# Lattice Automata for Program Analysis

**Mads Chr. Olesen** <mchro@cs.aau.dk>
joint work with
René Rydhof Hansen, Kim Guldstrand Larsen, Jiri Srba,
Andreas Engelbredt Dalsgaard and Petur Olsen

Department of Computer Science, Aalborg University

19th August 2010

# Model Checking or Static Analysis

## Model Checking

## Static Analysis

ala. the monotone framework

# Model Checking or Static Analysis

## Model Checking

- Explicit representation

## Static Analysis

ala. the monotone framework

- Summarizing representation

# Model Checking or Static Analysis

## Model Checking

- Explicit representation
- Path-based

## Static Analysis

ala. the monotone framework

- Summarizing representation
- Program-point based

# Model Checking or Static Analysis

## Model Checking

- Explicit representation
- Path-based
- State-space explosion

## Static Analysis

ala. the monotone framework

- Summarizing representation
- Program-point based
- Too coarse

# Model Checking or Static Analysis

## Model Checking

- Explicit representation
- Path-based
- State-space explosion

## Static Analysis

ala. the monotone framework

- Summarizing representation
- Program-point based
- Too coarse

## Lattice Automata

- Path-based
- Summarizing
- Counter-Example Guided Abstraction Refinement
- **Very much work in progress**

# Common Formalism: Lattice Automata

Generalises:

- Timed Automata
- Monotone Framework
- Boolean Programs

Advantages:

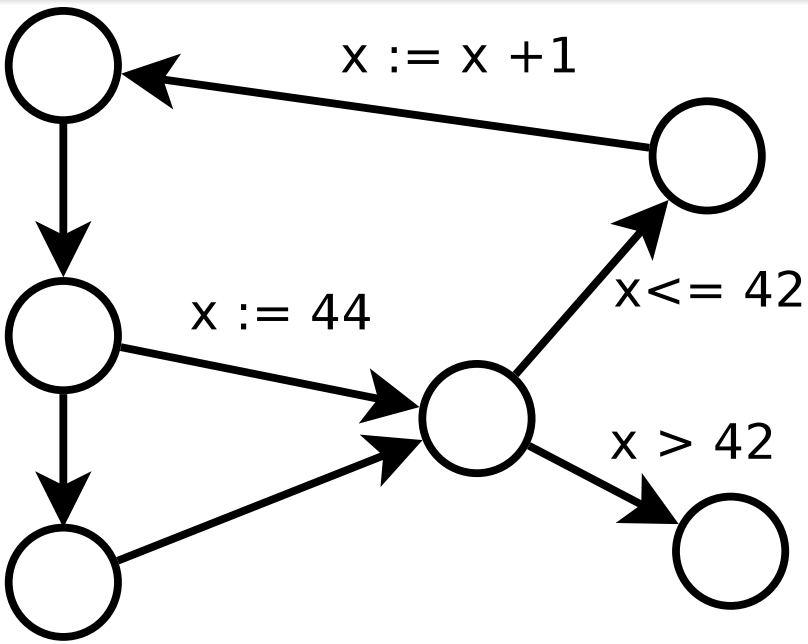
- Efficient common tools
- Technology transfer

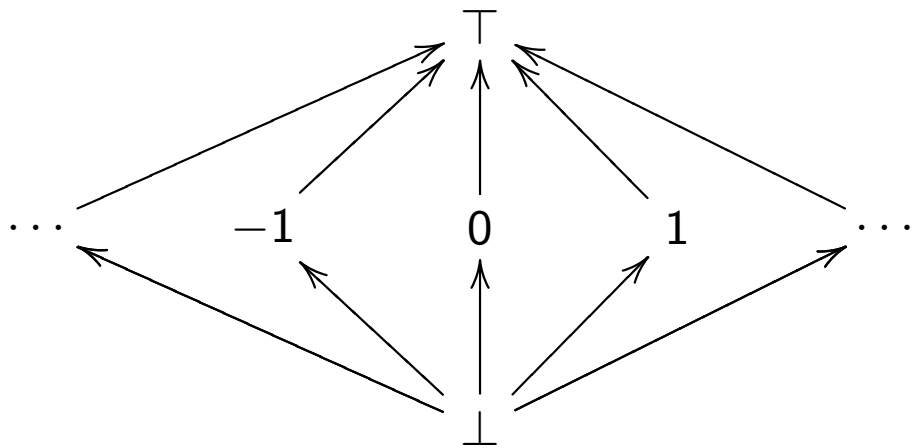
# Definition

## Definition (Lattice Automaton)

$\mathcal{T} = (S, \mathcal{L}, \longrightarrow)$

- $S$: finite set of states
- $\mathcal{L} = (L, \sqsubseteq)$ is a lattice of finite height
- $\longrightarrow \subseteq S \times L \times S \times L$ is the transition relation



