Reactive Synthesis
Lecture 01

Swen Jacobs and Martin Zimmermann
(Saarland University)
Reactive Systems

Systems that react to environment inputs in potentially infinite executions
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:

- **Requirements**
- **Formal Specification**
- **Implementation**

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

Verification Workflow:

- Requirements
- Formal Specification
- Implementation
- Verification
- Bug-fixing

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

Verification Workflow:

- Requirements
  - Formal Specification
  - Implementation
  - Verification
  - Bug-fixing

Synthesis Workflow:

- Requirements
  - Formal Specification
  - Implementation
  - Synthesis
  - Verification
  - Bug-fixing
The synthesis problem is also known as **Church’s problem**, since Alonso Church first defined it in 1962.
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”.

![Transition System Diagram](image-url)
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.

A winning condition determines which player wins the game, i.e., 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.
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**.

![Game Graph](image)

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.
**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**.

![Game Graph](image)

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

Resulting input-output behavior: same as system seen before.
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 B"uchi-Landweber Theorem

**Theorem [BL69]:** Every infinite game in a finite graph with \( \omega \)-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 $\omega$-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
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
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.
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.
More expressive winning conditions: Büchi, Co-Büchi, Parity, LTL

Solving such games
State-of-the-art algorithms for solving LTL Synthesis:
Bounded Synthesis
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)
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
**Arenas**

**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 $|V|$. 

Reactive Synthesis (2017/18) 18
Arenas

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

To start a play, a token is placed at some initial vertex $\rho_0$. Assume the token is at some vertex $\rho_n$ with $\rho_n \in V_i$. Then, Player $i$ moves the token to a successor $\rho_{n+1}$ of $\rho_n$. As the arena has no terminal vertices, the players can always make a move. Hence, the outcome is an infinite sequence of vertices.

Example

\[
\begin{align*}
&v_3 \\ &v_4 \quad v_5 \\ &v_6 \\ &v_7 \quad v_8
\end{align*}
\]

Definition

A play in $A$ 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 $n \in \mathbb{N}$. 

Reactive Synthesis (2017/18) 19
Plays

- To start a play, a token is placed at some initial vertex \( \rho_0 \).
Plays

- To start a play, a token is placed at some initial vertex $\rho_0$.

Example

```
   v_3
```

Definition

A play in $A$ 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 $n \in \mathbb{N}$. 
Plays

- To start a play, a token is placed at some initial vertex $\rho_0$.
- Assume the token is at some vertex $\rho_n$ with $\rho_n \in V_i$. Then, Player $i$ moves the token to a successor $\rho_{n+1}$ of $\rho_n$.

Example

$\quad v_3$

Definition

A play in $A$ is an infinite sequence $\rho = \rho_0, \rho_1, \rho_2, \ldots \in V^\omega$ such that $(\rho_n, \rho_{n+1}) \in E$ for every $n \in \mathbb{N}$. 
To start a play, a token is placed at some initial vertex \( \rho_0 \).

Assume the token is at some vertex \( \rho_n \) with \( \rho_n \in V_i \). Then, Player \( i \) moves the token to a successor \( \rho_{n+1} \) of \( \rho_n \).

**Example**

\[ v_3 \rightarrow v_4 \]
To start a play, a token is placed at some initial vertex $\rho_0$.

Assume the token is at some vertex $\rho_n$ with $\rho_n \in V_i$. Then, Player $i$ moves the token to a successor $\rho_{n+1}$ of $\rho_n$.

Example

Diagram:

- $v_0$ to $v_1$ to $v_2$
- $v_0$ to $v_3$ to $v_4$ to $v_5$
- $v_3$ to $v_6$ to $v_7$ to $v_8$
- $v_1$ to $v_4$
- $v_2$ to $v_5$
- $v_7$ to $v_8$

Definition

A play in $A$ 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 $n \in \mathbb{N}$. 
To start a play, a token is placed at some initial vertex $\rho_0$.

Assume the token is at some vertex $\rho_n$ with $\rho_n \in V_i$. Then, Player $i$ moves the token to a successor $\rho_{n+1}$ of $\rho_n$.

Example

$\rho_3 \rightarrow \rho_4 \rightarrow \rho_8 \rightarrow \rho_5$
Plays

- To start a play, a token is placed at some initial vertex $\rho_0$.
- Assume the token is at some vertex $\rho_n$ with $\rho_n \in V_i$. Then, Player $i$ moves the token to a successor $\rho_{n+1}$ of $\rho_n$.

Example

\[
\begin{align*}
  &v_3 \rightarrow v_4 \rightarrow v_8 \rightarrow v_5 \rightarrow v_7 \\
\end{align*}
\]
Plays

- To start a play, a token is placed at some initial vertex $\rho_0$.
- Assume the token is at some vertex $\rho_n$ with $\rho_n \in V_i$. Then, Player $i$ moves the token to a successor $\rho_{n+1}$ of $\rho_n$.

Example

Diagram showing a sequence of states $v_3, v_4, v_8, v_5, v_7, v_6$.
Plays

- To start a play, a token is placed at some initial vertex $\rho_0$.

- Assume the token is at some vertex $\rho_n$ with $\rho_n \in V_i$. Then, Player $i$ moves the token to a successor $\rho_{n+1}$ of $\rho_n$.

Example

![Graph diagram showing a sequence of vertices and edges representing a play.]

Definition

A play in $A$ 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 $n \in \mathbb{N}$. 
Plays

- To start a play, a token is placed at some initial vertex $\rho_0$.
- Assume the token is at some vertex $\rho_n$ with $\rho_n \in V_i$. Then, Player $i$ moves the token to a successor $\rho_{n+1}$ of $\rho_n$.
- As the arena has no terminal vertices, the players can always make a move.
- Hence, the outcome is an infinite sequence of vertices.

