Session 1: Games, Automata and Synthesis

Session 1: Games, Automata and Synthesis

Introduction

Traditional program verification

```
int cut_square(int i)
    return i <= 0 ? 0 : i*i;
```

Introduction

Traditional program verification

```
int cut_square(int i)
    return i <= 0 ? 0 : i*i;
```

Requirements (What needs proofing)

- The output is always `>= 0`
- The output is always larger or equal to the absolute value of the input

Introduction

Traditional program verification

```
int cut_square(int i)
    return i <= 0 ? 0 : i*i;
```

Requirements (What needs proofing)

- The output is always `>= 0`
- The output is always larger or equal to the absolute value of the input

Approach (How to prove)

- Establish invariant
- Use some form of induction
- May rely on external solvers like SAT or SMT

Needs

A clear first input then output relation

Introduction

Model checking

```
int next_move(state* cstate)
    // ...
    return nm;
```

```
void next_state(state* cstate,
                state* nstate, int nm)
    // ...
```

Works on *infinite* behavior, reasons about continuous systems

Model checking

```
int next_move(state* cstate)
    // ...
    return nm;
```

```
void next_state(state* cstate,
                state* nstate, int nm)
    // ...
```

Works on *infinite* behavior, reasons about continuous systems

Requirements (What needs proofing)

- Some logical specification The elevator will at some point open its door at every floor to which it is called.

Introduction

Model checking

```
int next_move(state* cstate)
    // ...
    return nm;
```

```
void next_state(state* cstate,
                state* nstate, int nm)
    // ...
```

Works on *infinite* behavior, reasons about continuous systems

Requirements (What needs proofing)

- Some logical specification The elevator will at some point open its door at every floor to which it is called.

Approach (How to prove)

- Translate (the negation of) the specification and the model to some finite automaton
- Check that there exists no run violating the specification

Introduction

Model checking

```
int next_move(state* cstate)
    // ...
    return nm;
```

```
void next_state(state* cstate,
                state* nstate, int nm)
    // ...
```

Works on *infinite* behavior, reasons about continuous systems

Requirements (What needs proofing)

- Some logical specification The elevator will at some point open its door at every floor to which it is called.

Approach (How to prove)

- Translate (the negation of) the specification and the model to some finite automaton
- Check that there exists no run violating the specification

Needs

A continuously executing system

Model checking: A closer look

Correctness

The model may not exhibit a single trace violating the specification

Model checking: A closer look

Correctness

The model may not exhibit a single trace violating the specification

The elevator model

- *Specification*: How the elevator should behave governed by the
- *Controller*: Choosing the next action as a function of
- *State*: the current position of the elevator and the
- *Environment*: behavior, that is which floors are demanded

Model checking: A closer look

Correctness

The model may not exhibit a single trace violating the specification

The elevator model

- *Specification*: How the elevator should behave governed by the
- *Controller*: Choosing the next action as a function of
- *State*: the current position of the elevator and the
- *Environment*: behavior, that is which floors are demanded

The run is a succession of actions of the environment (choices over the environment APs) and the controller (choices over the controlled APs).

Model checking: A closer look

Correctness

The model may not exhibit a single trace violating the specification

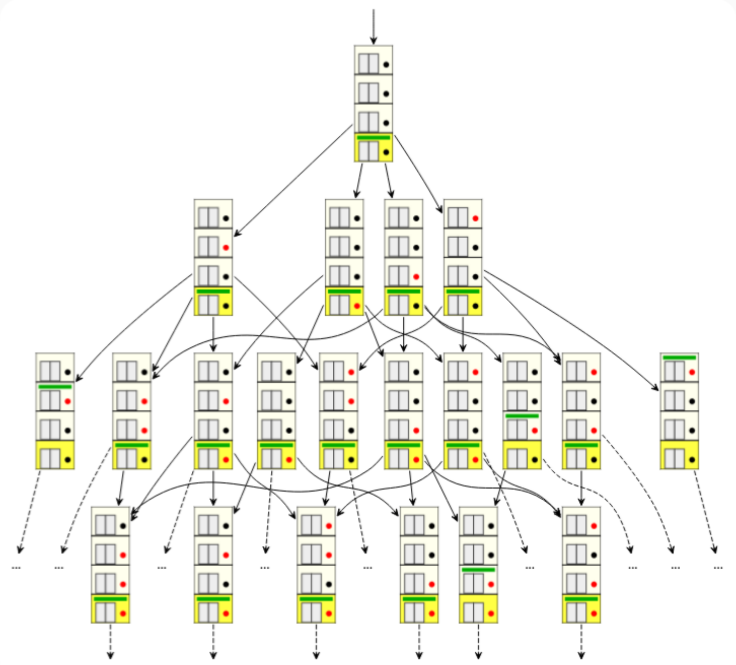
The elevator model

- *Specification*: How the elevator should behave governed by the
- *Controller*: Choosing the next action as a function of
- *State*: the current position of the elevator and the
- *Environment*: behavior, that is which floors are demanded

The run is a succession of actions of the environment (choices over the environment APs) and the controller (choices over the controlled APs).

