Reactive Synthesis

Lecture 5

Swen Jacobs and Martin Zimmermann
(Saarland University)

Recap

To solve a safety game in symbolic representation, we can compute the (Player 1) attractor of the unsafe states.

The controllable predecessor of Player 1 becomes

$$\mathrm{CPre}_1(F) = \exists X_u \forall X_c \exists L'.\ F(L') \wedge T(L, X_u, X_c, L')$$

and the attractor is defined as (with $F = \mathit{Unsafe}$)

- $\mathrm{Attr}_1^0(F) = F$,
- $\mathrm{Attr}_1^{n+1}(F) = \mathrm{Attr}_1^n(F) \vee \mathrm{CPre}_1(\mathrm{Attr}_1^n(F))$, and
- $\mathrm{Attr}_1(F) = \bigvee_{n \in \mathbb{N}} \mathrm{Attr}_1^n(F)$.

ROBDDs support (efficiently) all operations needed for attractor computation

Winning Region and Winning Strategy

The fixpoint of the attractor computation is the **winning region of Player 1**, and (since safety games are determined) the negation of the attractor is the **winning region of Player 0**.

If the initial state is in the winning region of Player 0, we know that there exists a **strategy for Player 0** to win the game.

How to find a strategy? What is a good strategy?

Recap

For games with explicit state representation:

- A strategy for Player $i \in \{0, 1\}$ is a function $\sigma\colon V^* V_i \to V$ such that $\sigma(wv) = v'$ implies $(v, v') \in E$ for every w and every $v \in V_i$.
- σ is positional if $\sigma(wv) = \sigma(v)$ for all $w \in V^*$ and all $v \in V_i$.
- σ is a winning strategy for $\mathcal{G}$ from $v \in V$ if every play that is consistent with σ and starts in v is winning for Player i.
- the winning region of Player i in $\mathcal{G}$:
  $W_i(\mathcal{G}) = \{v \in V \mid \text{Player } i \text{ has winning strategy for } \mathcal{G} \text{ from } v\}$

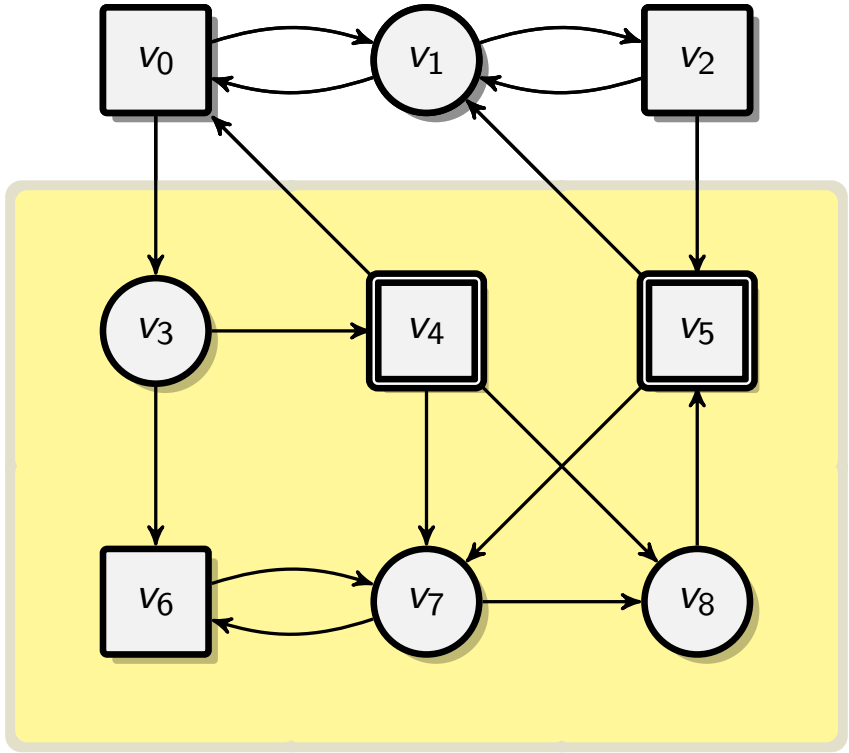
Lemma

Let $\mathcal{G} = (\mathcal{A}, \textsc{Reach}(R))$ be a reachability game. Then, $W_0(\mathcal{G}) = \mathrm{Attr}_0(R)$ and $W_1(\mathcal{G}) = V \setminus \mathrm{Attr}_0(R)$. In particular, $\mathcal{G}$ is determined (with positional strategies).