Example
To start a play, a token is placed at some initial vertex \( \rho_0 \).

Assume the token is at some vertex \( \rho_n \) with \( \rho_n \in V_i \). Then, Player \( i \) moves the token to a successor \( \rho_{n+1} \) of \( \rho_n \).

As the arena has no terminal vertices, the players can always make a move.

Hence, the outcome is an infinite sequence of vertices.

**Example**

```plaintext
v_3 \rightarrow v_4 \rightarrow v_8 \rightarrow v_5 \rightarrow v_7 \rightarrow v_6 \rightarrow \cdots
```

**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

A strategy for Player $i \in \{0, 1\}$ in an arena $(V, V_0, V_1, E)$ is a function $\sigma: V^* \times V_i \rightarrow V$ 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 $\sigma$ 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:

$\sigma(wv_1) = v_0$

$\sigma(wv_3) = v_6$

$\sigma(wv_7) = v_6$

$\sigma(wv_8) = v_5$

$v_0 v_1 v_0 v_1 v_0 v_3 (v_6 v_7)$ $\omega$ is consistent with $\sigma$. 

Reactive Synthesis (2017/18)
Strategies

Definition

A strategy for Player \( i \in \{0, 1\} \) in an arena \((V, V_0, V_1, E)\) is a function \( \sigma : V^* V_i \rightarrow V \) 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 \( \sigma \) if \( \rho_{n+1} = \sigma(\rho_0 \cdots \rho_n) \) for every \( n \in \mathbb{N} \) with \( \rho_n \in V_i \).
**Strategies**

**Definition**
A *strategy* for Player $i \in \{0, 1\}$ in an arena $(V, V_0, V_1, E)$ is a function $\sigma : V^* V_i \rightarrow V$ 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 $\sigma$ 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:

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

$v_0 v_1 v_0 v_1 v_0 v_3 (v_6 v_7)^\omega$ is consistent with $\sigma$. 


**Definition**

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

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

**Definition**
A game $\mathcal{G} = (\mathcal{A}, \text{Win})$ consists of an arena $\mathcal{A}$ with vertex set $\mathcal{V}$ and a winning condition $\text{Win} \subseteq \mathcal{V}^\omega$.
Player 0 wins a play $\rho$ if $\rho \in \text{Win}$; otherwise, Player 1 wins $\rho$.

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

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

A winning strategy for Player 0 from every vertex:

\[ \sigma(ww_1) = v_0 \]
\[ \sigma(ww_3) = v_4 \]
\[ \sigma(ww_7) = v_8 \]
\[ \sigma(ww_8) = v_5 \]

\[ \text{Win} = \{ \rho_0 \rho_1 \rho_2 \cdots \in V^\omega | \exists v \in V \text{ such that } \rho_n \neq v \text{ for all } n \} \]
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
$W_i(G) = \{v \in V \mid \text{Player } i \text{ has winning strategy for } G \text{ from } v\}$: the winning region of Player $i$ in $G$.

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

Proof.
On the blackboard.
Determinacy

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

Lemma
We have \( W_0(G) \cap W_1(G) = \emptyset \) for every game \( G \).

Proof.
On the blackboard.

Definition
A game \( G \) with vertex set \( V \) is determined if \( W_0(G) \cup W_1(G) = V \).
Solving a game
Given (a finite representation of) a game $G = (A, \text{Win})$, determine the winning regions $W_i(G)$ and (finite representations of) corresponding winning strategies.
Algorithmic Problems

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

Questions

1. How do we represent \( \text{Win} \) finitely?
2. How do we represent (winning) strategies finitely?
Positional Strategies

- A strategy $\sigma : V^* V_i \rightarrow V$ is an infinite object.
- Often strategies are finitely representable.
Positional Strategies

- A strategy $\sigma : V^* V_i \rightarrow V$ 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 $v$. 
A strategy $\sigma : V^* V_i \rightarrow V$ 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 $\sigma$ for Player $i$ is *positional* (or *memoryless*) if $\sigma(wv) = \sigma(v)$ for all $w \in V^*$ and all $v \in V_i$. 
A strategy $\sigma : V^* V_i \rightarrow V$ 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 $\sigma$ 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 $\sigma : V_i \rightarrow V$ instead of $\sigma : V^* V_i \rightarrow V$, if $\sigma$ is positional.
Typically, winning conditions $Win$ are obtained from acceptance conditions for $\omega$-automata or from specification logics, which specify $Win$ by finite objects (sets of vertices, labelings of vertices, formulas, etc.).
Typically, winning conditions $Win$ are obtained from acceptance conditions for $\omega$-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
Typically, winning conditions $\text{Win}$ are obtained from acceptance conditions for $\omega$-automata or from specification logics, which specify $\text{Win}$ 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
A foundational winning condition: staying safe.

Player 0 wins a play, if only safe vertices (marked by double line) are visited.
A foundational winning condition: staying safe.

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

**Notation**

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

**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 \( \text{SAFETY}(S) \) is defined as

\[
\text{SAFETY}(S) := \{ \rho \in V^\omega \mid \text{Occ}(\rho) \subseteq S \}.
\]

We call a game \( \mathcal{G} = (\mathcal{A}, \text{SAFETY}(S)) \) a safety game.
Another foundational winning condition: reaching a goal.

Player 0 wins a play, if at least one goal vertex (marked by double line) is visited.
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 $\text{Reach}(R)$ is defined as

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

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