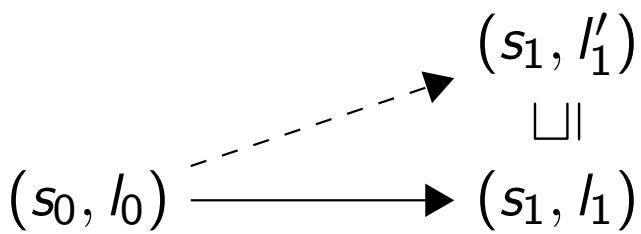
# Paths

## Definition (Path)

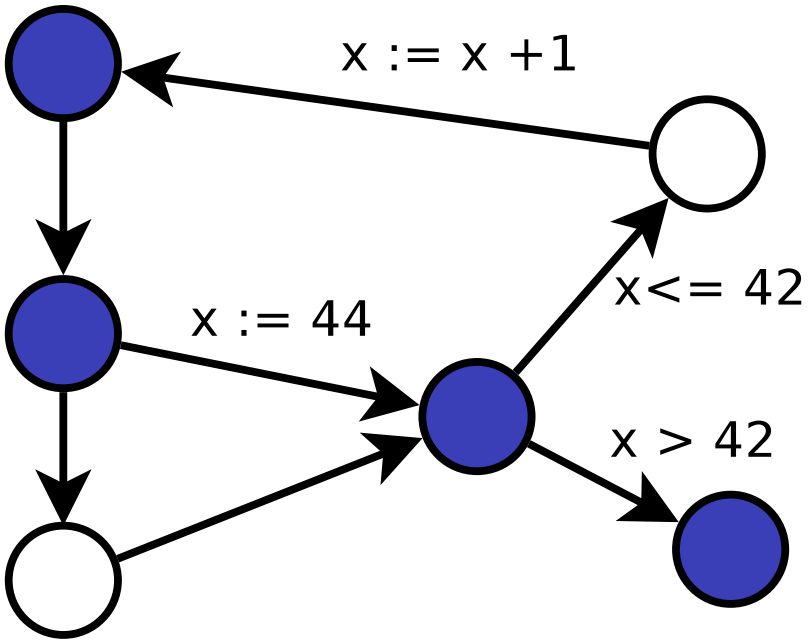
$\sigma = (s_0, \ell_0)(s_1, \ell_1) \cdots (s_n, \ell_n)$
such that $(s_i, \ell_i) \longrightarrow (s_{i+1}, \ell_{i+1})$

## Definition (Abstracted Path)

$\hat{\sigma} = (s_0, \ell_0)(s_1, \ell_1) \cdots (s_n, \ell_n)$
such that $\exists \ell'_{i+1} \in L : (s_i, \ell_i) \longrightarrow (s_{i+1}, \ell'_{i+1})$ and $\ell'_{i+1} \sqsubseteq \ell_{i+1}$