Proof.

On the blackboard. □

Recap



$$\mathrm{Attr}_0^4(\{v_4, v_5\}) = \{v_3, v_4, v_5, v_6, v_7, v_8\}$$

Plan for Today

- Basic Games
- Algorithms & Data Structures

- Advanced Games
- Temporal Logic Synthesis

Plan for Today

- Basic Games
- Algorithms & Data Structures
  - Strategy Extraction
- **Project Kickoff**
- Advanced Games
- Temporal Logic Synthesis

Extraction of Winning Strategies

Strategy Extraction

Based on winning region of Player 0, we can extract a non-deterministic winning strategy, i.e., a relation that determines all admissible values of X_c based on the current values of X_u and L.

Strategy Extraction

Based on winning region of Player 0, we can extract a non-deterministic winning strategy, i.e., a relation that determines all admissible values of X_c based on the current values of X_u and L.

Note: the next state follows from the current values of s, u, c and the transition relation T, i.e., this definition is equivalent to the original definition of a (positional) strategy.

Strategy Extraction

Based on winning region of Player 0, we can extract a non-deterministic winning strategy, i.e., a relation that determines all admissible values of X_c based on the current values of X_u and L.

Note: the next state follows from the current values of s, u, c and the transition relation T, i.e., this definition is equivalent to the original definition of a (positional) strategy.

Let $W_0(L)$ be the winning region for player 0.
Then a (non-deterministic) winning strategy *Strat* can be computed as:

Strategy Extraction

Based on winning region of Player 0, we can extract a non-deterministic winning strategy, i.e., a relation that determines all admissible values of X_c based on the current values of X_u and L.

Note: the next state follows from the current values of s, u, c and the transition relation T, i.e., this definition is equivalent to the original definition of a (positional) strategy.

Let $W_0(L)$ be the winning region for player 0.
Then a (non-deterministic) winning strategy *Strat* can be computed as:

$$Strat(L, X_u, X_c) = \exists L'.W_0(L) \wedge T(L, X_u, X_c, L') \wedge W_0(L')$$

I.e., *Strat* is a relation over tuples $(s, u, c) \in \mathbb{B}^L \times \mathbb{B}^{X_u} \times \mathbb{B}^{X_c}$.

Strategy Extraction

Based on winning region of Player 0, we can extract a non-deterministic winning strategy, i.e., a relation that determines all admissible values of X_c based on the current values of X_u and L.

Note: the next state follows from the current values of s, u, c and the transition relation T, i.e., this definition is equivalent to the original definition of a (positional) strategy.

Let $W_0(L)$ be the winning region for player 0.
Then a (non-deterministic) winning strategy *Strat* can be computed as:

$$Strat(L, X_u, X_c) = \exists L'.W_0(L) \wedge T(L, X_u, X_c, L') \wedge W_0(L')$$

I.e., *Strat* is a relation over tuples $(s, u, c) \in \mathbb{B}^L \times \mathbb{B}^{X_u} \times \mathbb{B}^{X_c}$.

Note: By construction, for every $(s, u) \in \mathbb{B}^L \times \mathbb{B}^{X_u}$ with $s \in W_0(L)$, *Strat* contains (s, u, c) for at least one $c \in \mathbb{B}^{X_c}$.

Functional Strategies

To implement *Strat* in a circuit, we need to make it deterministic, i.e., for every $(s, u) \in \mathbb{B}^L \times \mathbb{B}^{X_u}$ choose exactly one $c \in \mathbb{B}^{X_c}$.

Functional Strategies

To implement *Strat* in a circuit, we need to make it deterministic, i.e., for every $(s, u) \in \mathbb{B}^L \times \mathbb{B}^{X_u}$ choose exactly one $c \in \mathbb{B}^{X_c}$.

To this end, do the following:

1. for $x_c \in X_c$ do

$$\text{let } \mathit{Strat}_{x_c} = \exists X_c \setminus \{x_c\}.\ \mathit{Strat}$$

Functional Strategies

To implement *Strat* in a circuit, we need to make it deterministic, i.e., for every $(s, u) \in \mathbb{B}^L \times \mathbb{B}^{X_u}$ choose exactly one $c \in \mathbb{B}^{X_c}$.

