The Challenge

- Generalized parity game is an **NP-Complete** problem and the current algorithm (Chatterjee et al. 07) is computationally very expensive.
- Is there a simpler solution to the complex problem?
- Is there a way to deal with properties one at a time?
The Challenge

- Generalized parity game is an **NP-Complete** problem and the current algorithm (Chatterjee et. al 07) is computationally very expensive.

- Is there a simpler solution to the complex problem?

- Is there a way to deal with properties one at a time?
The Challenge

- Generalized parity game is an **NP-Complete** problem and the current algorithm (Chatterjee et. al 07) is computationally very expensive.
- Is there a simpler solution to the complex problem?
- Is there a way to deal with properties one at a time?
A safety condition for a game graph $G = (Q, E)$ is a function $\pi : Q \rightarrow \{0, 1\}$ such that there is no transition $(u, v) \in E$ such that $\pi(u) = 0$ and $\pi(v) = 1$. 
A safety condition for a game graph $G = (Q, E)$ is a function $\pi : Q \rightarrow \{0, 1\}$ such that there is no transition $(u, v) \in E$ such that $\pi(u) = 0$ and $\pi(v) = 1$. 
A persistence condition for a game graph $G = (Q, E)$ is a function $\pi : Q \rightarrow \{1, 2\}$. 

Safety First: A Two-Stage Algorithm for LTL Games
The Claim

- What is so unique about persistence properties?
- The winning states for persistence properties can be categorized into persistent and transient states.
- The computation of strategies is not necessary when we are only interested in determining the persistent and transient states.
- A transient state will stay a transient state for the subsequent games.
The Claim

- What is so unique about persistence properties?
- The winning states for persistence properties can be categorized into **persistent** and **transient** states.
- A transient state will stay a transient state for the subsequent games.
What is so unique about persistence properties?

- The winning states for persistence properties can be categorized into persistent and transient states.
- The computation of strategies is not necessary when we are only interested in determining the persistent and transient states.
- A transient state will stay a transient state for the subsequent games.
The Claim

- What is so unique about **persistence properties**?
- The winning states for persistence properties can be categorized into **persistent** and **transient** states.
- A transient state will stay a transient state for the subsequent games.
Introduction
Games
Two Stage Synthesis
Results
Conclusions

The Challenge
Algorithm
Optimizations
Implementation
Caveats

Safety First: A Two-Stage Algorithm for LTL Games

Input/Output based game $\rightarrow$ State based game

No spurious grant $\mathcal{G}(-r_0 \rightarrow X(-a_0))$
Example: Simple Arbiter revisited

Safety First: A Two-Stage Algorithm for LTL Games
Example: Simple Arbiter revisited

\[ \neg \alpha_0 \lor \neg \alpha_1 \]

\[ \alpha_0 \land \alpha_1 \]
Example: Simple Arbiter revisited
Example: Simple Arbiter revisited

\[ \neg a_0 \lor \neg a_1 \]
Example: Simple Arbiter revisited

Safety First: A Two-Stage Algorithm for LTL Games
Example: Simple Arbiter revisited
Example: Simple Arbiter revisited

Safety First: A Two-Stage Algorithm for LTL Games
Example: Simple Arbiter revisited

Safety First: A Two-Stage Algorithm for LTL Games
Example: Simple Arbiter revisited

Safety First: A Two-Stage Algorithm for LTL Games
Example: Simple Arbiter revisited

Safety First: A Two-Stage Algorithm for LTL Games
Example: Simple Arbiter revisited
Example: Simple Arbiter revisited

Safety First: A Two-Stage Algorithm for LTL Games
Example: Simple Arbiter revisited
Example: Simple Arbiter revisited

Safety First: A Two-Stage Algorithm for LTL Games
Example: Simple Arbiter revisited

Safety First: A Two-Stage Algorithm for LTL Games
Example: Simple Arbiter revisited

Safety First: A Two-Stage Algorithm for LTL Games
Example: Simple Arbiter revisited

Safety First: A Two-Stage Algorithm for LTL Games
Example: Simple Arbiter revisited

Safety First: A Two-Stage Algorithm for LTL Games
The complexity of “classical” algorithm of (Chatterjee et al 07) is given by

\[ O(m \cdot n^{2d}) \cdot \binom{d}{d_1, d_2, \ldots, d_k}, \]

\[ d_i = \lceil k_i / 2 \rceil \]

If \( \pi_k \) is a safety condition, solving the game in two stages leads to a better bound for the second stage, \( O(m \cdot n^{2d-2}) \cdot \binom{d-1}{d_1, \ldots, d_{k-1}} \), while the first stage runs in \( O(m \cdot n^2) \).