The model is **correct** if the controller can guarantee the specification for **all** valid moves of the environment.

Model checking: A closer look



Introducing: Synthesis

Synthesis is so to speak the flip-side of verification:

Instead of verifying that a model/controller verifies a specification, why not directly generate it such a manner?

Introducing: Synthesis

Synthesis is so to speak the flip-side of verification:

Instead of verifying that a model/controller verifies a specification, why not directly generate it such a manner?

Intuitively, the idea is to make the *choices* explicit and create a two-player game between the environment (*env*, player 0) and the controller (*player*, player 1).

Introducing: Synthesis

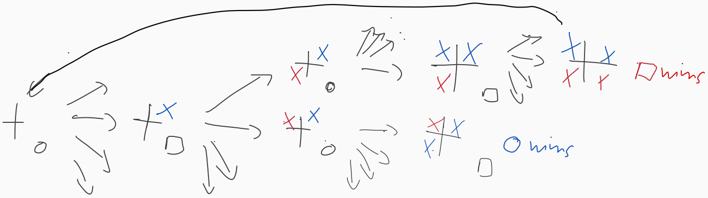
Synthesis is so to speak the flip-side of verification:

Instead of verifying that a model/controller verifies a specification, why not directly generate it such a manner?

Intuitively, the idea is to make the *choices* explicit and create a two-player game between the environment (*env*, player 0) and the controller (*player*, player 1).

If the controller can always *win* the game, he has an answer or strategy for all possible environment behaviors. This strategy then by construction verifies the specification.

Simplified 2 wins: Env wins if it can occupy a diagonal at any point



Definition 1

We are only concerned with 2 player games with perfect information.

The game is played between player 0 (the *Env*) and player 1 (the *Player*). To ease notations, we also define player i ($i \in [0, 1]$) and his opponent player $i - 1$.

Definition 1

We are only concerned with 2 player games with perfect information.

The game is played between player 0 (the *Env*) and player 1 (the *Player*). To ease notations, we also define player i ($i \in [0, 1]$) and his opponent player $i - 1$.

Arena

An arena $\mathcal{A} = (V, V_0, V_1, E)$ - a finite set of vertices V - the set of vertices V_i owned by player i partitioning V - $E \subseteq V \times V$ the set of directed edges - for every vertex v the set $\{v' | (v, v') \in E\}$ is non-empty

Definition 1

We are only concerned with 2 player games with perfect information.

The game is played between player 0 (the *Env*) and player 1 (the *Player*). To ease notations, we also define player i ($i \in [0, 1]$) and his opponent player $i - 1$.

Arena

An arena $\mathcal{A} = (V, V_0, V_1, E)$ - a finite set of vertices V - the set of vertices V_i owned by player i partitioning V - $E \subseteq V \times V$ the set of directed edges - for every vertex v the set $\{v' | (v, v') \in E\}$ is non-empty

We say arena is **alternating** if for every edge $(v, v') \in E$ we have $v \in V_i$ and $v' \in V_{i-1}$

Definition 2

Sub-Arena

Let $\mathcal{A}$ be an arena and $V' \subseteq V$ a subset of vertices.
The sub-arena of $\mathcal{A}$ induced by V' called $\mathcal{A}_{V'}$ is defined as
$\mathcal{A}_{V'} = (V', V_0 \cap V', V_1 \cap V', E \cap (V' \times V'))$

Definition 2

Sub-Arena

Let $\mathcal{A}$ be an arena and $V' \subseteq V$ a subset of vertices.
The sub-arena of $\mathcal{A}$ induced by V' called $\mathcal{A}_{V'}$ is defined as
$\mathcal{A}_{V'} = (V', V_0 \cap V', V_1 \cap V', E \cap (V' \times V'))$

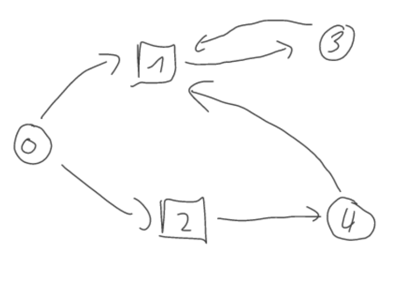
Note that not all sets V' induce a sub-arena: for instance the rule that a successor needs to exist may be violated.