To this end, do the following:

1. for $x_c \in X_c$ do
   1.1 // nondet. strategy for one controllable input:
   let $Strat_{x_c} = \exists X_c \setminus \{x_c\}.\ Strat$

Functional Strategies

To implement *Strat* in a circuit, we need to make it deterministic, i.e., for every $(s, u) \in \mathbb{B}^L \times \mathbb{B}^{X_u}$ choose exactly one $c \in \mathbb{B}^{X_c}$.

To this end, do the following:

1. for $x_c \in X_c$ do
   1.1 // nondet. strategy for one controllable input:
   let $Strat_{x_c} = \exists X_c \setminus \{x_c\}.\ Strat$

   let $f_{x_c} = Strat_{x_c}[x_c \leftarrow 1]$

Functional Strategies

To implement *Strat* in a circuit, we need to make it deterministic, i.e., for every $(s, u) \in \mathbb{B}^L \times \mathbb{B}^{X_u}$ choose exactly one $c \in \mathbb{B}^{X_c}$.

To this end, do the following:

1. for $x_c \in X_c$ do
   - 1.1 // nondet. strategy for one controllable input:
     let $Strat_{x_c} = \exists X_c \setminus \{x_c\}.\ Strat$
   - 1.2 // return 1 iff 1 is admissible:
     let $f_{x_c} = Strat_{x_c}[x_c \leftarrow 1]$

Functional Strategies

To implement *Strat* in a circuit, we need to make it deterministic, i.e., for every $(s, u) \in \mathbb{B}^L \times \mathbb{B}^{X_u}$ choose exactly one $c \in \mathbb{B}^{X_c}$.

To this end, do the following:

1. for $x_c \in X_c$ do
   - 1.1 // nondet. strategy for one controllable input:
     let $Strat_{x_c} = \exists X_c \setminus \{x_c\}.\ Strat$
   - 1.2 // return 1 iff 1 is admissible:
     let $f_{x_c} = Strat_{x_c}[x_c \leftarrow 1]$
   - 1.3 let $f[x_c] = f_{x_c}$

Functional Strategies

To implement *Strat* in a circuit, we need to make it deterministic, i.e., for every $(s, u) \in \mathbb{B}^L \times \mathbb{B}^{X_u}$ choose exactly one $c \in \mathbb{B}^{X_c}$.

To this end, do the following:

1. for $x_c \in X_c$ do

1.1 // nondet. strategy for one controllable input:
let $Strat_{x_c} = \exists X_c \setminus \{x_c\}.\ Strat$

1.2 // return 1 iff 1 is admissible:
let $f_{x_c} = Strat_{x_c}[x_c \leftarrow 1]$

1.3 let $f[x_c] = f_{x_c}$

let $Strat = Strat[x_c \leftarrow f_{x_c}]$

Functional Strategies

To implement *Strat* in a circuit, we need to make it deterministic, i.e., for every $(s, u) \in \mathbb{B}^L \times \mathbb{B}^{X_u}$ choose exactly one $c \in \mathbb{B}^{X_c}$.

To this end, do the following:

1. for $x_c \in X_c$ do

1.1 // nondet. strategy for one controllable input:
let $Strat_{x_c} = \exists X_c \setminus \{x_c\}.\ Strat$

1.2 // return 1 iff 1 is admissible:
let $f_{x_c} = Strat_{x_c}[x_c \leftarrow 1]$

1.3 let $f[x_c] = f_{x_c}$

1.4 // fix choice for x_c in following iterations
let $Strat = Strat[x_c \leftarrow f_{x_c}]$

Functional Strategies

To implement *Strat* in a circuit, we need to make it deterministic, i.e., for every $(s, u) \in \mathbb{B}^L \times \mathbb{B}^{X_u}$ choose exactly one $c \in \mathbb{B}^{X_c}$.

To this end, do the following:

1. for $x_c \in X_c$ do
   - 1.1 // nondet. strategy for one controllable input:
     let $Strat_{x_c} = \exists X_c \setminus \{x_c\}.\ Strat$
   - 1.2 // return 1 iff 1 is admissible:
     let $f_{x_c} = Strat_{x_c}[x_c \leftarrow 1]$
   - 1.3 let $f[x_c] = f_{x_c}$
   - 1.4 // fix choice for x_c in following iterations
     let $Strat = Strat[x_c \leftarrow f_{x_c}]$
2. return f

Obtaining smaller Solutions: The Careset

The functional strategies obtained so far make a choice how to resolve non-determinism: whenever a 1 is possible, they return a 1. But there are many other ways to resolve non-determinism.

