Reactive Synthesis

Lecture 01

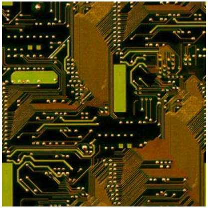
Swen Jacobs and Martin Zimmermann
(Saarland University)

Reactive Systems

Systems that **react** to environment inputs in potentially **infinite** executions

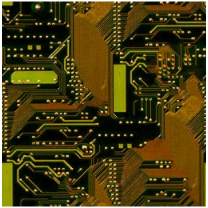
Reactive Systems

Systems that **react** to environment inputs in potentially **infinite** executions



Reactive Systems

Systems that **react** to environment inputs in potentially **infinite** executions



Properties of reactive systems:

- infinite (or unbounded) executions
- reacting to antagonistic environment
- usually discrete time steps and finite input range
- often used as controllers of physical processes

The Need for Provable Correctness

Reactive Systems are often **safety-critical**:

- controllers in planes, trains, cars
- power plants, electric grids

The Need for Provable Correctness

Reactive Systems are often **safety-critical**:

- controllers in planes, trains, cars
- power plants, electric grids

Under assumption of discrete time and finite input range, many properties can automatically be proved for a given reactive system. This is called **model checking**.

The Need for Provable Correctness

Reactive Systems are often **safety-critical**:

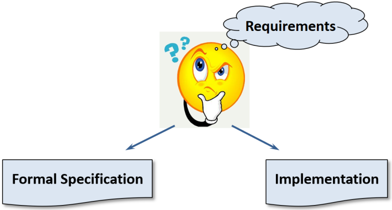
- controllers in planes, trains, cars
- power plants, electric grids

Under assumption of discrete time and finite input range, many properties can automatically be proved for a given reactive system. This is called **model checking**.

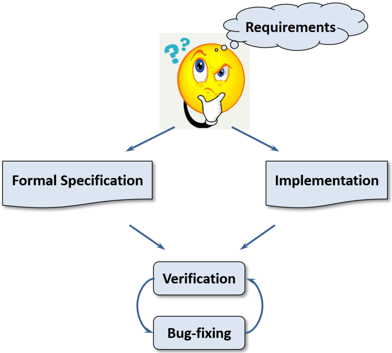
Model checking decides **whether a given system is correct**, but leaves significant effort for the human designer: develop the system and fix bugs until it is correct.

Verification versus Synthesis

Verification Workflow:

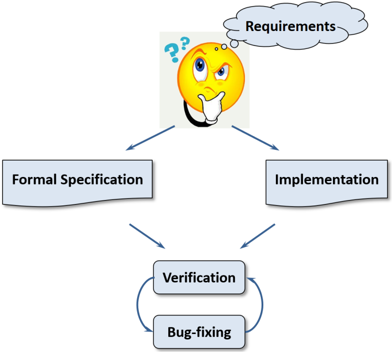


Verification Workflow:

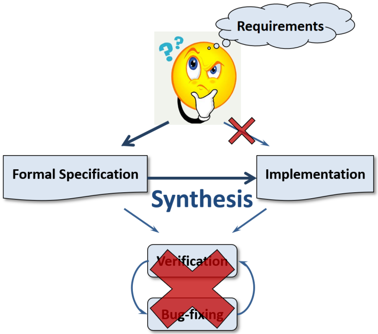


Verification versus Synthesis

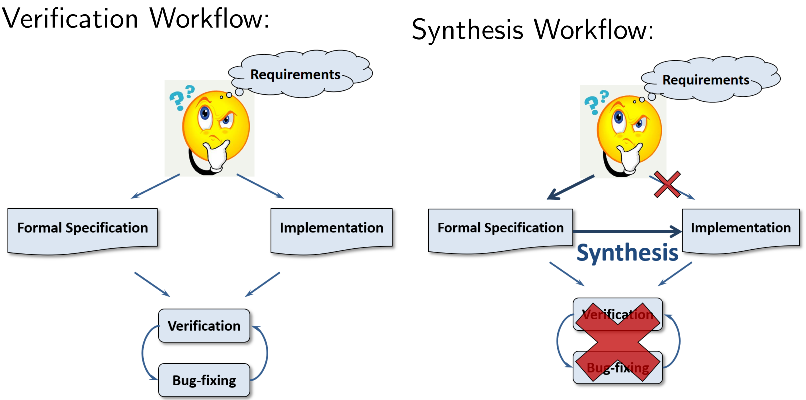
Verification Workflow:



Synthesis Workflow:



Verification versus Synthesis



The synthesis problem is also known as **Church's problem**, since Alonso Church first defined it in 1962.

Example: A Specification

Consider a very simple system with a **single bit of input** and a **single bit of output**.

Our specification is the conjunction of the following three requirements:

1. Whenever the input bit is 1, then the output bit is 1, too.
2. At least one out of every three consecutive output bits is a 1.
3. If there are infinitely many 0's in the input stream, then there are infinitely many 0's in the output stream.

Example: A Specification

Consider a very simple system with a **single bit of input** and a **single bit of output**.