Definition 2

Sub-Arena

Let $\mathcal{A}$ be an arena and $V' \subseteq V$ a subset of vertices.
The sub-arena of $\mathcal{A}$ induced by V' called $\mathcal{A}_{V'}$ is defined as
$\mathcal{A}_{V'} = (V', V_0 \cap V', V_1 \cap V', E \cap (V' \times V'))$

Note that not all sets V' induce a sub-arena: for instance the rule that a successor needs to exist may be violated.



- $V' = \{1, 3\}$ induces a sub-arena
- $V' = \{2, 4\}$ does not

Definition 3

Intuitively, during the play the players push a token along the edges of the arena, whoever own the vertex decides on the next edge to use. During this process, we record the vertices (and sometimes) edges seen.

A Play

A play in $\mathcal{A}$ is an infinite sequence $\rho = \rho_0 \rho_1 \rho_2 \rho_3 \cdots \in V^\omega$ such that $\rho_n \rho_{n+1} \in E$ holds for all $n \in \mathbb{N}$.

Definition 3

Intuitively, during the play the players push a token along the edges of the arena, whoever own the vertex decides on the next edge to use. During this process, we record the vertices (and sometimes) edges seen.

A Play

A play in $\mathcal{A}$ is an infinite sequence $\rho = \rho_0\rho_1\rho_2\rho_3\cdots \in V^\omega$ such that $\rho_n\rho_{n+1} \in E$ holds for all $n \in \mathbb{N}$.

A Strategy

A strategy for player i ($i \in \{0, 1\}$) in an arena $\mathcal{A}$ is a function $\delta_i : V^*V_i \to V$ s.t. $\delta_i(wv) = v'$ implies $(v, v') \in E$ for every w and v.

Definition 3

Intuitively, during the play the players push a token along the edges of the arena, whoever own the vertex decides on the next edge to use. During this process, we record the vertices (and sometimes) edges seen.

A Play

A play in $\mathcal{A}$ is an infinite sequence $\rho = \rho_0\rho_1\rho_2\rho_3\cdots \in V^\omega$ such that $\rho_n\rho_{n+1} \in E$ holds for all $n \in \mathbb{N}$.

A Strategy

A strategy for player i ($i \in \{0, 1\}$) in an arena $\mathcal{A}$ is a function $\delta_i : V^*V_i \to V$ s.t. $\delta_i(wv) = v'$ implies $(v, v') \in E$ for every w and v.

In other words, a strategy decides what to do next given the history of the game and its choice is possible in $\mathcal{A}$.

Definition 4

A consistent play

A play in $\mathcal{A}$ is **consistent** with a strategy δ_i for player i if $\rho_{n+i} = \delta_i(\rho_0, \dots, \rho_n)$ for every $n \in \mathbb{N}$.

Definition 4

A consistent play

A play in $\mathcal{A}$ is **consistent** with a strategy δ_i for player i if $\rho_{n+i} = \delta_i(\rho_0, \ldots, \rho_n)$ for every $n \in \mathbb{N}$.

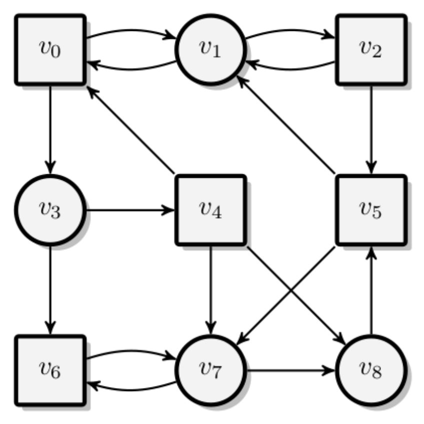
And

Positional strategy

A strategy δ_i is called positional (or memoryless) if $\delta_i(wv) = \delta_i(v)$ for all $w \in V^*$ and $v \in V$.

Since positional strategies are function from V to V (instead of $(V^* \times V)$ to V) we also denote them as such.

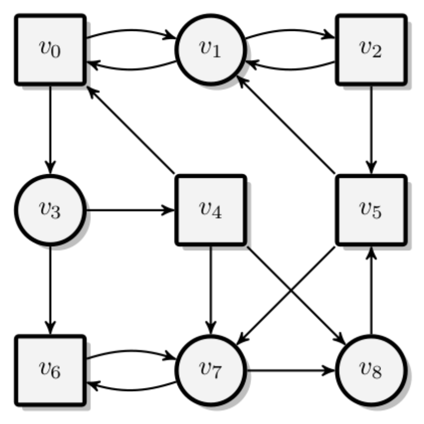
Definition 3&4 illustrations



player 0: circles
player 1: squares

- $v_0 v_3 (v_6 v_7)^\omega$ is a play
- *go right* is a positional strategy for player 0
- $v_0 (v_1 v_2 v_1 v_2 v_5 v_1)^\omega$ is a play consistent with the positional strategy *go right* for player 0