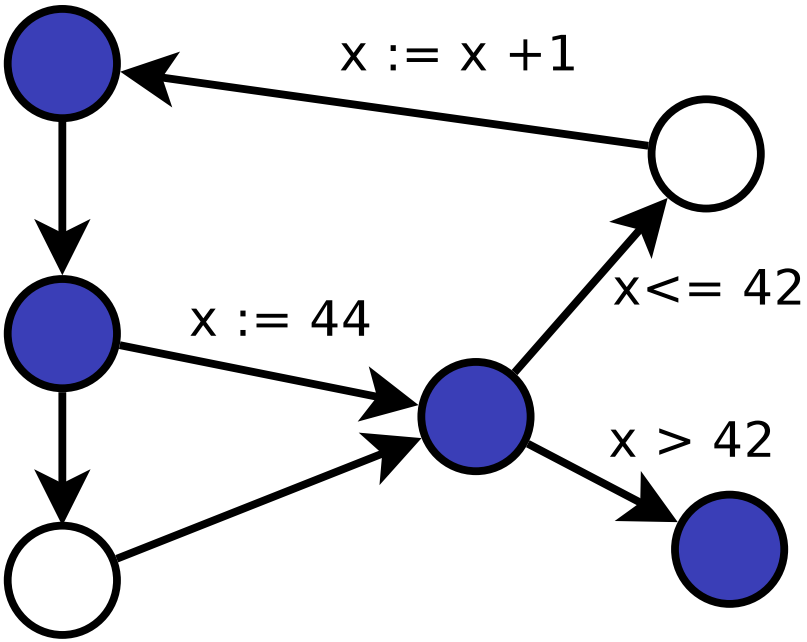
# Paths

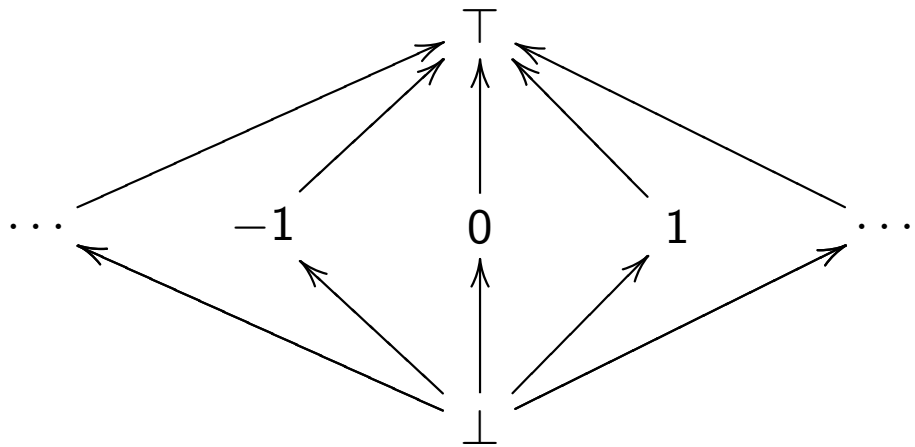



Concrete:

$(s_1, 0) \longrightarrow (s_2, 0) \longrightarrow (s_3, 44) \longrightarrow (s_4, 44)$

# Paths



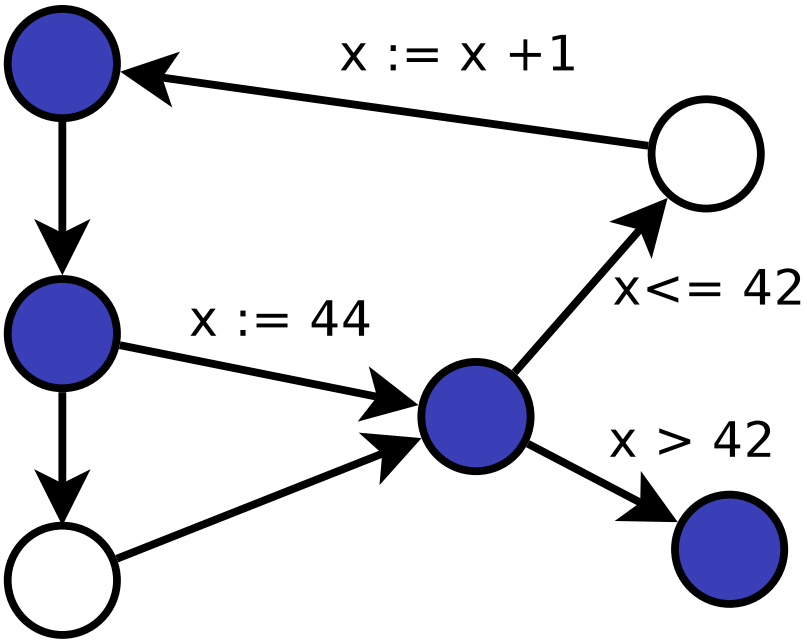

Concrete:

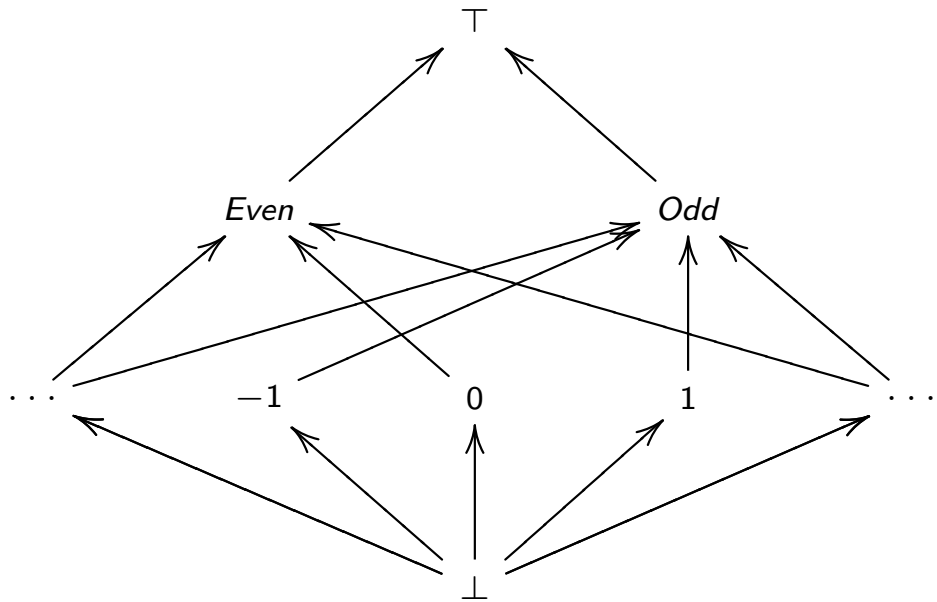
$(s_1, 0) \longrightarrow (s_2, 0) \longrightarrow (s_3, 44) \longrightarrow (s_4, 44)$