Our specification is the conjunction of the following three requirements:

1. Whenever the input bit is 1, then the output bit is 1, too.
2. At least one out of every three consecutive output bits is a 1.
3. If there are infinitely many 0's in the input stream, then there are infinitely many 0's in the output stream.

First two requirements can be satisfied by always outputting a 1, but not the third.

How to satisfy all three?

Example: A Specification

Consider a very simple system with a **single bit of input** and a **single bit of output**.

Our specification is the conjunction of the following three requirements:

1. Whenever the input bit is 1, then the output bit is 1, too.
2. At least one out of every three consecutive output bits is a 1.
3. If there are infinitely many 0's in the input stream, then there are infinitely many 0's in the output stream.

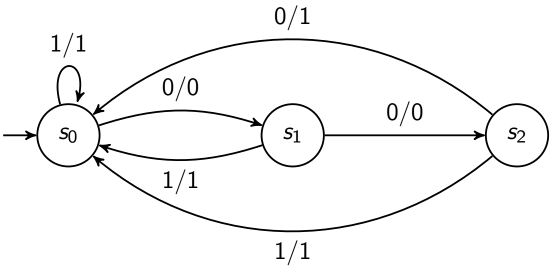
First two requirements can be satisfied by always outputting a 1, but not the third.

How to satisfy all three?

Correct behavior: Every 1 in the input stream is answered by a 1. If input bit is a 0, answer with a 0, unless the last two output bits were 0: in this case output a 1.

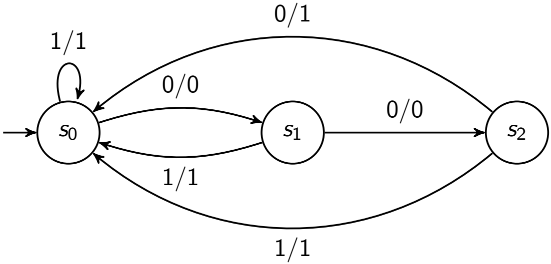
Example: Verification of a Transition System

Correct behavior is implemented by the following system, where the label 1/1 stands for "read input 1 and produce output 1".



Example: Verification of a Transition System

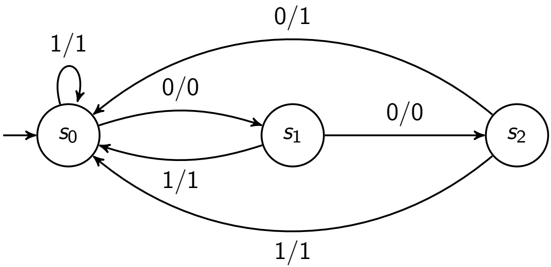
Correct behavior is implemented by the following system, where the label 1/1 stands for "read input 1 and produce output 1".



Verification: relatively easy (individual transition labels for first requirement, analysis of loops for the other two)

Example: Verification of a Transition System

Correct behavior is implemented by the following system, where the label 1/1 stands for "read input 1 and produce output 1".

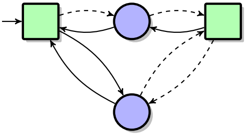


Verification: relatively easy (individual transition labels for first requirement, analysis of loops for the other two)

However: if an error is found, designer has to fix the system by hand, and may introduce new errors in the process

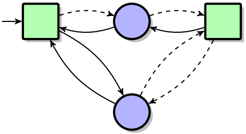
Reactive Synthesis is a Game

Idea: separate system behavior into choices of the environment (for inputs) and the system (for outputs). Encode requirements as a graph and let the two players play a turn-based **infinite game**.



Reactive Synthesis is a Game

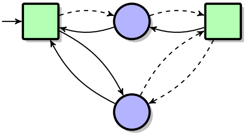
Idea: separate system behavior into choices of the environment (for inputs) and the system (for outputs). Encode requirements as a graph and let the two players play a turn-based **infinite game**.



A **winning condition** determines which player wins the game, i.e., is a condition on infinite paths through the game graph.

Reactive Synthesis is a Game

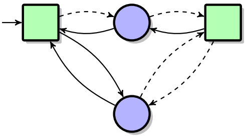
Idea: separate system behavior into choices of the environment (for inputs) and the system (for outputs). Encode requirements as a graph and let the two players play a turn-based **infinite game**.



A **winning condition** determines which player wins the game, i.e., is a condition on infinite paths through the game graph.

A **strategy** of a player is a function that returns a legal next move for the given player, based on what happened in the game so far.

Idea: separate system behavior into choices of the environment (for inputs) and the system (for outputs). Encode requirements as a graph and let the two players play a turn-based **infinite game**.



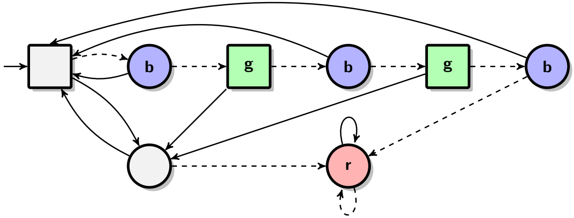
A **winning condition** determines which player wins the game, i.e., is a condition on infinite paths through the game graph.

