Introduction to Model Checking

Fabio Somenzi

Department of Electrical, Computer, and Energy Engineering
University of Colorado at Boulder

July 25, 2009

Outline

The Sales Pitch

- Two thirds of ASIC budget goes into verification
- Dynamic verification has improved, but . . .
- The verification crisis only got worse over the last decade
- Over 60% of IC designs requires a second *spin*
- Bugs that go undetected may end up costing hundreds of millions of dollars
  - The FDIV bug costed Intel over $ 500M
- Security and dependability are increasingly important

The Sales Pitch

- Two thirds of ASIC budget goes into verification
- Dynamic verification has improved, but . . .
- The verification crisis only got worse over the last decade
- Over 60% of IC designs requires a second *spin*
- Bugs that go undetected may end up costing hundreds of millions of dollars
  - The FDIV bug costed Intel over $ 500M
- Security and dependability are increasingly important

The Sales Pitch

- Two thirds of ASIC budget goes into verification
- Dynamic verification has improved, but . . .
- The verification crisis only got worse over the last decade
- Over 60% of IC designs requires a second *spin*
- Bugs that go undetected may end up costing hundreds of millions of dollars
  - The FDIV bug costed Intel over $ 500M
- Security and dependability are increasingly important

The Sales Pitch

- Two thirds of ASIC budget goes into verification
- Dynamic verification has improved, but . . .
- The verification crisis only got worse over the last decade
- Over 60% of IC designs requires a second *spin*
- Bugs that go undetected may end up costing hundreds of millions of dollars
  - The FDIV bug costed Intel over $ 500M
- Security and dependability are increasingly important

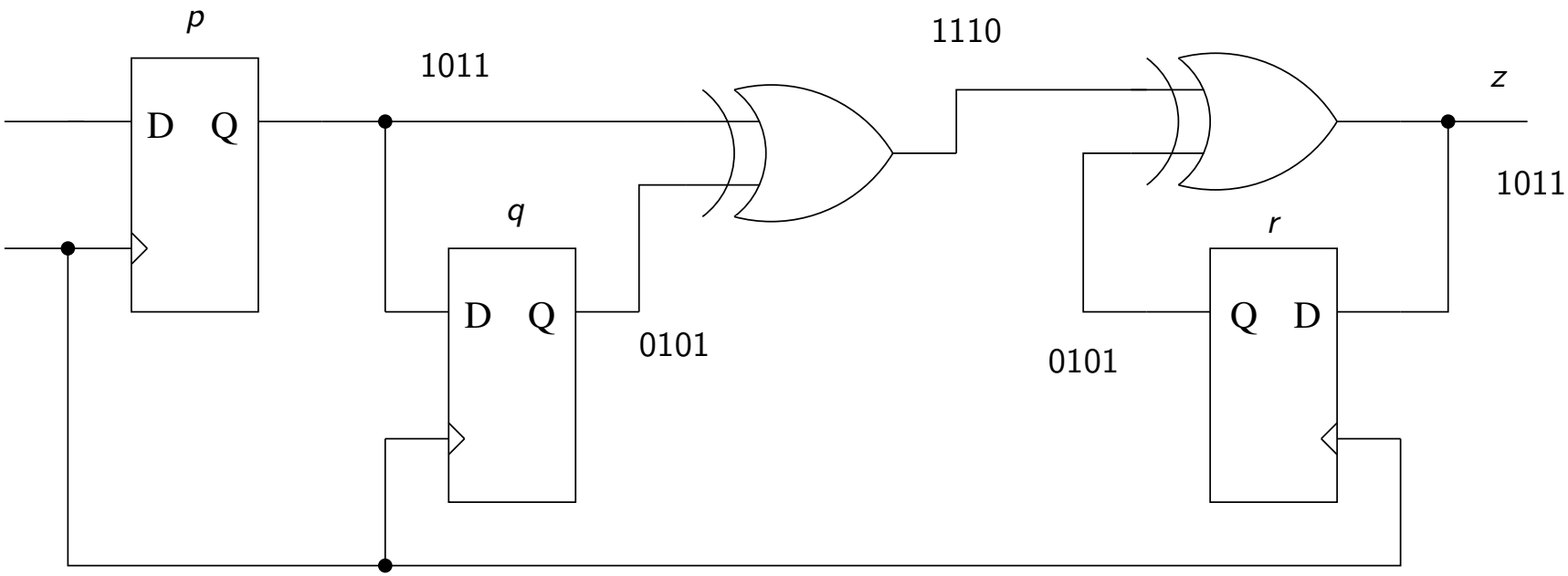
The Sales Pitch

- Two thirds of ASIC budget goes into verification
- Dynamic verification has improved, but . . .
- The verification crisis only got worse over the last decade
- Over 60% of IC designs requires a second *spin*
- Bugs that go undetected may end up costing hundreds of millions of dollars
  - The FDIV bug costed Intel over $ 500M
- Security and dependability are increasingly important

The Sales Pitch

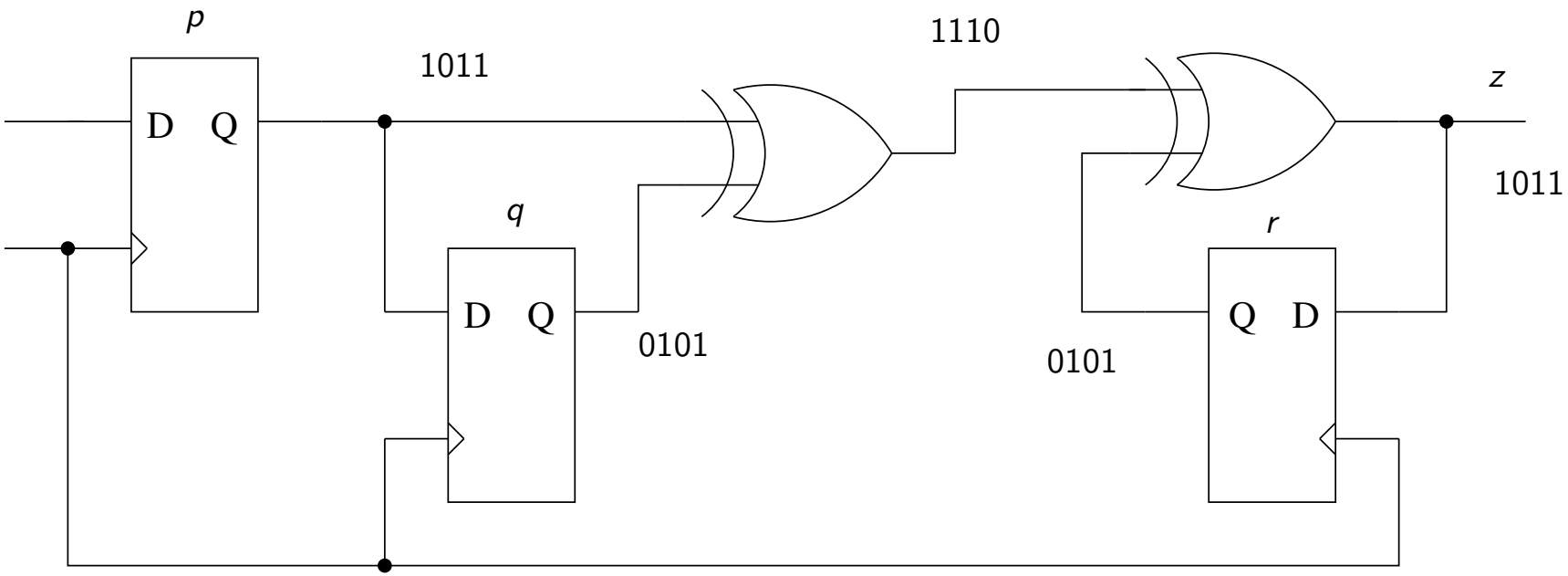
- Two thirds of ASIC budget goes into verification
- Dynamic verification has improved, but . . .
- The verification crisis only got worse over the last decade
- Over 60% of IC designs requires a second *spin*
- Bugs that go undetected may end up costing hundreds of millions of dollars
  - The FDIV bug costed Intel over $ 500M
- Security and dependability are increasingly important

Binary-Gray-Binary



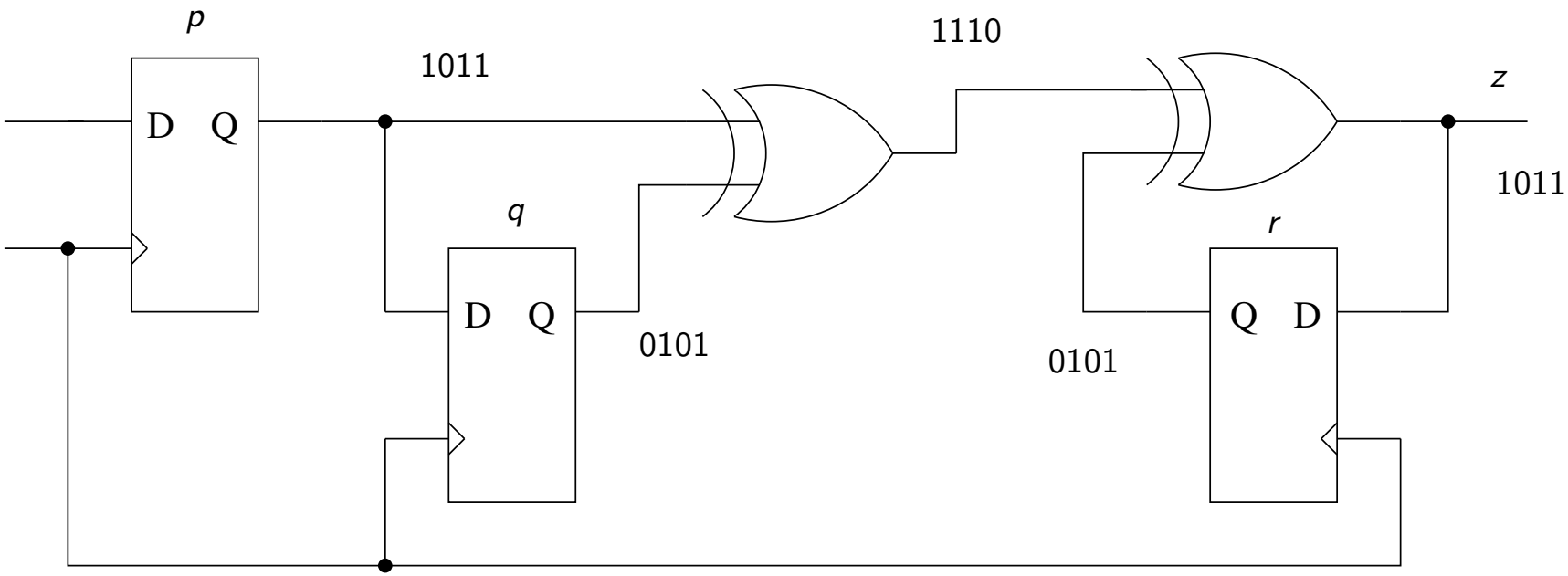
AG($p \leftrightarrow z$): invariably (AG) p and z have the same value

Binary-Gray-Binary



$\mathrm{AG}(p \leftrightarrow z)$: invariably (AG) p and z have the same value

Binary-Gray-Binary



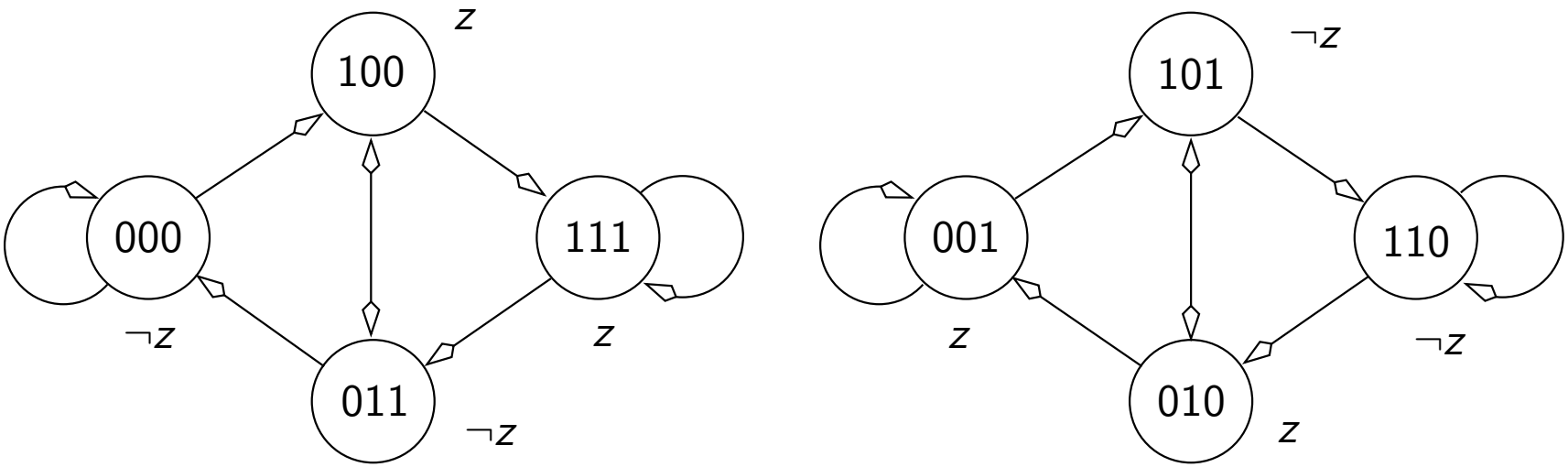
AG($p \leftrightarrow z$): invariably (AG) p and z have the same value

Verilog Description

```
module gray (clock, i, z);
    input clock, i;
    output z;
    reg p, q, r;
    wire w;
    always @ (posedge clock) begin
        r = z;
        q = p;
        p = i;
    end
    assign w = p^q, z = w^r;
endmodule // gray
```

States and Transitions

$100 : p \wedge \neg q \wedge \neg r$



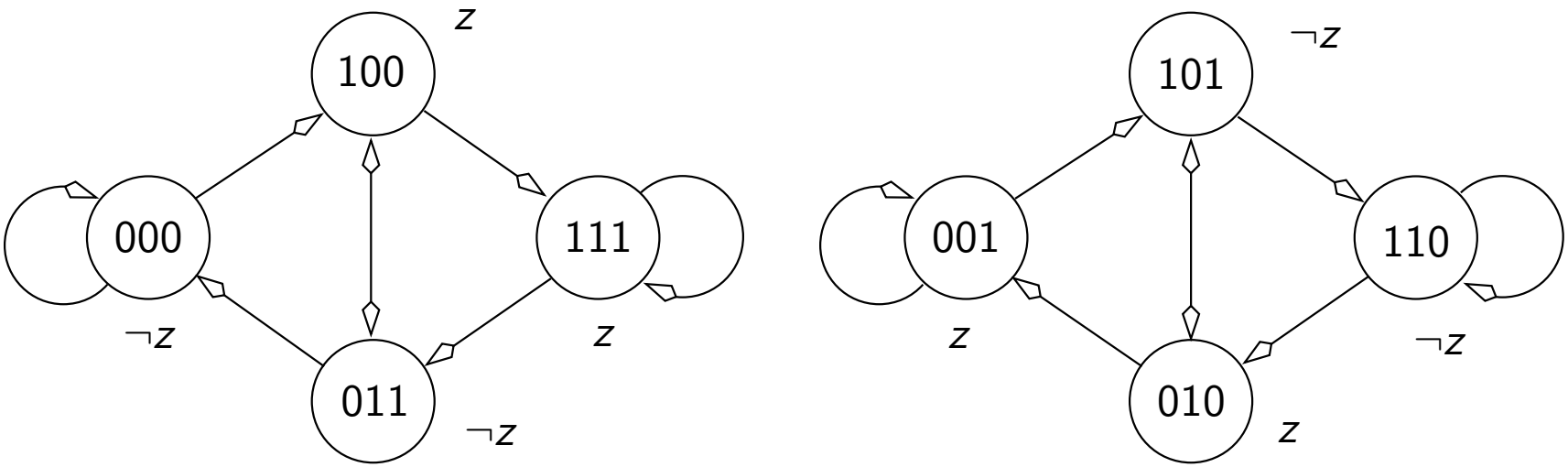
$(p \leftrightarrow z) \rightarrow \mathrm{AG}(p \leftrightarrow z)$ holds of all initial states

$(q \leftrightarrow r) \rightarrow \mathrm{AG}(p \leftrightarrow z)$ holds of all initial states

$\mathrm{AG}((p \leftrightarrow z) \leftrightarrow (q \leftrightarrow r))$ holds of all initial states

States and Transitions

$100 : p \wedge \neg q \wedge \neg r$



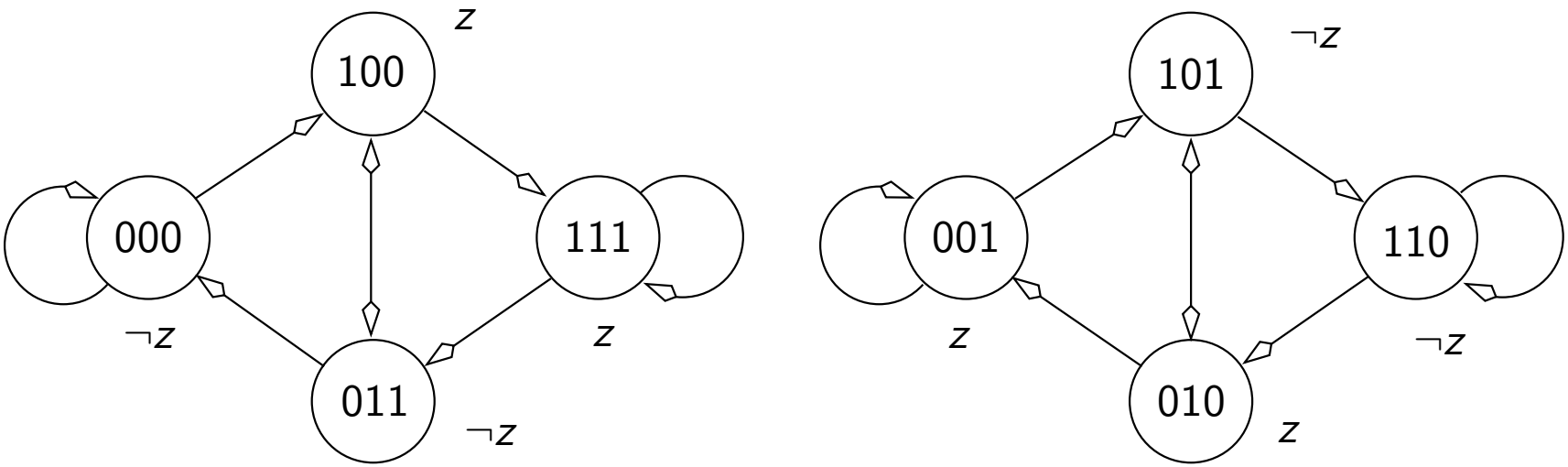
$(p \leftrightarrow z) \rightarrow \mathrm{AG}(p \leftrightarrow z)$ holds of all initial states

$(q \leftrightarrow r) \rightarrow \mathrm{AG}(p \leftrightarrow z)$ holds of all initial states

$\mathrm{AG}((p \leftrightarrow z) \leftrightarrow (q \leftrightarrow r))$ holds of all initial states

States and Transitions

$100 : p \wedge \neg q \wedge \neg r$



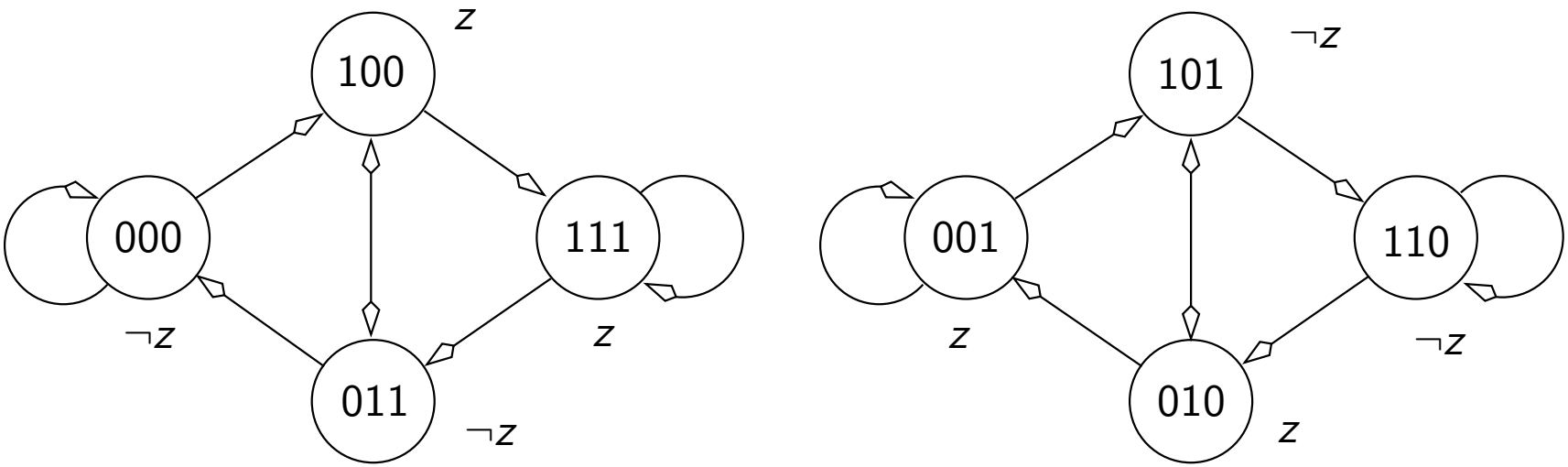
$(p \leftrightarrow z) \rightarrow \mathrm{AG}(p \leftrightarrow z)$ holds of all initial states

$(q \leftrightarrow r) \rightarrow \mathrm{AG}(p \leftrightarrow z)$ holds of all initial states

$\mathrm{AG}((p \leftrightarrow z) \leftrightarrow (q \leftrightarrow r))$ holds of all initial states

States and Transitions

$$100 : p \wedge \neg q \wedge \neg r$$

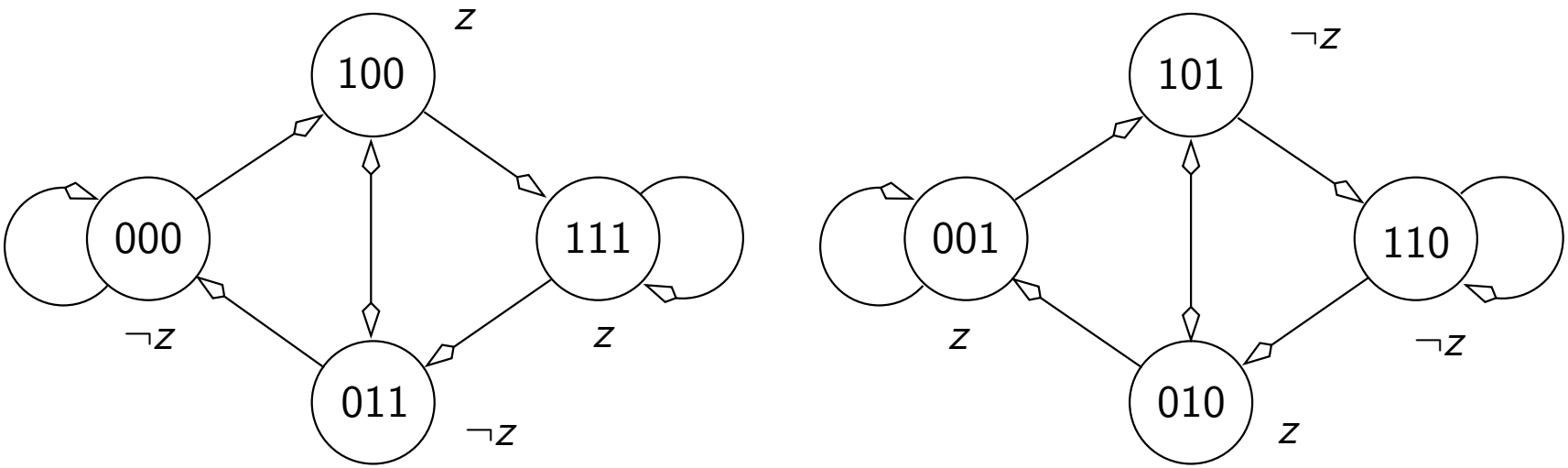


$(p \leftrightarrow z) \rightarrow \mathrm{AG}(p \leftrightarrow z)$ holds of all initial states

$(q \leftrightarrow r) \rightarrow \mathrm{AG}(p \leftrightarrow z)$ holds of all initial states

$\mathrm{AG}((p \leftrightarrow z) \leftrightarrow (q \leftrightarrow r))$ holds of all initial states

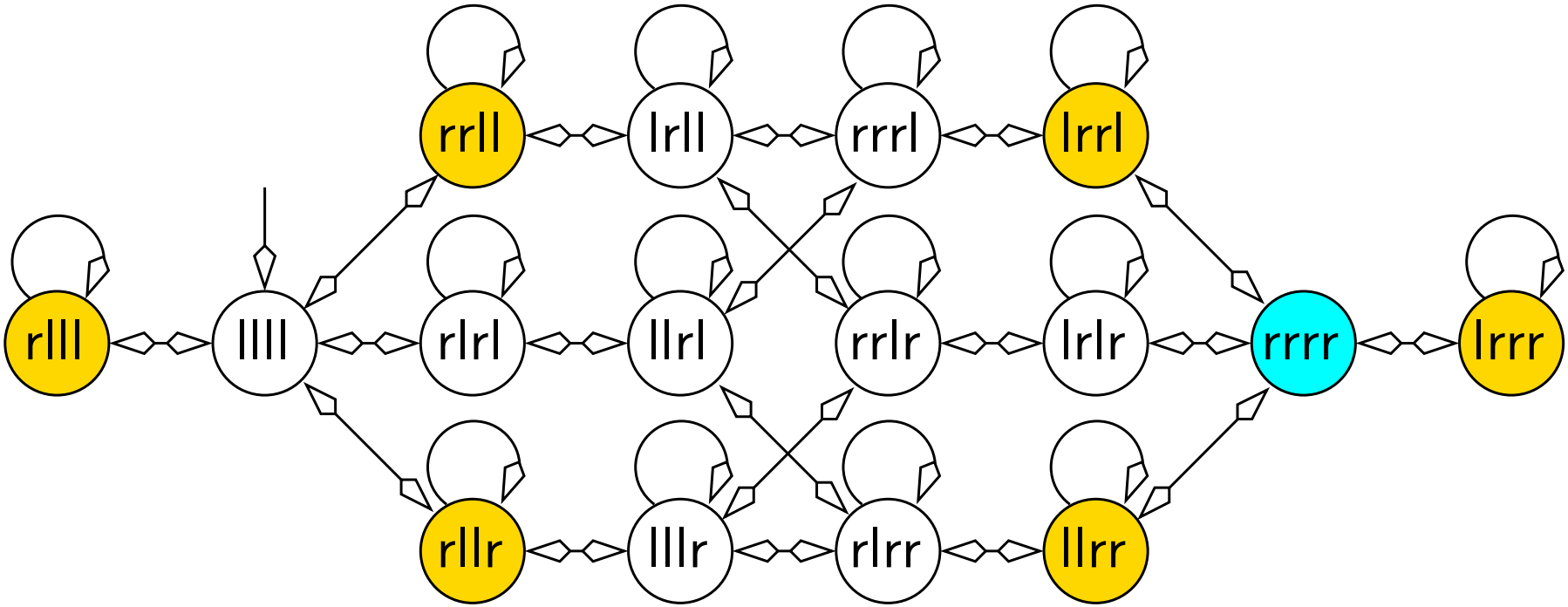
Another Property



$\mathrm{EF}(p \leftrightarrow z)$ "p and z may become equal"

does not hold of all initial states

The CGW Puzzle



rlrl: boat and goat on right bank; cabbage and wolf on left

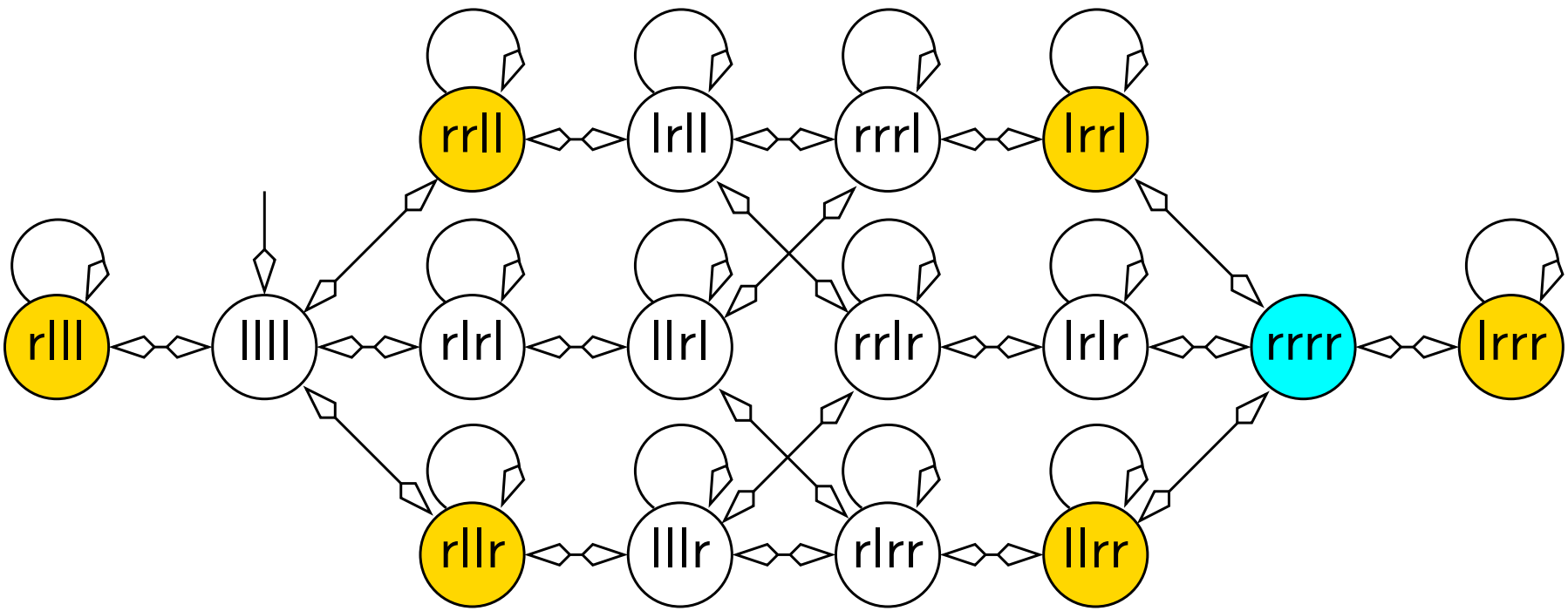
$$\mathsf{E}\,\neg\,\text{yellow}\;\;\mathsf{U}\;\;\text{cyan}$$

there is a path to a cyan state not going through any yellow states

Solving the Puzzle

- Compute

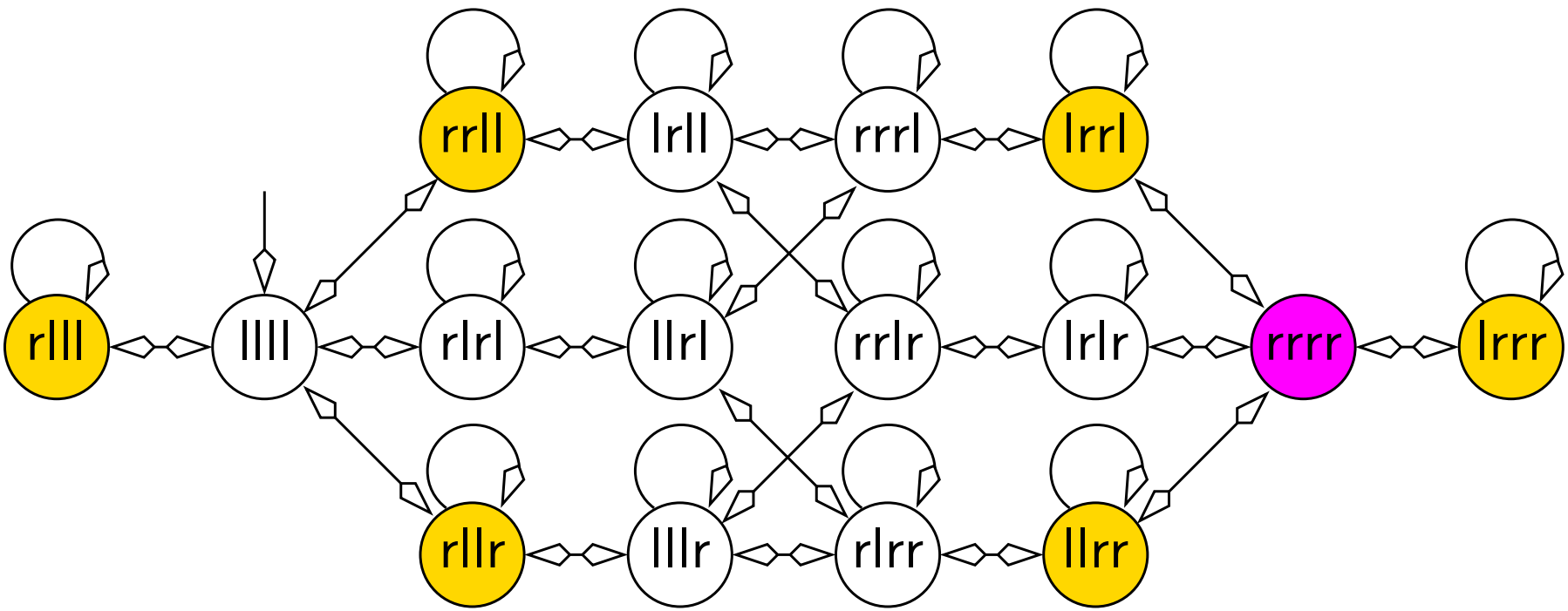
$$\mu Z \,.\, \text{cyan} \vee (\neg \text{yellow} \wedge \text{EX}\, Z)$$