Obtaining smaller Solutions: The Careset

The functional strategies obtained so far make a choice how to resolve non-determinism: whenever a 1 is possible, they return a 1. But there are many other ways to resolve non-determinism.

Each of these ways must respect the cases where **only** 1 or **only** 0 is possible.

In boolean functions where some output values are fixed and others may be arbitrary, the set of inputs for which there is a fixed output value is called the **careset**.

Obtaining smaller Solutions: The Careset

The functional strategies obtained so far make a choice how to resolve non-determinism: whenever a 1 is possible, they return a 1. But there are many other ways to resolve non-determinism.

Each of these ways must respect the cases where **only** 1 or **only** 0 is possible.

In boolean functions where some output values are fixed and others may be arbitrary, the set of inputs for which there is a fixed output value is called the **careset**.

Goal: obtain a function that behaves just like the original functional strategy on the careset (and arbitrarily otherwise), and can be represented by a smaller ROBDD.

Obtaining smaller Solutions: Restrict

Input: boolean function f, careset C
Output: $f \downarrow C$ with $f \downarrow C(\vec{x}) = f(\vec{x})$ if $C(\vec{x})$

Obtaining smaller Solutions: Restrict

Input: boolean function f, careset C
Output: $f \downarrow C$ with $f \downarrow C(\vec{x}) = f(\vec{x})$ if $C(\vec{x})$

Restrict(f,C)

1. if $C = 0$ then return error
2. if $C = 1$ or $f = 0$ or $f = 1$ then return f;

Obtaining smaller Solutions: Restrict

Input: boolean function f, careset C
Output: $f \downarrow C$ with $f \downarrow C(\vec{x}) = f(\vec{x})$ if $C(\vec{x})$

Restrict(f,C)

1. if $C = 0$ then return error
2. if $C = 1$ or $f = 0$ or $f = 1$ then return f;
3. let $x = var(root(C))$
4. if $C[x \leftarrow 0] = 0$ then return $Restrict(f[x \leftarrow 1], C[x \leftarrow 1])$

Obtaining smaller Solutions: Restrict

Input: boolean function f, careset C
Output: $f \downarrow C$ with $f \downarrow C(\vec{x}) = f(\vec{x})$ if $C(\vec{x})$

Restrict(f,C)

1. if $C = 0$ then return error
2. if $C = 1$ or $f = 0$ or $f = 1$ then return f;
3. let $x = var(root(C))$
4. if $C[x \leftarrow 0] = 0$ then return $Restrict(f[x \leftarrow 1], C[x \leftarrow 1])$
5. if $C[x \leftarrow 1] = 0$ then return $Restrict(f[x \leftarrow 0], C[x \leftarrow 0])$

Obtaining smaller Solutions: Restrict

Input: boolean function f, careset C
Output: $f \downarrow C$ with $f \downarrow C(\vec{x}) = f(\vec{x})$ if $C(\vec{x})$

Restrict(f,C)

1. if $C = 0$ then return error
2. if $C = 1$ or $f = 0$ or $f = 1$ then return f;
3. let $x = var(root(C))$
4. if $C[x \leftarrow 0] = 0$ then return $Restrict(f[x \leftarrow 1], C[x \leftarrow 1])$
5. if $C[x \leftarrow 1] = 0$ then return $Restrict(f[x \leftarrow 0], C[x \leftarrow 0])$
6. if $f[x \leftarrow 0] = f[x \leftarrow 1]$ then return
   $Restrict(f, C[x \leftarrow 0] \vee C[x \leftarrow 1])$

Obtaining smaller Solutions: Restrict

Input: boolean function f, careset C
Output: $f \downarrow C$ with $f \downarrow C(\vec{x}) = f(\vec{x})$ if $C(\vec{x})$

Restrict(f,C)

1. if $C = 0$ then return error
2. if $C = 1$ or $f = 0$ or $f = 1$ then return f;
3. let $x = var(root(C))$
4. if $C[x \leftarrow 0] = 0$ then return $Restrict(f[x \leftarrow 1], C[x \leftarrow 1])$
5. if $C[x \leftarrow 1] = 0$ then return $Restrict(f[x \leftarrow 0], C[x \leftarrow 0])$
6. if $f[x \leftarrow 0] = f[x \leftarrow 1]$ then return
   $Restrict(f, C[x \leftarrow 0] \vee C[x \leftarrow 1])$
7. return $(\bar{x} \wedge Restrict(f[x \leftarrow 0], C[x \leftarrow 0]))$
   $\vee (x \wedge Restrict(f[x \leftarrow 1], C[x \leftarrow 1]))$

Properties of Restrict

Theorem (Coudert,Berthet,Madre 1989)

For any boolean functions f and $C \neq 0$, if $C(\vec{x}) = 1$ then $f \downarrow C(\vec{x}) = f(\vec{x})$.