A **strategy** of a player is a function that returns a legal next move for the given player, based on what happened in the game so far.

A **winning strategy** is a strategy such that the given player will win the game, regardless of the moves of their opponent.

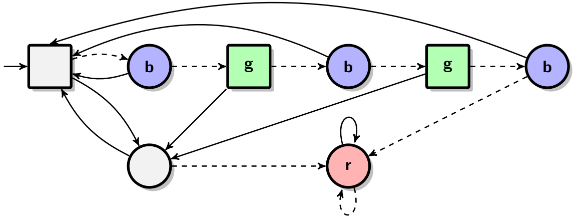
Example: Reactive Synthesis is a Game

Game graph that models the example specification (solid edges model picking a 1, dashed edges picking a 0):



Example: Reactive Synthesis is a Game

Game graph that models the example specification (solid edges model picking a 1, dashed edges picking a 0):

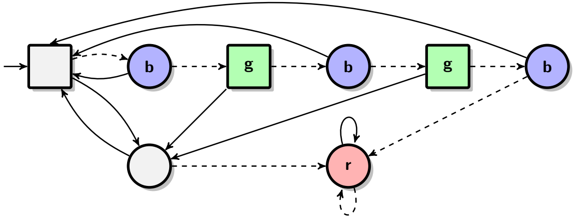


Winning condition:

- never visit red vertex (properties 1 & 2)

Example: Reactive Synthesis is a Game

Game graph that models the example specification (solid edges model picking a 1, dashed edges picking a 0):

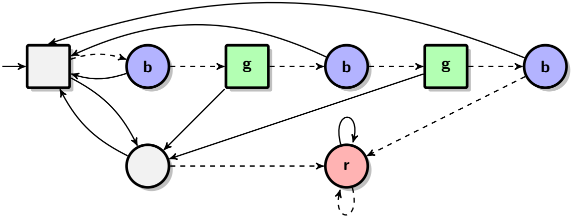


Winning condition:

- never visit red vertex (properties 1 & 2)
- if infinitely many blue vertices are visited, then inf. many green vertices are visited (property 3).

Example: Reactive Synthesis is a Game

Game graph that models the example specification (solid edges model picking a 1, dashed edges picking a 0):



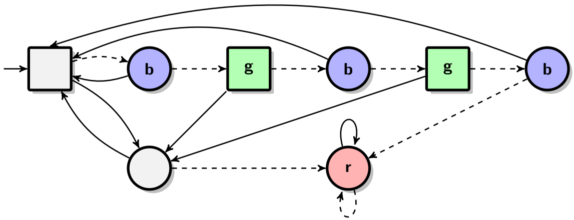
Winning condition:

- never visit red vertex (properties 1 & 2)
- if infinitely many blue vertices are visited, then inf. many green vertices are visited (property 3).

Winning strategy for Player 0:

Example: Reactive Synthesis is a Game

Game graph that models the example specification (solid edges model picking a 1, dashed edges picking a 0):



Winning condition:

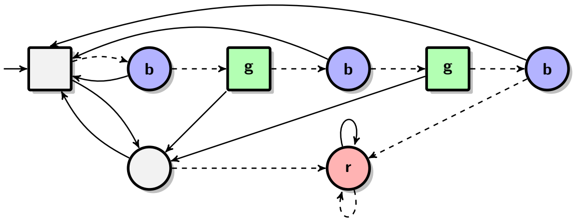
- never visit red vertex (properties 1 & 2)
- if infinitely many blue vertices are visited, then inf. many green vertices are visited (property 3).

Winning strategy for Player 0:

- Never move to red vertex

Example: Reactive Synthesis is a Game

Game graph that models the example specification (solid edges model picking a 1, dashed edges picking a 0):



Winning condition:

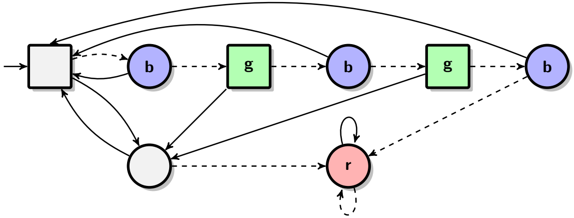
- never visit red vertex (properties 1 & 2)
- if infinitely many blue vertices are visited, then inf. many green vertices are visited (property 3).

Winning strategy for Player 0:

- Never move to red vertex
- From blue vertex, move to green if possible

Example: Reactive Synthesis is a Game

Game graph that models the example specification (solid edges model picking a 1, dashed edges picking a 0):



Winning condition:

- never visit red vertex (properties 1 & 2)
- if infinitely many blue vertices are visited, then inf. many green vertices are visited (property 3).

Winning strategy for Player 0:

- Never move to red vertex
- From blue vertex, move to green if possible
- Resulting input-output behavior: same as system seen before.

The Büchi-Landweber Theorem

Theorem [BL69]: Every infinite game in a finite graph with ω-regular winning condition is determined. Finite-state strategies are sufficient to win these games, and can be computed effectively.