Definition 3&4 illustrations



player 0: circles
player 1: squares

- $v_0 v_3 (v_6 v_7)^\omega$ is a play
- *go right* is a positional strategy for player 0
- $v_0 (v_1 v_2 v_1 v_2 v_5 v_1)^\omega$ is a play consistent with the positional strategy *go right* for player 0
- Give the positional strategy δ_1 consistent with $v_0 (v_1 v_2 v_1 v_2 v_5 v_1)^\omega$
- Given we start in v_0, is there a strategy for player 1 to reach v_6?

Definition 5 (and final) [at least for now]

We can finally properly define a game

Game

A game $\mathcal{G}$ is defined by the tuple $(\mathcal{A}, Win)$ with $\mathcal{A}$ being the arena and $Win \subseteq V^\omega$ being the winning sequences.
We say the game is won by player 1 if and only if $\rho \in Win$. Otherwise it is won by player 0.

Definition 5 (and final) [at least for now]

We can finally properly define a game

Game

A game $\mathcal{G}$ is defined by the tuple $(\mathcal{A}, Win)$ with $\mathcal{A}$ being the arena and $Win \subseteq V^\omega$ being the winning sequences.
We say the game is won by player 1 if and only if $\rho \in Win$. Otherwise it is won by player 0.

Winning Strategy

A strategy δ_i for player i is called a **winning strategy** for vertex v if all plays starting v and consistent with δ_i are won by player i.

Definition 5 (and final) [at least for now]

We can finally properly define a game

Game

A game $\mathcal{G}$ is defined by the tuple $(\mathcal{A}, Win)$ with $\mathcal{A}$ being the arena and $Win \subseteq V^\omega$ being the winning sequences.
We say the game is won by player 1 if and only if $\rho \in Win$. Otherwise it is won by player 0.

Winning Strategy

A strategy δ_i for player i is called a **winning strategy** for vertex v if all plays starting v and consistent with δ_i are won by player i.

A Winning Region

The winning region $W_i(\mathcal{G})$ of player i is the set of vertices from which player i has a winning strategy.

On winning regions

Lemma: Winning regions do not intersect

or $W_0(\mathcal{G}) \cap W_1(G) = \emptyset$

On winning regions

Lemma: Winning regions do not intersect

or $W_0(\mathcal{G}) \cap W_1(G) = \emptyset$

Proof. Let $\mathcal{G} = (\mathcal{A}, Win)$. Towards a contradiction, assume there exists a vertex $v \in W_0(\mathcal{G}) \cap W_1(\mathcal{G})$.

Then, both players have a winning strategy from v, call them δ_0 and δ_1. Let $\rho = \rho(v, \delta_0, \delta_1)$, i.e., we let the players both use their winning strategy against each other, the play is consistent with both strategies.

Then, $\rho \in Win$, as δ_1 is a winning strategy for player 1 and $\rho \notin Win$, as δ_0 is a winning strategy for player 0.

Hence, we have derived the desired contradiction.

On winning regions and determinacy

Last slide we have shown that winning regions are always disjoint.

On winning regions and determinacy

Last slide we have shown that winning regions are always disjoint.

The other question is whether they also partition the vertices, that is whether $W_0(\mathcal{G}) \cup W_1(\mathcal{G}) = V$.

On winning regions and determinacy

Last slide we have shown that winning regions are always disjoint.

The other question is whether they also partition the vertices, that is whether $W_0(\mathcal{G}) \cup W_1(\mathcal{G}) = V$.

[Positional] Determinancy

Let $\mathcal{G}$ be a game with vertex set V.
We say that $\mathcal{G}$ is determined if $W_0($\mathcal{G}$) \cupW_1($\mathcal{G}$) = V. Furthermore, we say that $\mathcal{G}$ is positionally determined if, from every vertex $v \in V$ one of the players has a positional winning strategy.

On winning regions and determinacy

Last slide we have shown that winning regions are always disjoint.

The other question is whether they also partition the vertices, that is whether $W_0(\mathcal{G}) \cup W_1(\mathcal{G}) = V$.

[Positional] Determinancy

Let $\mathcal{G}$ be a game with vertex set V.
We say that $\mathcal{G}$ is determined if $W_0(\mathcal{G}) ∪W_1(\mathcal{G}) = V. Furthermore, we say that $\mathcal{G}$ is positionally determined if, from every vertex $v \in V$ one of the players has a positional winning strategy.