Solving the Puzzle

- Compute

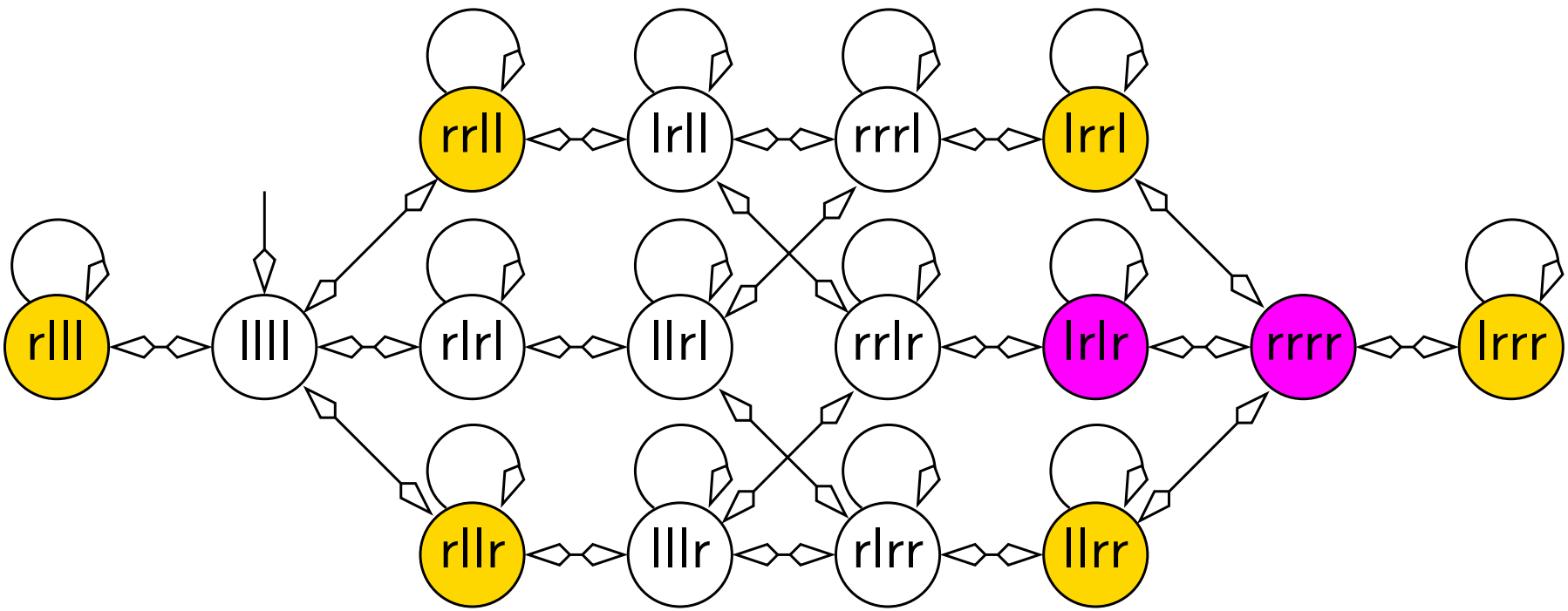
$$\mu Z \,.\, \text{cyan} \vee (\neg \text{yellow} \wedge \text{EX}\, Z)$$



Solving the Puzzle

- Compute

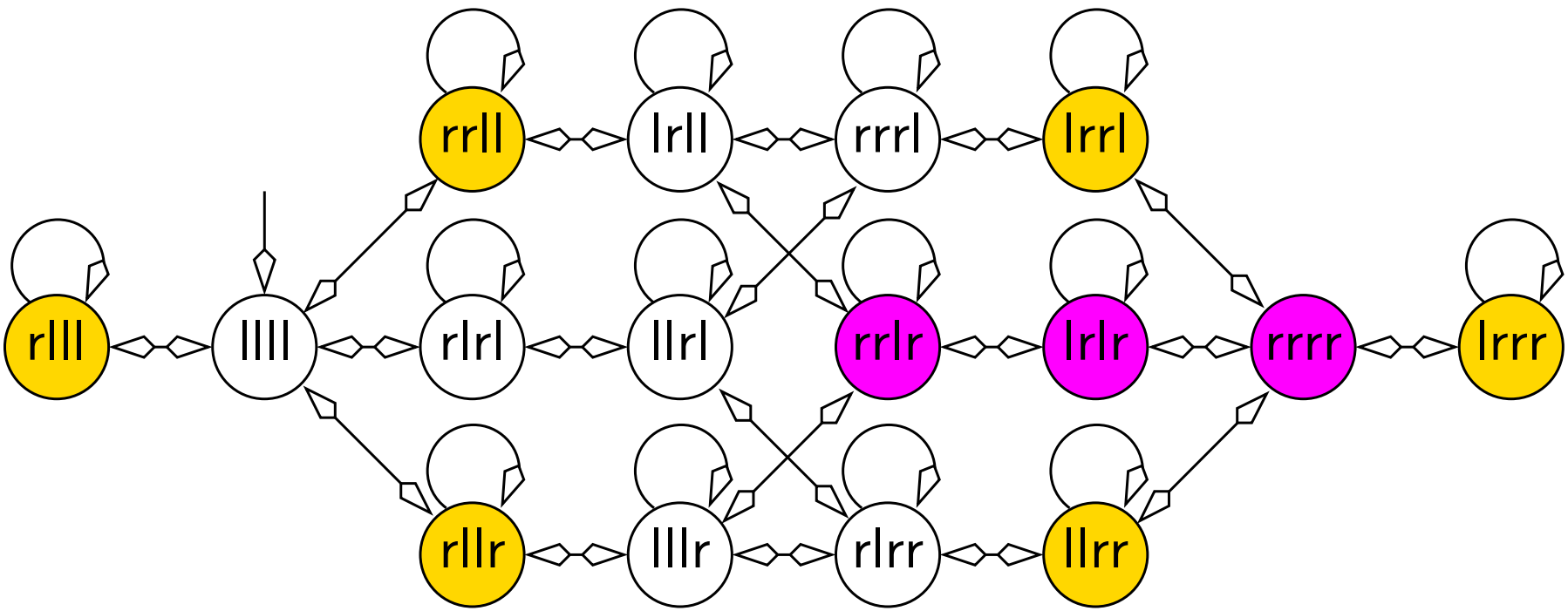
$$\mu Z \,.\, \text{cyan} \vee (\neg \text{yellow} \wedge \text{EX}\, Z)$$



Solving the Puzzle

- Compute

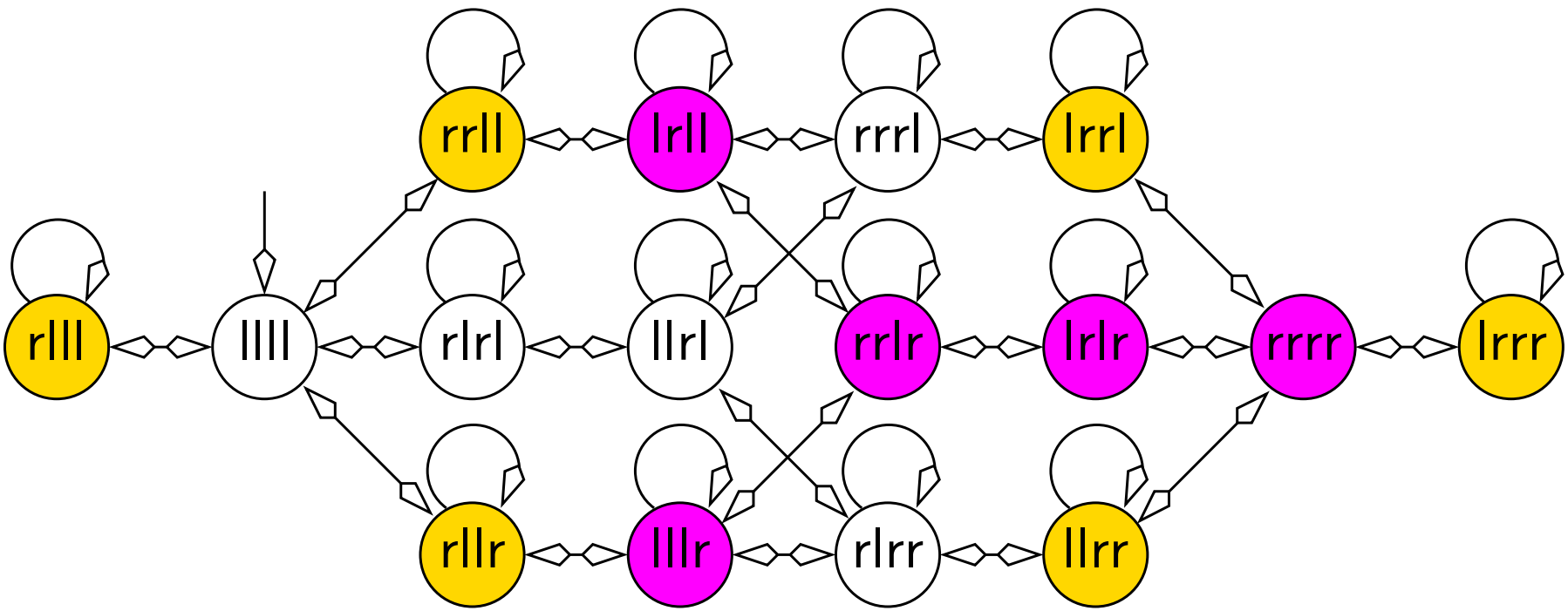
$$\mu Z \,.\, \text{cyan} \vee (\neg \text{yellow} \wedge \text{EX}\, Z)$$



Solving the Puzzle

- Compute

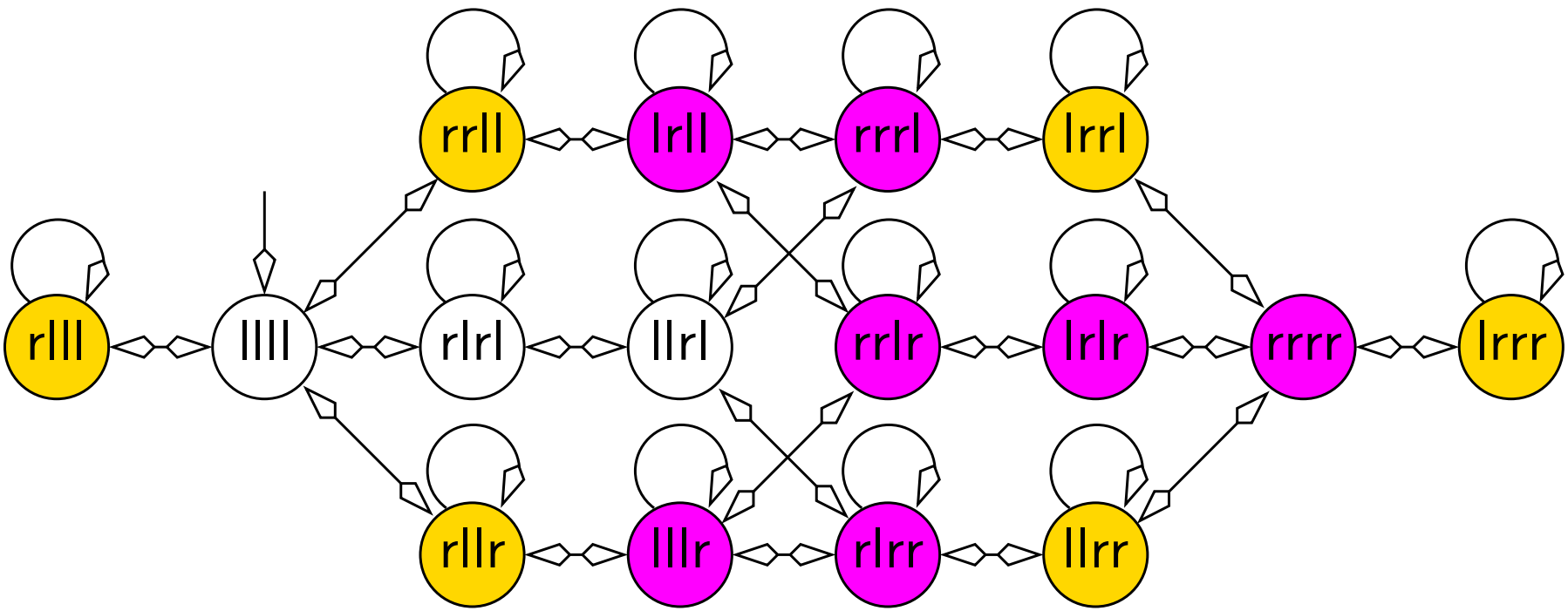
$$\mu Z \,.\, \text{cyan} \vee (\neg \text{yellow} \wedge \text{EX}\, Z)$$



Solving the Puzzle

- Compute

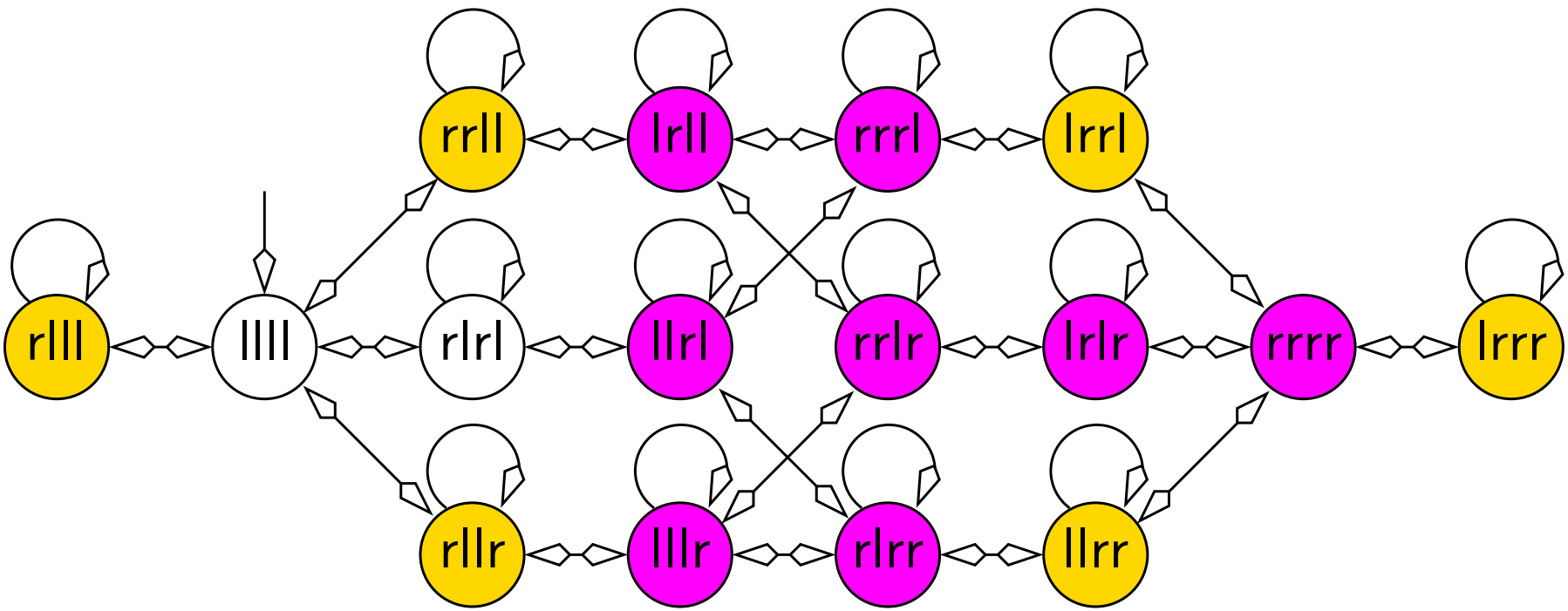
$$\mu Z \,.\, \text{cyan} \vee (\neg \text{yellow} \wedge \text{EX}\, Z)$$



Solving the Puzzle

- Compute

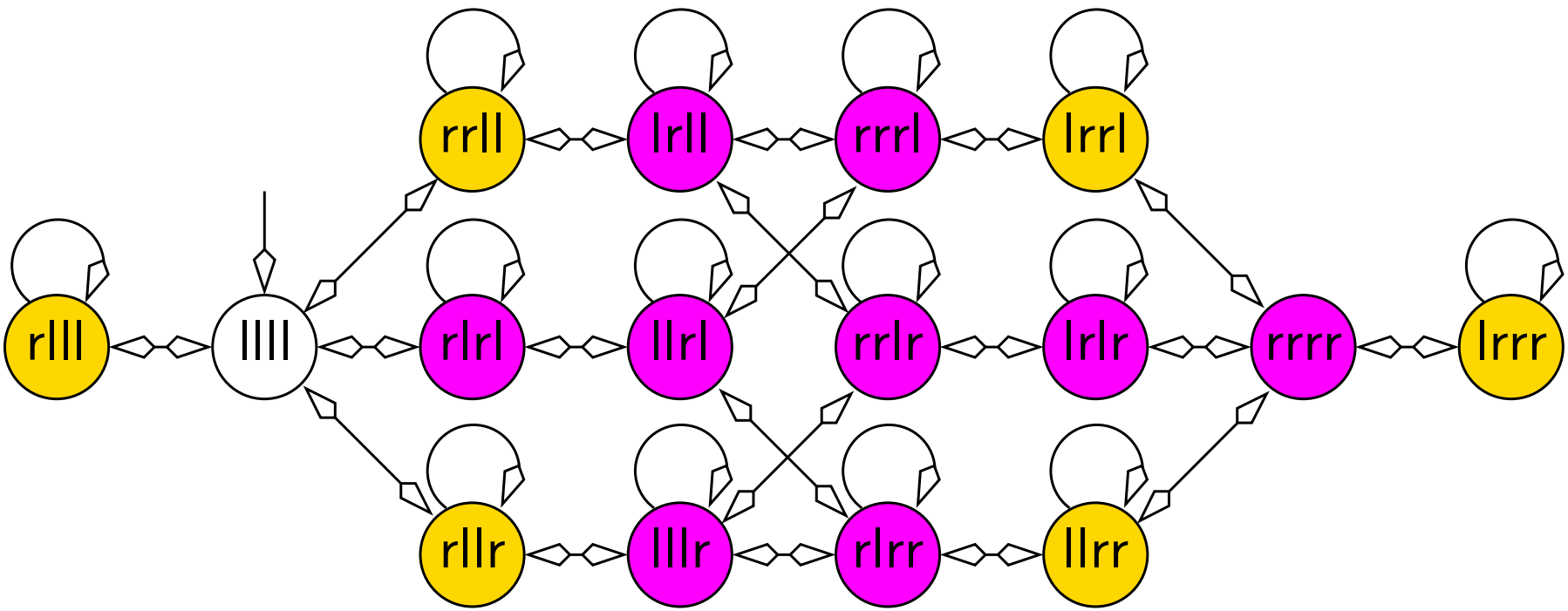
$$\mu Z \,.\, \text{cyan} \vee (\neg \text{yellow} \wedge \text{EX}\, Z)$$



Solving the Puzzle

- Compute

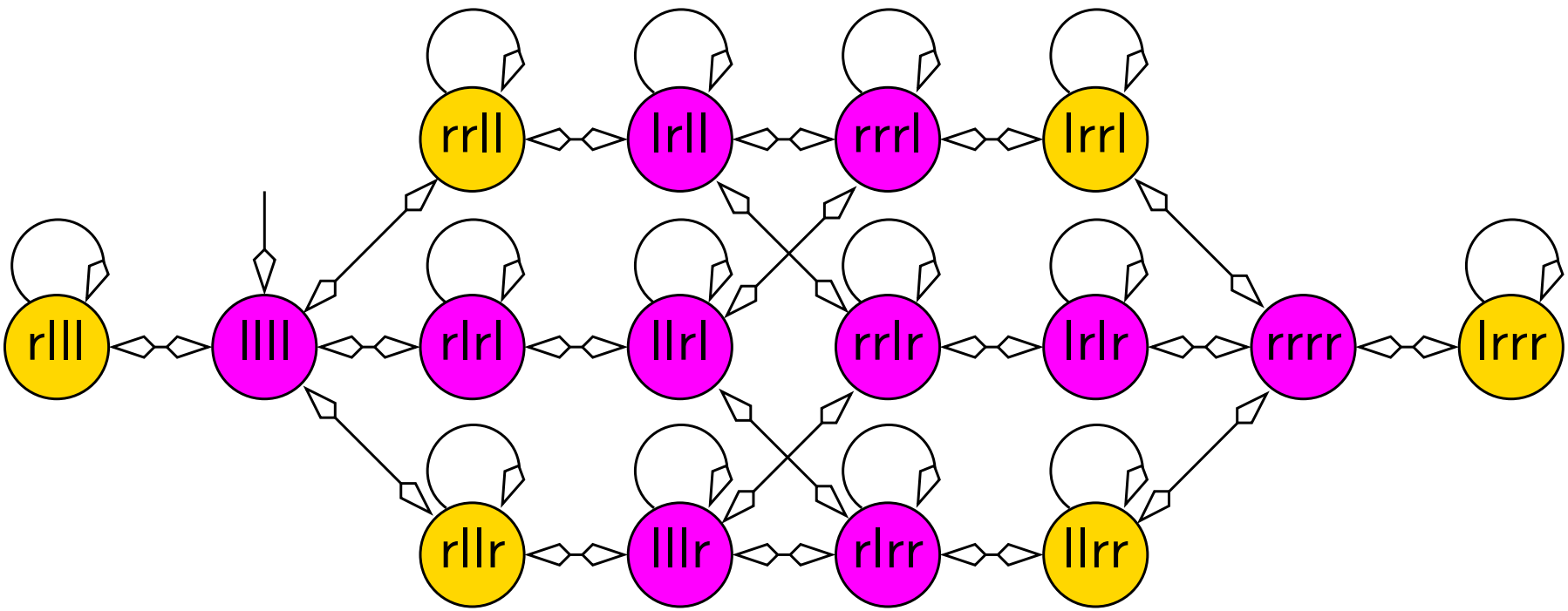
$$\mu Z \,.\, \text{cyan} \vee (\neg \text{yellow} \wedge \text{EX}\, Z)$$



Solving the Puzzle

- Compute

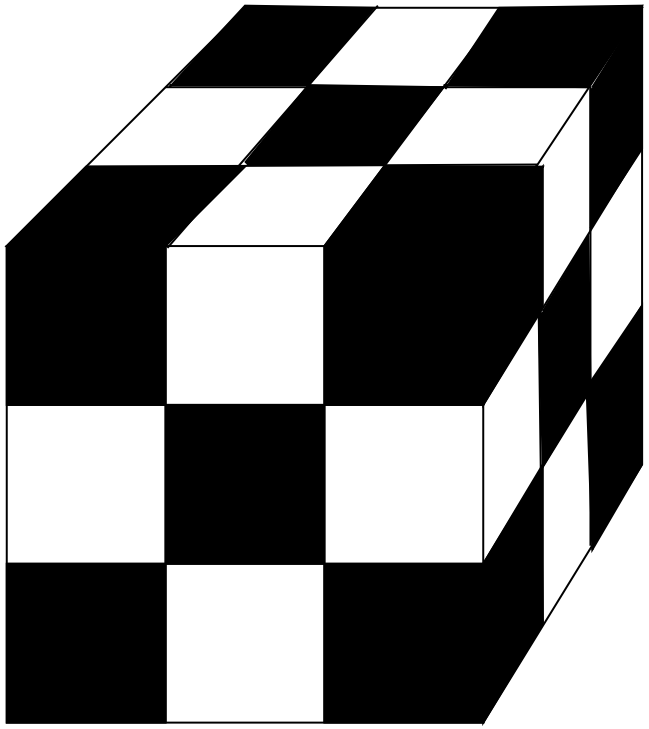
$$\mu Z \,.\, \text{cyan} \vee (\neg \text{yellow} \wedge \text{EX}\, Z)$$



Model Checking

- Check whether the given finite-state system is a model for a property
- That is, check whether the computations of the system satisfy the property
- The check is based on exploring the states of the system

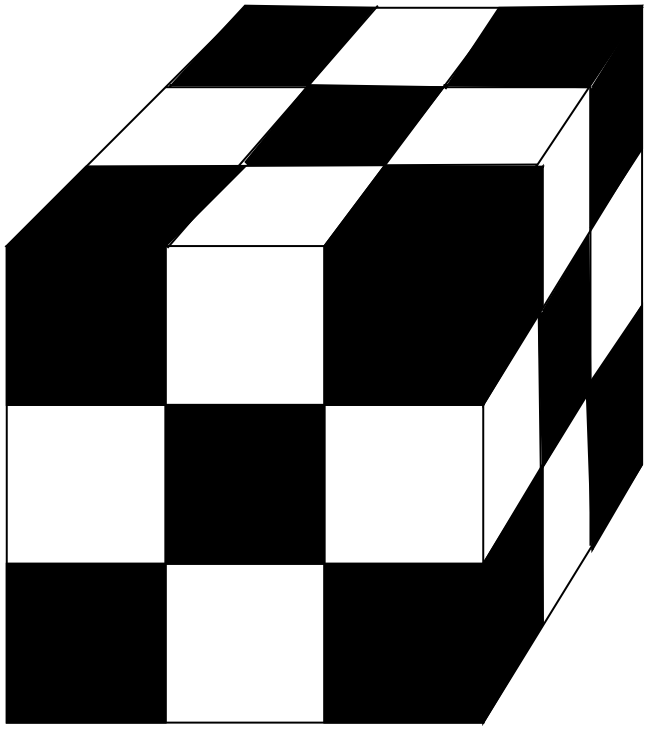
The Cube Puzzle



$$\mathrm{AG}(\mathrm{position} = \mathrm{center} \rightarrow \bigvee_{0 \leq i < 27} \neg \mathrm{visited}_i)$$

There are $3.46 \cdot 10^7$ reachable states out of $4.29 \cdot 10^9$

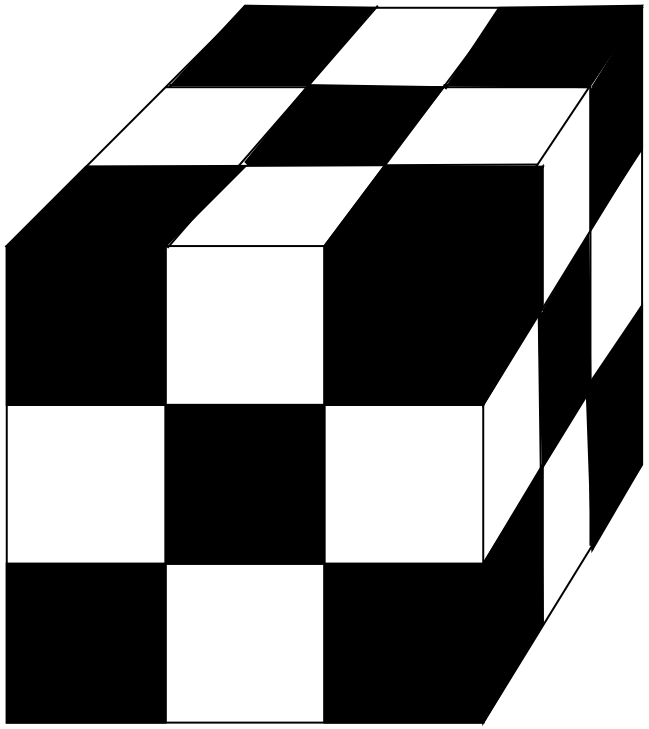
The Cube Puzzle



$$\mathsf{AG}(\mathsf{position} = \mathsf{center} \rightarrow \bigvee_{0 \leq i < 27} \neg \mathsf{visited}_i)$$

There are $3.46 \cdot 10^7$ reachable states out of $4.29 \cdot 10^9$

The Cube Puzzle

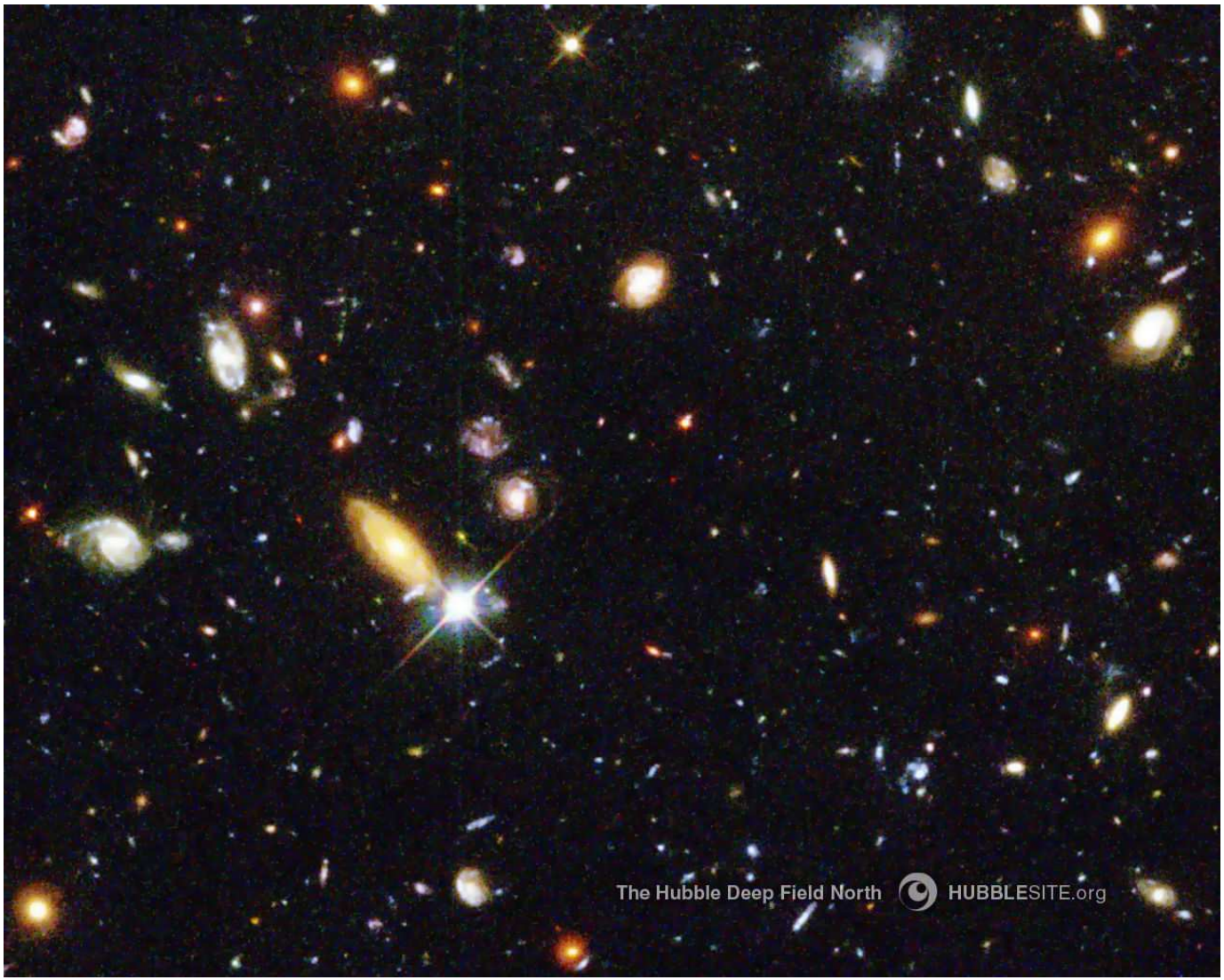


$$\mathrm{AG}(\mathrm{position} = \mathrm{center} \rightarrow \bigvee_{0 \leq i < 27} \neg \mathrm{visited}_i)$$

There are $3.46 \cdot 10^7$ reachable states out of $4.29 \cdot 10^9$

Glacier Gorge

Further Up



The Starry Sky Above Me...