The Büchi-Landweber Theorem

Theorem [BL69]: Every infinite game in a finite graph with ω-regular winning condition is determined. Finite-state strategies are sufficient to win these games, and can be computed effectively.

In particular, this means **we can solve Church's problem**:

1. model the specification as
   - a finite game graph that describes the interaction between system and environment, and
   - a winning condition on this graph
2. find a winning strategy for the system player
3. generate a system that implements this strategy

Overview of Course

- Basic Games
- Algorithms & Data Structures
  - Project Kickoff
  - Project Submission
- Advanced Games
- Temporal Logic Synthesis
  - Project Evaluation

Basic Games

Formalization of **basic game-theoretic notions**: arena (game graph), play, winning condition, strategy, winning strategy

Solving reachability and safety games

Properties of basic games: decidability, determinacy, existence of positional (memoryless) strategies

Algorithms & Data Structures

Definition of

- **algorithms** to find winning strategies for a given game,
- **data structures** that allow us to symbolically represent properties of the game that are computed in these algorithms, and
- **optimizations and heuristics** that allow us to find winning strategies more efficiently.

Project

After second block, students will start their **project** in which they implement their own synthesis tool based on the contents of the course so far. This should be done in groups of two students.

In January, first an initial version of the implementation will be **submitted and checked for correctness**. After that, we will allow a short time period for bug fixes.

Until the end of January, final versions will be run on a large benchmark set in a competition. **Results of the competition** will be announced in the final lecture.

Advanced Games

More expressive winning conditions: Büchi, Co-Büchi, Parity, LTL

Solving such games

State-of-the-art algorithms for solving LTL Synthesis:
Bounded Synthesis

Organization

Tutorials:

Place: Seminar Room 15 in Building E1.3

Time: Tue **either** 14:15-16:00 **or** 16:15-18:00

One problem set per week (except for project weeks), to be solved in groups of two. Handed out during the lecture, collected **before** the next tutorial.

Exams, Grading:

The final grade will be composed of the project grade (1/3) and the grade from an exam (2/3).

For admission to the exam, students must obtain 50% of the exercises points from the problem sets during the course.

(Dates for exams will be announced soon)

Registration

If you have not done so, please register on
`https://courses.react.uni-saarland.de/rs1718/`

Plan for (the Rest of) Today

- Basic Games

- Algorithms & Data Structures
- Advanced Games
- Temporal Logic Synthesis

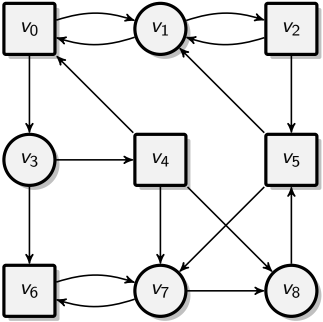
Plan for (the Rest of) Today

- Basic Games
  - Arenas, strategies, and games
  - Reachability and safety games
- Algorithms & Data Structures
- Advanced Games
- Temporal Logic Synthesis

Definition

An *arena* $\mathcal{A} = (V, V_0, V_1, E)$ consists of

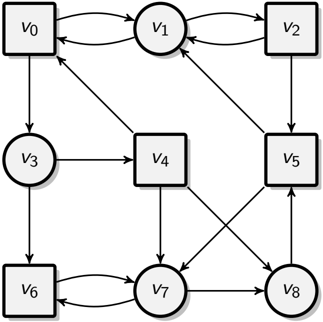
- a finite set V of vertices,
- disjoint sets $V_0, V_1 \subseteq V$ with $V = V_0 \cup V_1$ denoting the vertices of Player 0 and Player 1 respectively, and
- a set $E \subseteq V \times V$ of (directed) edges without terminal vertices, i.e., $\{v' \in V \mid (v, v') \in E\}$ is non-empty for every $v \in V$.



Definition

An *arena* $\mathcal{A} = (V, V_0, V_1, E)$ consists of

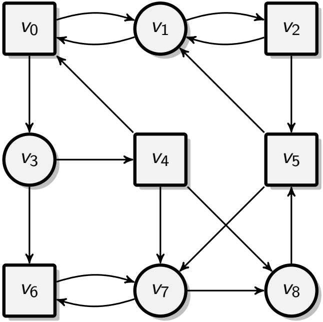
- a finite set V of vertices,
- disjoint sets $V_0, V_1 \subseteq V$ with $V = V_0 \cup V_1$ denoting the vertices of Player 0 and Player 1 respectively, and
- a set $E \subseteq V \times V$ of (directed) edges without terminal vertices, i.e., $\{v' \in V \mid (v, v') \in E\}$ is non-empty for every $v \in V$.



Remark

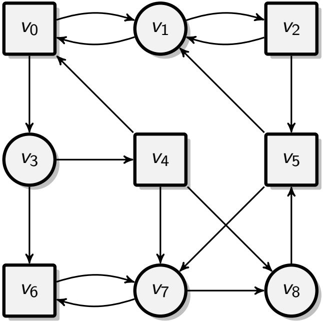
The size of $\mathcal{A}$, denoted by $|\mathcal{A}|$, is defined to be $|V|$.