Note the subtlety “one of the playes has **a** positional winning strategy [for each vertex v]”.

This indicates that the positional strategy may depend on the initial vertex.

On winning regions and determinacy

Last slide we have shown that winning regions are always disjoint.

The other question is whether they also partition the vertices, that is whether $W_0(\mathcal{G}) \cup W_1(\mathcal{G}) = V$.

[Positional] Determinancy

Let $\mathcal{G}$ be a game with vertex set V.
We say that $\mathcal{G}$ is determined if \$W_0($\mathcal{G}$) ∪W_1($\mathcal{G}$) = V. Furthermore, we say that $\mathcal{G}$ is positionally determined if, from every vertex $v \in V$ one of the players has a positional winning strategy.

Note the subtlety “one of the playes has **a** positional winning strategy [for each vertex v]”.

This indicates that the positional strategy may depend on the initial vertex.

If a positional strategy does **not** depend on the initial vertex, we call it a **uniform** positional winning strategy.

It is a Trap

Intuitively, a trap is a set of vertices from which the trapped player may not escape without the help of the other.

Let $\mathcal{A} = (V, V_0, V_1, E)$ be an arena and let $T \subseteq V$.
Then, T is a trap for player i, if - every vertex $v \in T \cap V_i$ of player i in T has only successors in T, i.e., $(v, v') \in E$ implies $v' \in T$, and - every vertex $v \in T \cap V_{1-i}$ of player $1 - i$ in T there is a successor in T , i.e., there is some $v' \in T$ with $(v, v') \in E$.

Trap, take #2

Sub-Arenas and nested traps

Let T be a trap for $\mathcal{A}$, then

- The sub-arena A_T induced by T is a valid sub-arena
- The sub-arena $A_{V \setminus T}$ is a valid sub-arena
- If T' is a trap for player i in A_T, then it is also trap for player i in $\mathcal{A}$

. . .

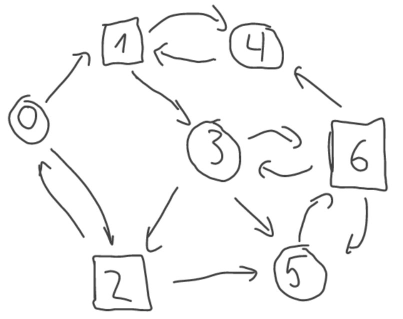
Sketch of Proof

Lemmas and exercises

- Let $\mathcal{G} = (\mathcal{A}, Win)$ be a game with prefix-independent winning condition Win. Then, $W_i(\mathcal{G})$ is a trap for player $1 - i$.

Lemmas and exercises

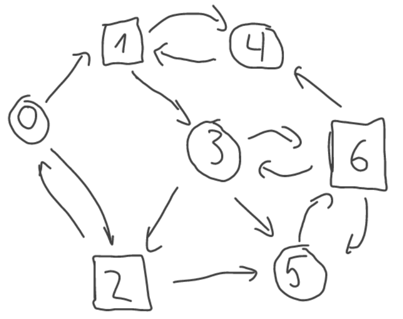
- Let $\mathcal{G} = (\mathcal{A}, Win)$ be a game with prefix-independent winning condition Win. Then, $W_i(\mathcal{G})$ is a trap for player $1 - i$.
- Consider the game $\mathcal{G} = (\mathcal{A}, Win)$ with the arena $\mathcal{A}$ depicted below and the winning condition $Win = \{\rho \in V^\omega | Occ(\rho) = V\}$



- Give a trap for player 1
- Give a winning strategy and initial state for some player
- Is the strategy found (uniformly) positional

Lemmas and exercises

- Let $\mathcal{G} = (\mathcal{A}, Win)$ be a game with prefix-independent winning condition Win. Then, $W_i(\mathcal{G})$ is a trap for player $1 - i$.
- Consider the game $\mathcal{G} = (\mathcal{A}, Win)$ with the arena $\mathcal{A}$ depicted below and the winning condition $Win = \{\rho \in V^\omega | Occ(\rho) = V\}$



- Give a trap for player 1
- Give a winning strategy and initial state for some player
- Is the strategy found (uniformly) positional

- Prove or disprove: If player i has a positional winning strategy from each vertex $v \in W_i(\mathcal{G})$ for some game $\mathcal{G}$, then player i has a uniform positional winning strategy for $\mathcal{G}$.

Reachability Games

Reachability Game

Reachability Game

Let $\mathcal{A}= (V, V_0 , V_1 , E) be an arena and let $R \subseteq V$ be a subset of $\mathcal{A}$'s vertices.