In practice, in the second stage, the number of transitions may decrease, and the removal of losing positions for \( \pi_1 \) may reduce the number of colors in the remaining conditions.
How significant is the improvement?

- The complexity of “classical” algorithm of (Chatterjee et al 07) is given by

\[ O(m \cdot n^{2d}) \cdot \binom{d}{d_1, d_2, \ldots, d_k}, \]

\[ d_i = \lceil k_i / 2 \rceil \]

If \( \pi_k \) is a safety condition, solving the game in two stages leads to a better bound for the second stage, \( O(m \cdot n^{2d-2}) \cdot \binom{d-1}{d_1, \ldots, d_{k-1}} \), while the first stage runs in \( O(m \cdot n^2) \).

- In practice, in the second stage, the number of transitions may decrease, and the removal of losing positions for \( \pi_1 \) may reduce the number of colors in the remaining conditions.
3 Two Stage Synthesis
   - The Challenge
   - Algorithm
   - Optimizations
   - Implementation
   - Caveats
Methodology

- **Identify the safety/persistent properties in the specification.**
- Translate each property into a deterministic automaton.
- Compose the automaton with already existing game-graph and then playing the 2-player game on the relevant section of the graph.
- Determinize all the remaining non-safety/non-persistent properties and then compose with the game-graph and play the final generalized parity game on the relevant section of the graph.
- Select an appropriate strategy which in conjunction with the property automata can be translated into software/hardware.
Methodology

- Identify the safety/persistent properties in the specification.
- Translate each property into a deterministic automaton.
- Compose the automaton with already existing game-graph and then playing the 2-player game on the relevant section of the graph.
- Determinize all the remaining non-safety/non-persistent properties and then compose with the game-graph and play the final generalized parity game on the relevant section of the graph.
- Select an appropriate strategy which in conjunction with the property automata can be translated into software/hardware.
Methodology

- Identify the safety/persistent properties in the specification.
- Translate each property into a deterministic automaton.
- Compose the automaton with already existing game-graph and then playing the 2-player game on the relevant section of the graph.
- Determinize all the remaining non-safety/non-persistent properties and then compose with the game-graph and play the final generalized parity game on the relevant section of the graph.
- Select an appropriate strategy which in conjunction with the property automata can be translated into software/hardware.
Methodology

- Identify the safety/persistent properties in the specification.
- Translate each property into a deterministic automaton.
- Compose the automaton with already existing game-graph and then playing the 2-player game on the relevant section of the graph.
- Determine all the remaining non-safety/non-persistent properties and then compose with the game-graph and play the final generalized parity game on the relevant section of the graph.
- Select an appropriate strategy which in conjunction with the property automata can be translated into software/hardware.
Methodology

- Identify the safety/persistent properties in the specification.
- Translate each property into a deterministic automaton.
- Compose the automaton with already existing game-graph and then playing the 2-player game on the relevant section of the graph.
- Determinize all the remaining non-safety/non-persistent properties and then compose with the game-graph and play the final generalized parity game on the relevant section of the graph.
- Select an appropriate strategy which in conjunction with the property automata can be translated into software/hardware.
Algorithm