Plays



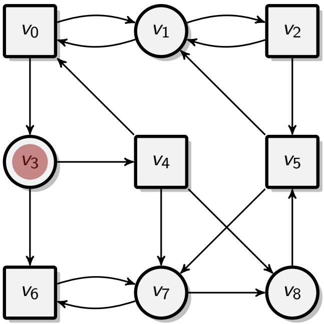
Plays

- To start a play, a token is placed at some initial vertex ρ_0.



Plays

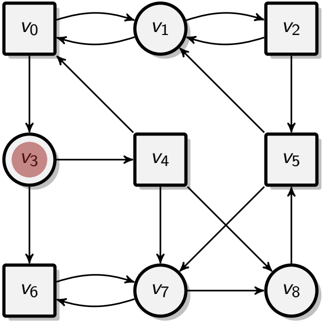
- To start a play, a token is placed at some initial vertex ρ_0.



Example

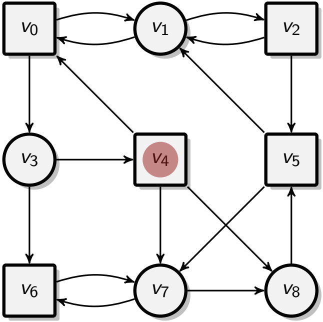
Plays

- To start a play, a token is placed at some initial vertex ρ_0.
- Assume the token is at some vertex ρ_n with $\rho_n \in V_i$. Then, Player i moves the token to a successor ρ_{n+1} of ρ_n.

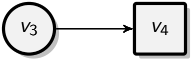


Example

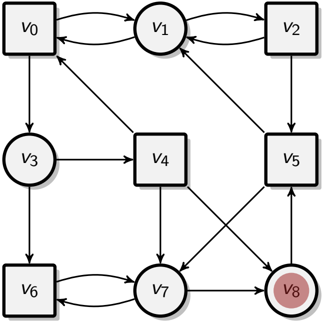
- To start a play, a token is placed at some initial vertex ρ_0.
- Assume the token is at some vertex ρ_n with $\rho_n \in V_i$. Then, Player i moves the token to a successor ρ_{n+1} of ρ_n.



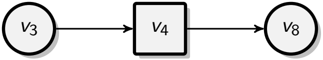
Example



- To start a play, a token is placed at some initial vertex ρ_0.
- Assume the token is at some vertex ρ_n with $\rho_n \in V_i$. Then, Player i moves the token to a successor ρ_{n+1} of ρ_n.

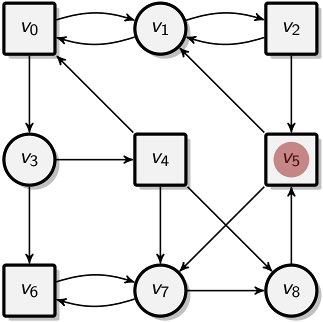


Example



Plays

- To start a play, a token is placed at some initial vertex ρ_0.
- Assume the token is at some vertex ρ_n with $\rho_n \in V_i$. Then, Player i moves the token to a successor ρ_{n+1} of ρ_n.

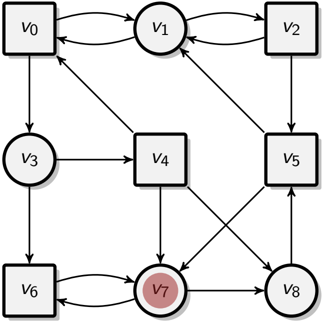


Example

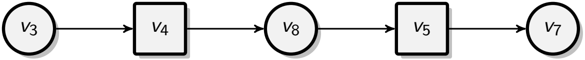
$v_3 \to v_4 \to v_8 \to v_5$

Plays

- To start a play, a token is placed at some initial vertex ρ_0.
- Assume the token is at some vertex ρ_n with $\rho_n \in V_i$. Then, Player i moves the token to a successor ρ_{n+1} of ρ_n.

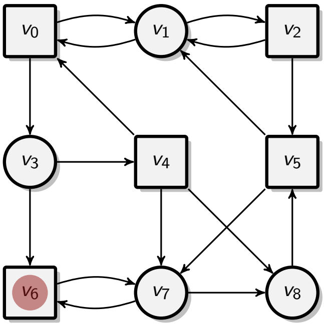


Example

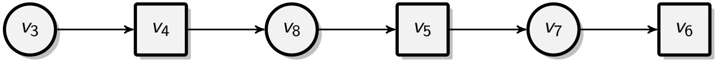


Plays

- To start a play, a token is placed at some initial vertex ρ_0.
- Assume the token is at some vertex ρ_n with $\rho_n \in V_i$. Then, Player i moves the token to a successor ρ_{n+1} of ρ_n.