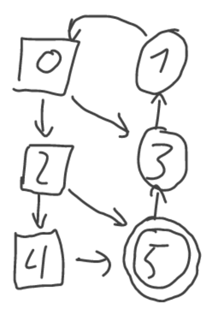
Then, the reachability condition $Reach(R)$ is defined as $Reach(R) := \{\rho \in V^{\omega} | Occ(\rho) \cap R \neq \emptyset\}$.

We call a game $\mathcal{G} = (\mathcal{A}, Reach(R))$ a reachability game with reachability set R.

Reachability Game

Reachability Game

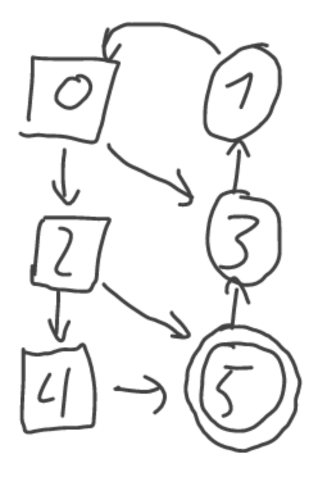
Let $\mathcal{A}$= (V, V_0 , V_1 , E) be an arena and let $R \subseteq V$ be a subset of $\mathcal{A}$'s vertices.
Then, the reachability condition $Reach(R)$ is defined as $Reach(R) := \{\rho \in V^\omega | Occ(\rho) \cap R \neq \emptyset\}$.
We call a game $\mathcal{G} = (\mathcal{A}, Reach(R))$ a reachability game with reachability set R.



Example of a reachability game with $R = \{5\}$

- Are there traps for player 0
- What is the winning region for player 1
- Is the strategy of player 1 uniform? . . .
- Is this *really* an infinite game?

Solving the game by hand



Predecessors and attractors 1

As we have argued, the winning region of player 1 W_1 corresponds to the set of all states which we can attract to / force to pass by a state in R.

Predecessors and attractors 1

As we have argued, the winning region of player 1 W_1 corresponds to the set of all states which we can attract to / force to pass by a state in R.

Adding states one step at a time

Controlled Predecessor

Let $\mathcal{A}$ be some arena and $S \subseteq V$ an arbitrary set of states.
We call $CPre_i(S)$ the set of controlled predecessors for player i and set S defined as

$$\begin{aligned} CPre_i(S) = &\{v \in V_i | v' \in S \text{ for some successor } v' \text{ of } v\} \cup \\ &\{v \in V_{1-i} | v' \in S \text{ for all successors } v' \text{ of } v\}. \end{aligned}$$

Predecessors and attractors 1

As we have argued, the winning region of player 1 W_1 corresponds to the set of all states which we can attract to / force to pass by a state in R.

Adding states one step at a time

Controlled Predecessor

Let $\mathcal{A}$ be some arena and $S \subseteq V$ an arbitrary set of states.
We call $CPre_i(S)$ the set of controlled predecessors for player i and set S defined as

$$\begin{aligned} CPre_i(S) = &\{v \in V_i | v' \in S \text{ for some successor } v' \text{ of } v\} \cup \\ &\{v \in V_{1-i} | v' \in S \text{ for all successors } v' \text{ of } v\}. \end{aligned}$$

These are all states which player i can **force** to be in S in the next step.

Predecessors and attractors 2

We can now iteratively call $CPre$ to compute the attractor via a fixed point computation

Predecessors and attractors 2

We can now iteratively call $CPre$ to compute the attractor via a fixed point computation

- $Attr_i^0(S) = S$ initialization
- $Attr_i^{n+1}(S) = Attr_i^n(S) \cup CPre_i(Attr_i^n(S))$ iteration
- \$Attr_i(S) = $Attr_i^n(S)$ termination

Predecessors and attractors 2

We can now iteratively call $CPre$ to compute the attractor via a fixed point computation

- $Attr_i^0(S) = S$ initialization
- $Attr_i^{n+1}(S) = Attr_i^n(S) \cup CPre_i(Attr_i^n(S))$ iteration
- $Attr_i(S) = $Attr_i^n(S)$ termination

Note that $Attr_i^{n-1}(S)$ is always a subset of $Attr_i^n(S)$

A note on termination for attractor computation

The different $Attr_i^n(S)$ form a monotonic subset relation.

That is in each iteration $Attr_i^{n+1}(S)$ is at least equal to $Attr_i^n(S)$.

This implies that after at most $|V|$ step the computation becomes stationary ensuring termination.

A note on termination for attractor computation

The different $Attr_i^n(S)$ form a monotonic subset relation.

That is in each iteration $Attr_i^{n+1}(S)$ is at least equal to $Attr_i^n(S)$.

This implies that after at most $|V|$ step the computation becomes stationary ensuring termination.