1. **SAFETY-FIRST**($G, \text{SPECIFICATION}$)
2. ($\text{SAFETY, NON-PERSISTENT} \leftarrow \text{SPECIFICATION}$)
3. foreach $\varphi \in \text{SAFETY}$
   3.1 $G = G \parallel \text{automaton}^{\varphi}_{\text{det}}$
   3.2 $(Q_{\text{sys}}, E_{\text{sys}}) \leftarrow \text{CHATTERJEE}(G, \varphi)$
   3.3 $(Q_{\text{new}}, E_{\text{new}}) \leftarrow \text{OPTIMIZE}(Q_{\text{sys}}, E_{\text{sys}})$
   3.4 $G = (Q_{\text{new}}, E_{\text{new}})$
end foreach
4. foreach $\varphi \in \text{NON-PERSISTENT}$
   4.1 $G = G \parallel \text{automaton}^{\varphi}_{\text{det}}$
end foreach
5. $(Q_{\text{sys}}, E_{\text{sys}}, \sigma_{\text{sys}}) \leftarrow \text{CHATTERJEE}(G, \varphi_1, \varphi_2, \ldots, \varphi_n)$
6. $\text{SYNTHESIZE}(Q_{\text{sys}}, E_{\text{sys}}, \sigma_{\text{sys}})$
Safety First: A Two-Stage Algorithm for LTL Games

1. **SAFETY-FIRST**(\(G, SPECIFICATION\))
2. \((SAFETY, NON − PERSISTENT) \leftarrow SPECIFICATION\)

3. foreach \(\varphi \in SAFETY\)
   3.1 \(G = G \parallel \text{automaton}_{\varphi_{det}}\)
   3.2 \((Q_{sys}, E_{sys}) \leftarrow \text{CHATTERJEE}(G, \varphi)\)
   3.3 \((Q_{new}, E_{new}) \leftarrow \text{OPTIMIZE}(Q_{sys}, E_{sys})\)
   3.4 \(G = (Q_{new}, E_{new})\)
end foreach

4. foreach \(\varphi \in NON − PERSISTENT\)
   4.1 \(G = G \parallel \text{automaton}_{\varphi_{det}}\)
end foreach

5. \((Q_{sys}, E_{sys}, \sigma_{sys}) \leftarrow \text{CHATTERJEE}(G, \varphi_1, \varphi_2 \ldots, \varphi_n)\)

6. **SYNTHESIZE**\((Q_{sys}, E_{sys}, \sigma_{sys})\)
Algorithm

1. \textsc{Safety-First}(G, \textsc{Specification})
2. (\textsc{Safety, Non-Persistent}) $\leftarrow$ \textsc{Specification}
3. \textbf{foreach} $\varphi \in \textsc{Safety}$
   \hspace{1em} $G = G \parallel \text{automaton}_{\text{det}}^\varphi$
   \hspace{1em} $(Q_{\text{sys}}, E_{\text{sys}}) \leftarrow \text{Chatterjee}(G, \varphi)$
   \hspace{1em} $(Q_{\text{new}}, E_{\text{new}}) \leftarrow \text{Optimize}(Q_{\text{sys}}, E_{\text{sys}})$
   \hspace{1em} $G = (Q_{\text{new}}, E_{\text{new}})$
\textbf{end foreach}
4. \textbf{foreach} $\varphi \in \text{Non-Persistent}$
   \hspace{1em} $G = G \parallel \text{automaton}_{\text{det}}^\varphi$
\textbf{end foreach}
5. $(Q_{\text{sys}}, E_{\text{sys}}, \sigma_{\text{sys}}) \leftarrow \text{Chatterjee}(G, \varphi_1, \varphi_2, \ldots, \varphi_n)$
6. \textsc{Synthesize}(Q_{\text{sys}}, E_{\text{sys}}, \sigma_{\text{sys}})
Algorithm

1. **SAFETY-FIRST**($G, \text{SPECIFICATION}$)
2. ($\text{SAFETY, NON-PERSISTENT}$) $\leftarrow$ SPECIFICATION
3. foreach $\varphi \in \text{SAFETY}$
   3.1 $G = G \parallel \text{automaton}^\varphi_{\text{det}}$
   3.2 $(Q_{\text{sys}}, E_{\text{sys}}) \leftarrow \text{CHATTERJEE}(G, \varphi)$
   3.3 $(Q_{\text{new}}, E_{\text{new}}) \leftarrow \text{OPTIMIZE}(Q_{\text{sys}}, E_{\text{sys}})$
   3.4 $G = (Q_{\text{new}}, E_{\text{new}})$
   end foreach
4. foreach $\varphi \in \text{NON-PERSISTENT}$
   4.1 $G = G \parallel \text{automaton}^\varphi_{\text{det}}$
   end foreach
5. $(Q_{\text{sys}}, E_{\text{sys}}, \sigma_{\text{sys}}) \leftarrow \text{CHATTERJEE}(G, \varphi_1, \varphi_2, \ldots, \varphi_n)$
6. $\text{SYNTHESIZE}(Q_{\text{sys}}, E_{\text{sys}}, \sigma_{\text{sys}})$
Algorithm