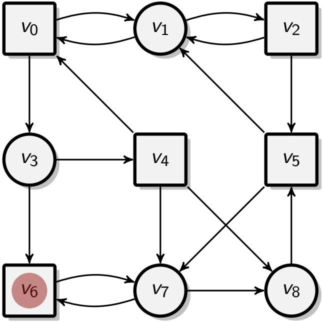


Example

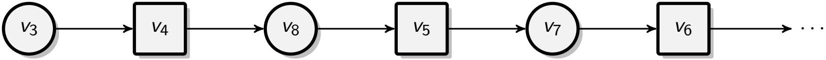


Plays

- To start a play, a token is placed at some initial vertex ρ_0.
- Assume the token is at some vertex ρ_n with $\rho_n \in V_i$. Then, Player i moves the token to a successor ρ_{n+1} of ρ_n.

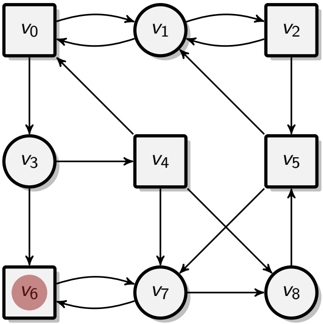


Example

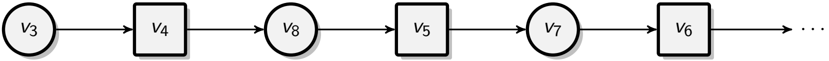


Plays

- To start a play, a token is placed at some initial vertex ρ_0.
- Assume the token is at some vertex ρ_n with $\rho_n \in V_i$. Then, Player i moves the token to a successor ρ_{n+1} of ρ_n.
- As the arena has no terminal vertices, the players can always make a move.
- Hence, the outcome is an infinite sequence of vertices.

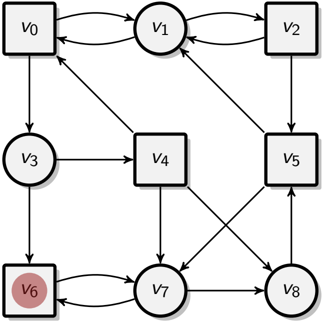


Example

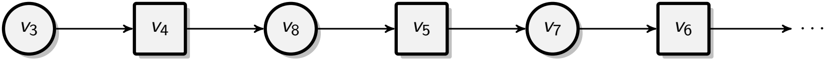


Plays

- To start a play, a token is placed at some initial vertex ρ_0.
- Assume the token is at some vertex ρ_n with $\rho_n \in V_i$. Then, Player i moves the token to a successor ρ_{n+1} of ρ_n.
- As the arena has no terminal vertices, the players can always make a move.
- Hence, the outcome is an infinite sequence of vertices.



Example



Definition

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

Strategies

Strategies

Definition

A *strategy* for Player $i \in \{0, 1\}$ in an arena (V, V_0, V_1, E) is a function $\sigma \colon V^* V_i \to V$ such that $\sigma(wv) = v'$ implies $(v, v') \in E$ for every $w \in V^*$ and every $v \in V_i$.

A play $\rho_0 \rho_1 \rho_2 \cdots$ is *consistent* with σ if $\rho_{n+1} = \sigma(\rho_0 \cdots \rho_n)$ for every $n \in \mathbb{N}$ with $\rho_n \in V_i$.

Definition

A *strategy* for Player $i \in \{0, 1\}$ in an arena (V, V_0, V_1, E) is a function $\sigma\colon V^*V_i \to V$ such that $\sigma(wv) = v'$ implies $(v, v') \in E$ for every $w \in V^*$ and every $v \in V_i$.

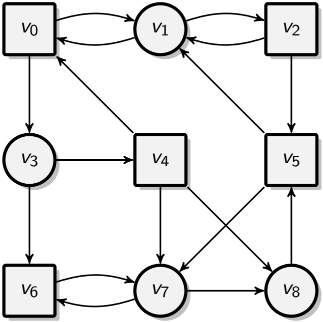
A play $\rho_0\rho_1\rho_2\cdots$ is *consistent* with σ if $\rho_{n+1} = \sigma(\rho_0 \cdots \rho_n)$ for every $n \in \mathbb{N}$ with $\rho_n \in V_i$.

Example

Consider the following strategy:

- $\sigma(wv_1) = v_0$
- $\sigma(wv_3) = v_6$
- $\sigma(wv_7) = v_6$
- $\sigma(wv_8) = v_5$

$v_0v_1v_0v_1v_0v_3(v_6v_7)^\omega$ is consistent with σ.



Games

Definition

A *game* $\mathcal{G} = (\mathcal{A}, \mathrm{Win})$ consists of an arena $\mathcal{A}$ with vertex set V and a winning condition $\mathrm{Win} \subseteq V^\omega$.

Player 0 wins a play ρ if $\rho \in \mathrm{Win}$; otherwise, Player 1 wins ρ.

Games

Definition

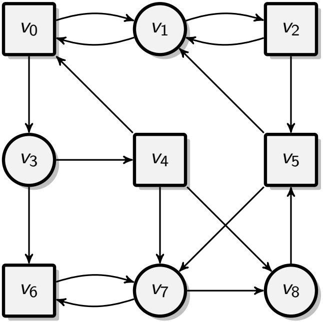
A *game* $\mathcal{G} = (\mathcal{A}, \mathrm{Win})$ consists of an arena $\mathcal{A}$ with vertex set V and a winning condition $\mathrm{Win} \subseteq V^\omega$.