Properties of Restrict

Theorem (Coudert,Berthet,Madre 1989)

For any boolean functions f and $C \neq 0$, if $C(\vec{x}) = 1$ then $f \downarrow C(\vec{x}) = f(\vec{x})$.

Theorem (Coudert,Berthet,Madre 1989)

For any boolean functions f and $C \neq 0$, the ROBDD for $f \downarrow C$ is of smaller or equal size to the ROBDD of f.

Properties of Restrict

Theorem (Coudert,Berthet,Madre 1989)

For any boolean functions f and $C \neq 0$, if $C(\vec{x}) = 1$ then $f \downarrow C(\vec{x}) = f(\vec{x})$.

Theorem (Coudert,Berthet,Madre 1989)

For any boolean functions f and $C \neq 0$, the ROBDD for $f \downarrow C$ is of smaller or equal size to the ROBDD of f.

Example

$f = (x_2 \wedge (x_1 \leftrightarrow (x_3 \rightarrow x_4))) \vee \neg(x_2 \vee ((x_4 \rightarrow x_1) \wedge (x_1 \vee x_3)))$
restricted under $C = (\bar{x_2} \wedge x_3 \wedge x_4) \vee (x_2 \wedge (x_3 \leftrightarrow x_4))$ is
$f \downarrow C = x_1 \wedge x_2$.
The ROBDD for $f \downarrow C$ is much smaller than that of f.

Optimization for Functional Strategies

To obtain smaller solutions:

1. for $x_c \in X_c$ do
   - 1.1 // nondet. strategy for one controllable input:
     let $Strat_{x_c} = \exists X_c \setminus \{x_c\}.\ Strat$
   - 1.8 let $f[x_c] = f_{x_c}$
   - 1.9 // fix choice for x_c in following iterations
     let $Strat = Strat[x_c \leftarrow f_{x_c}]$
2. return f

Optimization for Functional Strategies

To obtain smaller solutions:

1. for $x_c \in X_c$ do
   - 1.1 // nondet. strategy for one controllable input:
     let $Strat_{x_c} = \exists X_c \setminus \{x_c\}.\ Strat$
   - 1.2 let $maybe\_true = Strat_{x_c}[x_c \leftarrow 1]$ //1 is admissible
   - 1.8 let $f[x_c] = f_{x_c}$
   - 1.9 // fix choice for x_c in following iterations
     let $Strat = Strat[x_c \leftarrow f_{x_c}]$
2. return f

Optimization for Functional Strategies

To obtain smaller solutions:

1. for $x_c \in X_c$ do
   - 1.1 // nondet. strategy for one controllable input:
     let $Strat_{x_c} = \exists X_c \setminus \{x_c\}.\ Strat$
   - 1.2 let $maybe\_true = Strat_{x_c}[x_c \leftarrow 1]$ //1 is admissible
   - 1.3 let $maybe\_false = Strat_{x_c}[x_c \leftarrow 0]$ // 0 is admissible
   - 1.8 let $f[x_c] = f_{x_c}$
   - 1.9 // fix choice for x_c in following iterations
     let $Strat = Strat[x_c \leftarrow f_{x_c}]$
2. return f

Optimization for Functional Strategies

To obtain smaller solutions:

1. for $x_c \in X_c$ do
   - 1.1 // nondet. strategy for one controllable input:
     let $Strat_{x_c} = \exists X_c \setminus \{x_c\}.\ Strat$
   - 1.2 let $maybe\_true = Strat_{x_c}[x_c \leftarrow 1]$ //1 is admissible
   - 1.3 let $maybe\_false = \overline{Strat_{x_c}[x_c \leftarrow 0]}$ // 0 is admissible
   - 1.4 let $mustbe\_true = \overline{maybe\_false} \wedge maybe\_true$
   - 1.8 let $f[x_c] = f_{x_c}$
   - 1.9 // fix choice for x_c in following iterations
     let $Strat = Strat[x_c \leftarrow f_{x_c}]$
2. return f

Optimization for Functional Strategies

To obtain smaller solutions:

1. for $x_c \in X_c$ do

1.1 // nondet. strategy for one controllable input:
let $Strat_{x_c} = \exists X_c \setminus \{x_c\}.\ Strat$