1. \textbf{SAFETY-FIRST}(G, \textit{SPECIFICATION})
2. \((\textit{SAFETY, NON} - \textit{PERSISTENT}) \leftarrow \textit{SPECIFICATION})
3. \textbf{foreach} \(\varphi \in \textit{SAFETY}\)
   - \(G = G \parallel \text{automanon}^{\varphi}_{\text{det}}\)
   - \((Q_{\text{sys}}, E_{\text{sys}}) \leftarrow \text{CHATTERJEE}(G, \varphi)\)
   - \((Q_{\text{new}}, E_{\text{new}}) \leftarrow \text{OPTIMIZE}(Q_{\text{sys}}, E_{\text{sys}})\)
   - \(G = (Q_{\text{new}}, E_{\text{new}})\)
   \textbf{end foreach}
4. \textbf{foreach} \(\varphi \in \textit{NON} - \textit{PERSISTENT}\)
   - \(G = G \parallel \text{automanon}^{\varphi}_{\text{det}}\)
   \textbf{end foreach}
5. \((Q_{\text{sys}}, E_{\text{sys}}, \sigma_{\text{sys}}) \leftarrow \text{CHATTERJEE}(G, \varphi_1, \varphi_2, ..., \varphi_n)\)
6. \textbf{SYNTHESIZE}(Q_{\text{sys}}, E_{\text{sys}}, \sigma_{\text{sys}})
Safety First: A Two-Stage Algorithm for LTL Games

1. \textbf{SAFETY-FIRST}(G, \textit{SPECIFICATION})
2. \((\textit{SAFETY}, \textit{NON − PERSISTENT}) \leftarrow \textit{SPECIFICATION})
3. \textbf{foreach} \(\varphi \in \textit{SAFETY}\)
   \hspace{1em} 3.1 \(G = G \parallel \text{automaton}^{\varphi}_{\text{det}}\)
   \hspace{1em} 3.2 \((Q_{\text{sys}}, E_{\text{sys}}) \leftarrow \text{CHATTERJEE}(G, \varphi)\)
   \hspace{1em} 3.3 \((Q_{\text{new}}, E_{\text{new}}) \leftarrow \text{OPTIMIZE}(Q_{\text{sys}}, E_{\text{sys}})\)
   \hspace{1em} 3.4 \(G = (Q_{\text{new}}, E_{\text{new}})\)
\textbf{end foreach}
4. \textbf{foreach} \(\varphi \in \textit{NON − PERSISTENT}\)
   \hspace{1em} 4.1 \(G = G \parallel \text{automaton}^{\varphi}_{\text{det}}\)
\textbf{end foreach}
5. \((Q_{\text{sys}}, E_{\text{sys}}, \sigma_{\text{sys}}) \leftarrow \text{CHATTERJEE}(G, \varphi_1, \varphi_2, \ldots, \varphi_n)\)
6. \textbf{SYNTHESIZE}(Q_{\text{sys}}, E_{\text{sys}}, \sigma_{\text{sys}})\)
1 Introduction

2 Games

3 Two Stage Synthesis
   - The Challenge
   - Algorithm
   - Optimizations
   - Implementation
   - Caveats

4 Results

5 Conclusions
Restrict the state space with the reachable winning states.

- Remove the constant bits in the reachable winning state space.
- Find dependencies between state-variables and remove the dependant variables.
- (Efficiently re-encode the state space).
■ Restrict the state space with the reachable winning states.

■ Remove the constant bits in the reachable winning state space.

■ Find dependencies between state-variables and remove the dependant variables.

■ (Efficiently re-encode the state space).
- Restrict the state space with the reachable winning states.
- Remove the constant bits in the reachable winning state space.
- Find dependencies between state-variables and remove the dependant variables.
  (Efficiently re-encode the state space).
- Restrict the state space with the reachable winning states.
- Remove the constant bits in the reachable winning state space.
- Find dependencies between state-variables and remove the dependant variables.
- (Efficiently re-encode the state space).
Outline

1 Introduction
2 Games
3 Two Stage Synthesis
   - The Challenge
   - Algorithm
   - Optimizations
   - Implementation
   - Caveats
4 Results
5 Conclusions
The LTL formula is determinized by the tool Wring using explicit state based translation. It is able to detect persistence properties and determinizes them using subset-construction otherwise uses Piterman’s determination procedure.