A more refined approach:

Let $\mathcal{A}$ be an arena and $0 \leq m \leq |V|$ some index.

Then we can write

$$
\begin{aligned}
S = Attr_i^0(S) \subseteq Attr_i^1(S) \subseteq Attr_i^2(S) \subseteq \ldots \subseteq Attr_i^m(S) \\
= Attr_i^{m+1}(S) = \ldots = Attr_i^{|V|}(S) = Attr_i(S)
\end{aligned}
$$

A note on termination for attractor computation

The different $Attr_i^n(S)$ form a monotonic subset relation.

That is in each iteration $Attr_i^{n+1}(S)$ is at least equal to $Attr_i^n(S)$.

This implies that after at most $|V|$ step the computation becomes stationary ensuring termination.

A more refined approach:

Let $\mathcal{A}$ be an arena and $0 \leq m \leq |V|$ some index.

Then we can write

$$
\begin{aligned}
S = Attr_i^0(S) \subseteq Attr_i^1(S) \subseteq Attr_i^2(S) \subseteq \ldots \subseteq Attr_i^m(S) \\
= Attr_i^{m+1}(S) = \ldots = Attr_i^{|V|}(S) = Attr_i(S)
\end{aligned}
$$

Giving us a valid criterion for *early* termination.

From Attractors to Reachability Games

The attractor provides the winning regions

Let $\mathcal{G} = (\mathcal{A}, Reach(R))$ with $\mathcal{A} = (V, V_0, V_1, E)$.
Then, $W_1(\mathcal{G}) = Attr_1(R)$ and $W_0(\mathcal{G}) = V \setminus Attr_1(R)$

From Attractors to Reachability Games

The attractor provides the winning regions

Let $\mathcal{G} = (\mathcal{A}, Reach(R))$ with $\mathcal{A} = (V, V_0, V_1, E)$.
Then, $W_1(\mathcal{G}) = Attr_1(R)$ and $W_0(\mathcal{G}) = V \setminus Attr_1(R)$

- $Attr_1(R)$ is precisely the set of states which player 1 can force to reach R, fulfilling the winning condition.
- $V \setminus Attr_1(R)$ is a trap for player 1 not containing any R states at all.

From Attractors to Winning Strategies

The attractor computation provides a positional winning strategy

Let $\mathcal{G} = (\mathcal{A}, Reach(R))$ with $\mathcal{A} = (V, V_0, V_1, E)$.
The sequence of attractors $Attr_1^n(R)$ can be transformed into a (positional) winning strategy.

From Attractors to Winning Strategies

The attractor computation provides a positional winning strategy

Let $\mathcal{G} = (\mathcal{A}, Reach(R))$ with $\mathcal{A} = (V, V_0, V_1, E)$.
The sequence of attractors $Attr_1^n(R)$ can be transformed into a (positional) winning strategy.

Sketch of Proof

Define the metric $\mu(v) = min\{n \in \mathbb{N} | v \in Attr_1^n(R)\}$

Intuitively this corresponds to the *distance* between the current vertex v and the *closest* vertex in R.

Now we can define a strategy:

Whenever v belongs to player i and is in $Attr_1(R)$ two cases arise:

- We have $v \in R$, in which case the objective is fulfilled
- Otherwise, by construction, exists a successor v' such that $\mu(v') < \mu(v)$.

Büchi Games

Büchi Games

Büchi Games

Let $\mathcal{A} = (V, V_0, V_1, E)$ be an arena and let $F \subseteq V$ be a subset of $\mathcal{A}$'s vertices.
Then, the Büchi condition $Buchi(F)$ is defined as
$Buchi(F) = \{\rho \in V^\omega | Inf(\rho) \cap F \neq \emptyset\}$.
We call a game $\mathcal{G} = (\mathcal{A}, Buchi(F))$ a Büchi game with recurrence set F.

Büchi Games

Büchi Games

Let $\mathcal{A} = (V, V_0, V_1, E)$ be an arena and let $F \subseteq V$ be a subset of $\mathcal{A}$'s vertices.
Then, the Büchi condition $Buchi(F)$ is defined as
$Buchi(F) = \{\rho \in V^\omega | Inf(\rho) \cap F \neq \emptyset\}$.
We call a game $\mathcal{G} = (\mathcal{A}, Buchi(F))$ a Büchi game with recurrence set F.

Recall that this corresponds to a liveness condition:

Something *good* (visiting a good state in F) is guaranteed to eventually happen.

Reachability vs Büchi

Note that this is very different from the reachability condition.

For reachability it is not possible to *unaccept* a word, so for some $w \in V^*$ with $Occ(w) \cap R \neq \emptyset$ then for all $u \in V^\omega$ we have $wu \in Reach(R)$.