1.2 let $maybe\_true = Strat_{x_c}[x_c \leftarrow 1]$ //1 is admissible

1.3 let $maybe\_false = \overline{Strat_{x_c}[x_c \leftarrow 0]}$ // 0 is admissible

1.4 let $mustbe\_true = \overline{maybe\_false} \wedge maybe\_true$

1.5 let $mustbe\_false = \overline{maybe\_true} \wedge maybe\_false$

1.8 let $f[x_c] = f_{x_c}$

1.9 // fix choice for x_c in following iterations
let $Strat = Strat[x_c \leftarrow f_{x_c}]$

2. return f

Optimization for Functional Strategies

To obtain smaller solutions:

1. for $x_c \in X_c$ do
   - 1.1 // nondet. strategy for one controllable input:
     let $Strat_{x_c} = \exists X_c \setminus \{x_c\}.\ Strat$
   - 1.2 let $maybe\_true = Strat_{x_c}[x_c \leftarrow 1]$ //1 is admissible
   - 1.3 let $maybe\_false = \overline{Strat_{x_c}[x_c \leftarrow 0]}$ // 0 is admissible
   - 1.4 let $mustbe\_true = \overline{maybe\_false} \wedge maybe\_true$
   - 1.5 let $mustbe\_false = \overline{maybe\_true} \wedge maybe\_false$
   - 1.6 let $care\_set = mustbe\_true \vee mustbe\_false$
   - 1.8 let $f[x_c] = f_{x_c}$
   - 1.9 // fix choice for x_c in following iterations
     let $Strat = Strat[x_c \leftarrow f_{x_c}]$
2. return f

Optimization for Functional Strategies

To obtain smaller solutions:

1. for $x_c \in X_c$ do
   - 1.1 // nondet. strategy for one controllable input:
     let $Strat_{x_c} = \exists X_c \setminus \{x_c\}.\ Strat$
   - 1.2 let $maybe\_true = Strat_{x_c}[x_c \leftarrow 1]$ //1 is admissible
   - 1.3 let $maybe\_false = \overline{Strat_{x_c}[x_c \leftarrow 0]}$ // 0 is admissible
   - 1.4 let $mustbe\_true = \overline{maybe\_false} \wedge maybe\_true$
   - 1.5 let $mustbe\_false = \overline{maybe\_true} \wedge maybe\_false$
   - 1.6 let $care\_set = mustbe\_true \vee mustbe\_false$
   - 1.7 let $f_{x_c} = Restrict(maybe\_true, care\_set)$
   - 1.8 let $f[x_c] = f_{x_c}$
   - 1.9 // fix choice for x_c in following iterations
     let $Strat = Strat[x_c \leftarrow f_{x_c}]$
2. return f

Project Kickoff

Project Overview

Phase I:

We provide: a basic implementation (Python, CUDD)

You will:

- implement new and improve existing procedures (based on lectures)
- provide an engineering notebook as documentation

Project Overview

Phase I:

We provide: a basic implementation (Python, CUDD)

You will:

- implement new and improve existing procedures (based on lectures)
- provide an engineering notebook as documentation

Phase II:

We provide:

- a benchmarking framework
- additional suggestions for optimizations

You will:

- optimize for solving time and solution circuit size (based on suggestions and own ideas)
- choose optimizations for competition

Project: Important Dates

Check-point: December 12, 2017

- implement mandatory procedures and optimizations
- submit preliminary version and documentation

Project: Important Dates

Check-point: December 12, 2017

- implement mandatory procedures and optimizations
- submit preliminary version and documentation

Final submission: January 14, 2018

- implement additional optimizations
- choose optimizations for competition
- submit final version and documentation
  (for grading and competition)

Project: Important Dates

Check-point: December 12, 2017

- implement mandatory procedures and optimizations
- submit preliminary version and documentation

Final submission: January 14, 2018

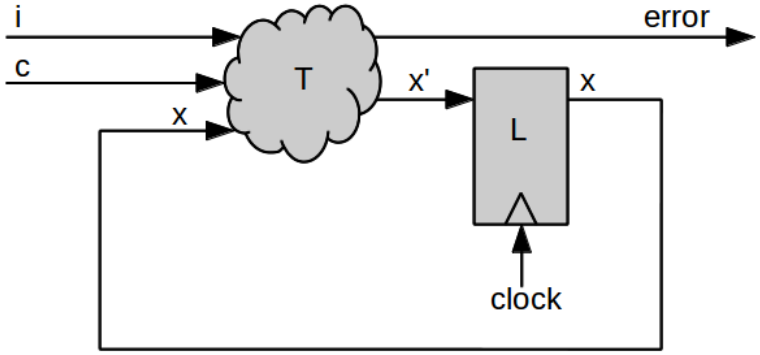
- implement additional optimizations
- choose optimizations for competition
- submit final version and documentation
  (for grading and competition)