Player 0 wins a play ρ if $\rho \in \mathrm{Win}$; otherwise, Player 1 wins ρ.

Definition

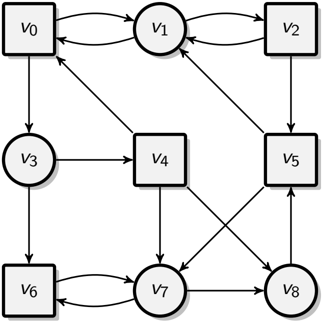
A strategy σ for Player i is a *winning strategy* for $\mathcal{G}$ from a vertex v if every play that is consistent with σ and starts in v is winning for Player i.

An Example



$$\mathrm{Win} = \{\rho_0\rho_1\rho_2\cdots \in V^\omega \mid \exists v \in V \text{ such that } \rho_n \neq v \text{ for all } n\}$$

An Example

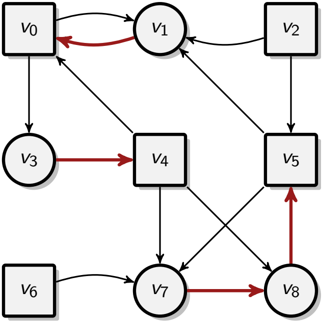


$$\mathrm{Win} = \{\rho_0\rho_1\rho_2\cdots \in V^\omega \mid \exists v \in V \text{ such that } \rho_n \neq v \text{ for all } n\}$$

A winning strategy for Player 0 from every vertex:

- $\sigma(wv_1) = v_0$
- $\sigma(wv_3) = v_4$
- $\sigma(wv_7) = v_8$
- $\sigma(wv_8) = v_5$

An Example



$$\mathrm{Win} = \{\rho_0\rho_1\rho_2\cdots \in V^\omega \mid \exists v \in V \text{ such that } \rho_n \neq v \text{ for all } n\}$$

A winning strategy for Player 0 from every vertex:

- $\sigma(wv_1) = v_0$
- $\sigma(wv_3) = v_4$
- $\sigma(wv_7) = v_8$
- $\sigma(wv_8) = v_5$

Determinacy

Notation

$W_i(\mathcal{G}) = \{v \in V \mid \text{Player } i \text{ has winning strategy for } \mathcal{G} \text{ from } v\}$:
the *winning region* of Player i in $\mathcal{G}$.

Lemma

We have $W_0(\mathcal{G}) \cap W_1(\mathcal{G}) = \emptyset$ *for every game* $\mathcal{G}$.

Determinacy

Notation

$W_i(\mathcal{G}) = \{v \in V \mid \text{Player } i \text{ has winning strategy for } \mathcal{G} \text{ from } v\}$: the *winning region* of Player i in $\mathcal{G}$.

Lemma

We have $W_0(\mathcal{G}) \cap W_1(\mathcal{G}) = \emptyset$ for every game $\mathcal{G}$.

Proof.

On the blackboard. □

Determinacy

Notation

$W_i(\mathcal{G}) = \{v \in V \mid \text{Player } i \text{ has winning strategy for } \mathcal{G} \text{ from } v\}$: the *winning region* of Player i in $\mathcal{G}$.

Lemma

We have $W_0(\mathcal{G}) \cap W_1(\mathcal{G}) = \emptyset$ *for every game* $\mathcal{G}$.

Proof.

On the blackboard. $\square$

Definition

A game $\mathcal{G}$ with vertex set V is *determined* if $W_0(\mathcal{G}) \cup W_1(\mathcal{G}) = V$.

Algorithmic Problems

Solving a game

Given (a finite representation of) a game $\mathcal{G} = (\mathcal{A}, \mathrm{Win})$, determine the winning regions $W_i(\mathcal{G})$ and (finite representations of) corresponding winning strategies.

Algorithmic Problems

Solving a game

Given (a finite representation of) a game $\mathcal{G} = (\mathcal{A}, \mathrm{Win})$, determine the winning regions $W_i(\mathcal{G})$ and (finite representations of) corresponding winning strategies.

Questions

1. How do we represent Win finitely?
2. How do we represent (winning) strategies finitely?

Positional Strategies

- A strategy $\sigma \colon V^* V_i \to V$ is an infinite object.
- Often strategies are finitely representable.

Positional Strategies

- A strategy $\sigma\colon V^* V_i \to V$ is an infinite object.
- Often strategies are finitely representable.
- Recall the last example:
  - $\sigma(wv_1) = v_0$
  - $\sigma(wv_3) = v_4$
  - $\sigma(wv_7) = v_8$
  - $\sigma(wv_8) = v_5$
- The output $\sigma(wv)$ only depends on the input's last vertex v.