Chatterjee’s algorithm for generalized-parity games has been implemented in VIS which uses BDDs for internal representation and computation. The game-graph is represented as an input-based game but the algorithm virtually converts it into a turn-based game.
The LTL formula is determinized by the tool `Wring` using explicit state based translation. It is able to detect persistence properties and determinizes them using subset-construction otherwise uses Piterman’s determinization procedure.

Chatterjee’s algorithm for generalized-parity games has been implemented in `VIS` which uses BDDs for internal representation and computation. The game-graph is represented as an input-based game but the algorithm virtually converts it into a turn-based game.
Outline

1 Introduction

2 Games

3 Two Stage Synthesis
   - The Challenge
   - Algorithm
   - Optimizations
   - Implementation
   - Caveats

4 Results

5 Conclusions
Caveats

- The game-theoretic approach in synthesizing the safety properties introduces more state variables compared to a manual implementation where the programmer can take advantage by combining internal signals.

- Aggressive dependency removal of state-variables has a negative impact on performance as it affects the early quantification schedule, dependencies up to 3 state variables results in enhanced performance times.
Caveats

- The game-theoretic approach in synthesizing the safety properties introduces more state variables compared to a manual implementation where the programmer can take advantage by combining internal signals.

- Aggressive dependency removal of state-variables has a negative impact on performance as it affects the early quantification schedule, dependencies up to 3 state variables results in enhanced performance times.
Results

- Anzu (Bloem et al 07)

- Why Safety-First?
  - Full LTL.
  - No pre-synthesis
Results

- Anzu (Bloem et al 07)
- Why Safety-First?
  - Full LTL.
  - No pre-synthesis
- Anzu (Bloem et al 07)
- Why Safety-First?
  - Full LTL.
  - No pre-synthesis
Results

- Anzu (Bloem et al 07)
- Why Safety-First?
  - Full LTL.
  - No pre-synthesis
Results

Safety First: A Two-Stage Algorithm for LTL Games

Amba-bus and General-Buffer Examples

Safety-First (seconds)

Anzu (seconds)

10^{-1} 10^{0} 10^{1} 10^{2} 10^{3} 10^{4} 10^{5}
In practice large chunk of the Specification is of safety type.

Splitting the synthesis process in two stages has opened the door for optimizations which may not affect the worst-case complexity but are practically very significant.

Without loss of generality in the LTL specification, Safety-First is already competitive.

Incrementally compute a good BDD order.
Conclusions

- In practice large chunk of the Specification is of safety type.
- Splitting the synthesis process in two stages has opened the door for optimizations which may not affect the worst-case complexity but are practically very significant.
- Without loss of generality in the LTL specification, Safety-First is already competitive.
- Incrementally compute a good BDD order.
In practice large chunk of the Specification is of safety type.

Splitting the synthesis process in two stages has opened the door for optimizations which may not affect the worst-case complexity but are practically very significant.

Without loss of generality in the LTL specification, Safety-First is already competitive.

Incrementally compute a good BDD order.
In practice large chunk of the Specification is of safety type.

Splitting the synthesis process in two stages has opened the door for optimizations which may not affect the worst-case complexity but are practically very significant.

Without loss of generality in the LTL specification, Safety-First is already competitive.

Incrementally compute a good BDD order.
THANK YOU
Example: Simple Arbiter revisited

Safety First: A Two-Stage Algorithm for LTL Games
Example: Simple Arbiter revisited
Example: Simple Arbiter revisited

Safety First: A Two-Stage Algorithm for LTL Games
Example: Simple Arbiter revisited

No spurious grant

\[ G(\neg r_0 \rightarrow X(\neg a_0)) \]

Safety First: A Two-Stage Algorithm for LTL Games
Example: Simple Arbiter revisited

Safety First: A Two-Stage Algorithm for LTL Games
Example: Simple Arbiter revisited