- Galaxies in the (observable) universe: $2.5 \cdot 10^{11}$
- Stars in the Milky Way: $4 \cdot 10^{11}$
- Stars in the universe: 10^{23}
- Average number of neutrinos per cubic meter: $3.3 \cdot 10^{8}$
- Neutrinos in the universe: 10^{93} (wild guess)
- A small sequential circuit may have more than 10^{100} states ($10^{100} = 1$ googol)

The Starry Sky Above Me...

- Galaxies in the (observable) universe: $2.5 \cdot 10^{11}$
- Stars in the Milky Way: $4 \cdot 10^{11}$
- Stars in the universe: 10^{23}
- Average number of neutrinos per cubic meter: $3.3 \cdot 10^{8}$
- Neutrinos in the universe: 10^{93} (wild guess)
- A small sequential circuit may have more than 10^{100} states ($10^{100} = 1$ googol)

The Starry Sky Above Me...

- Galaxies in the (observable) universe: $2.5 \cdot 10^{11}$
- Stars in the Milky Way: $4 \cdot 10^{11}$
- Stars in the universe: 10^{23}
- Average number of neutrinos per cubic meter: $3.3 \cdot 10^{8}$
- Neutrinos in the universe: 10^{93} (wild guess)
- A small sequential circuit may have more than 10^{100} states ($10^{100} = 1$ googol)

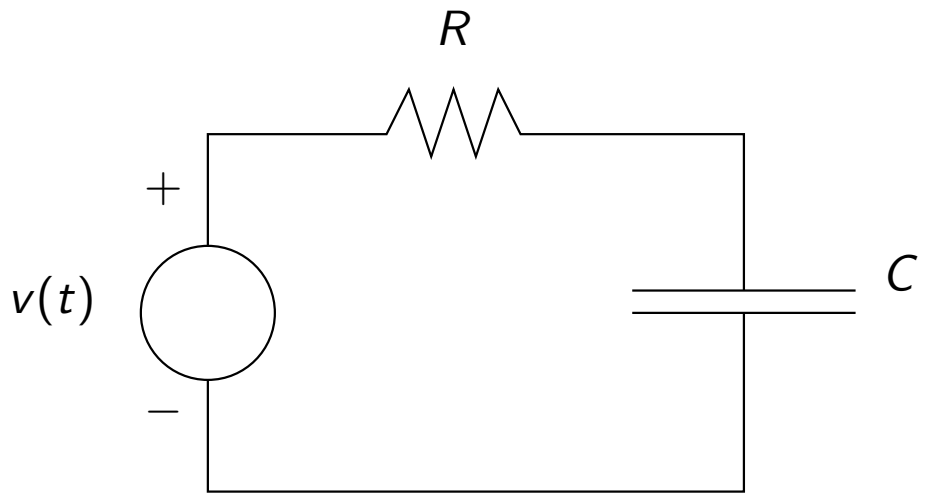
Combating State Explosion

- Symbolic Model Checking
  - Represent sets (of states and transitions) by their characteristic functions
    - $\{00, 01, 10\} \longrightarrow \neg x_1 \vee \neg x_2$
    - $x_1 \vee x_{100}$ represents $3 \cdot 2^{98}$ elements
  - BDDs and CNF popular representations
  - Do not enumerate the elements of the sets
    - Manipulate the characteristic functions instead
- Abstraction
  - Infer properties of the concrete model from the analysis of a simplified abstract model

Combating State Explosion

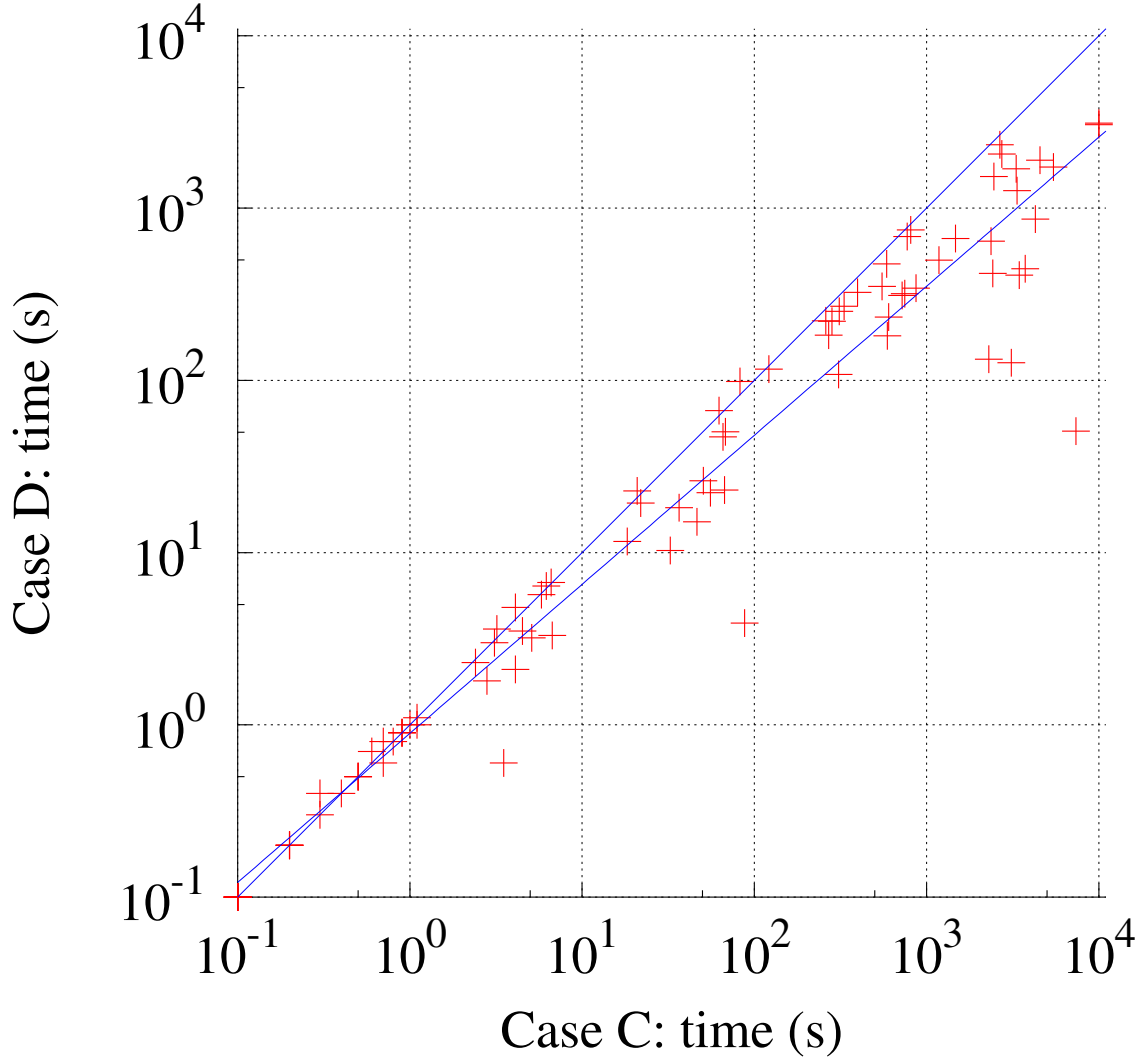
- Symbolic Model Checking
  - Represent sets (of states and transitions) by their characteristic functions
    - $\{00, 01, 10\} \longrightarrow \neg x_1 \vee \neg x_2$
    - $x_1 \vee x_{100}$ represents $3 \cdot 2^{98}$ elements
  - BDDs and CNF popular representations
  - Do not enumerate the elements of the sets
    - Manipulate the characteristic functions instead
- Abstraction
  - Infer properties of the concrete model from the analysis of a simplified abstract model

Familiar Abstractions




```
#include <iostream>
int main()
{
   std::cout << "Hello, world!" << std::endl;
   return 0;
}
```

Layered Abstractions



Peterson Mutex Algorithm

```
always @ (posedge clock) begin
    self = select;
    case (pc[self])
    L0: if (!pause) pc[self] = L1;
    L1: begin intr[self] = 1; pc[self] = L2; end
    L2: begin turn = ~self; pc[self] = L3; end
    L3: if (!intr[~self] || turn == self) pc[self] = L4;
    L4: if (!pause) pc[self] = L5; // critical
    L5: begin intr[self] = 0; pc[self] = L0; end
    endcase
end
```

Properties for Mutex Algorithm

- Mutual exclusion
  - $\mathsf{AG}\,\neg(\mathtt{pc}[0] = \mathtt{L4} \wedge \mathtt{pc}[1] = \mathtt{L4})$
- Absence of starvation
  - $\mathsf{AG}(\mathtt{pc}[0] = \mathtt{L1} \rightarrow \mathsf{AF}\,\mathtt{pc}[0] = \mathtt{L4})$
  - $\mathsf{AG}(\mathtt{pc}[1] = \mathtt{L1} \rightarrow \mathsf{AF}\,\mathtt{pc}[1] = \mathtt{L4})$
- Temporal logic operators
  - AG: invariably, AF: inevitably

Nondeterminism

```
always @ (posedge clock) begin
    self = select;
    case (pc[self])
    L0: if (!pause) pc[self] = L1;
    L1: begin intr[self] = 1; pc[self] = L2; end
    L2: begin turn = ~self; pc[self] = L3; end
    L3: if (!intr[~self] || turn == self) pc[self] = L4;
    L4: if (!pause) pc[self] = L5; // critical
    L5: begin intr[self] = 0; pc[self] = L0; end
    endcase
end
```

Fairness Conditions

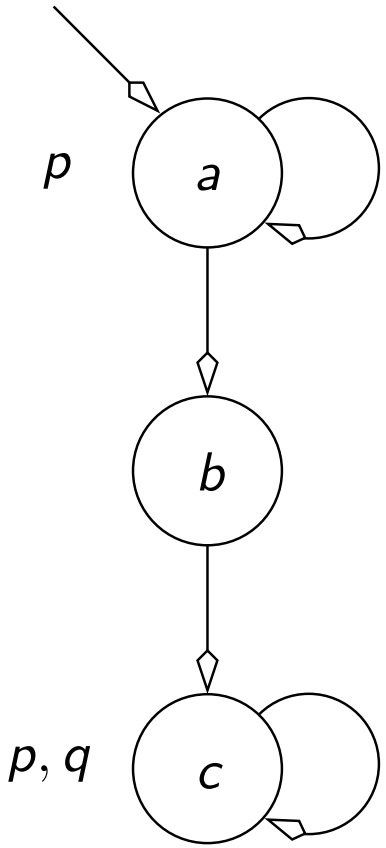
- Fair scheduling
  - $\mathsf{G}\,\mathsf{F}\,\texttt{self} = 1 \wedge \mathsf{G}\,\mathsf{F}\,\texttt{self} = 0$
- Neither process dwells forever in the critical section
  - $\mathsf{G}\,\mathsf{F}\,\texttt{pc}[0] \neq \texttt{L4} \wedge \mathsf{G}\,\mathsf{F}\,\texttt{pc}[1] \neq \texttt{L4}$
- Temporal logic operators
  - $\mathsf{G}\,\mathsf{F}$: infinitely often

Outline

Kripke Structures

- Finite transition systems without inputs
- $\langle S, T, S_0, A, L \rangle$
  - S: finite set of states
  - $T \subseteq S \times S$: transition relation
  - $S_0 \subseteq S$: set of initial states
  - A: set of atomic propositions
  - $L : S \to 2^A$: labeling function

Kripke Structure Example



- $K = \langle S, T, S_0, A, L \rangle$
- $S = \{a, b, c\}$
- $T = \{(a, a), (a, b), (b, c), (c, c)\}$
- $S_0 = \{a\}$
- $A = \{p, q\}$
- $L : \{a, b, c\} \rightarrow \{\{\}, \{p\}, \{q\}, \{p, q\}\}$
  - $L(a) = \{p\}$
  - $L(b) = \{\}$
  - $L(c) = \{p, q\}$

Composition

- Complex systems are composed of several modules
- Each module is described as a finite state machine (FSM)
- The overall Kripke structure is obtained as the product of the FSMs
  - State explosion!
- The product can be either synchronous or asynchronous (interleaving)

Specifications

- Properties are sets of behaviors
- Various specification mechanisms are in use: Temporal logics and automata are popular
- The examples we have seen are formulae of the temporal logic CTL
- Syntactic sugar often useful (e.g., PSL/Sugar)

Specifications

- Properties are sets of behaviors
- Various specification mechanisms are in use: Temporal logics and automata are popular
- The examples we have seen are formulae of the temporal logic CTL
- Syntactic sugar often useful (e.g., PSL/Sugar)

Specifications

- Properties are sets of behaviors
- Various specification mechanisms are in use: Temporal logics and automata are popular
- The examples we have seen are formulae of the temporal logic CTL
- Syntactic sugar often useful (e.g., PSL/Sugar)

Specifications

- Properties are sets of behaviors
- Various specification mechanisms are in use: Temporal logics and automata are popular
- The examples we have seen are formulae of the temporal logic CTL
- Syntactic sugar often useful (e.g., PSL/Sugar)

Temporal Logics

- Temporal logics add temporal operators and path quantifiers to standard (e.g., propositional) logics
- Temporal operators allow one to conveniently describe the order of occurrence of events and other statements involving time without explicitly mentioning time
- Expressiveness of propositional temporal logic in between those of propositional logic and predicate logic

Temporal Logics

- Temporal logics add temporal operators and path quantifiers to standard (e.g., propositional) logics
- Temporal operators allow one to conveniently describe the order of occurrence of events and other statements involving time without explicitly mentioning time
- Expressiveness of propositional temporal logic in between those of propositional logic and predicate logic

Operators and Quantifiers

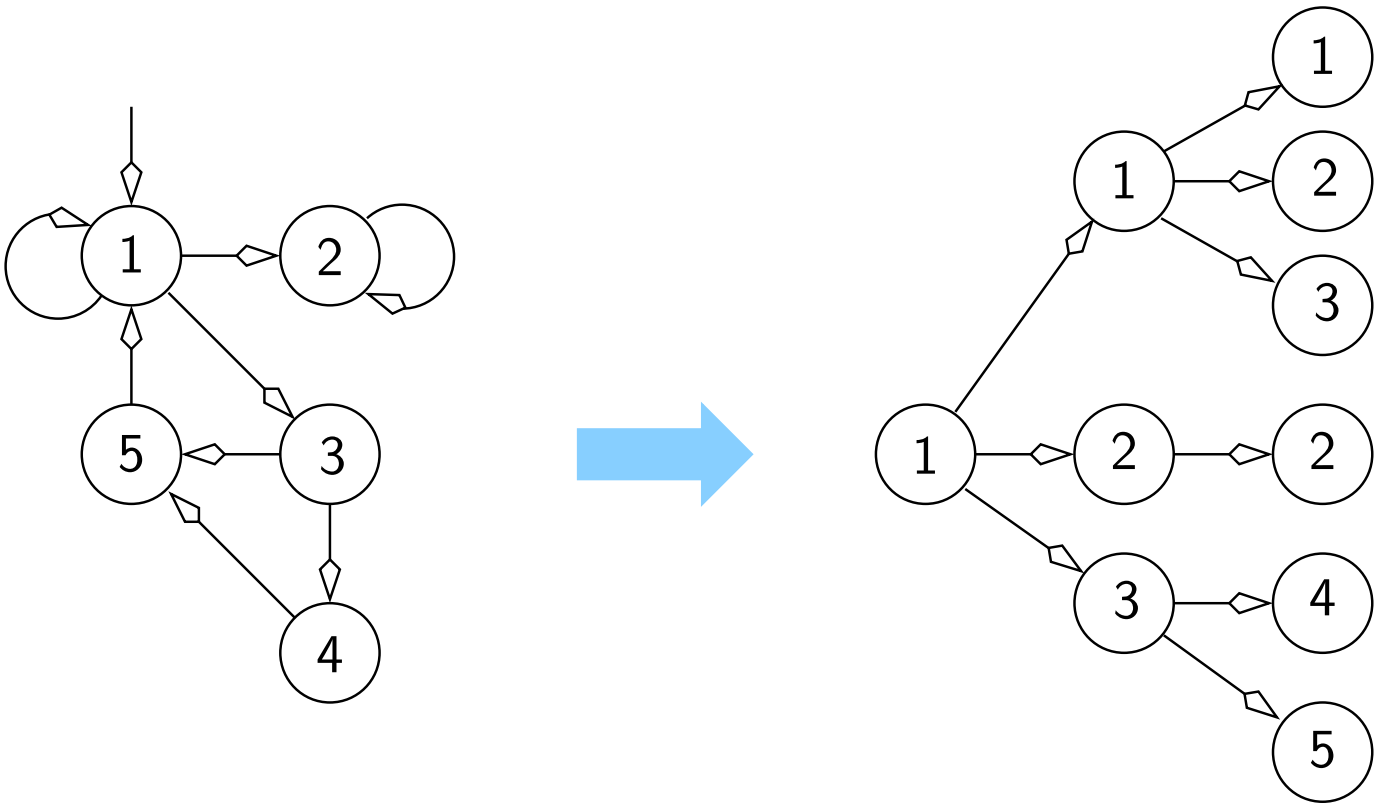
- G φ: φ holds globally ($\Box\varphi$)
- F φ: φ holds eventually ($\Diamond\varphi$)
- ψ U φ: ψ holds until φ holds
- ψ R φ: ψ releases φ
- X φ: φ holds at the next state ($\bigcirc\varphi$)
- E: along at least one path
- A: along all paths
- In CTL all temporal operators are immediately preceeded by a path quantifier

Operators and Quantifiers

- G φ: φ holds globally ($\Box\varphi$)
- F φ: φ holds eventually ($\Diamond\varphi$)
- ψ U φ: ψ holds until φ holds
- ψ R φ: ψ releases φ
- X φ: φ holds at the next state ($\bigcirc\varphi$)
- E: along at least one path
- A: along all paths
- In CTL all temporal operators are immediately preceeded by a path quantifier

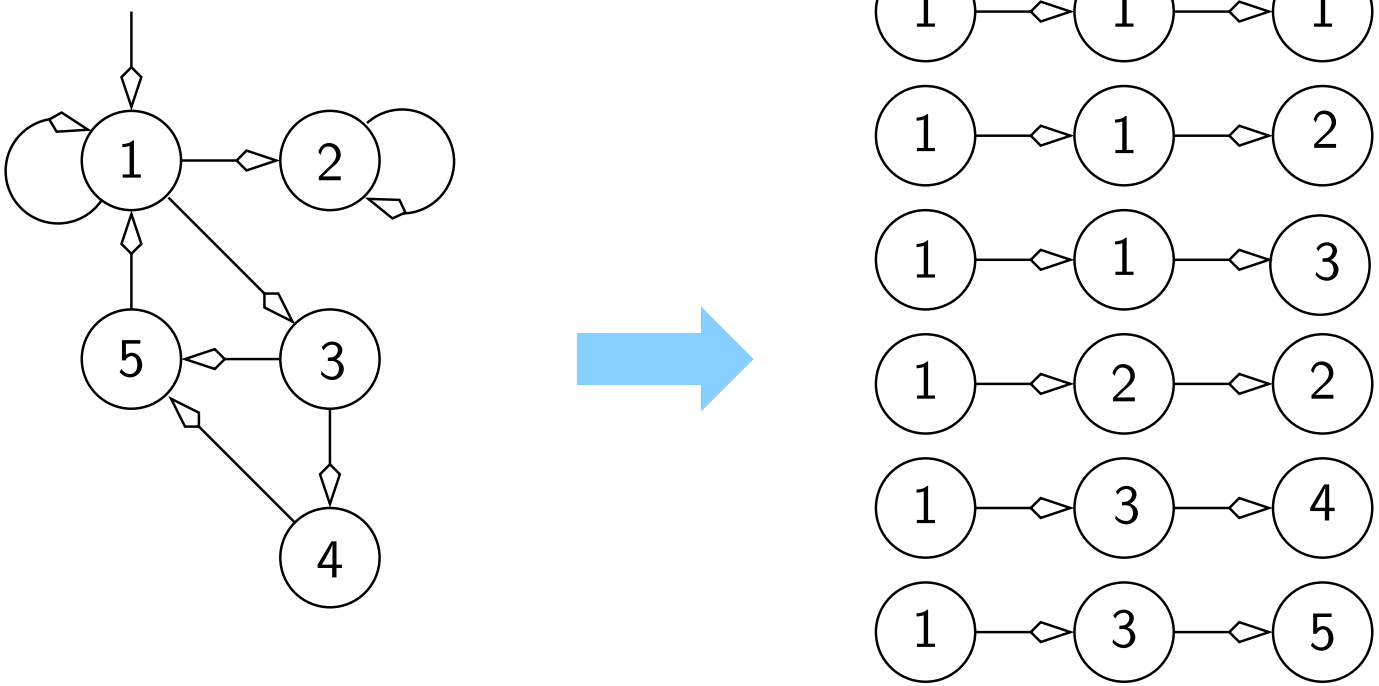
Branching Time

Branching time logics reason about computation trees



Linear Time

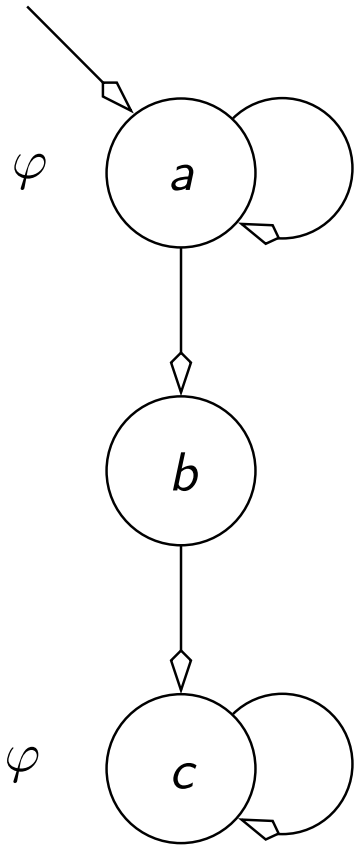
Linear time logics reason about sets of computation paths



Linear vs. Branching Time: Syntax

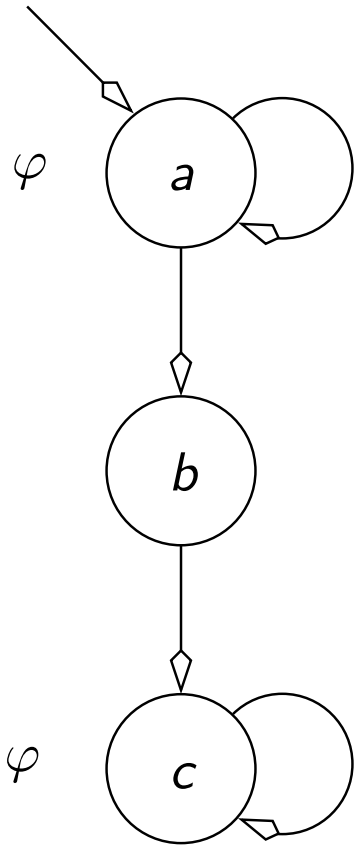
- A linear-time temporal logic formula can contain only one path quantifier at the beginning
  - $\mathsf{A}\,\mathsf{F}\,\mathsf{G}\,p$
  - $\mathsf{A}(\mathsf{F}\,p \rightarrow \mathsf{F}\,q)$
  - $\mathsf{E}\,\mathsf{G}\,\mathsf{F}\,p$ (existential linear-time formula)
- A branching-time formula may have multiple quantifiers
  - $\mathsf{AF}\,\mathsf{AG}\,p$
  - $\mathsf{AG}\,\mathsf{EF}\,p$
  - $\mathsf{EG}\,\mathsf{EF}\,p$

Linear vs. Branching Time



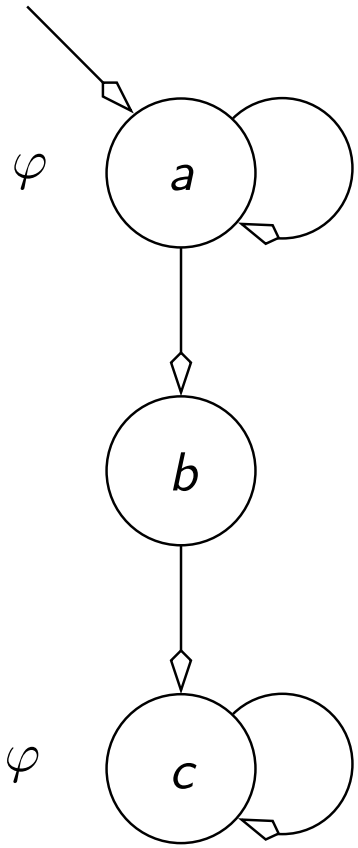
- A F G φ holds in this structure
  - Infinite paths eventually dwell in either a or c where φ holds
- AF AG φ does not hold in this structure
  - As long as a run dwells in a, it can always go to a state where φ does not hold
- No CTL formula exists that is equivalent to A F G φ