Positional Strategies

- A strategy $\sigma\colon V^* V_i \to V$ is an infinite object.
- Often strategies are finitely representable.
- Recall the last example:
  - $\sigma(wv_1) = v_0$
  - $\sigma(wv_3) = v_4$
  - $\sigma(wv_7) = v_8$
  - $\sigma(wv_8) = v_5$
- The output $\sigma(wv)$ only depends on the input's last vertex v.

Definition

A strategy σ for Player i is *positional* (or *memoryless*) if $\sigma(wv) = \sigma(v)$ for all $w \in V^*$ and all $v \in V_i$.

Positional Strategies

- A strategy $\sigma \colon V^* V_i \to V$ is an infinite object.
- Often strategies are finitely representable.
- Recall the last example:
  - $\sigma(wv_1) = v_0$
  - $\sigma(wv_3) = v_4$
  - $\sigma(wv_7) = v_8$
  - $\sigma(wv_8) = v_5$
- The output $\sigma(wv)$ only depends on the input's last vertex v.

Definition

A strategy σ for Player i is *positional* (or *memoryless*) if $\sigma(wv) = \sigma(v)$ for all $w \in V^*$ and all $v \in V_i$.

Notation

We write $\sigma \colon V_i \to V$ instead of $\sigma \colon V^* V_i \to V$, if σ is positional.

Winning Conditions

Typically, winning conditions Win are obtained from acceptance conditions for ω-automata or from specification logics, which specify Win by finite objects (sets of vertices, labelings of vertices, formulas, etc.)

Winning Conditions

Typically, winning conditions Win are obtained from acceptance conditions for ω-automata or from specification logics, which specify Win by finite objects (sets of vertices, labelings of vertices, formulas, etc.)

- Automata theoretic conditions:
  - Reachability and Safety
  - Büchi and co-Büchi
  - Parity
  - Rabin, Streett, and Muller

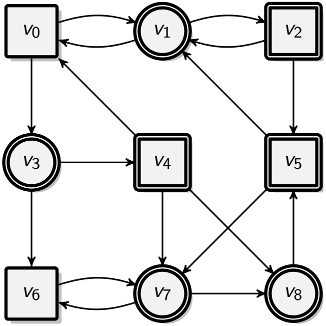
Winning Conditions

Typically, winning conditions Win are obtained from acceptance conditions for ω-automata or from specification logics, which specify Win by finite objects (sets of vertices, labelings of vertices, formulas, etc.)

- Automata theoretic conditions:
  - Reachability and Safety
  - Büchi and co-Büchi
  - Parity
  - Rabin, Streett, and Muller
- Specification Logics:
  - Linear Temporal Logic (LTL)
  - several industrial logics based on LTL

Safety Games

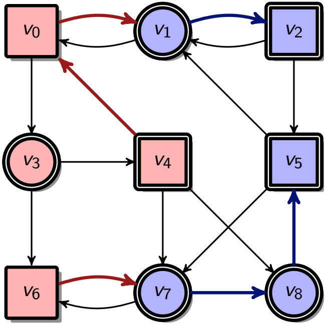
A foundational winning condition: staying safe.



Player 0 wins a play, if only safe vertices (marked by double line) are visited.

Safety Games

A foundational winning condition: staying safe.



Player 0 wins a play, if only safe vertices (marked by double line) are visited.

Notation

$\mathrm{Occ}(\rho) := \{v \in V \mid \rho_n = v \text{ for some } n\}$: the set of vertices occurring in ρ.

Definition

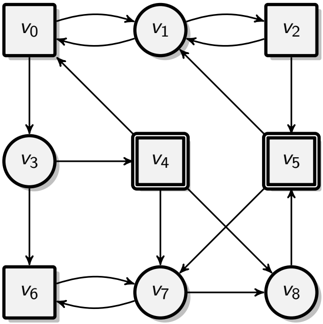
Let $\mathcal{A} = (V, V_0, V_1, E)$ be an arena and let $S \subseteq V$ be a subset of $\mathcal{A}$'s vertices. Then, the safety condition $\textsc{Safety}(S)$ is defined as

$$\textsc{Safety}(S) := \{\rho \in V^\omega \mid \mathrm{Occ}(\rho) \subseteq S\}.$$

We call a game $\mathcal{G} = (\mathcal{A}, \textsc{Safety}(S))$ a *safety game*.

Reachability Games

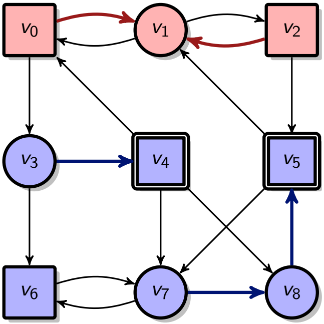
Another foundational winning condition: reaching a goal.



Player 0 wins a play, if at least one goal vertex (marked by double line) is visited.

Reachability Games

Another foundational winning condition: reaching a goal.



Player 0 wins a play, if at least one goal vertex (marked by double line) is visited.

Reachability Games

Definition

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 $\textsc{Reach}(R)$ is defined as

$$\textsc{Reach}(R) := \{\rho \in V^\omega \mid \mathrm{Occ}(\rho) \cap R \neq \emptyset\}.$$

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