Reachability vs Büchi

Note that this is very different from the reachability condition.

For reachability it is not possible to *unaccept* a word, so for some $w \in V^*$ with $Occ(w) \cap R \neq \emptyset$ then for all $u \in V^\omega$ we have $wu \in Reach(R)$.

However for Büchi we actually need to know the "entire infinite word" in order to accept or reject.

So we can never decide acceptance after seeing some finite prefix. Accepted words need to form a lasso: A finite prefix u and finite cycle w which must contain states from F giving $u.w^\omega$.

Solving Büchi Games

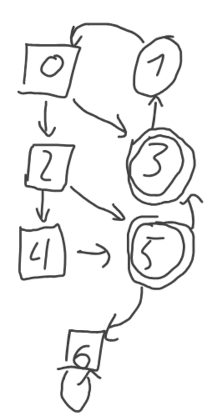
Solving Büchi Games is basically a nested to problem:

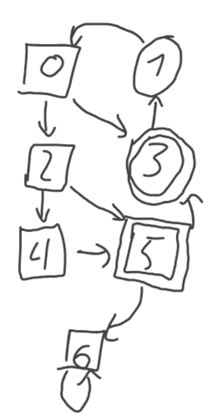
- We need to be able to reach a state of the recurrence set (in a finite number of steps)
  - This can be done via attractor computation
- We need to make sure we can repeat this 🤔

Solving Büchi Games

Solving Büchi Games is basically a nested to problem:

- We need to be able to reach a state of the recurrence set (in a finite number of steps)
  - This can be done via attractor computation
- We need to make sure we can repeat this 🤔





Recurrence Construction

Solving the Reccurence Game

We use again an iterative approach

- $F^0 = F$ initialization
- $W_0^n = V \setminus Attr_1(F^n)$ iteration with nested computation
- $F^{n+1} = F \setminus CPre_0(W_0^n)$ update

Recurrence Construction

Solving the Reccurence Game

We use again an iterative approach

- $F^0 = F$ initialization
- $W_0^n = V \setminus Attr_1(F^n)$ iteration with nested computation
- $F^{n+1} = F \setminus CPre_0(W_0^n)$ update

What about termination

We have two monotonic sequences

$F = F^0 \supseteq F^1 \supseteq \dots \supseteq F^m = F^{m+1}$

$W_0^0 \subseteq W_0^1 \subseteq \dots \subseteq W_0^m = W_0^{m+1}$

Recurrence Construction

Solving the Reccurence Game

We use again an iterative approach

- $F^0 = F$ initialization
- $W_0^n = V \setminus Attr_1(F^n)$ iteration with nested computation
- $F^{n+1} = F \setminus CPre_0(W_0^n)$ update

What about termination

We have two monotonic sequences

$F = F^0 \supseteq F^1 \supseteq \ldots \supseteq F^m = F^{m+1}$

$W_0^0 \subseteq W_0^1 \subseteq \ldots \subseteq W_0^m = W_0^{m+1}$

And what about

- The winning region of player i
- The complexity of the algorithm?
- How to extract a winning strategy?
- What type of strategy is it?

Synthesis in a nutshell

From game solving to synthesis

We have treated some of the classics for games on graphs. However this was fairly disconnected from our original synthesis problem!

From game solving to synthesis

We have treated some of the classics for games on graphs. However this was fairly disconnected from our original synthesis problem!

Let us fix that!

Textbook automata theoretic approach

Recall that we have a specification, given as LTL formula. This formula is built over a set of atomic propositions AP.

These propositions are partitioned into two sets:

- The set of controllable AP (controller output), O
- The set of uncontrollable AP (environment input) I

Textbook automata theoretic approach

Recall that we have a specification, given as LTL formula. This formula is built over a set of atomic propositions AP.

These propositions are partitioned into two sets:

- The set of controllable AP (controller output), O
- The set of uncontrollable AP (environment input) I

The formula specifies what output we expect given an input sequence.

Textbook automata theoretic approach

Recall that we have a specification, given as LTL formula. This formula is built over a set of atomic propositions AP.

These propositions are partitioned into two sets:

- The set of controllable AP (controller output), O
- The set of uncontrollable AP (environment input) I

The formula specifies what output we expect given an input sequence.

Our controller can be seen as a transducer from I^* to O.

This is also called a Mealy Machine and this is our goal.

Textbook automata theoretic approach

The basic approach is

- Translate the formula into a deterministic (parity) automaton
- Split all transitions in two, first over I, then over O
- Solve the associated game
- Prune the automaton with the winning strategy to obtain Mealy Machine

Textbook automata theoretic approach - Demo

Time for the notebook

That’s all for today

thanks and see you in practice