Linear vs. Branching Time



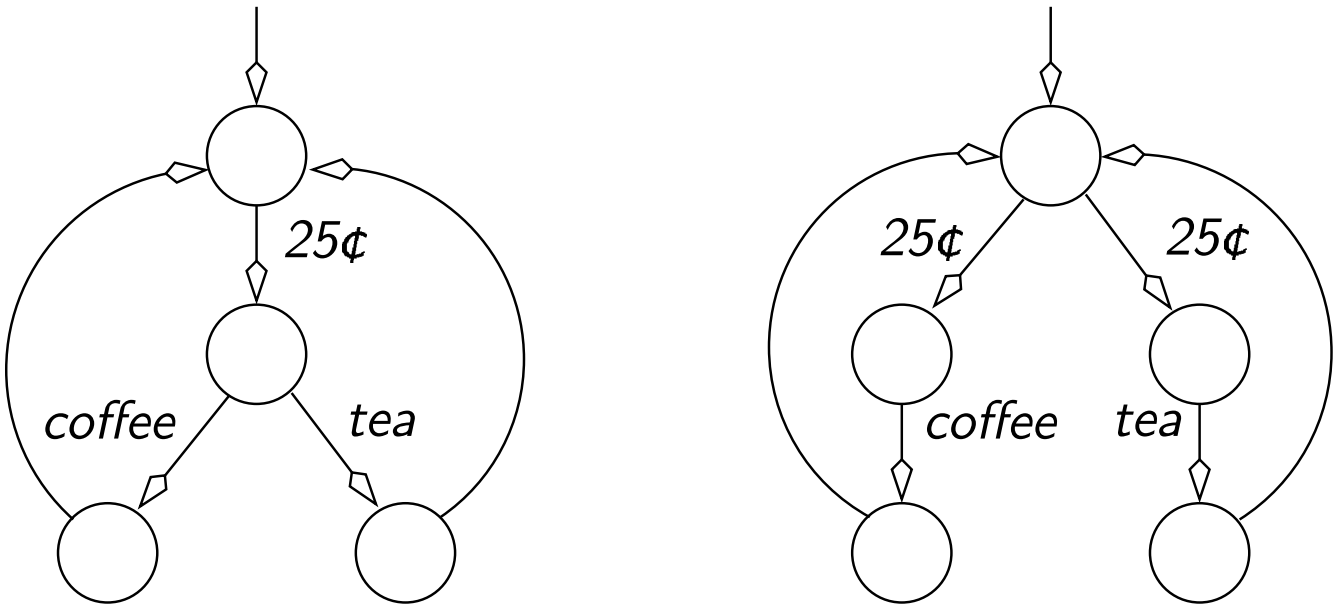
- A F G φ holds in this structure
  - Infinite paths eventually dwell in either a or c where φ holds
- AF AG φ does not hold in this structure
  - As long as a run dwells in a, it can always go to a state where φ does not hold
- No CTL formula exists that is equivalent to A F G φ

Linear vs. Branching Time



- A F G φ holds in this structure
  - Infinite paths eventually dwell in either a or c where φ holds
- AF AG φ does not hold in this structure
  - As long as a run dwells in a, it can always go to a state where φ does not hold
- No CTL formula exists that is equivalent to A F G φ

Linear vs. Branching Time



Linear-time properties cannot distinguish these two

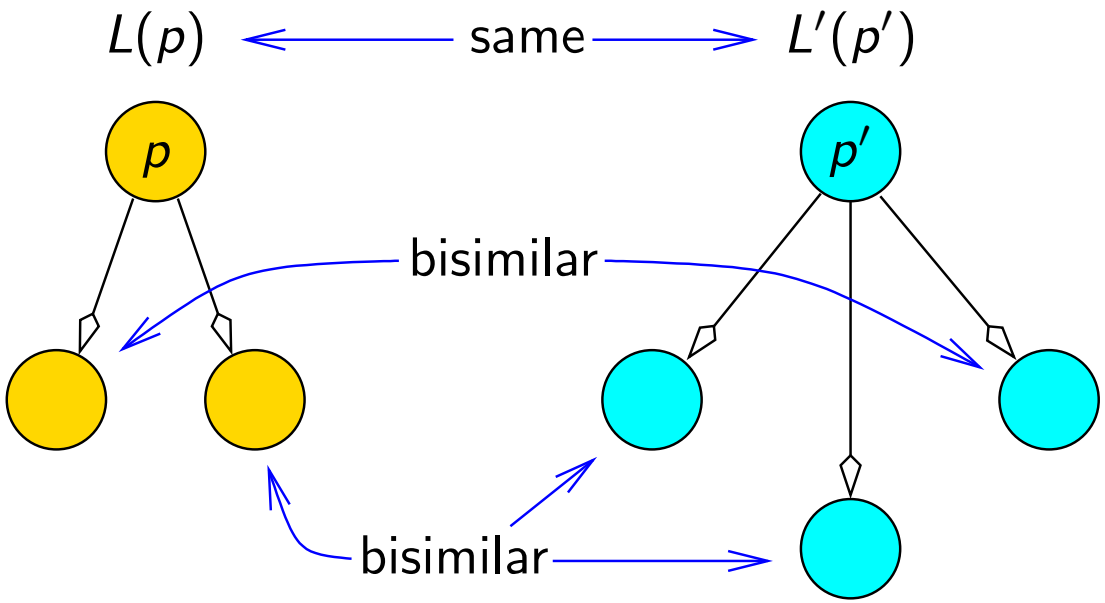
Trace Equivalence vs. Bisimilarity

- Linear time properties cannot distinguish two structures if they are trace (language) equivalent
- Branching time properties cannot distinguish two structures if they are bisimilar
- Bisimilarity is stronger than trace equivalence (see previous example)

Trace Equivalence vs. Bisimilarity

- Linear time properties cannot distinguish two structures if they are trace (language) equivalent
- Branching time properties cannot distinguish two structures if they are bisimilar
- Bisimilarity is stronger than trace equivalence (see previous example)

Bisimilar States



p and p' are bisimilar

Bisimulation Relation

- A relation B among the states of two Kripke structures K and K' with $A = A'$ is a bisimulation relation if $(p, p') \in B$ implies
  - $L(p) = L(p')$
  - $(p, q) \in T \rightarrow \exists q' . (p', q') \in T' \wedge (q, q') \in B$
  - $(p', q') \in T' \rightarrow \exists q . (p, q) \in T \wedge (q, q') \in B$

Bisimilar Structures

- Two Kripke structures K and K' are bisimulation equivalent ($K \equiv K'$) if there is a bisimulation relation between their states such that every initial state of one structure is bisimilar to some initial state of the other structure
- Two structures are bisimulation equivalent iff they satisfy the same branching-time properties
  - CTL properties suffice (Browne et al. [1988])

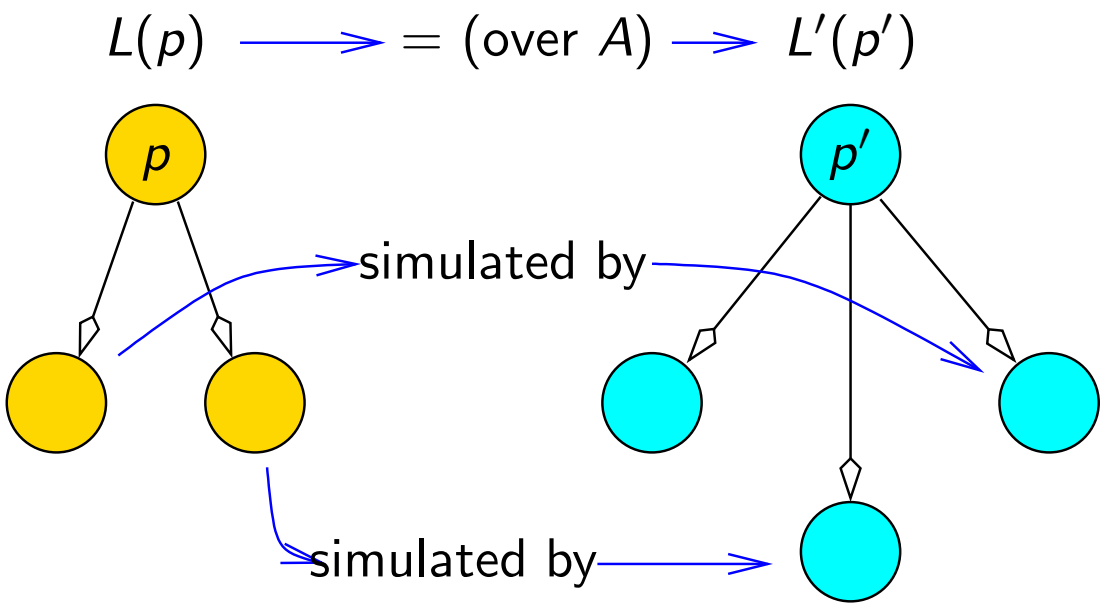
Bisimilar Structures

- Two Kripke structures K and K' are bisimulation equivalent ($K \equiv K'$) if there is a bisimulation relation between their states such that every initial state of one structure is bisimilar to some initial state of the other structure
- Two structures are bisimulation equivalent iff they satisfy the same branching-time properties
  - CTL properties suffice (Browne et al. [1988])

Bisimilar Structures

- Two Kripke structures K and K' are bisimulation equivalent ($K \equiv K'$) if there is a bisimulation relation between their states such that every initial state of one structure is bisimilar to some initial state of the other structure
- Two structures are bisimulation equivalent iff they satisfy the same branching-time properties
  - CTL properties suffice (Browne et al. [1988])

Similar States

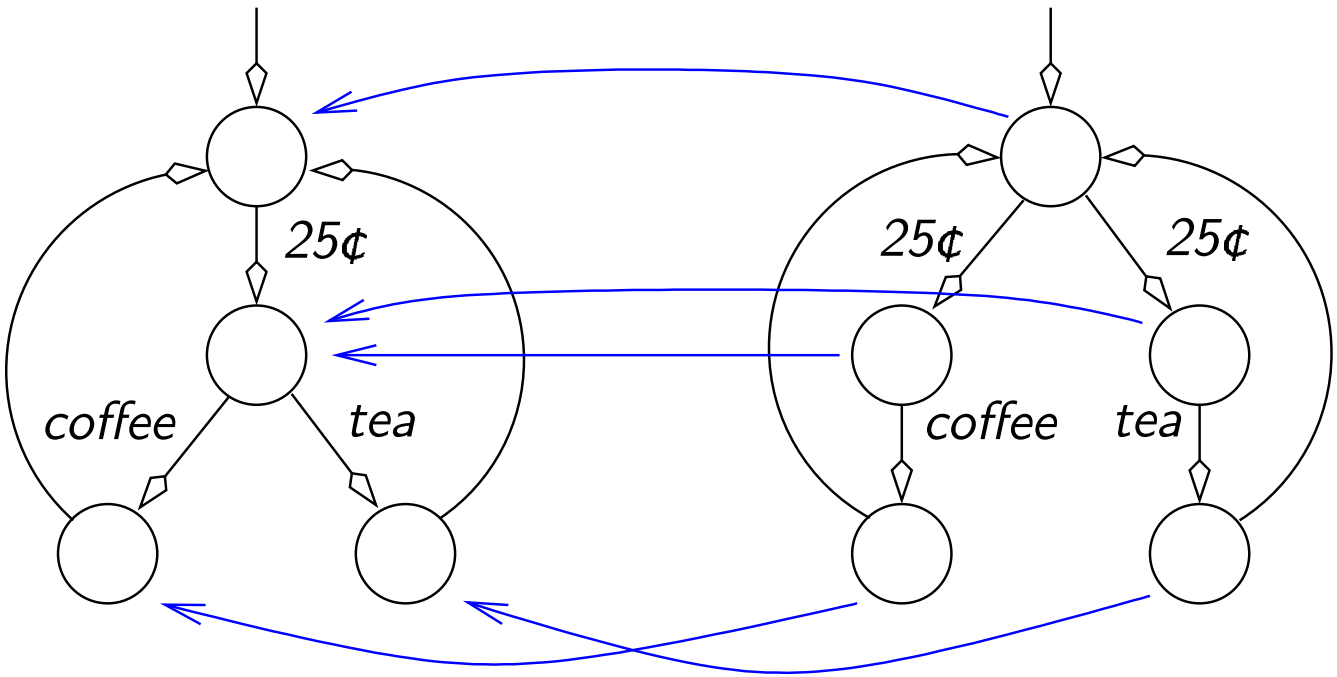


p is simulated by p'

Simulation Relation

- A relation Σ among the states of two Kripke structures K and K' with $A \subseteq A'$ is a simulation relation if $(p, p') \in \Sigma$ implies
  - $L(p) = L'(p') \cap A$
  - $(p, q) \in T \rightarrow \exists q' . (p', q') \in T' \wedge (q, q') \in \Sigma$
- We say that p is simulated by p'

Example of Simulation



Similar and Simulation-Equivalent Structures

- Kripke structure K' simulates structure K (written $K \preceq K'$) if there exists a simulation relation $\Sigma \subseteq S \times S'$ such that for every initial state s of K there is an initial state s' of K' such that $(s, s') \in \Sigma$
- If $K \preceq K'$ and φ is a universal branching-time formula over A, then $K' \models \varphi$ implies $K \models \varphi$.
- Two Kripke structures K and K' are simulation equivalent ($K \sim K'$) iff $K \preceq K'$ and $K' \preceq K$
- Two structures are simulation equivalent iff they satisfy the same branching-time universal properties

Similar and Simulation-Equivalent Structures

- Kripke structure K' simulates structure K (written $K \preceq K'$) if there exists a simulation relation $\Sigma \subseteq S \times S'$ such that for every initial state s of K there is an initial state s' of K' such that $(s, s') \in \Sigma$
- If $K \preceq K'$ and φ is a universal branching-time formula over A, then $K' \models \varphi$ implies $K \models \varphi$.
- Two Kripke structures K and K' are simulation equivalent ($K \sim K'$) iff $K \preceq K'$ and $K' \preceq K$
- Two structures are simulation equivalent iff they satisfy the same branching-time universal properties

Similar and Simulation-Equivalent Structures

- Kripke structure K' simulates structure K (written $K \preceq K'$) if there exists a simulation relation $\Sigma \subseteq S \times S'$ such that for every initial state s of K there is an initial state s' of K' such that $(s, s') \in \Sigma$
- If $K \preceq K'$ and φ is a universal branching-time formula over A, then $K' \models \varphi$ implies $K \models \varphi$.
- Two Kripke structures K and K' are simulation equivalent ($K \sim K'$) iff $K \preceq K'$ and $K' \preceq K$
- Two structures are simulation equivalent iff they satisfy the same branching-time universal properties

Similar and Simulation-Equivalent Structures

- Kripke structure K' simulates structure K (written $K \preceq K'$) if there exists a simulation relation $\Sigma \subseteq S \times S'$ such that for every initial state s of K there is an initial state s' of K' such that $(s, s') \in \Sigma$
- If $K \preceq K'$ and φ is a universal branching-time formula over A, then $K' \models \varphi$ implies $K \models \varphi$.
- Two Kripke structures K and K' are simulation equivalent ($K \sim K'$) iff $K \preceq K'$ and $K' \preceq K$
- Two structures are simulation equivalent iff they satisfy the same branching-time universal properties

Linear vs. Branching Time: Summary

- Branching time is more powerful, but also trickier
  - Two vending machine models
  - $\mathsf{A}\,\mathsf{F}\,\mathsf{G}\,\varphi$ vs. $\mathsf{AF}\,\mathsf{AG}\,\varphi$
- Linear time is more suitable for compositional verification and Bounded Model Checking
- Counterexample generation simpler for linear time

Linear vs. Branching Time: Summary

- Branching time is more powerful, but also trickier
  - Two vending machine models
  - $\mathsf{A}\,\mathsf{F}\,\mathsf{G}\,\varphi$ vs. $\mathsf{AF}\,\mathsf{AG}\,\varphi$
- Linear time is more suitable for compositional verification and Bounded Model Checking
- Counterexample generation simpler for linear time

Linear vs. Branching Time: Summary

- Branching time is more powerful, but also trickier
  - Two vending machine models
  - $\mathsf{A}\,\mathsf{F}\,\mathsf{G}\,\varphi$ vs. $\mathsf{AF}\,\mathsf{AG}\,\varphi$
- Linear time is more suitable for compositional verification and Bounded Model Checking
- Counterexample generation simpler for linear time

Safety

- A safety property describes something bad that should not happen
- $\mathsf{AG}\,\neg(\texttt{grant}_0 \wedge \texttt{grant}_1)$: requestors 0 and 1 should not be granted access to the shared resource simultaneously
- $\mathsf{AG}\,\neg(\texttt{door} = \texttt{open} \wedge \texttt{engine} = \texttt{running})$

Liveness

- A liveness property describes something good that should happen
- AG($\texttt{req} \rightarrow \mathsf{F}\,\texttt{ack}$): requests should be acknowledged

Safety or Liveness?

- The definitions given so far may be confusing
- $\mathsf{AG}(\texttt{command} = \texttt{stop} \rightarrow \mathsf{X}\,\texttt{state} = \texttt{halt})$
- $\mathsf{AG}(\texttt{command} = \texttt{stop} \rightarrow \mathsf{F}\,\texttt{state} = \texttt{halt})$
- $\mathsf{AG}(\texttt{req} \rightarrow \mathsf{F}\,\texttt{ack})$

Safety in Linear Time

- A linear-time property is a set of (linear) traces or infinite sequences over the atomic propositions
- A property φ is a safety property if every trace not in φ has a prefix that cannot be extended to a trace in φ
- Intuitively, the prefix includes the bad event that causes the property to fail
- $\mathrm{AG}(p \rightarrow \mathrm{X}\, q)$ is a safety property
  - A counterexample includes a state where p holds followed by a state where $\neg q$ holds
  - The prefix that includes these two states cannot be extended to an infinite path satisfying the property

Safety in Linear Time

- A linear-time property is a set of (linear) traces or infinite sequences over the atomic propositions
- A property φ is a safety property if every trace not in φ has a prefix that cannot be extended to a trace in φ
- Intuitively, the prefix includes the bad event that causes the property to fail
- $\mathsf{AG}(p \rightarrow \mathsf{X}\, q)$ is a safety property
  - A counterexample includes a state where p holds followed by a state where $\neg q$ holds
  - The prefix that includes these two states cannot be extended to an infinite path satisfying the property

Liveness in Linear Time

- A linear-time property φ is a liveness property if every finite sequence over the atomic propositions can be extended to an infinite sequence in φ
- Intuitively, finite prefixes do not affect the fulfillment of the eventualities that characterize liveness properties
- $\mathrm{AG}(p \rightarrow \mathrm{F}\, q)$ is a liveness property
  - One can add a state where q holds to any finite prefix

Liveness in Linear Time

- A linear-time property φ is a liveness property if every finite sequence over the atomic propositions can be extended to an infinite sequence in φ
- Intuitively, finite prefixes do not affect the fulfillment of the eventualities that characterize liveness properties
- $\mathsf{AG}(p \rightarrow \mathsf{F}\, q)$ is a liveness property
  - One can add a state where q holds to any finite prefix

Neither Safe Nor Live

- $\mathsf{A}(p\,\mathsf{U}\,q)$ is neither a safety property nor a liveness property
  - A trace satisfying $\mathsf{G}(p \wedge \neg q)$ is a counterexample with no prefix that cannot be extended
  - A trace satisfying $\neg q\,\mathsf{U}(\neg p \neg q)$ is a counterexample with a finite prefix that cannot be extended
- Every property can be written as the intersection of a safety property and a liveness property
  - $\mathsf{A}(p\,\mathsf{U}\,q) = \mathsf{A}((q\,\mathsf{R}(p \vee q)) \wedge \mathsf{F}\,q)$

Neither Safe Nor Live

- $\mathsf{A}(p\,\mathsf{U}\,q)$ is neither a safety property nor a liveness property
  - A trace satisfying $\mathsf{G}(p \wedge \neg q)$ is a counterexample with no prefix that cannot be extended
  - A trace satisfying $\neg q\,\mathsf{U}(\neg p \neg q)$ is a counterexample with a finite prefix that cannot be extended
- Every property can be written as the intersection of a safety property and a liveness property
  - $\mathsf{A}(p\,\mathsf{U}\,q) = \mathsf{A}((q\,\mathsf{R}(p \vee q)) \wedge \mathsf{F}\,q)$

Classification of Properties

- Time structure: branching, linear
- Bisimilarity, trace equivalence
- Safety, liveness
- Existential, universal
- Tense: future, past
  - X versus Y (neXt vs. Yesterday)
  - See (Laroussinie and Schnoebelen [2000])

Outline

CTL*

- CTL* is a powerful branching-time temporal logic
- We use a subset of the operators (X and U) and the E quantifier to define the syntax
- The remaining operators are defined as abbreviations
- We need both state formulae and path formulae to recursively define the logic
  - The state formulae give CTL*
  - Path formulae can only appear as subformulae

CTL*

- CTL* is a powerful branching-time temporal logic
- We use a subset of the operators (X and U) and the E quantifier to define the syntax
- The remaining operators are defined as abbreviations
- We need both state formulae and path formulae to recursively define the logic
  - The state formulae give CTL*
  - Path formulae can only appear as subformulae

CTL*

- CTL* is a powerful branching-time temporal logic
- We use a subset of the operators (X and U) and the E quantifier to define the syntax
- The remaining operators are defined as abbreviations
- We need both state formulae and path formulae to recursively define the logic
  - The state formulae give CTL*
  - Path formulae can only appear as subformulae

CTL*

- CTL* is a powerful branching-time temporal logic
- We use a subset of the operators (X and U) and the E quantifier to define the syntax
- The remaining operators are defined as abbreviations
- We need both state formulae and path formulae to recursively define the logic
  - The state formulae give CTL*
  - Path formulae can only appear as subformulae

CTL* Syntax

- An atomic proposition is a state formula
- A state formula is also a path formula
- If φ and ψ are state formulae, so are $\neg\varphi$ and $\varphi \wedge \psi$
- If φ is a path formula, $\mathsf{E}\,\varphi$ is a state formula
- If φ and ψ are path formulae, so are $\neg\varphi$ and $\varphi \wedge \psi$
- If φ and ψ are path formulae, so are $\mathsf{X}\,\varphi$ and $\psi\,\mathsf{U}\,\varphi$

CTL* Abbreviations

- $\varphi \vee \psi = \neg(\neg\varphi \wedge \neg\psi)$
- $\mathsf{true} = \varphi \vee \neg\varphi$
- $\mathsf{false} = \varphi \wedge \neg\varphi$
- $\psi \,\mathsf{R}\, \varphi = \neg(\neg\psi \,\mathsf{U}\, \neg\varphi)$
- $\mathsf{F}\,\varphi = \mathsf{true}\,\mathsf{U}\,\varphi$
- $\mathsf{G}\,\varphi = \mathsf{false}\,\mathsf{R}\,\varphi$
- $\mathsf{A}\,\varphi = \neg\,\mathsf{E}\,\neg\varphi$

"Pushing Down" Negations

- $\neg \mathsf{A}\,\mathsf{G}\,\mathsf{F}\,\varphi = \mathsf{E}\,\mathsf{F}\,\mathsf{G}\,\neg\varphi$
- Negations can be pushed "down" toward the atomic propositions
- The basic rules are (beside DeMorgan)
  - $\psi\,\mathsf{R}\,\varphi = \neg(\neg\psi\,\mathsf{U}\,\neg\varphi)$
  - $\mathsf{A}\,\varphi = \neg\,\mathsf{E}\,\neg\varphi$
- From the first we get $\mathsf{G}\,\varphi = \neg\,\mathsf{F}\,\neg\varphi$

CTL* Semantics

- The semantics of CTL* formulae are defined with respect to a Kripke structure K
- If formula φ holds of state s (path π) of K, we write $K, s \models \varphi$ ($K, \pi \models \varphi$)
- The double turnstile is read "models"
- K is omitted when no ambiguity arises
- π^i is π without the first i states

CTL* Semantics

- The semantics of CTL* formulae are defined with respect to a Kripke structure K
- If formula φ holds of state s (path π) of K, we write $K, s \models \varphi$ ($K, \pi \models \varphi$)
- The double turnstile is read "models"
- K is omitted when no ambiguity arises
- π^i is π without the first i states

CTL* Semantics

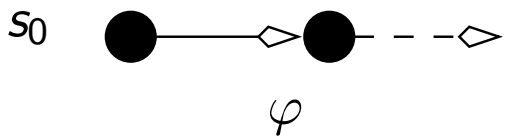
- The semantics of CTL* formulae are defined with respect to a Kripke structure K
- If formula φ holds of state s (path π) of K, we write $K, s \models \varphi$ ($K, \pi \models \varphi$)
- The double turnstile is read "models"
- K is omitted when no ambiguity arises
- π^i is π without the first i states

CTL* Semantics

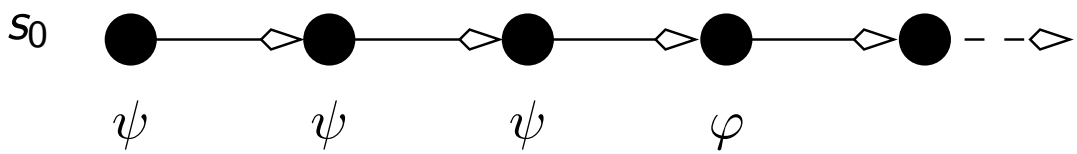
- $s \models p, p \in A$ iff $p \in L(s)$
- $s \models \neg\varphi$ iff $s \not\models \varphi$
- $s \models \varphi \wedge \psi$ iff $s \models \varphi$ and $s \models \psi$
- $s \models \mathsf{E}\,\varphi$ iff $\exists\pi$ from s such that $\pi \models \varphi$
- $\pi \models \neg\varphi$ iff $\pi \not\models \varphi$
- $\pi \models \varphi \wedge \psi$ iff $\pi \models \varphi$ and $\pi \models \psi$
- $\pi \models \mathsf{X}\,\varphi$ iff $\pi^1 \models \varphi$
- $\pi \models \psi\,\mathsf{U}\,\varphi$ iff $\exists i \geq 0\,.\,\pi^i \models \varphi$ and $0 \leq j < i \rightarrow \pi^j \models \psi$

Semantics of X and U

$\mathsf{X}\,\varphi$



$\psi\,\mathsf{U}\,\varphi$



CTL

- Computational Tree Logic is a branching time fragment of CTL*
- In CTL every temporal operator must be immediately preceded by a path quantifier
  - $\mathsf{AF}\,\mathsf{AG}\,\varphi$ is a CTL formula
  - $\mathsf{A}\,\mathsf{F}\,\mathsf{G}\,\varphi$ is not a CTL formula
- Model checking easy relative to CTL*

CTL

- Computational Tree Logic is a branching time fragment of CTL*
- In CTL every temporal operator must be immediately preceded by a path quantifier
  - $\mathsf{AF}\,\mathsf{AG}\,\varphi$ is a CTL formula
  - $\mathsf{A}\,\mathsf{F}\,\mathsf{G}\,\varphi$ is not a CTL formula
- Model checking easy relative to CTL*

LTL

- Linear Temporal Logic is a linear-time fragment of CTL*
- In LTL there can be only one quantifier at the beginning of the formula
- The quantifier is usually A, in which case it is usually omitted
  - F G φ means A F G φ

LTL

- Linear Temporal Logic is a linear-time fragment of CTL*
- In LTL there can be only one quantifier at the beginning of the formula
- The quantifier is usually A, in which case it is usually omitted
  - $\mathsf{F}\,\mathsf{G}\,\varphi$ means $\mathsf{A}\,\mathsf{F}\,\mathsf{G}\,\varphi$

Omega-Automata

- Properties can be described by automata that take the computation of the system as input and either accept it or reject it
- For computations that finish regular automata suffice
- For non-terminating computations we need ω-automata

Omega-Automata

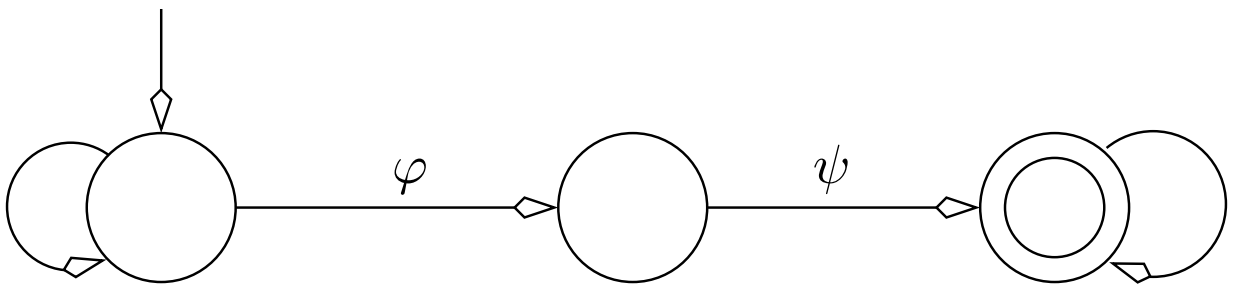
- Properties can be described by automata that take the computation of the system as input and either accept it or reject it
- For computations that finish regular automata suffice
- For non-terminating computations we need ω-automata

Omega-Automata

- Properties can be described by automata that take the computation of the system as input and either accept it or reject it
- For computations that finish regular automata suffice
- For non-terminating computations we need ω-automata

Automaton for Safety Property

- $\mathsf{AG}(\varphi \rightarrow \mathsf{X}\,\neg\psi)$ is negated to get the formula $\mathsf{EF}(\varphi \wedge \mathsf{X}\,\psi)$



Why Omega Automata?