Competition results: January 30, 2018

- all tools run on a large set of challenging benchmarks
- we compare number of benchmarks solved and solution sizes
- competition results do not enter grading
  (but winners will get a prize)

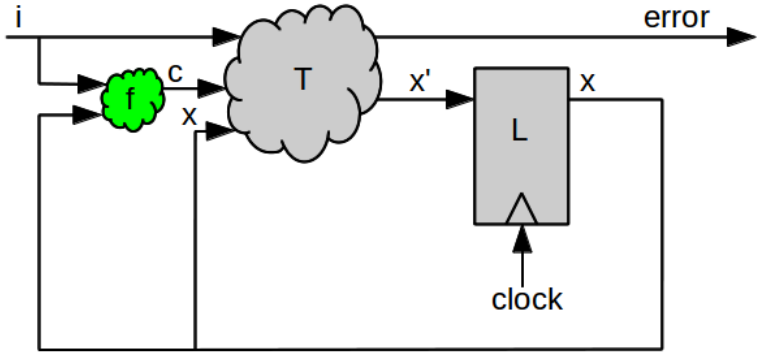
Our Synthesis Problem

Input:



AIGER format

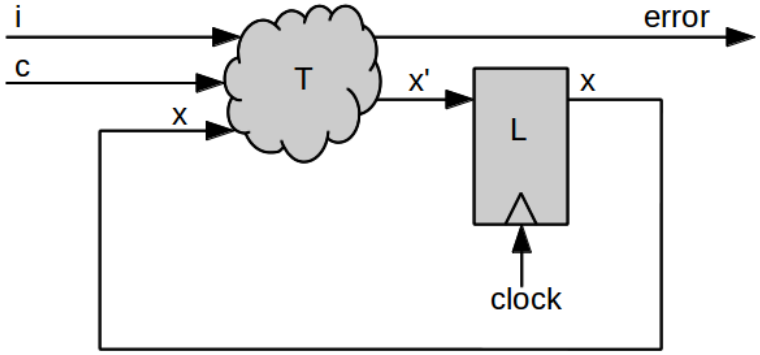
Output:



AIGER format

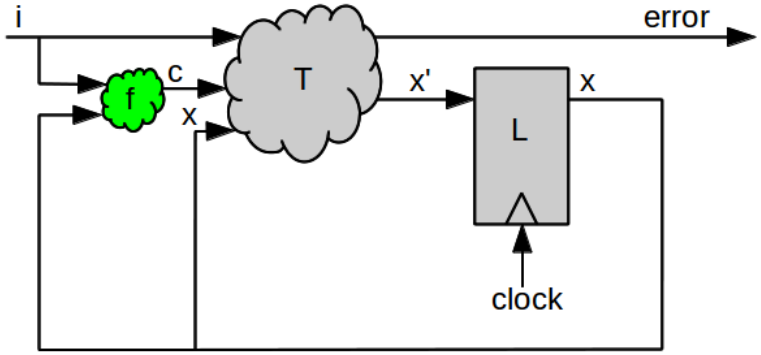
Our Synthesis Problem

Input:

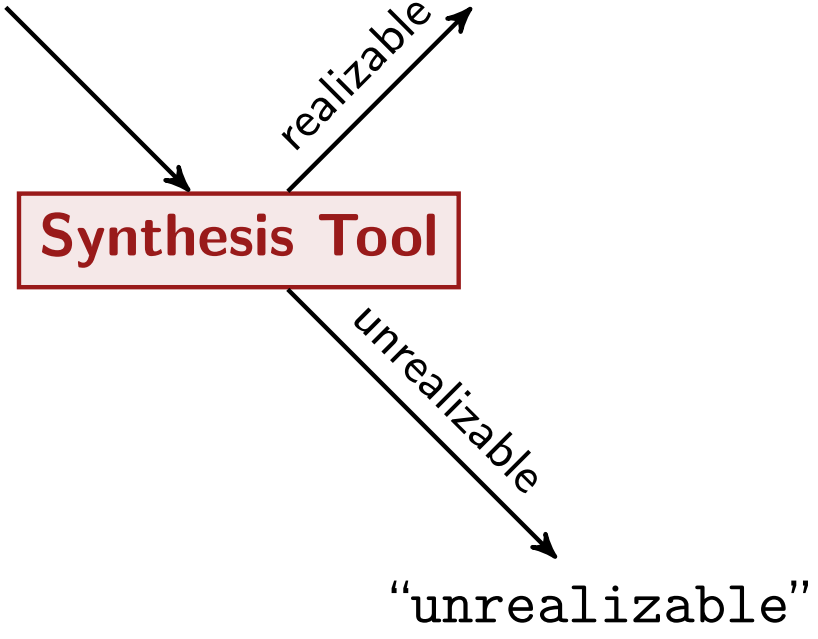


AIGER format

Output:



AIGER format



AIGER Format

```
aag 13 4 2 1 7
2
4
6
8
10 2
12 4
27
14 7 10
16 6 11
18 15 17
20 9 12
22 8 13
24 21 23
26 18 24
i0 i_r1
i1 i_r2
i2 controllable_g1
i3 controllable_g2
l0 l_r1
l1 l_r2
c
G((r1 <-> X(g1)) & (r2 <-> X(g2)))
realizable
```

header: M I L O A

inputs

latches

single output (should stay 0)

AND gates

comments that also specify which inputs are controllable

Correctness of a Synthesized Circuit

A synthesized circuit is a **solution** for an input circuit if:

1. it adheres to syntactical restrictions for solutions:
   - values of controllable inputs are defined in terms of other inputs and latches
   - first line of file is changed accordingly
   - other lines of the input circuit description remain untouched
2. the synthesized circuit is safe, i.e., no error state is reachable.

We provide tools to check syntactical correctness and safety of a solution.

Basic Implementation

Implemented in Python, tested on Ubuntu

High-level Algorithm:

- `parse_into_spec()`
- `synthesize()`
  - `compose_init_state_bdd()`
  - `compose_transition_bdd()`
  - `not_error_bdd()`
  - `calc_win_region()`
  - `get_nondet_strategy()`
  - `extract_output_functions()`
- `model_to_aiger()`
  - traverse solution BDD recursively
  - redefine inputs in aiger file

Useful Tools and references

- `aigtodot` and others in aiger-1.9.4: `http://fmv.jku.at/aiger/` (also supplied in correctness/aiger directory of basic implementation)
- AIGER format description: `http://fmv.jku.at/aiger/FORMAT` (also supplied in correctness/aiger directory of basic implementation)
- AIGER format modification for synthesis tasks: `https://arxiv.org/abs/1405.5793`

Project Phase I: Tasks

1. **form group of 2 students** (if not done yet)
2. **install and test** existing implementation
3. **understand** existing implementation
   (attractor computation "flipped")
4. **implement** the following procedures from lecture
   (make selectable with input arguments):
   - standard attractor computation
   - substitution-based attractor computation
   - optimized strategy extraction
   - enable dynamic reordering
5. **evaluate** performance differences (time, solution size)
6. **keep track** of everything in engineering notebook

Engineering Notebook

Record your work on the program:

- Which part of the program did you work on?
- What was the goal?
- What was the result?

Record project meetings, including discussion items and action items that result from the meeting.

Record related work that influenced you, including books, research papers, webpages, source code, etc.
(be inspired, but do not copy code!)

Obtain a timestamp: Upload notebook to rcms at the end of the workday, **at least once a week**.

Format: simple textfile or pdf; **only add, never delete**.

Organization

Next 3 weeks:

- tutorials are optional and only for discussion of project
- no lectures, office hours during lecture time

Important dates:

December 12: project check-point

December 13: next lecture

December 20: lecture

from January 3, 2018: regular lecture and tutorial schedule

January 14, 2018: final tool submission

January 30, 2018 (tutorial slot): tool competition results

January 31, 2018 (lecture slot): final exam

Obtaining the Basic Implementation

The basic implementation can be downloaded from the "Information/Materials" category in our rCMS at

`https://courses.react.uni-saarland.de/rs1718/`

Please read carefully the installation instructions (in `readme/INSTALL` files).

The setup was tested on Ubuntu, we recommend using a linux OS (natively or in a virtual machine).

For questions, we have also set up a forum in the rCMS. Feel free to ask questions. We will try to answer, but you can also help out each other (or just discuss).