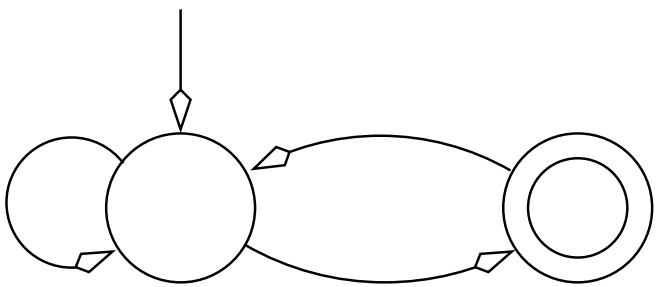
- For safety properties we can "stretch" regular automata because it suffices to find the non-extensible prefixes of the counterexamples
- For liveness properties we need to address the fact that the computations we consider do not terminate

Büchi Automata

- Büchi automata are similar to (nondeterministic) regular automata except for the acceptance conditions
- A Büchi automaton accepts an infinite sequence if there is a run of it that goes through an accepting state infinitely often

Büchi Automaton to Model Check $\mathsf{A}\,\mathsf{F}\,\mathsf{G}\,\varphi$

- We negate the property to get $\mathsf{E}\,\mathsf{G}\,\mathsf{F}\,\neg\varphi$

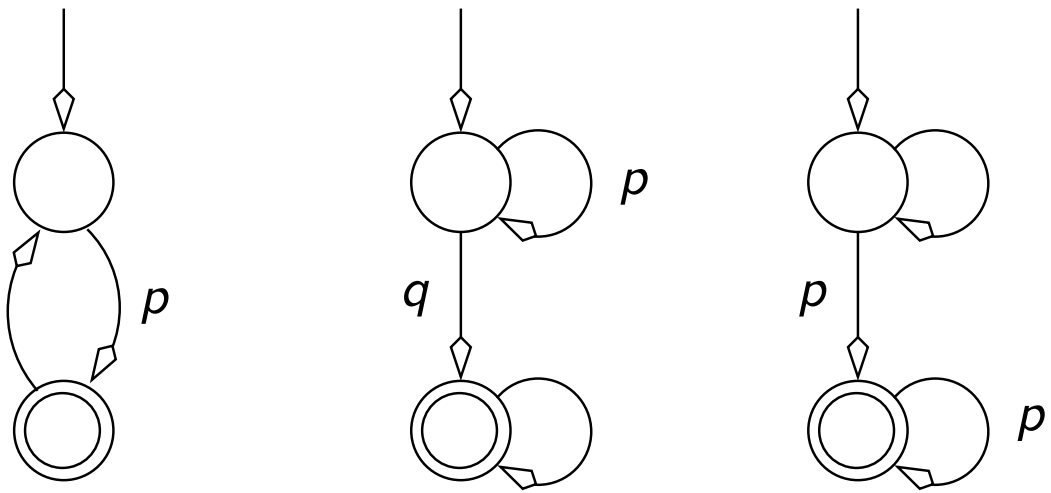


$\neg\varphi$

- A computation accepted by the automaton is a counterexample to $\mathsf{A}\,\mathsf{F}\,\mathsf{G}\,\varphi$

Omega-Automata

- Omega-automata describe linear-time properties
- They are more expressive than LTL



Fairness

- Fairness constraints are used to
  - Model features of the environment
  - Mitigate the effects of simplifications of the model
- They instruct the model checker to disregard certain (unfair) computations

Modeling the Environment

- Property: The combination lock will open unless we keep making mistakes
- Fairness constraint: provided we never give up
- This constraint implies that the knob is turned infinitely often

Modeling the Environment

- Property: The combination lock will open unless we keep making mistakes
- Fairness constraint: provided we never give up
- This constraint implies that the knob is turned infinitely often

Modeling the Environment

- Property: The combination lock will open unless we keep making mistakes
- Fairness constraint: provided we never give up
- This constraint implies that the knob is turned infinitely often

Mitigating Abstraction

- Property: All requests are eventually acknowledged
- Fairness constraints: provided all grantees eventually relinquish the shared resource
- These constraints imply that each requestor is not using the shared resource infinitely often

Mitigating Abstraction

- Property: All requests are eventually acknowledged
- Fairness constraints: provided all grantees eventually relinquish the shared resource
- These constraints imply that each requestor is not using the shared resource infinitely often

Mitigating Abstraction

- Property: All requests are eventually acknowledged
- Fairness constraints: provided all grantees eventually relinquish the shared resource
- These constraints imply that each requestor is not using the shared resource infinitely often

Büchi Fairness Constraints

- A Büchi fairness constraint is a set of states that a fair run must intersect infinitely often
- LTL can express fairness constraints
  - $\mathsf{A}(\mathsf{G}\,\mathsf{F}\,\psi \rightarrow \varphi)$ says that φ must hold only along paths where ψ holds infinitely often
- CTL cannot express fairness constraints
  - They must be specified separately

Outline

CTL Model Checking

- Given a Kripke structure K and a CTL formula φ, we want to determine whether $K \models \varphi$
- We find $[\![\varphi]\!]_K$, the set of all states s such that $K, s \models \varphi$
- We then check whether $S_0 \subseteq [\![\varphi]\!]_K$
- When no confusion arises we write simply φ for $[\![\varphi]\!]_K$

CTL Model Checking

- Given a Kripke structure K and a CTL formula φ, we want to determine whether $K \models \varphi$
- We find $[\![\varphi]\!]_K$, the set of all states s such that $K, s \models \varphi$
- We then check whether $S_0 \subseteq [\![\varphi]\!]_K$
- When no confusion arises we write simply φ for $[\![\varphi]\!]_K$

CTL Model Checking

- Given a Kripke structure K and a CTL formula φ, we want to determine whether $K \models \varphi$
- We find $[\![\varphi]\!]_K$, the set of all states s such that $K, s \models \varphi$
- We then check whether $S_0 \subseteq [\![\varphi]\!]_K$
- When no confusion arises we write simply φ for $[\![\varphi]\!]_K$

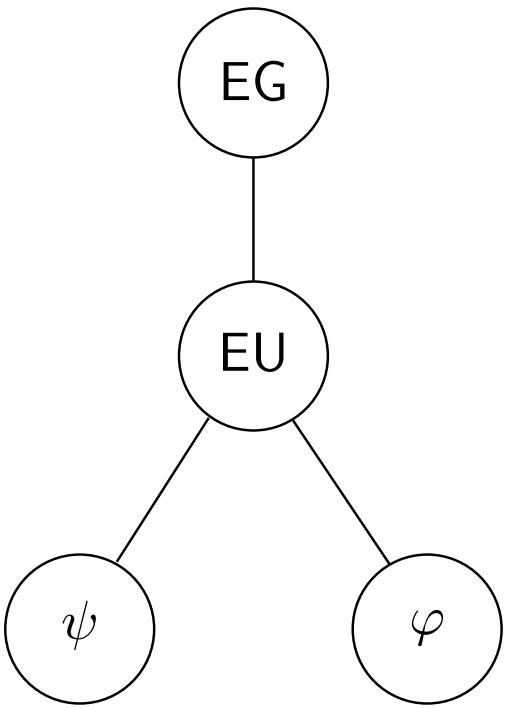
CTL Model Checking

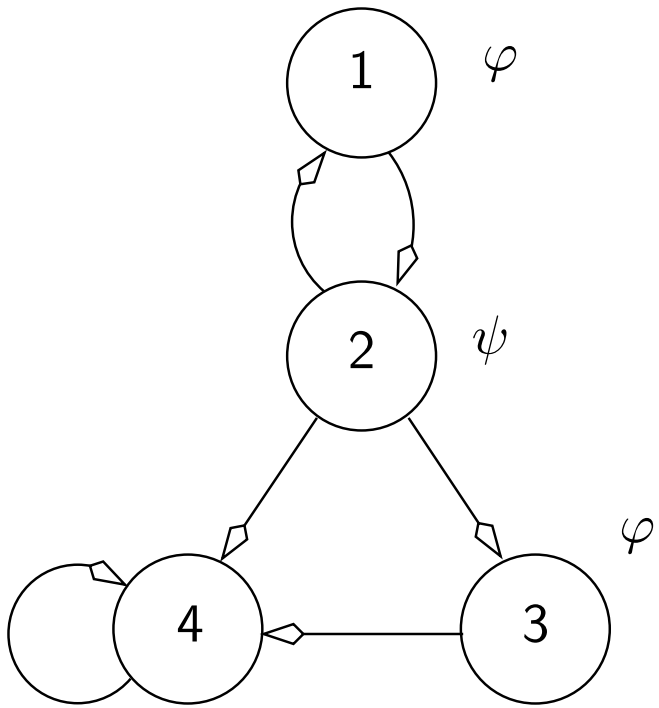
- Given a Kripke structure K and a CTL formula φ, we want to determine whether $K \models \varphi$
- We find $[\![\varphi]\!]_K$, the set of all states s such that $K, s \models \varphi$
- We then check whether $S_0 \subseteq [\![\varphi]\!]_K$
- When no confusion arises we write simply φ for $[\![\varphi]\!]_K$

Computing Satisfying Sets

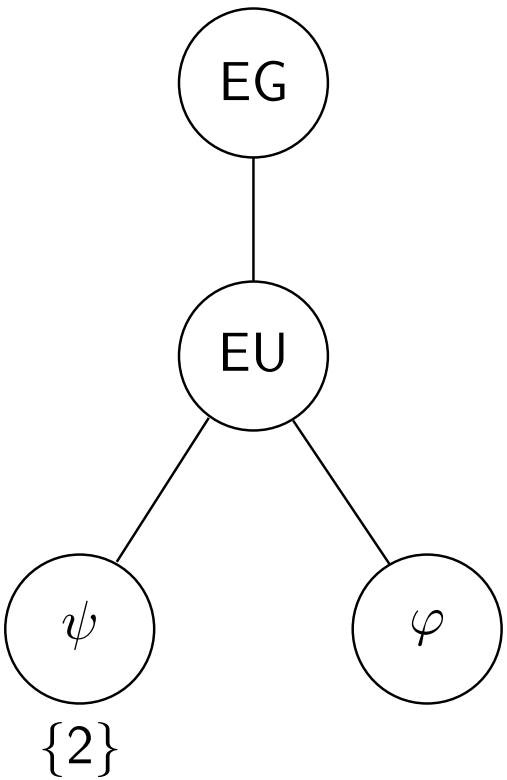
- Work bottom-up on the parse graph of the CTL formula
- Annotate every node with the satisfying set of the subformula rooted at the node
- The computation at each node of the parse graph depends on its label

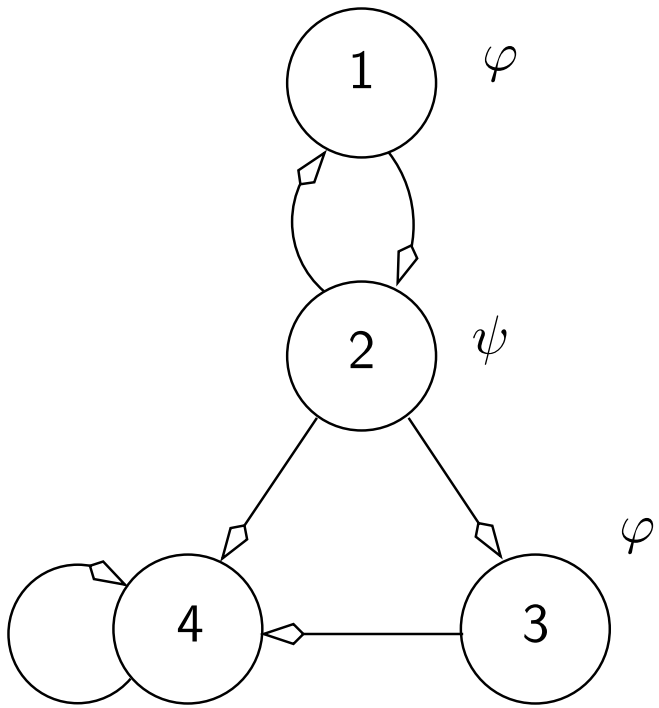
Satisfying Sets for $\mathsf{EG}\,\mathsf{E}\,\psi\,\mathsf{U}\,\varphi$



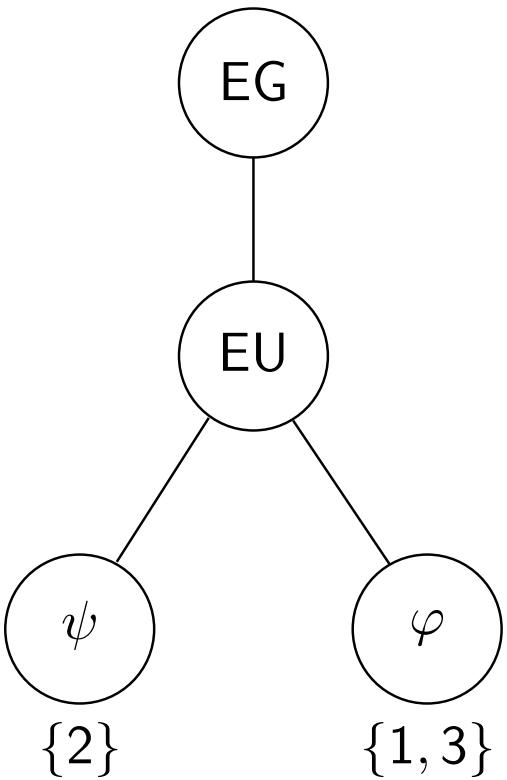


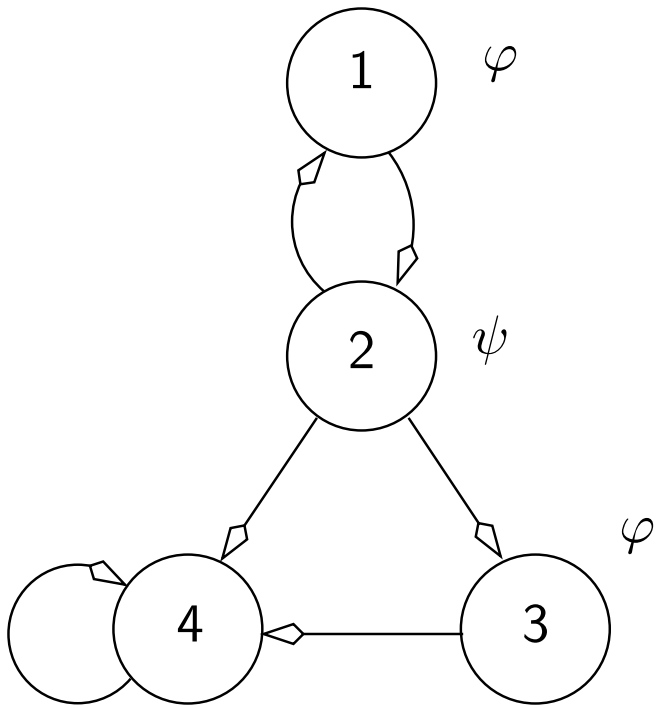
Satisfying Sets for $\mathsf{EG}\,\mathsf{E}\,\psi\,\mathsf{U}\,\varphi$



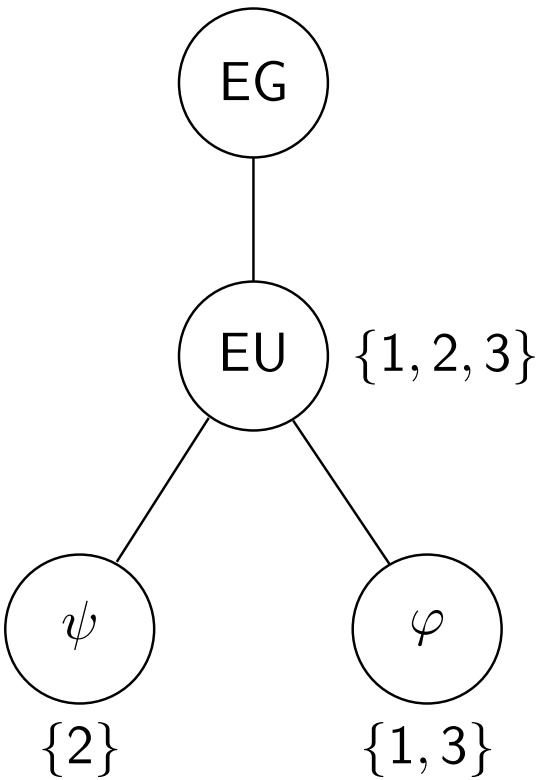


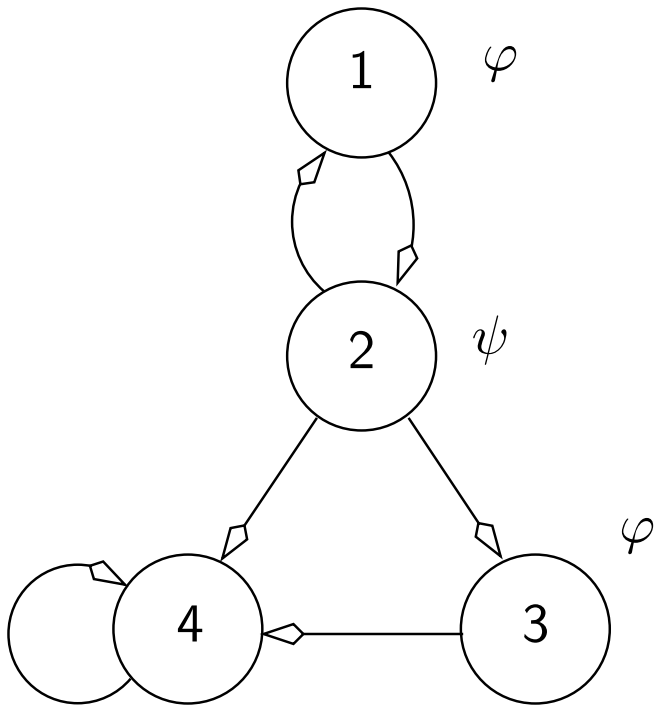
Satisfying Sets for $\mathsf{EG}\,\mathsf{E}\,\psi\,\mathsf{U}\,\varphi$



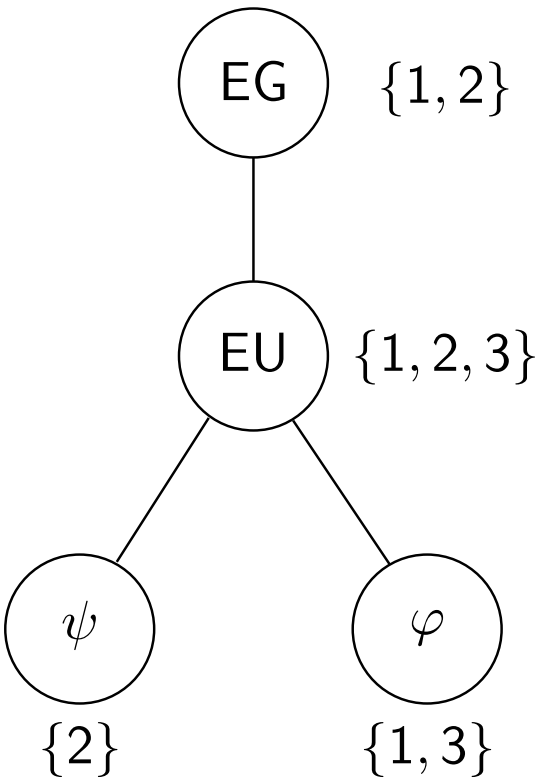


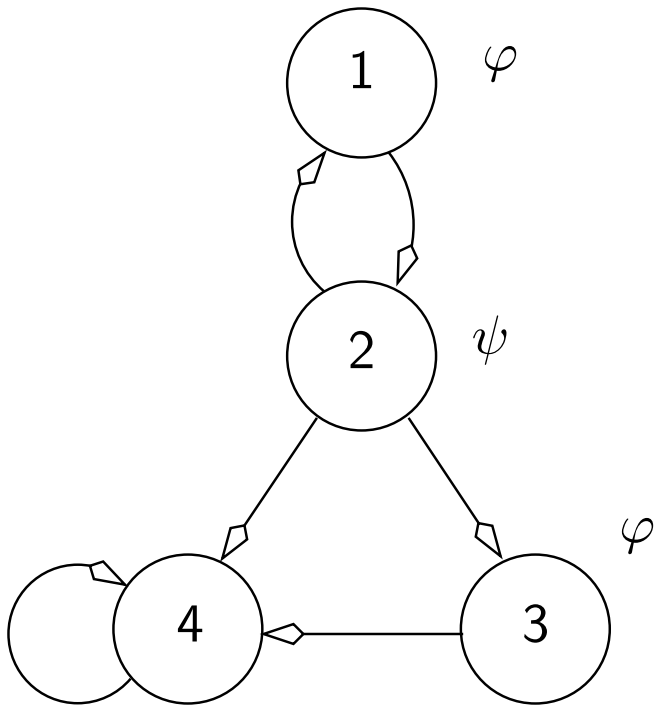
Satisfying Sets for $\mathsf{EG}\,\mathsf{E}\,\psi\,\mathsf{U}\,\varphi$





Satisfying Sets for $\mathsf{EG}\,\mathsf{E}\,\psi\,\mathsf{U}\,\varphi$





Action for Each Label

- Atomic proposition p: return set of states with p in their labels
- $\neg\varphi$: return the complement of $[\![\varphi]\!]_K$
- $\varphi \wedge \psi$: return $[\![\varphi]\!]_K \cap [\![\psi]\!]_K$
- $\mathsf{EX}\,\varphi$: return the set of predecessors in K of the states in $[\![\varphi]\!]_K$
- $\mathsf{E}\,\psi\,\mathsf{U}\,\varphi$: return the set of states with paths to states in $[\![\varphi]\!]_K$ that are entirely contained in $[\![\psi]\!]_K$ (except possibly for the last state of each path)
- $\mathsf{EG}\,\varphi$: return the set of states on infinite paths entirely contained in $[\![\varphi]\!]_K$

Action for Each Label

- Atomic proposition p: return set of states with p in their labels
- $\neg\varphi$: return the complement of $[\![\varphi]\!]_K$
- $\varphi \wedge \psi$: return $[\![\varphi]\!]_K \cap [\![\psi]\!]_K$
- $\mathsf{EX}\,\varphi$: return the set of predecessors in K of the states in $[\![\varphi]\!]_K$
- $\mathsf{E}\,\psi\,\mathsf{U}\,\varphi$: return the set of states with paths to states in $[\![\varphi]\!]_K$ that are entirely contained in $[\![\psi]\!]_K$ (except possibly for the last state of each path)
- $\mathsf{EG}\,\varphi$: return the set of states on infinite paths entirely contained in $[\![\varphi]\!]_K$

Action for Each Label

- Atomic proposition p: return set of states with p in their labels
- $\neg\varphi$: return the complement of $[\![\varphi]\!]_K$
- $\varphi \wedge \psi$: return $[\![\varphi]\!]_K \cap [\![\psi]\!]_K$
- $\mathsf{EX}\,\varphi$: return the set of predecessors in K of the states in $[\![\varphi]\!]_K$
- $\mathsf{E}\,\psi\,\mathsf{U}\,\varphi$: return the set of states with paths to states in $[\![\varphi]\!]_K$ that are entirely contained in $[\![\psi]\!]_K$ (except possibly for the last state of each path)
- $\mathsf{EG}\,\varphi$: return the set of states on infinite paths entirely contained in $[\![\varphi]\!]_K$

Action for Each Label

- Atomic proposition p: return set of states with p in their labels
- $\neg\varphi$: return the complement of $[\![\varphi]\!]_K$
- $\varphi \wedge \psi$: return $[\![\varphi]\!]_K \cap [\![\psi]\!]_K$
- $\mathsf{EX}\,\varphi$: return the set of predecessors in K of the states in $[\![\varphi]\!]_K$
- $\mathsf{E}\,\psi\,\mathsf{U}\,\varphi$: return the set of states with paths to states in $[\![\varphi]\!]_K$ that are entirely contained in $[\![\psi]\!]_K$ (except possibly for the last state of each path)
- $\mathsf{EG}\,\varphi$: return the set of states on infinite paths entirely contained in $[\![\varphi]\!]_K$

Action for Each Label

- Atomic proposition p: return set of states with p in their labels
- $\neg\varphi$: return the complement of $[\![\varphi]\!]_K$
- $\varphi \wedge \psi$: return $[\![\varphi]\!]_K \cap [\![\psi]\!]_K$
- $\mathsf{EX}\,\varphi$: return the set of predecessors in K of the states in $[\![\varphi]\!]_K$
- $\mathsf{E}\,\psi\,\mathsf{U}\,\varphi$: return the set of states with paths to states in $[\![\varphi]\!]_K$ that are entirely contained in $[\![\psi]\!]_K$ (except possibly for the last state of each path)
- $\mathsf{EG}\,\varphi$: return the set of states on infinite paths entirely contained in $[\![\varphi]\!]_K$

Explicit Model Checking

- To fix ideas, suppose Kripke structures are represented by adjacency lists and sets by bit vectors
- Boolean operations on sets can be performed in linear time
- Computing EU amounts to reachability in a graph
- Computing EG amounts to finding the strongly connected components (SCCs) of a subgraph of K
- Both reachability and SCC computation are based on depth-first search and take time linear in the size of the Kripke structure
- In practice sets are often represented by hash tables

Explicit Model Checking

- To fix ideas, suppose Kripke structures are represented by adjacency lists and sets by bit vectors
- Boolean operations on sets can be performed in linear time
- Computing EU amounts to reachability in a graph
- Computing EG amounts to finding the strongly connected components (SCCs) of a subgraph of K
- Both reachability and SCC computation are based on depth-first search and take time linear in the size of the Kripke structure
- In practice sets are often represented by hash tables

Explicit Model Checking

- To fix ideas, suppose Kripke structures are represented by adjacency lists and sets by bit vectors
- Boolean operations on sets can be performed in linear time
- Computing EU amounts to reachability in a graph
- Computing EG amounts to finding the strongly connected components (SCCs) of a subgraph of K
- Both reachability and SCC computation are based on depth-first search and take time linear in the size of the Kripke structure
- In practice sets are often represented by hash tables

Complexity of Explicit CTL MC

- Considering fairness constraints does not change the complexity of the algorithm
  - It suffices to discard SCCs that do not intersect all fair sets
- CTL model checking is linear in the size of the formula and the size of the structure

Symbolic Model Checking

- Linear time complexity is great unless your system has 10^{50} states
  - Number of states grows exponentially with number of state variables
- Explicit model checking limited to a few billion states
- Symbolic model checking can do much more (though not uniformly)

Symbolic Model Checking

- Linear time complexity is great unless your system has 10^{50} states
  - Number of states grows exponentially with number of state variables
- Explicit model checking limited to a few billion states
- Symbolic model checking can do much more (though not uniformly)

Symbolic Model Checking

- Linear time complexity is great unless your system has 10^{50} states
  - Number of states grows exponentially with number of state variables
- Explicit model checking limited to a few billion states
- Symbolic model checking can do much more (though not uniformly)

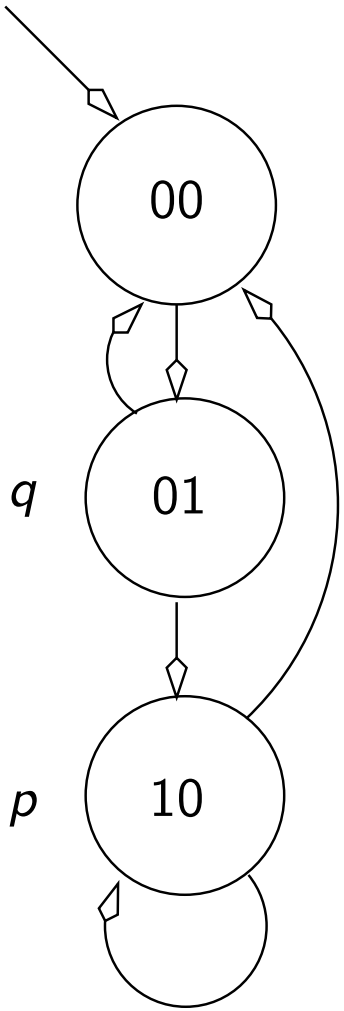
Characteristic Functions

- States are encoded as binary strings of length n (the number of binary state variables)
- A set of states V is represented by a characteristic function $\chi_V : B^n \rightarrow B$ that returns 1 for all elements of the set and 0 for all other states
- We often drop the "χ" from χ_V when there is no ambiguity

Characteristic Functions

- States are encoded as binary strings of length n (the number of binary state variables)
- A set of states V is represented by a characteristic function $\chi_V : B^n \rightarrow B$ that returns 1 for all elements of the set and 0 for all other states
- We often drop the "χ" from χ_V when there is no ambiguity

Symbolic Representation



- $T(x_1, x_0, y_1, y_0) = (\neg x_1 \wedge \neg x_0 \wedge \neg y_1 \wedge y_0) \vee (\neg x_1 \wedge x_0 \wedge \neg y_0) \vee (x_1 \wedge \neg x_0 \wedge \neg y_0)$
- $S_0(x_1, x_0) = \neg x_1 \wedge \neg x_0$
- $p(x_1, x_0) = x_1 \wedge \neg x_0$
- $q(x_1, x_0) = \neg x_1 \wedge x_0$

Characteristic Functions

- Let V contain all states with either the first or the last bit set to 1

$$\chi_V = x_1 \vee x_n$$

- Set V has $3 \cdot 2^{n-2}$ elements
- Great, but let us not get carried away, because it is not possible to find a representation that is compact for most functions

Characteristic Functions

- Let V contain all states with either the first or the last bit set to 1

$$\chi_V = x_1 \vee x_n$$

- Set V has $3 \cdot 2^{n-2}$ elements
- Great, but let us not get carried away, because it is not possible to find a representation that is compact for most functions

Characteristic Functions

- Let V contain all states with either the first or the last bit set to 1

$$\chi_V = x_1 \vee x_n$$

- Set V has $3 \cdot 2^{n-2}$ elements
- Great, but let us not get carried away, because it is not possible to find a representation that is compact for most functions

Implicit Enumeration

- Symbolic model checking enumerates states implicitly
- No explicit loop on the states or the transitions is used
- The cost of implicit enumeration is more affected by the size of the representation than by the cardinality of the set

Implicit Enumeration

- Symbolic model checking enumerates states implicitly
- No explicit loop on the states or the transitions is used
- The cost of implicit enumeration is more affected by the size of the representation than by the cardinality of the set

Computing Predecessors

- Let x (y) be the vector of current (next) state variables
- The set of predecessors of the states in V is given by

$$\exists y \,.\, T(x, y) \wedge V(y)$$

- No loops over states and transitions

Computing Predecessors

- Let x (y) be the vector of current (next) state variables
- The set of predecessors of the states in V is given by

$$\exists y \,.\, T(x, y) \wedge V(y)$$

- No loops over states and transitions

Symbolic Model Checking

- Boolean connectives give no difficulties
- Complementation turns into negation
- Union becomes disjunction and intersection becomes conjunction
- We have seen how to deal with $\mathrm{EX}\,\varphi$
- For EU and EG we use a fixpoint characterization
  - A fixpoint x of f is such that $f(x) = x$

Fixpoint Characterization

- The satisfying sets of EU and EG computations are fixpoints of monotonic functions over 2^S
  - $\mathsf{E}\,\psi\,\mathsf{U}\,\varphi = \varphi \vee [\psi \wedge \mathsf{EX}\,\mathsf{E}\,\psi\,\mathsf{U}\,\varphi]$
  - $\mathsf{EG}\,\varphi = \varphi \wedge \mathsf{EX}\,\mathsf{EG}\,\varphi$
- Specifically, $\mathsf{E}\,\psi\,\mathsf{U}\,\varphi$ is the least fixpoint and $\mathsf{EG}\,\varphi$ is the greatest fixpoint. This is written
  - $\mathsf{E}\,\psi\,\mathsf{U}\,\varphi = \mu Z\,.\,\varphi \vee [\psi \wedge \mathsf{EX}\,Z]$
  - $\mathsf{EG}\,\varphi = \nu Z\,.\,\varphi \wedge \mathsf{EX}\,Z$
- μ-calculus notation

Fixpoint Characterization

- The satisfying sets of EU and EG computations are fixpoints of monotonic functions over 2^S
  - $\mathsf{E}\,\psi\,\mathsf{U}\,\varphi = \varphi \vee [\psi \wedge \mathsf{EX}\,\mathsf{E}\,\psi\,\mathsf{U}\,\varphi]$
  - $\mathsf{EG}\,\varphi = \varphi \wedge \mathsf{EX}\,\mathsf{EG}\,\varphi$
- Specifically, $\mathsf{E}\,\psi\,\mathsf{U}\,\varphi$ is the least fixpoint and $\mathsf{EG}\,\varphi$ is the greatest fixpoint. This is written
  - $\mathsf{E}\,\psi\,\mathsf{U}\,\varphi = \mu Z\,.\,\varphi \vee [\psi \wedge \mathsf{EX}\,Z]$
  - $\mathsf{EG}\,\varphi = \nu Z\,.\,\varphi \wedge \mathsf{EX}\,Z$
- μ-calculus notation

Fixpoint Characterization

- The satisfying sets of EU and EG computations are fixpoints of monotonic functions over 2^S
  - $\mathsf{E}\,\psi\,\mathsf{U}\,\varphi = \varphi \vee [\psi \wedge \mathsf{EX}\,\mathsf{E}\,\psi\,\mathsf{U}\,\varphi]$
  - $\mathsf{EG}\,\varphi = \varphi \wedge \mathsf{EX}\,\mathsf{EG}\,\varphi$
- Specifically, $\mathsf{E}\,\psi\,\mathsf{U}\,\varphi$ is the least fixpoint and $\mathsf{EG}\,\varphi$ is the greatest fixpoint. This is written
  - $\mathsf{E}\,\psi\,\mathsf{U}\,\varphi = \mu Z\,.\,\varphi \vee [\psi \wedge \mathsf{EX}\,Z]$
  - $\mathsf{EG}\,\varphi = \nu Z\,.\,\varphi \wedge \mathsf{EX}\,Z$
- μ-calculus notation

Tarski's Theorem

- A function f is monotonic if

$$x \leq y \rightarrow f(x) \leq f(y)$$

- A monotonic function over a finite lattice has a least fixpoint that can be computed as the limit of the sequence

$$0, f(0), f(f(0)), f(f(f(0))), \ldots$$

- For greatest fixpoints

$$1, f(1), f(f(1)), f(f(f(1))), \ldots$$

- This is the kitty version of the theorem for finite lattices

Tarski's Theorem

- A function f is monotonic if

$$x \leq y \rightarrow f(x) \leq f(y)$$

- A monotonic function over a finite lattice has a least fixpoint that can be computed as the limit of the sequence

$$0, f(0), f(f(0)), f(f(f(0))), \ldots$$

- For greatest fixpoints

$$1, f(1), f(f(1)), f(f(f(1))), \ldots$$

- This is the kitty version of the theorem for finite lattices

Tarski's Theorem

- A function f is monotonic if

$$x \leq y \rightarrow f(x) \leq f(y)$$

- A monotonic function over a finite lattice has a least fixpoint that can be computed as the limit of the sequence

$$0, f(0), f(f(0)), f(f(f(0))), \ldots$$

- For greatest fixpoints

$$1, f(1), f(f(1)), f(f(f(1))), \ldots$$

- This is the kitty version of the theorem for finite lattices

Tarski's Theorem

- A function f is monotonic if

$$x \leq y \rightarrow f(x) \leq f(y)$$

- A monotonic function over a finite lattice has a least fixpoint that can be computed as the limit of the sequence

$$0, f(0), f(f(0)), f(f(f(0))), \ldots$$

- For greatest fixpoints

$$1, f(1), f(f(1)), f(f(f(1))), \ldots$$

- This is the kitty version of the theorem for finite lattices

Computing $\mathsf{E}\,\psi\,\mathsf{U}\,\varphi$ and $\mathsf{E}\,\psi\,\mathsf{R}\,\varphi$

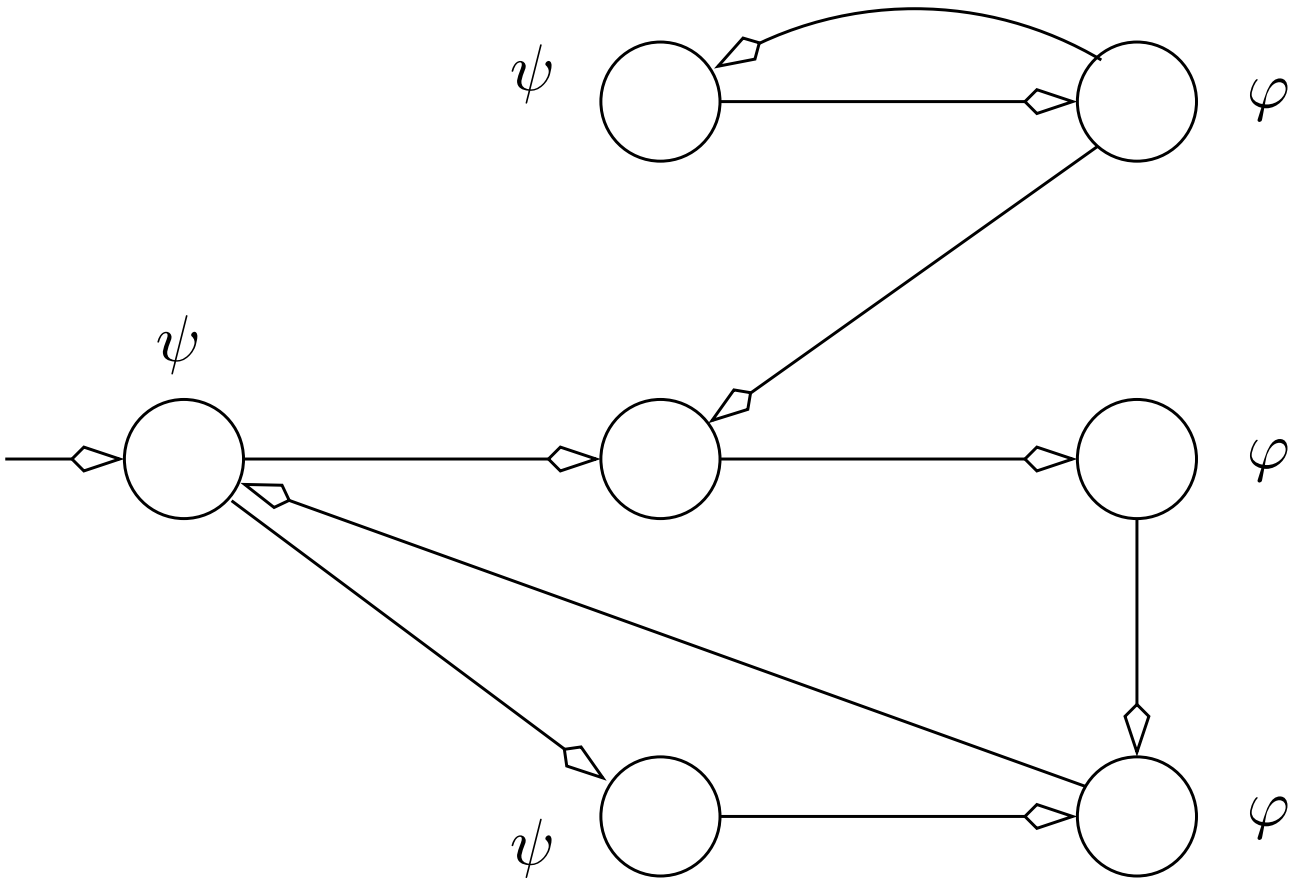
$Z = \varnothing;$
$\zeta = S;$

while $(Z \neq \zeta)$ {
    $\zeta = Z;$
    $Z = \varphi \vee (\psi \wedge \mathsf{EX}\, Z);$
}

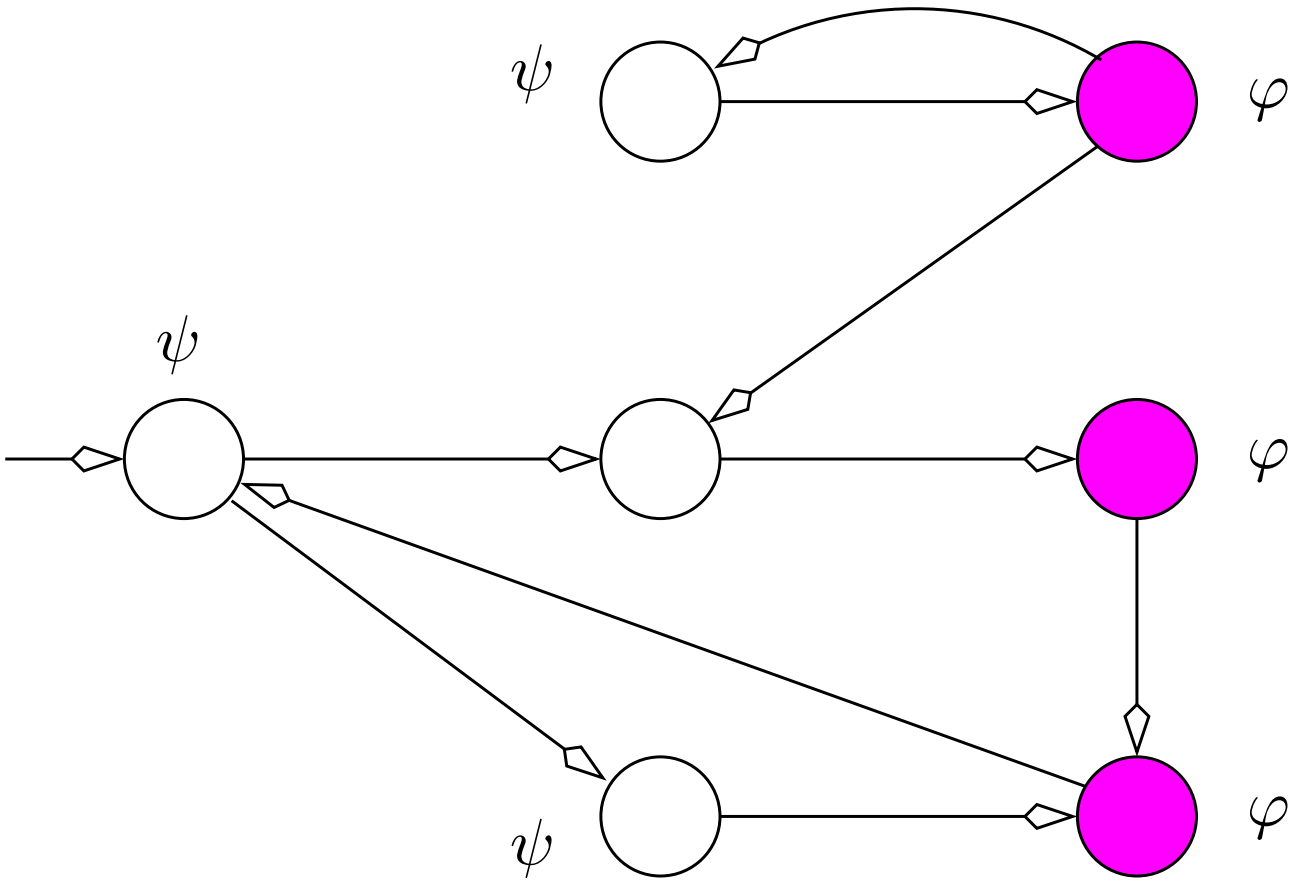
$Z = S;$
$\zeta = \varnothing;$

while $(Z \neq \zeta)$ {
    $\zeta = Z;$
    $Z = \varphi \wedge (\psi \vee \mathsf{EX}\, Z);$
}

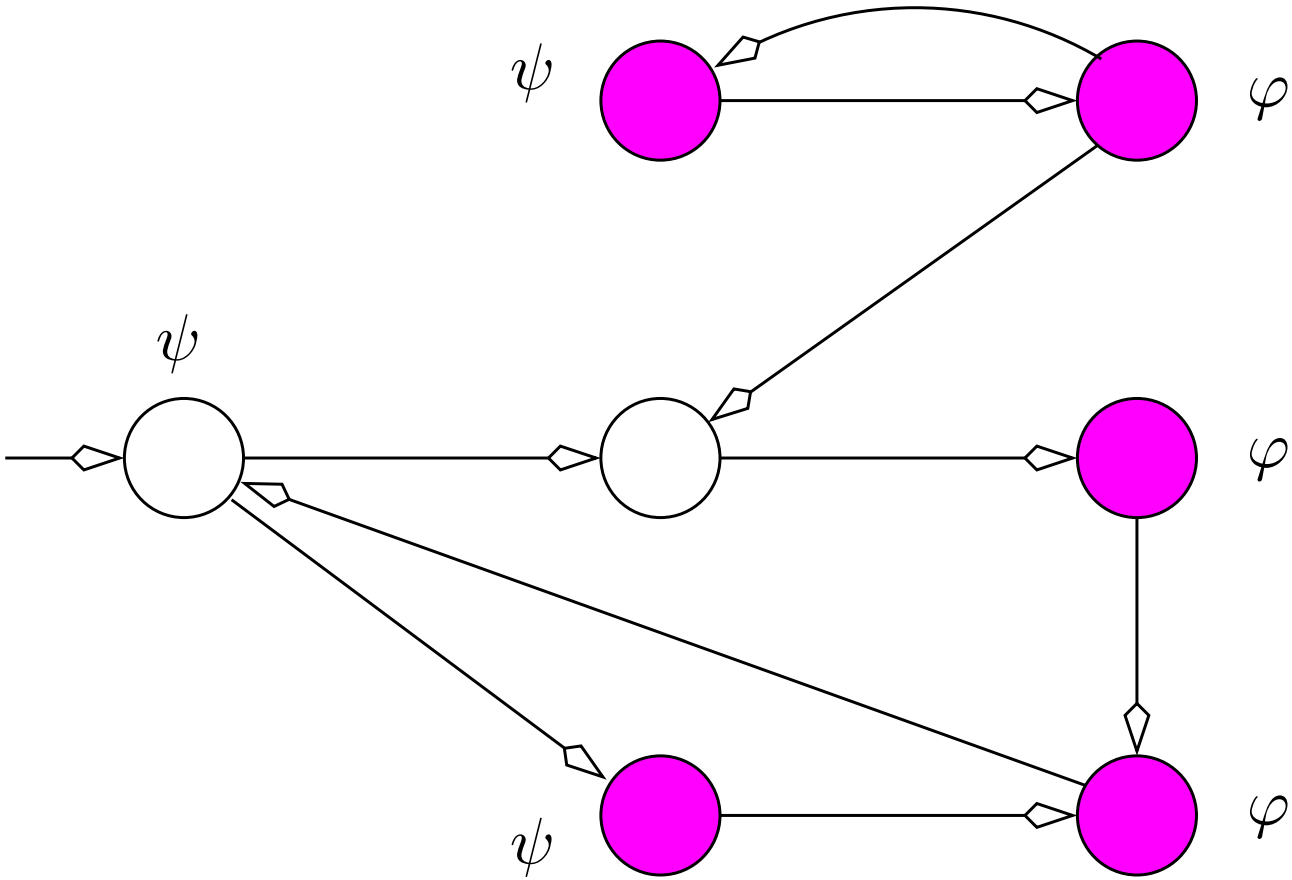
Computing $\mathsf{E}\,\psi\,\mathsf{U}\,\varphi$



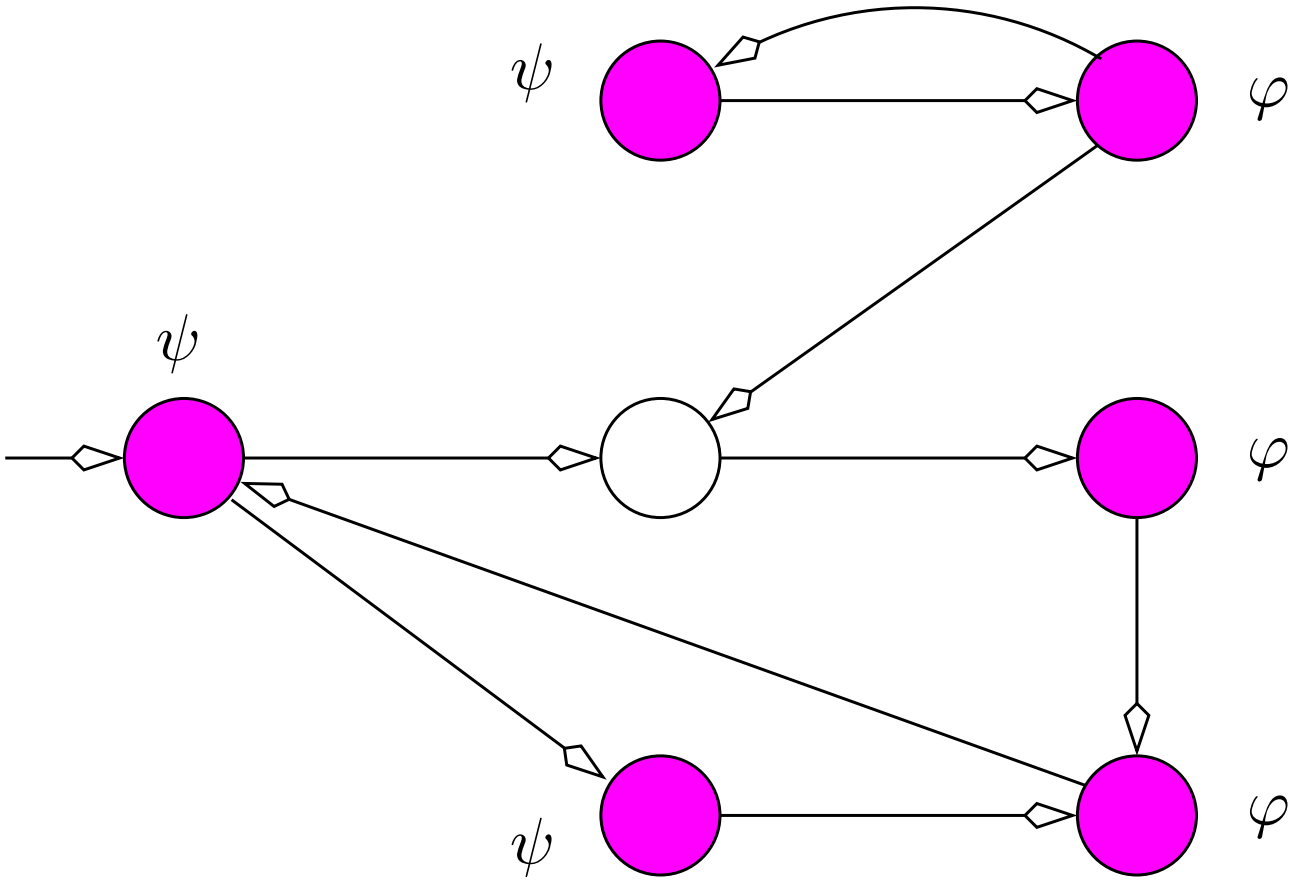
Computing $\mathsf{E}\,\psi\,\mathsf{U}\,\varphi$



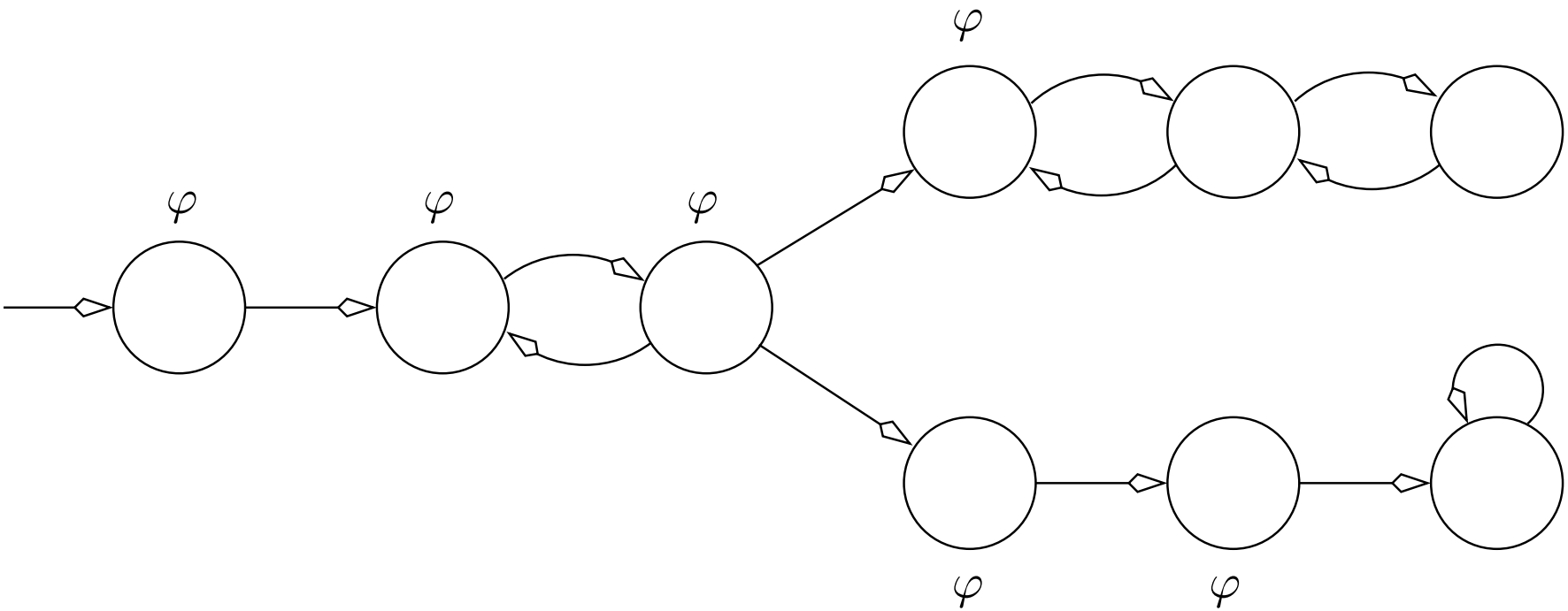
Computing $\mathsf{E}\,\psi\,\mathsf{U}\,\varphi$



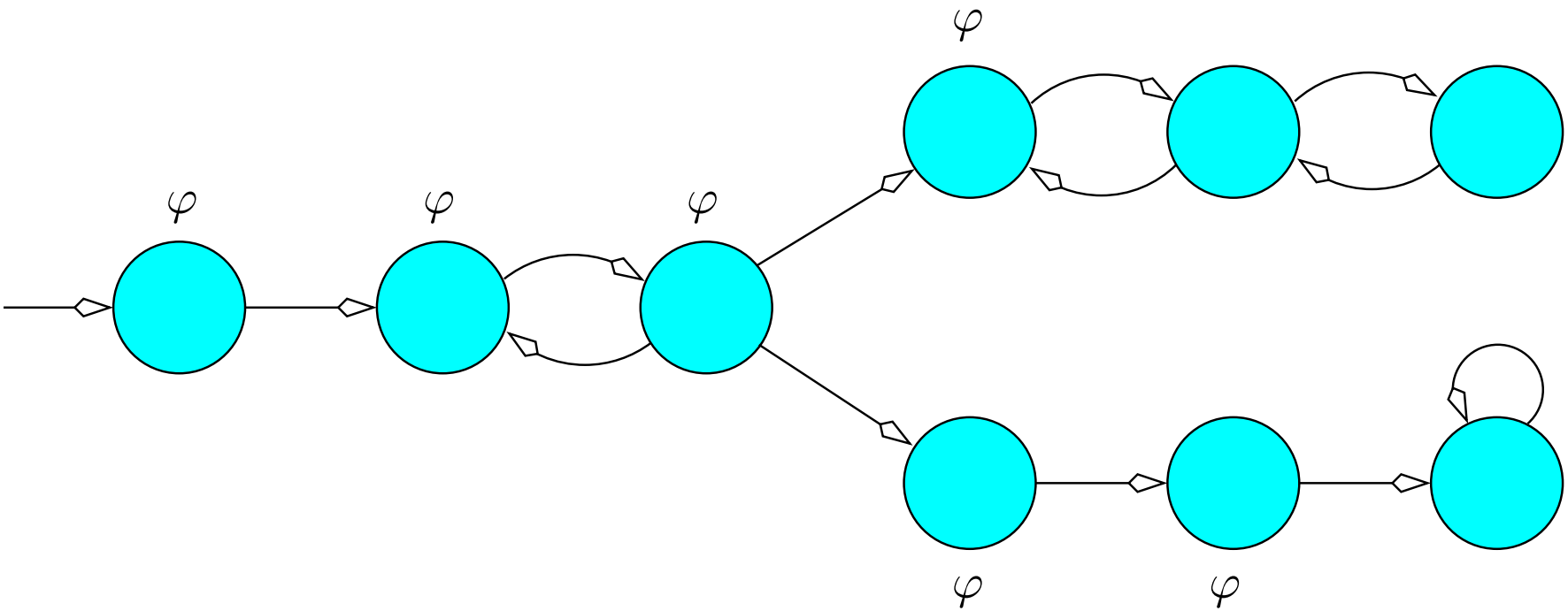
Computing $\mathsf{E}\,\psi\,\mathsf{U}\,\varphi$



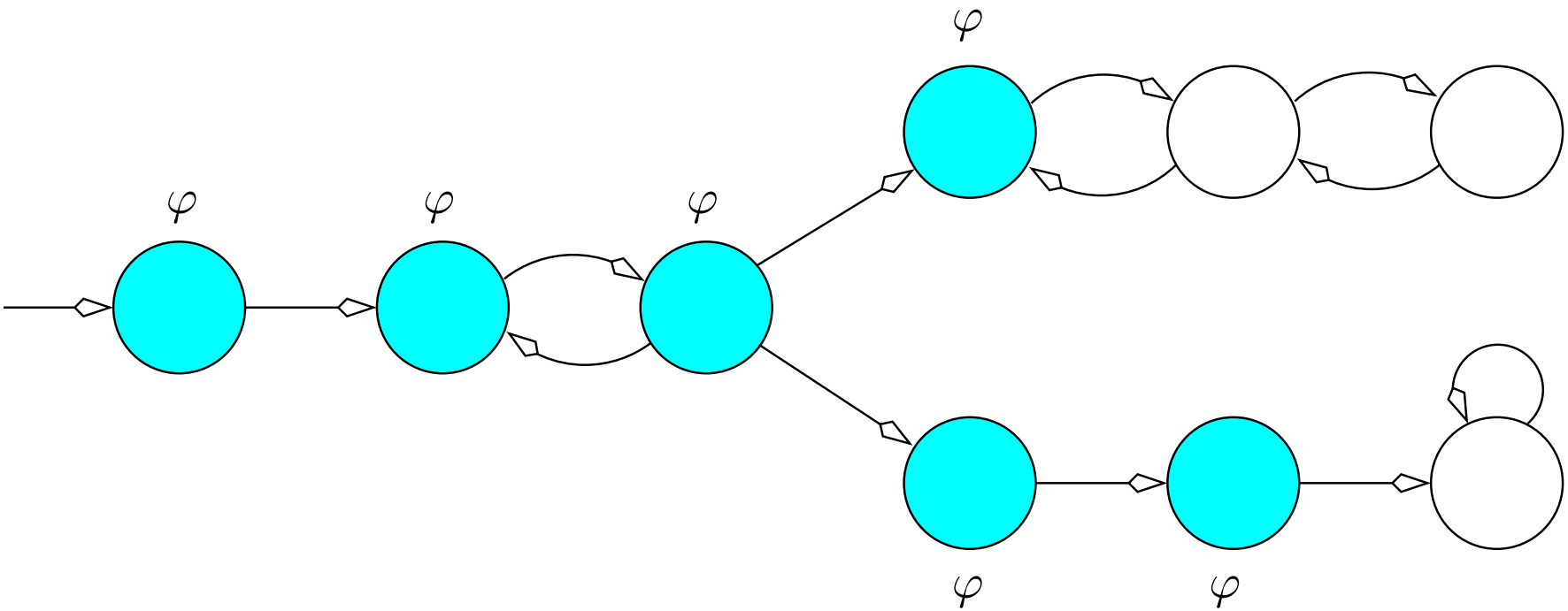
Computing EG φ



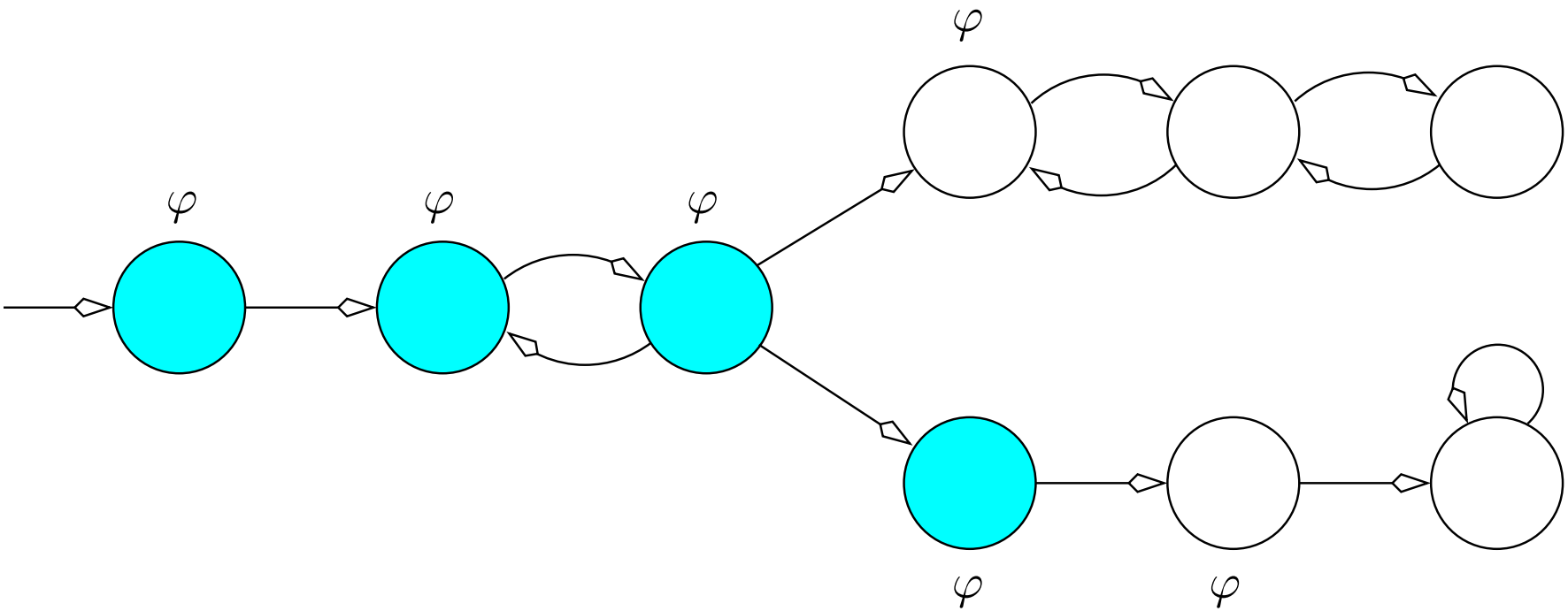
Computing EG φ



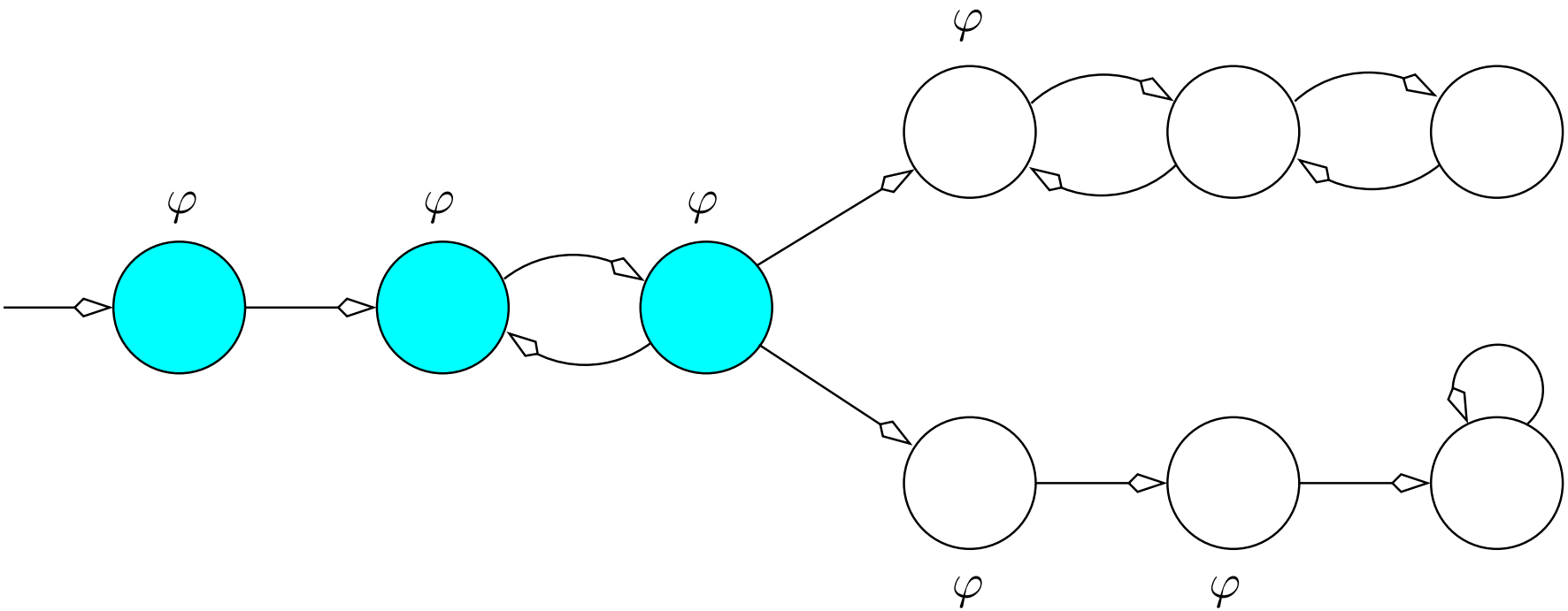
Computing EG φ



Computing EG φ



Computing EG φ



Backward vs. Forward Search

- The algorithm we have presented is based on backward search in the Kripke structure (EX computes predecessors)
- Part of CTL can be model checked using forward search (using EY that computes successors)
- Forward search popular for invariants (AG p)
- Forward search is also used to find states reachable from initial states and prune backward search

On-the-Fly Model Checking

- In checking $K \models \mathrm{EF}\, p$ we can stop the computation of the least fixpoint as soon as all initial states are acquired
- This is an instance of on-the-fly model checking
- In general, on-the-fly model checking stops as soon as the answer is known

ECTL and ACTL

- ECTL is a fragment of CTL that consists of existential properties
- Negation can only occur in front of atomic propositions
- The path quantifier A cannot be used
- ACTL consists of the negation of ECTL formulae (universal properties)

ECTL and ACTL

- ECTL is a fragment of CTL that consists of existential properties
- Negation can only occur in front of atomic propositions
- The path quantifier A cannot be used
- ACTL consists of the negation of ECTL formulae (universal properties)

ECTL and ACTL Examples

- $\mathsf{EF}(\varphi \wedge \mathsf{EG}\,\neg\psi)$ ECTL
- $\mathsf{AG}(\varphi \rightarrow \mathsf{AF}\,\psi)$ ACTL
  - Negation of the previous formula
- $\mathsf{AG}(\mathsf{AF}\,\varphi \rightarrow \mathsf{AF}\,\psi)$ mixed
- $\mathsf{EF}(\mathsf{EG}\,\varphi \vee \mathsf{EG}\,\psi)$ ECTL

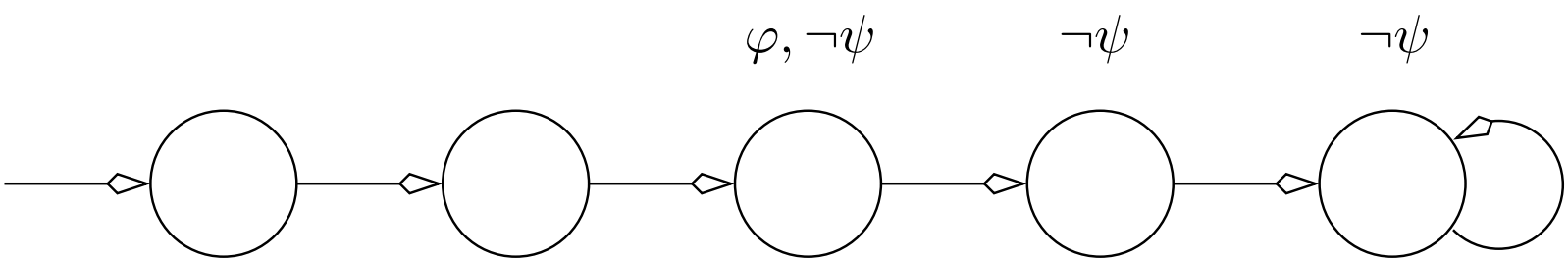
Counterexamples and Witnesses

- If a property fails it is useful to show an execution of the system that is a counterexample
- For a property that holds, it may be useful to show an execution of the system that is a witness
- A witness to $K, s \models \varphi$ is a counterexample to $K, s \models \neg\varphi$ and vice versa

Witness Example

- Witness for the ECTL property

$$\mathsf{EF}(\varphi \wedge \mathsf{EG}\,\neg\psi)$$



A Witness May Not Suffice

- One witness computation does not suffice to show that AG φ holds in a structure with multiple paths
- One counterexample computation does not suffice to show that EF $\neg\varphi$ fails
- Complete witnesses can be found for ECTL formulae and complete counterexamples for ACTL formulae

A Witness May Not Suffice

- One witness computation does not suffice to show that $\mathsf{AG}\,\varphi$ holds in a structure with multiple paths
- One counterexample computation does not suffice to show that $\mathsf{EF}\,\neg\varphi$ fails
- Complete witnesses can be found for ECTL formulae and complete counterexamples for ACTL formulae

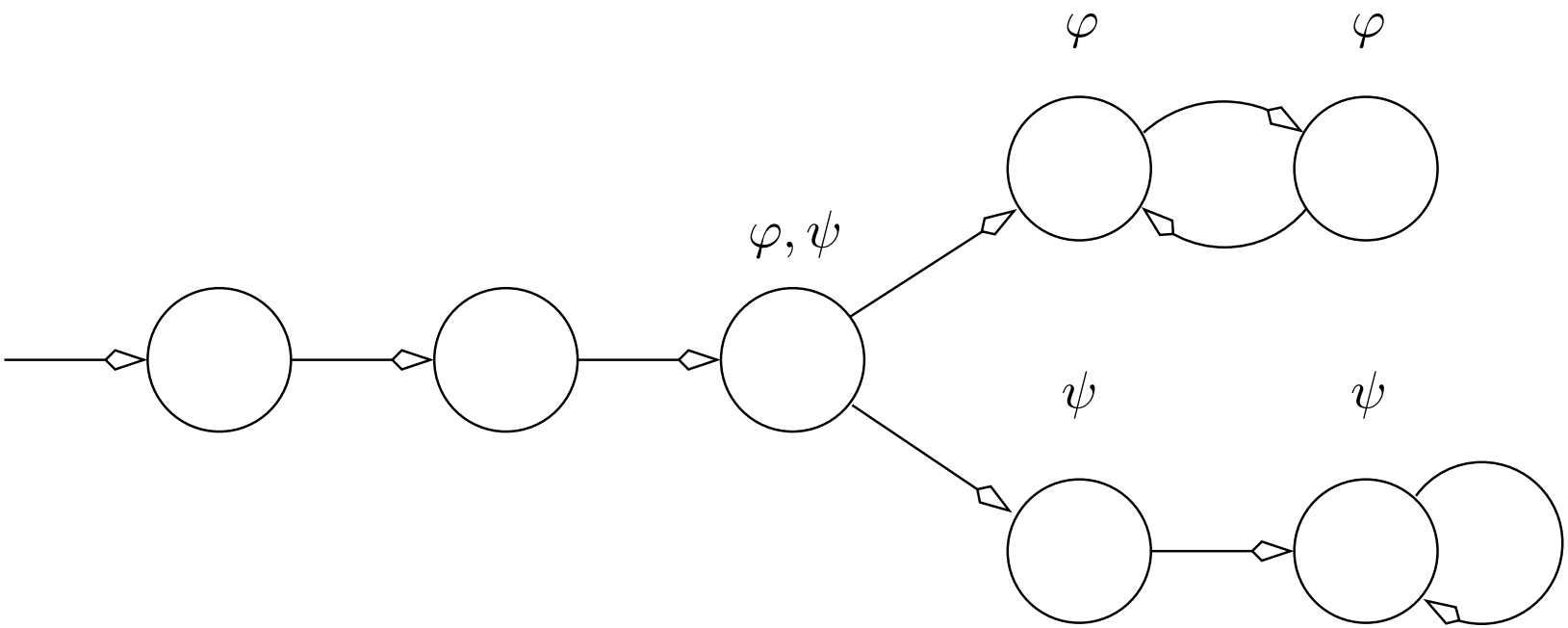
A Witness May Not Suffice

- One witness computation does not suffice to show that $\mathsf{AG}\,\varphi$ holds in a structure with multiple paths
- One counterexample computation does not suffice to show that $\mathsf{EF}\,\neg\varphi$ fails
- Complete witnesses can be found for ECTL formulae and complete counterexamples for ACTL formulae

Branching Witnesses

- Consider a witness for the ECTL formula

$$\mathsf{EF}(\mathsf{EG}\,\varphi \wedge \mathsf{EG}\,\psi)$$



Adding Fairness Constraints

- Fairness constraints are given as sets of states that must be traversed infinitely often (Büchi fairness)
- In CTL the fair sets are the satisfying sets of CTL formulae (distinct from the formulae for the properties to be verified)
- Quantifiers are restricted to fair paths
  - Paths that intersect all the fair sets infinitely often

Adding Fairness Constraints

- Fairness constraints are given as sets of states that must be traversed infinitely often (Büchi fairness)
- In CTL the fair sets are the satisfying sets of CTL formulae (distinct from the formulae for the properties to be verified)
- Quantifiers are restricted to fair paths
  - Paths that intersect all the fair sets infinitely often

Adding Fairness Constraints

- Fairness constraints are given as sets of states that must be traversed infinitely often (Büchi fairness)
- In CTL the fair sets are the satisfying sets of CTL formulae (distinct from the formulae for the properties to be verified)
- Quantifiers are restricted to fair paths
  - Paths that intersect all the fair sets infinitely often

Computing the Fair States

- The fair states are those along fair paths
- They are computed by the μ-calculus formula

$$\nu Z \,.\, \mathsf{EX}[\mathsf{E}\, Z\, \mathsf{U}(Z \wedge c)]$$

in the case of one fairness constraint c

MC for CTL with Fairness

- Let $\Phi = \nu Z \,.\, \bigwedge_i \mathsf{EX}[\mathsf{E}\, Z\, \mathsf{U}(Z \wedge c_i)]$
  - The set of fair states with $C = \{c_i\}$
- $\mathsf{E}_C\, \mathsf{G}\, \varphi = \nu Z \,.\, \varphi \wedge \bigwedge_i \mathsf{EX}[\mathsf{E}\, Z\, \mathsf{U}(Z \wedge c_i)]$
- $\mathsf{E}_C\, \mathsf{X}\, \varphi = \mathsf{EX}(\varphi \wedge \Phi)$
- $\mathsf{E}_C\, \psi\, \mathsf{U}\, \varphi = \mathsf{E}\, \psi\, \mathsf{U}(\varphi \wedge \Phi)$

Fair Witnesses

- For EX and EU, we append witness to EG true to finite witness
- For EG φ, while tracing a loop in a fair SCC we need to insure that all fair sets are visited
- Finding a shortest witness is hard, but heuristics work reasonably well

Outline

LTL Model Checking

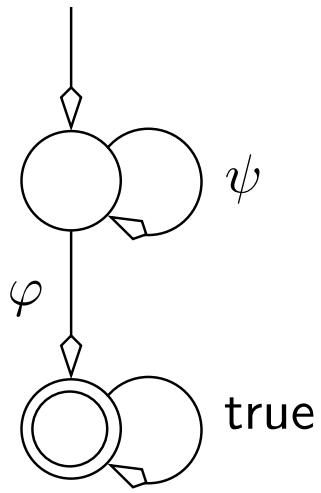
- LTL operators can be characterized in terms of fixpoints like their CTL counterparts
  - $\psi \,\mathsf{U}\, \varphi = \varphi \vee [\psi \wedge \mathsf{X}(\psi \,\mathsf{U}\, \varphi)]$
  - $\psi \,\mathsf{R}\, \varphi = \varphi \wedge [\psi \vee \mathsf{X}(\psi \,\mathsf{R}\, \varphi)]$
- The problem is that we have sets of paths instead of sets of states
- We take a different approach

Model Checking LTL

- Negate the given formula φ to get $\neg\varphi$
- Build a Büchi automaton $A_{\neg\varphi}$ that accepts exactly the computations that model $\neg\varphi$
- Compose $A_{\neg\varphi}$ with the system to be verified
- Check whether there is a fair path in the composition

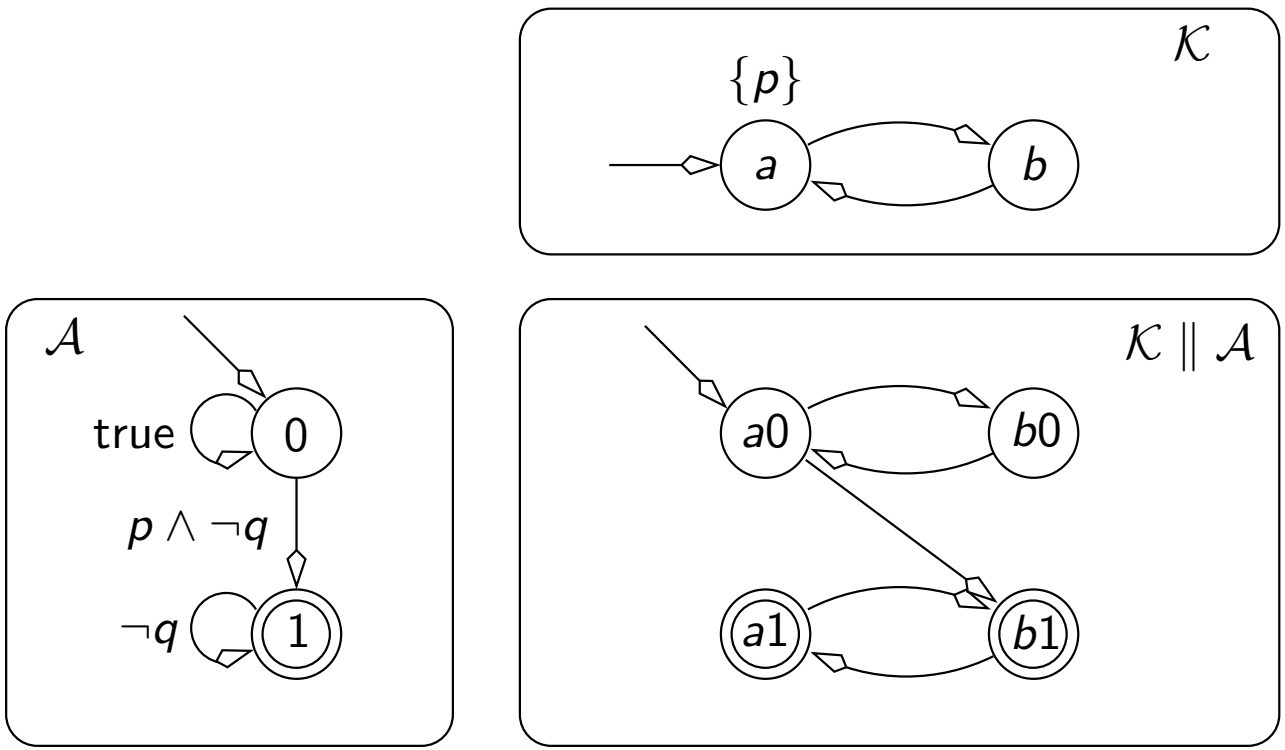
From Formula to Automaton

$$\psi \,\mathsf{U}\, \varphi = \varphi \vee [\psi \wedge \mathsf{X}(\psi \,\mathsf{U}\, \varphi)]$$



Model Checking $\mathrm{AG}(p \rightarrow \mathrm{F}\, q)$

- A counterexample to $\mathrm{AG}(p \rightarrow \mathrm{F}\, q)$ is an execution in which p happens, but q does not follow it
- Compose automaton for all counterexamples with model and check for accepting path



CTL* Model Checking

- If formula is propositional, compute its satisfying set directly, otherwise
- There is a subformula that is an LTL formula $\mathsf{E}\,\psi$: find its satisfying set with LTL model checking algorithm and use it as a new atomic proposition
- Repeat until the satisfying set of the root of the parse tree is found

Outline

Bounded Model Checking

- Checks for a counterexample to a property of a model
  - We assume finite state and LTL
- Encodes the property checking problem as propositional satisfiability (SAT)
- Constructs a propositional formula that is satisfiable iff there exits a length-k counterexample, e.g.,

$$I(s_0) \wedge \bigwedge_{0 \leq i < k} T(s_i, s_{i+1}) \wedge \neg p(s_k)$$

- If no counterexample is found, BMC increases k until
  - a counterexample is found,
  - the search becomes intractable, or
  - k reaches a certain bound

Bounded Model Checking

- Checks for a counterexample to a property of a model
  - We assume finite state and LTL
- Encodes the property checking problem as propositional satisfiability (SAT)
- Constructs a propositional formula that is satisfiable iff there exits a length-k counterexample, e.g.,

$$I(s_0) \wedge \bigwedge_{0 \leq i < k} T(s_i, s_{i+1}) \wedge \neg p(s_k)$$

- If no counterexample is found, BMC increases k until
  - a counterexample is found,
  - the search becomes intractable, or
  - k reaches a certain bound

Bounded Model Checking

- Checks for a counterexample to a property of a model
  - We assume finite state and LTL
- Encodes the property checking problem as propositional satisfiability (SAT)
- Constructs a propositional formula that is satisfiable iff there exits a length-k counterexample, e.g.,

$$I(s_0) \wedge \bigwedge_{0 \leq i < k} T(s_i, s_{i+1}) \wedge \neg p(s_k)$$

- If no counterexample is found, BMC increases k until
  - a counterexample is found,
  - the search becomes intractable, or
  - k reaches a certain bound

Bounded Model Checking

- Checks for a counterexample to a property of a model
  - We assume finite state and LTL
- Encodes the property checking problem as propositional satisfiability (SAT)
- Constructs a propositional formula that is satisfiable iff there exits a length-k counterexample, e.g.,

$$I(s_0) \wedge \bigwedge_{0 \leq i < k} T(s_i, s_{i+1}) \wedge \neg p(s_k)$$

- If no counterexample is found, BMC increases k until
  - a counterexample is found,
  - the search becomes intractable, or
  - k reaches a certain bound

Falsification vs. Verification

- Sometime, proving the absence of an error is more useful than finding one
  - When checking an abstract model for a universal property
    - Finding an error in the abstract model does not imply its existence in the original model
    - However, proving that the property passes in the abstract model guarantee the absence of errors in the original model
- BMC efficiency reduces as the length of the counterexample increases
  - It may be more efficient to prove the property and stop early than keep searching for a counterexample

Proving Properties with BMC

- The original BMC algorithm (Biere et al. [1999]), although complete in theory, is limited in practice to falsification of LTL properties
- BMC can prove that an LTL property ψ passes on a model $\mathcal{M}$ only if a bound, κ, is known such that:
  - if no counterexample of length up to κ is found, then $\mathcal{M} \models \mathsf{A}\,\psi$
- Several methods exist to compute a suitable κ
- The optimum value of κ, however, is usually very expensive to obtain
  - Finding it is at least as hard as checking whether $M \models \mathsf{A}\,\psi$ (Clarke et al. [2004])

Proving Properties with BMC

- The original BMC algorithm (Biere et al. [1999]), although complete in theory, is limited in practice to falsification of LTL properties
- BMC can prove that an LTL property ψ passes on a model $\mathcal{M}$ only if a bound, κ, is known such that:
  - if no counterexample of length up to κ is found, then $\mathcal{M} \models \mathsf{A}\,\psi$
- Several methods exist to compute a suitable κ
- The optimum value of κ, however, is usually very expensive to obtain
  - Finding it is at least as hard as checking whether $M \models \mathsf{A}\,\psi$ (Clarke et al. [2004])

Proving Properties with BMC

- The original BMC algorithm (Biere et al. [1999]), although complete in theory, is limited in practice to falsification of LTL properties
- BMC can prove that an LTL property ψ passes on a model $\mathcal{M}$ only if a bound, κ, is known such that:
  - if no counterexample of length up to κ is found, then $\mathcal{M} \models \mathsf{A}\,\psi$
- Several methods exist to compute a suitable κ
- The optimum value of κ, however, is usually very expensive to obtain
  - Finding it is at least as hard as checking whether $M \models \mathsf{A}\,\psi$ (Clarke et al. [2004])

Proving Properties with BMC

- The original BMC algorithm (Biere et al. [1999]), although complete in theory, is limited in practice to falsification of LTL properties
- BMC can prove that an LTL property ψ passes on a model $\mathcal{M}$ only if a bound, κ, is known such that:
  - if no counterexample of length up to κ is found, then $\mathcal{M} \models \mathsf{A}\,\psi$
- Several methods exist to compute a suitable κ
- The optimum value of κ, however, is usually very expensive to obtain
  - Finding it is at least as hard as checking whether $M \models \mathsf{A}\,\psi$ (Clarke et al. [2004])

Interpolation

- Suppose
  $I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \neg p(s_k)$ is unsatisfiable
- Let $F_1 = I(s_0) \wedge T(s_0, s_1)$ and
  $F_2 = T(s_1, s_2) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \neg p(s_k)$
- Then $F_1(s_0, s_1) \wedge F_2(s_1, \ldots, s_k)$ is unsatisfiable
- Interpolant $\mathcal{I}_1(s_1)$ (McMillan [2003]) is such that
  - $F_1(s_0, s_1) \rightarrow \mathcal{I}_1(s_1)$
  - $\mathcal{I}_1(s_1) \wedge F_2(s_1, \ldots, s_k)$ is unsatisfiable
- $\mathcal{I}_1(s_1)$ can be computed in linear time from a resolution proof that $F_1(s_0, s_1) \wedge F_2(s_1, \ldots, s_k)$ is unsatisfiable
- $\exists s_0 \,.\, I(s_0) \wedge T(s_0, s_1)$ is the strongest interpolant
  - set of states reachable from $I(s_0)$ in one step

Interpolation

- Suppose
  $I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \neg p(s_k)$ is unsatisfiable
- Let $F_1 = I(s_0) \wedge T(s_0, s_1)$ and
  $F_2 = T(s_1, s_2) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \neg p(s_k)$
- Then $F_1(s_0, s_1) \wedge F_2(s_1, \ldots, s_k)$ is unsatisfiable
- Interpolant $\mathcal{I}_1(s_1)$ (McMillan [2003]) is such that
  - $F_1(s_0, s_1) \rightarrow \mathcal{I}_1(s_1)$
  - $\mathcal{I}_1(s_1) \wedge F_2(s_1, \ldots, s_k)$ is unsatisfiable
- $\mathcal{I}_1(s_1)$ can be computed in linear time from a resolution proof that $F_1(s_0, s_1) \wedge F_2(s_1, \ldots, s_k)$ is unsatisfiable
- $\exists s_0 \,.\, I(s_0) \wedge T(s_0, s_1)$ is the strongest interpolant
  - set of states reachable from $I(s_0)$ in one step

Interpolation

- Suppose
  $I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \neg p(s_k)$ is unsatisfiable
- Let $F_1 = I(s_0) \wedge T(s_0, s_1)$ and
  $F_2 = T(s_1, s_2) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \neg p(s_k)$
- Then $F_1(s_0, s_1) \wedge F_2(s_1, \ldots, s_k)$ is unsatisfiable
- Interpolant $\mathcal{I}_1(s_1)$ (McMillan [2003]) is such that
  - $F_1(s_0, s_1) \rightarrow \mathcal{I}_1(s_1)$
  - $\mathcal{I}_1(s_1) \wedge F_2(s_1, \ldots, s_k)$ is unsatisfiable
- $\mathcal{I}_1(s_1)$ can be computed in linear time from a resolution proof that $F_1(s_0, s_1) \wedge F_2(s_1, \ldots, s_k)$ is unsatisfiable
- $\exists s_0 \,.\, I(s_0) \wedge T(s_0, s_1)$ is the strongest interpolant
  - set of states reachable from $I(s_0)$ in one step

Interpolation

- Suppose
  $I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \neg p(s_k)$ is unsatisfiable
- Let $F_1 = I(s_0) \wedge T(s_0, s_1)$ and
  $F_2 = T(s_1, s_2) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \neg p(s_k)$
- Then $F_1(s_0, s_1) \wedge F_2(s_1, \ldots, s_k)$ is unsatisfiable
- Interpolant $\mathcal{I}_1(s_1)$ (McMillan [2003]) is such that
  - $F_1(s_0, s_1) \rightarrow \mathcal{I}_1(s_1)$
  - $\mathcal{I}_1(s_1) \wedge F_2(s_1, \ldots, s_k)$ is unsatisfiable
- $\mathcal{I}_1(s_1)$ can be computed in linear time from a resolution proof that $F_1(s_0, s_1) \wedge F_2(s_1, \ldots, s_k)$ is unsatisfiable
- $\exists s_0 \,.\, I(s_0) \wedge T(s_0, s_1)$ is the strongest interpolant
  - set of states reachable from $I(s_0)$ in one step

Interpolation

- Suppose
  $I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \neg p(s_k)$ is unsatisfiable
- Let $F_1 = I(s_0) \wedge T(s_0, s_1)$ and
  $F_2 = T(s_1, s_2) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \neg p(s_k)$
- Then $F_1(s_0, s_1) \wedge F_2(s_1, \ldots, s_k)$ is unsatisfiable
- Interpolant $\mathcal{I}_1(s_1)$ (McMillan [2003]) is such that
  - $F_1(s_0, s_1) \rightarrow \mathcal{I}_1(s_1)$
  - $\mathcal{I}_1(s_1) \wedge F_2(s_1, \ldots, s_k)$ is unsatisfiable
- $\mathcal{I}_1(s_1)$ can be computed in linear time from a resolution proof that $F_1(s_0, s_1) \wedge F_2(s_1, \ldots, s_k)$ is unsatisfiable
- $\exists s_0 \,.\, I(s_0) \wedge T(s_0, s_1)$ is the strongest interpolant
  - set of states reachable from $I(s_0)$ in one step

Interpolation

- Suppose
  $I(s_0) \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \neg p(s_k)$ is unsatisfiable
- Let $F_1 = I(s_0) \wedge T(s_0, s_1)$ and
  $F_2 = T(s_1, s_2) \wedge \cdots \wedge T(s_{k-1}, s_k) \wedge \neg p(s_k)$
- Then $F_1(s_0, s_1) \wedge F_2(s_1, \ldots, s_k)$ is unsatisfiable
- Interpolant $\mathcal{I}_1(s_1)$ (McMillan [2003]) is such that
  - $F_1(s_0, s_1) \rightarrow \mathcal{I}_1(s_1)$
  - $\mathcal{I}_1(s_1) \wedge F_2(s_1, \ldots, s_k)$ is unsatisfiable
- $\mathcal{I}_1(s_1)$ can be computed in linear time from a resolution proof that $F_1(s_0, s_1) \wedge F_2(s_1, \ldots, s_k)$ is unsatisfiable
- $\exists s_0 \,.\, I(s_0) \wedge T(s_0, s_1)$ is the strongest interpolant
  - set of states reachable from $I(s_0)$ in one step

Interpolation

- $\mathcal{I}_1(s_1)$ is a superset of the states reachable in one step such that no member state has a path of length $k-1$ to a bad state
- Replace $I(s_0)$ with $I(s_0) \vee \mathcal{I}_1(s_0)$ and repeat
  - If formula still unsatisfiable, interpolant $\mathcal{I}_2(s_1)$ is a superset of states reachable in one or two steps such that no member state has a path of length $k-1$ to a bad state
- A converging sequence of interpolants means that no states satisfying $\neg p$ (bad states) are reachable

Interpolation

- $\mathcal{I}_1(s_1)$ is a superset of the states reachable in one step such that no member state has a path of length $k-1$ to a bad state
- Replace $I(s_0)$ with $I(s_0) \vee \mathcal{I}_1(s_0)$ and repeat
  - If formula still unsatisfiable, interpolant $\mathcal{I}_2(s_1)$ is a superset of states reachable in one or two steps such that no member state has a path of length $k-1$ to a bad state
- A converging sequence of interpolants means that no states satisfying $\neg p$ (bad states) are reachable

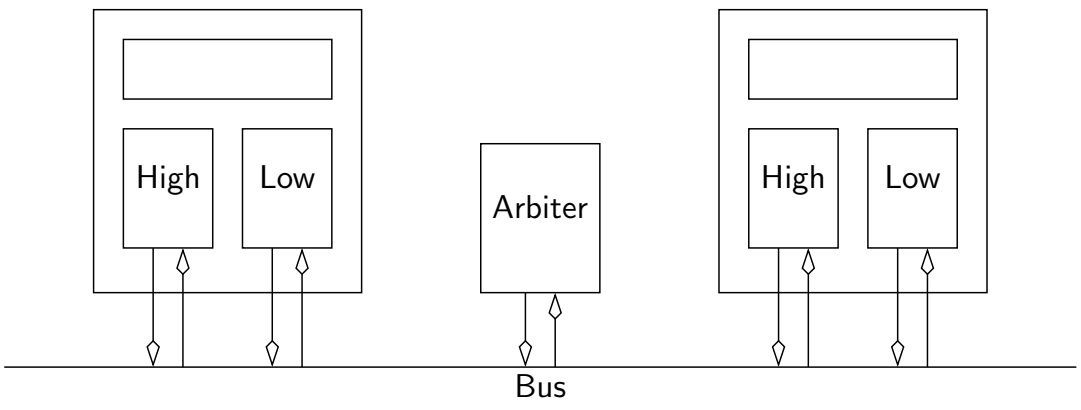
Interpolation

- $\mathcal{I}_1(s_1)$ is a superset of the states reachable in one step such that no member state has a path of length $k-1$ to a bad state
- Replace $I(s_0)$ with $I(s_0) \vee \mathcal{I}_1(s_0)$ and repeat
  - If formula still unsatisfiable, interpolant $\mathcal{I}_2(s_1)$ is a superset of states reachable in one or two steps such that no member state has a path of length $k-1$ to a bad state
- A converging sequence of interpolants means that no states satisfying $\neg p$ (bad states) are reachable

Outline

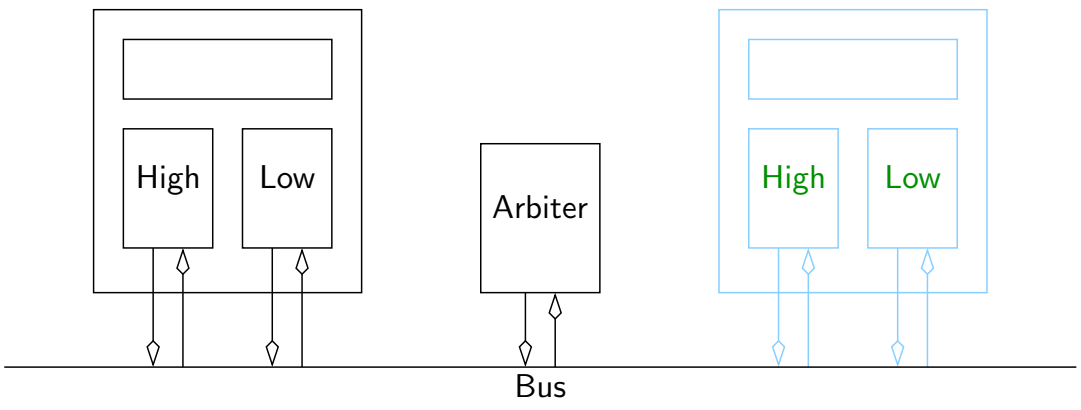
Informal Abstraction

- In use since the early days of simulation
  - What if we write to a memory when the address contains Xs?
- Still indispensable, but
  - May be laborious
  - Comes with no implied warranty



Informal Abstraction

- In use since the early days of simulation
  - What if we write to a memory when the address contains Xs?
- Still indispensable, but
  - May be laborious
  - Comes with no implied warranty

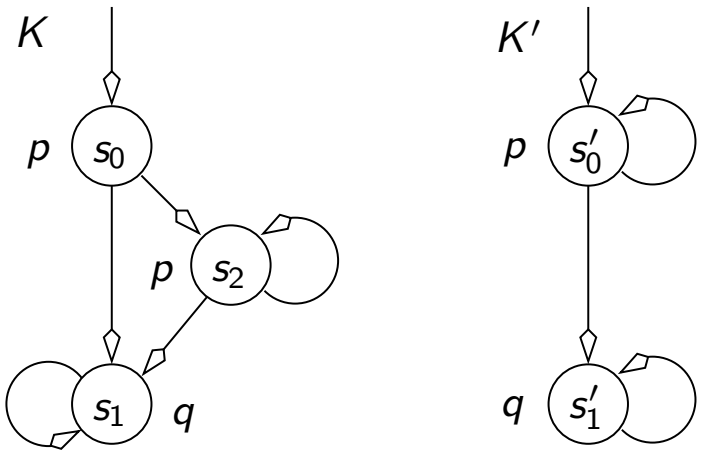


Informal Abstraction

- Given concrete model C and property φ
- Derive somehow an abstract model A from C
  - Abstraction should preserve detail relevant to φ
- Check whether $A \models \varphi$
- If $A \models \varphi$ confidence in correctness of C is increased
- If $A \not\models \varphi$ check counterexample
  - Is failure due to a real bug or an artifact of abstraction?
  - Either fix bug or refine abstraction

Formal Abstraction

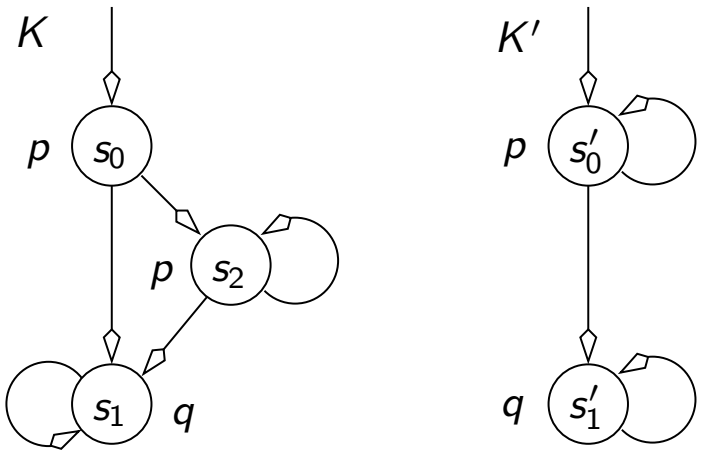
- Comes with warranty
- Simulation Relations



- bisimilarity preserves all of μ-calculus
- simulation equivalence preserves linear-time properties
- simulation preserves passing universal properties

Formal Abstraction

- Comes with warranty
- Simulation Relations



  - bisimilarity preserves all of μ-calculus
  - simulation equivalence preserves linear-time properties
  - simulation preserves passing universal properties

Simulation Relations

- Computing simulation relations is expensive
- Avoid direct computation
  - Cone of Influence (COI) Reduction and slicing yield a bisimilar model
  - The symmetry quotient induced by an invariance group is bisimilar to the original model
  - Predicate abstraction leads to bisimulation or simulation
  - Freeing some components leads to simulation
- Simulation is sufficient for language containment

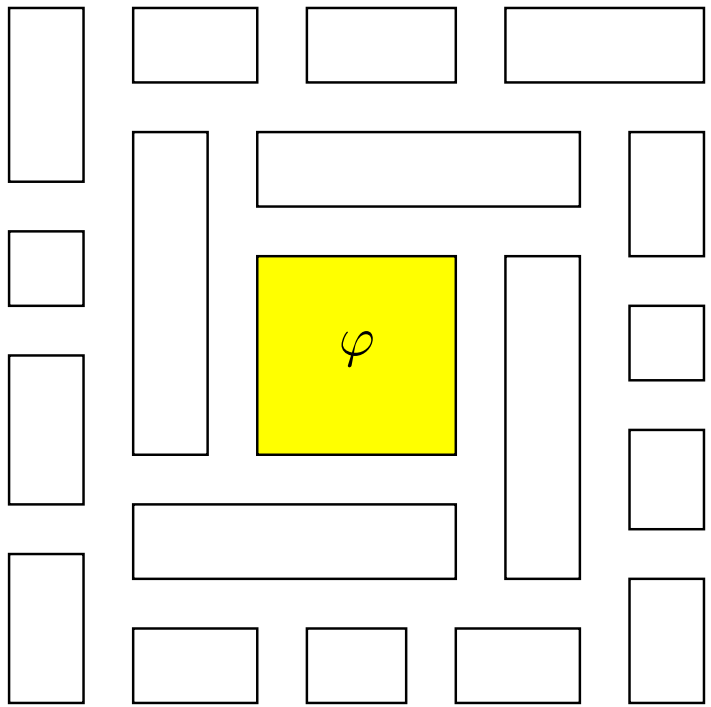
Property Driven Abstraction

Rule of thumb: The more you want to preserve, the less you can abstract

- Bisimulation does not allow much abstraction
- Property driven abstraction
  - focus on preserving only the property of interest
  - Start with a coarse abstraction
  - Let the property guide the initial abstraction and the refinement

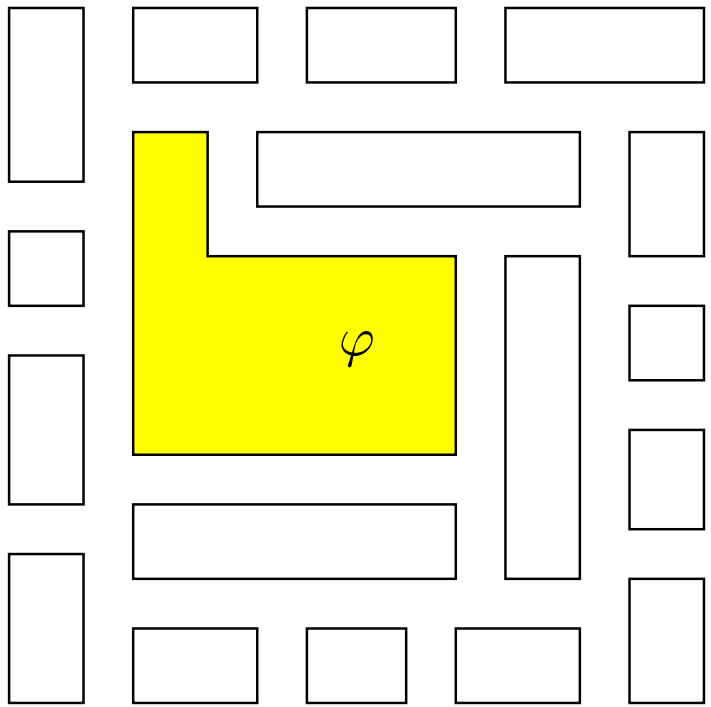
Abstraction Refinement

- What is not in the abstract model is ignored
- Refinement brings in something that was previously ignored



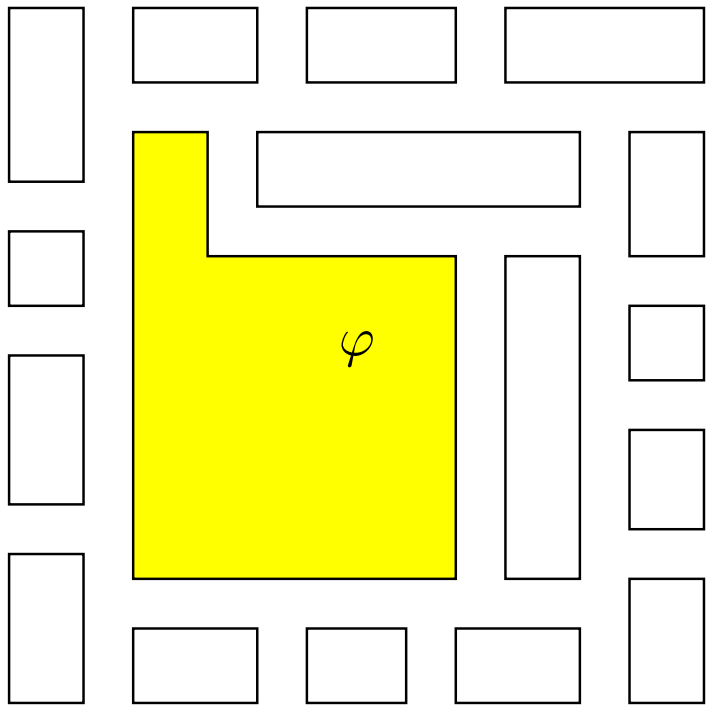
Abstraction Refinement

- What is not in the abstract model is ignored
- Refinement brings in something that was previously ignored



Abstraction Refinement

- What is not in the abstract model is ignored
- Refinement brings in something that was previously ignored



Abstraction Refinement

- For universal properties:
  - Choose initial abstract model
  - while the property fails
    - if concrete model also fails, report failure
    - refine
  - report success
- For mixed properties, use two abstractions
- If property fails on abstract model, an abstract counterexample is produced

Back to the Cube

- A cube is divided into 27 equal smaller cubes (blocks hereafter). Is it possible to trace a path from any point on the surface to the center that always travels parallel to some side of the cube and visits each block exactly once?
- If we number the smaller cubes consecutively, top-down, front to back, and left to right, we find that there are 14 odd-numbered blocks and 13 even-numbered blocks
- Since all adjacent blocks have different parity, we cannot find a path of length 27 ending with an even-numbered block.
- Since the center block is numbered 14, the answer is "no"
- The same argument applies whenever the side of the cube is divided into an odd number of parts, so that the number of blocks is $(2n+1)^3$ for some $n > 0$

Back to the Cube

- A cube is divided into 27 equal smaller cubes (blocks hereafter). Is it possible to trace a path from any point on the surface to the center that always travels parallel to some side of the cube and visits each block exactly once?
- If we number the smaller cubes consecutively, top-down, front to back, and left to right, we find that there are 14 odd-numbered blocks and 13 even-numbered blocks
- Since all adjacent blocks have different parity, we cannot find a path of length 27 ending with an even-numbered block.
- Since the center block is numbered 14, the answer is "no"
- The same argument applies whenever the side of the cube is divided into an odd number of parts, so that the number of blocks is $(2n+1)^3$ for some $n > 0$

Back to the Cube

- A cube is divided into 27 equal smaller cubes (blocks hereafter). Is it possible to trace a path from any point on the surface to the center that always travels parallel to some side of the cube and visits each block exactly once?
- If we number the smaller cubes consecutively, top-down, front to back, and left to right, we find that there are 14 odd-numbered blocks and 13 even-numbered blocks
- Since all adjacent blocks have different parity, we cannot find a path of length 27 ending with an even-numbered block.
- Since the center block is numbered 14, the answer is "no"
- The same argument applies whenever the side of the cube is divided into an odd number of parts, so that the number of blocks is $(2n+1)^3$ for some $n > 0$

Back to the Cube

- A cube is divided into 27 equal smaller cubes (blocks hereafter). Is it possible to trace a path from any point on the surface to the center that always travels parallel to some side of the cube and visits each block exactly once?
- If we number the smaller cubes consecutively, top-down, front to back, and left to right, we find that there are 14 odd-numbered blocks and 13 even-numbered blocks
- Since all adjacent blocks have different parity, we cannot find a path of length 27 ending with an even-numbered block.
- Since the center block is numbered 14, the answer is "no"
- The same argument applies whenever the side of the cube is divided into an odd number of parts, so that the number of blocks is $(2n+1)^3$ for some $n > 0$

Back to the Cube

- A cube is divided into 27 equal smaller cubes (blocks hereafter). Is it possible to trace a path from any point on the surface to the center that always travels parallel to some side of the cube and visits each block exactly once?
- If we number the smaller cubes consecutively, top-down, front to back, and left to right, we find that there are 14 odd-numbered blocks and 13 even-numbered blocks
- Since all adjacent blocks have different parity, we cannot find a path of length 27 ending with an even-numbered block.
- Since the center block is numbered 14, the answer is "no"
- The same argument applies whenever the side of the cube is divided into an odd number of parts, so that the number of blocks is $(2n+1)^3$ for some $n > 0$

Model Checking the Cube

We encode the problem with two state variables:

$$\texttt{posn} \in \{1, \ldots, 27\} \text{ and } \texttt{visited} \subseteq \{1, \ldots, 27\}$$

$$S = \{1, \ldots, 27\} \times 2^{\{1,\ldots,27\}}$$

$$S_0 = \{(p, \{p\}) \mid p \in \{1, \ldots, 27\} \setminus \{14\}\}$$

Let $\mathrm{adj}(p)$ return the set of positions adjacent to block p

$$T = \{((p, v), (p', v')) \mid (p, v) \in S \wedge p' \in \mathrm{adj}(p) \setminus v \wedge v' = v \setminus \{p'\}\}$$

Finally, the property:

$$\varphi = \mathrm{AG}(\mathrm{posn} = 14 \rightarrow \mathrm{visited} \neq \{1, \ldots, 27\}) \ .$$

In Verilog, we need 32 binary variables. Out of the 2^{32} states, $3.46426 \cdot 10^7$ are reachable and φ is proved in about one minute

Model Checking the Cube

We encode the problem with two state variables:

$$\texttt{posn} \in \{1, \ldots, 27\} \text{ and } \texttt{visited} \subseteq \{1, \ldots, 27\}$$

$$S = \{1, \ldots, 27\} \times 2^{\{1,\ldots,27\}}$$

$$S_0 = \{(p, \{p\}) \mid p \in \{1, \ldots, 27\} \setminus \{14\}\}$$

Let $\mathrm{adj}(p)$ return the set of positions adjacent to block p

$$T = \{((p, v), (p', v')) \mid (p, v) \in S \wedge p' \in \mathrm{adj}(p) \setminus v \wedge v' = v \setminus \{p'\}\}$$

Finally, the property:

$$\varphi = \mathrm{AG}(\mathrm{posn} = 14 \rightarrow \mathrm{visited} \neq \{1, \ldots, 27\}) \ .$$

In Verilog, we need 32 binary variables. Out of the 2^{32} states, $3.46426 \cdot 10^7$ are reachable and φ is proved in about one minute

Model Checking the Cube

We encode the problem with two state variables:

$$\texttt{posn} \in \{1, \ldots, 27\} \text{ and } \texttt{visited} \subseteq \{1, \ldots, 27\}$$

$$S = \{1, \ldots, 27\} \times 2^{\{1,\ldots,27\}}$$

$$S_0 = \{(p, \{p\}) \mid p \in \{1, \ldots, 27\} \setminus \{14\}\}$$

Let $\mathrm{adj}(p)$ return the set of positions adjacent to block p

$$T = \{((p, v), (p', v')) \mid (p, v) \in S \wedge p' \in \mathrm{adj}(p) \setminus v \wedge v' = v \setminus \{p'\}\}$$

Finally, the property:

$$\varphi = \mathsf{AG}(\mathrm{posn} = 14 \rightarrow \mathrm{visited} \neq \{1, \ldots, 27\}) \ .$$

In Verilog, we need 32 binary variables. Out of the 2^{32} states, $3.46426 \cdot 10^7$ are reachable and φ is proved in about one minute

Augmenting the Transition Relation

- We can do significantly better by keeping the same state encoding, initial states and property, but augmenting the transition relation
- Let $\mathrm{opp}(p)$ returns the set of positions in the cube that have parity opposite to block p. The transition relation can then be defined as before, but with $\mathrm{opp}(p)$ replacing $\mathrm{adj}(p)$
- The new relation has many more transitions. For instance, it is now possible to go from Block 1 to Block 6
- The number of reachable states correspondingly increases to $5.41574 \cdot 10^8$, but the model checking time decreases to less than a second
- Since φ is universal, the fact that it passes on the model with the augmented transition relation guarantees that it passes also on the original model

Augmenting the Transition Relation

- We can do significantly better by keeping the same state encoding, initial states and property, but augmenting the transition relation
- Let $\mathrm{opp}(p)$ returns the set of positions in the cube that have parity opposite to block p. The transition relation can then be defined as before, but with $\mathrm{opp}(p)$ replacing $\mathrm{adj}(p)$
- The new relation has many more transitions. For instance, it is now possible to go from Block 1 to Block 6
- The number of reachable states correspondingly increases to $5.41574 \cdot 10^8$, but the model checking time decreases to less than a second
- Since φ is universal, the fact that it passes on the model with the augmented transition relation guarantees that it passes also on the original model

Augmenting the Transition Relation

- We can do significantly better by keeping the same state encoding, initial states and property, but augmenting the transition relation
- Let $\mathrm{opp}(p)$ returns the set of positions in the cube that have parity opposite to block p. The transition relation can then be defined as before, but with $\mathrm{opp}(p)$ replacing $\mathrm{adj}(p)$
- The new relation has many more transitions. For instance, it is now possible to go from Block 1 to Block 6
- The number of reachable states correspondingly increases to $5.41574 \cdot 10^8$, but the model checking time decreases to less than a second
- Since φ is universal, the fact that it passes on the model with the augmented transition relation guarantees that it passes also on the original model

Augmenting the Transition Relation

- We can do significantly better by keeping the same state encoding, initial states and property, but augmenting the transition relation
- Let $\mathrm{opp}(p)$ returns the set of positions in the cube that have parity opposite to block p. The transition relation can then be defined as before, but with $\mathrm{opp}(p)$ replacing $\mathrm{adj}(p)$
- The new relation has many more transitions. For instance, it is now possible to go from Block 1 to Block 6
- The number of reachable states correspondingly increases to $5.41574 \cdot 10^8$, but the model checking time decreases to less than a second
- Since φ is universal, the fact that it passes on the model with the augmented transition relation guarantees that it passes also on the original model

Augmenting the Transition Relation

- We can do significantly better by keeping the same state encoding, initial states and property, but augmenting the transition relation
- Let $\mathrm{opp}(p)$ returns the set of positions in the cube that have parity opposite to block p. The transition relation can then be defined as before, but with $\mathrm{opp}(p)$ replacing $\mathrm{adj}(p)$
- The new relation has many more transitions. For instance, it is now possible to go from Block 1 to Block 6
- The number of reachable states correspondingly increases to $5.41574 \cdot 10^8$, but the model checking time decreases to less than a second
- Since φ is universal, the fact that it passes on the model with the augmented transition relation guarantees that it passes also on the original model

Changing State Space

The proof that there is no path refers to the parity of the block numbers and the total number of blocks of each parity.

$$\texttt{parity} \in \{0,1\},\ \texttt{visited0} \in \{1,\ldots,14\},\ \texttt{visitedE} \in \{1,\ldots,13\}$$

$$S = \{0,1\} \times \{1,\ldots,14\} \times \{1,\ldots,13\}$$

$$S_0 = \{(0,0,1),(1,1,0)\}$$

The transition relation is

$$\{((0,v_O,v_E),(1,v_O,v'_E)) \mid (0,v_O,v_E) \in S \wedge v_E < 13 \wedge v'_E = v_E{+}1\} \cup$$
$$\{((1,v_O,v_E),(0,v'_O,v_E)) \mid (1,v_O,v_E) \in S \wedge v_O < 14 \wedge v'_O = v_O{+}1\}$$

Transitions are possible to states of opposite parity with one of the two counters incremented (if it has not saturated).

Changing State Space

The proof that there is no path refers to the parity of the block numbers and the total number of blocks of each parity.

$$\texttt{parity} \in \{0,1\},\ \texttt{visited0} \in \{1,\ldots,14\},\ \texttt{visitedE} \in \{1,\ldots,13\}$$

$$S = \{0,1\} \times \{1,\ldots,14\} \times \{1,\ldots,13\}$$

$$S_0 = \{(0,0,1),(1,1,0)\}$$

The transition relation is

$$\{((0,v_O,v_E),(1,v_O,v'_E)) \mid (0,v_O,v_E) \in S \wedge v_E < 13 \wedge v'_E = v_E{+}1\} \cup$$
$$\{((1,v_O,v_E),(0,v'_O,v_E)) \mid (1,v_O,v_E) \in S \wedge v_O < 14 \wedge v'_O = v_O{+}1\}$$

Transitions are possible to states of opposite parity with one of the two counters incremented (if it has not saturated).

Changing the State Space

The property needs to be changed

$$\varphi_1 = \mathsf{AG}(\texttt{parity} = 0 \rightarrow (\texttt{visited0} \neq 14 \vee \texttt{visitedE} \neq 13))$$

Since $\texttt{parity} = 0$ corresponds to $\texttt{posn} \in \{2, 4, \ldots, 26\}$, our new property actually corresponds to

$$\varphi_2 = \mathsf{AG}(\texttt{posn} \in \{2, 4, \ldots, 26\} \rightarrow \texttt{visited} \neq \{1, \ldots, 27\})$$

If φ_2 is satisfied on the original model, φ is satisfied as well. The number of binary variables has been reduced from 32 to 9, there are only 53 reachable states, and verification takes negligible time. Interestingly, the following property also holds:

$$\varphi_1' = \mathsf{AG}(\texttt{parity} = 0 \rightarrow \texttt{visited0} \neq 14)$$

Changing the State Space

The property needs to be changed

$$\varphi_1 = \mathsf{AG}(\texttt{parity} = 0 \rightarrow (\texttt{visited0} \neq 14 \vee \texttt{visitedE} \neq 13))$$

Since $\texttt{parity} = 0$ corresponds to $\texttt{posn} \in \{2, 4, \ldots, 26\}$, our new property actually corresponds to

$$\varphi_2 = \mathsf{AG}(\texttt{posn} \in \{2, 4, \ldots, 26\} \rightarrow \texttt{visited} \neq \{1, \ldots, 27\})$$

If φ_2 is satisfied on the original model, φ is satisfied as well.

The number of binary variables has been reduced from 32 to 9, there are only 53 reachable states, and verification takes negligible time. Interestingly, the following property also holds:

$$\varphi_1' = \mathsf{AG}(\texttt{parity} = 0 \rightarrow \texttt{visited0} \neq 14)$$

Changing the State Space

The property needs to be changed

$$\varphi_1 = \mathsf{AG}(\texttt{parity} = 0 \rightarrow (\texttt{visited0} \neq 14 \vee \texttt{visitedE} \neq 13))$$

Since $\texttt{parity} = 0$ corresponds to $\texttt{posn} \in \{2, 4, \ldots, 26\}$, our new property actually corresponds to

$$\varphi_2 = \mathsf{AG}(\texttt{posn} \in \{2, 4, \ldots, 26\} \rightarrow \texttt{visited} \neq \{1, \ldots, 27\})$$

If φ_2 is satisfied on the original model, φ is satisfied as well. The number of binary variables has been reduced from 32 to 9, there are only 53 reachable states, and verification takes negligible time. Interestingly, the following property also holds:

$$\varphi_1' = \mathsf{AG}(\texttt{parity} = 0 \rightarrow \texttt{visited0} \neq 14)$$

Another Twist

In the reachable states of our model, the values of `visited0` and `visitedE` never differ by more than one in absolute value. If we let

$$\texttt{diff} = \texttt{visitedE} - \texttt{visited0}$$

then we can choose the following encoding:

$$\texttt{parity} \in \{0, 1\} \quad \text{and} \quad \texttt{diff} \in \{-1, 0, +1\}$$

Each state is a pair consisting of parity and difference:

$$S = \{0, 1\} \times \{-1, 0, +1\}$$

$$S_0 = \{(0, +1), (1, -1)\}$$

The transition relation is

$$\{((0, d), (1, d-1)) \mid d \in \{0, +1\}\} \cup \{((1, d), (0, d+1)) \mid d \in \{-1, 0\}\}$$

Another Twist

In the reachable states of our model, the values of `visited0` and `visitedE` never differ by more than one in absolute value. If we let

$$\texttt{diff} = \texttt{visitedE} - \texttt{visited0}$$

then we can choose the following encoding:

$$\texttt{parity} \in \{0, 1\} \quad \text{and} \quad \texttt{diff} \in \{-1, 0, +1\}$$

Each state is a pair consisting of parity and difference:

$$S = \{0, 1\} \times \{-1, 0, +1\}$$

$$S_0 = \{(0, +1), (1, -1)\}$$

The transition relation is

$$\{((0, d), (1, d-1)) \mid d \in \{0, +1\}\} \cup \{((1, d), (0, d+1)) \mid d \in \{-1, 0\}\}$$

Another Twist

Once again, we need to rewrite the property to be checked because the state encoding has changed:

$$\varphi_3 = \mathsf{AG}(\texttt{parity} = 0 \rightarrow \texttt{diff} \neq -1)$$

Since $\texttt{diff} \neq -1$ implies $\texttt{visited0} \neq 14 \vee \texttt{visitedE} \neq 13$, we have further strengthened our property.

Since φ_3 holds in our new model, as long as satisfaction of φ_3 in the new, 3-bit model implies satisfaction of φ_3 in the previos, 9-bit model, then we can conclude that φ holds in the original model.

There are four reachable states in the 3-bit model, and φ_3 is proved in almost no time.

Another Twist

Once again, we need to rewrite the property to be checked because the state encoding has changed:

$$\varphi_3 = \mathsf{AG}(\texttt{parity} = 0 \rightarrow \texttt{diff} \neq -1)$$

Since $\texttt{diff} \neq -1$ implies $\texttt{visited0} \neq 14 \vee \texttt{visitedE} \neq 13$, we have further strengthened our property.
Since φ_3 holds in our new model, as long as satisfaction of φ_3 in the new, 3-bit model implies satisfaction of φ_3 in the previos, 9-bit model, then we can conclude that φ holds in the original model.
There are four reachable states in the 3-bit model, and φ_3 is proved in almost no time.

Another Twist

Once again, we need to rewrite the property to be checked because the state encoding has changed:

$$\varphi_3 = \mathsf{AG}(\texttt{parity} = 0 \rightarrow \texttt{diff} \neq -1)$$

Since $\texttt{diff} \neq -1$ implies $\texttt{visited0} \neq 14 \vee \texttt{visitedE} \neq 13$, we have further strengthened our property.
Since φ_3 holds in our new model, as long as satisfaction of φ_3 in the new, 3-bit model implies satisfaction of φ_3 in the previos, 9-bit model, then we can conclude that φ holds in the original model.
There are four reachable states in the 3-bit model, and φ_3 is proved in almost no time.

State Encoding

If we use the following encoding:

$$\begin{aligned} \texttt{diff} = -1 &\rightarrow \texttt{diff}_1 = 0 \wedge \texttt{diff}_0 = 0 \\ \texttt{diff} = 0 &\rightarrow \texttt{diff}_1 = 0 \wedge \texttt{diff}_0 = 1 \\ \texttt{diff} = +1 &\rightarrow \texttt{diff}_1 = 1 \wedge \texttt{diff}_0 = 0 \end{aligned}$$

then, over all four reachable states, we have

$$\texttt{diff}_1 \leftrightarrow \texttt{parity} = 0 \wedge \texttt{diff}_0 = 0$$

We can replace one state bit with a combinational function of the other two. In the resulting model, φ_3 becomes

$$\mathrm{AG}(\texttt{parity} = 0 \rightarrow (\texttt{parity} = 0 \vee \texttt{diff}_0 = 1))$$

which simplifies to AG true; this property trivially holds.

State Encoding

If we use the following encoding:

$$\begin{aligned} \texttt{diff} = -1 &\rightarrow \texttt{diff}_1 = 0 \wedge \texttt{diff}_0 = 0 \\ \texttt{diff} = 0 &\rightarrow \texttt{diff}_1 = 0 \wedge \texttt{diff}_0 = 1 \\ \texttt{diff} = +1 &\rightarrow \texttt{diff}_1 = 1 \wedge \texttt{diff}_0 = 0 \end{aligned}$$

then, over all four reachable states, we have

$$\texttt{diff}_1 \leftrightarrow \texttt{parity} = 0 \wedge \texttt{diff}_0 = 0$$

We can replace one state bit with a combinational function of the other two. In the resulting model, φ_3 becomes

$$\mathrm{AG}(\texttt{parity} = 0 \rightarrow (\texttt{parity} = 0 \vee \texttt{diff}_0 = 1))$$

which simplifies to AG true; this property trivially holds.