Abstracted:

$(s_1, 0) \longrightarrow (s_2, \top) \longrightarrow (s_3, 44) \longrightarrow (s_4, \top)$

# Paths




Concrete:

$(s_1, 0) \longrightarrow (s_2, 0) \longrightarrow (s_3, 44) \longrightarrow (s_4, 44)$

Abstracted:

$(s_1, 0) \longrightarrow (s_2, \top) \longrightarrow (s_3, 44) \longrightarrow (s_4, \top)$ or

$(s_1, 0) \longrightarrow (s_2, \mathit{Even}) \longrightarrow (s_3, \mathit{Even}) \longrightarrow (s_4, \mathit{Even})$

# "Model Checking"

Given $\mathcal{T} = (S, \mathcal{L}, \longrightarrow)$, initial configuration $(s_0, \ell_0)$, a goal state $s_g \in S$, is $s_g$ reachable?

1. Put $(s_0, \ell_0)$ on waiting list
2. Find immediate successors of some state on waiting list
   1. Is goal state? If so return "reachable"
   2. Else put on waiting list, if not seen before
3. return "Unreachable"

# "Model Checking"

Given $\mathcal{T} = (S, \mathcal{L}, \longrightarrow)$, initial configuration $(s_0, \ell_0)$, a goal state $s_g \in S$, is $s_g$ reachable?

1. Put $(s_0, \ell_0)$ on waiting list
2. Find immediate successors of some state on waiting list
   1. Is goal state? If so return "reachable"
   2. Else put on waiting list, if not seen before
3. return "Unreachable"

Paths concrete.

# "Static Analysis"

Given $\mathcal{T} = (S, \mathcal{L}, \longrightarrow)$, initial configuration $(s_0, \ell_0)$, a goal state $s_g \in S$, is $s_g$ reachable?
Worklist algorithm:

1. Put $s_0, \ldots s_n$ on worklist
2. For each program state: $\mathit{result}(s_i) = \bot$
3. Take $s_i$ of worklist:
   1. $\mathit{result}(s_i) = \mathit{result}(s_i) \sqcup \hat{\longrightarrow}(\mathit{result}(\text{ predecessors of } s_i))$
   2. Put successors of $s_i$ on worklist
4. If $\mathit{result}(s_g) \neq \bot$ return "Reachable"

# "Static Analysis"

Given $\mathcal{T} = (S, \mathcal{L}, \longrightarrow)$, initial configuration $(s_0, \ell_0)$, a goal state $s_g \in S$, is $s_g$ reachable?
Worklist algorithm:

1. Put $s_0, \ldots s_n$ on worklist
2. For each program state: $\mathit{result}(s_i) = \bot$
3. Take $s_i$ of worklist:
   1. $\mathit{result}(s_i) = \mathit{result}(s_i) \sqcup \hat{\longrightarrow}(\mathit{result}(\text{ predecessors of } s_i))$
   2. Put successors of $s_i$ on worklist
4. If $\mathit{result}(s_g) \neq \bot$ return "Reachable"

"Paths" abstracted.

# Counter-Example Guided Abstraction Refinement

Is error state reachable?

- Start as coarsely as possible ("static analysis")
- Given counter-example, determine if spurious
- Refine abstraction if spurious
- Check if reachable with new abstraction
- Continue until error state unreachable

# Heuristics

To use CEGAR, application specific heuristics needed:

## Definition (Path feasibility function)

Is abstracted path feasible
$pathfeasible : (S \times L)^* \times partitioning \rightarrow \{True, False\}$

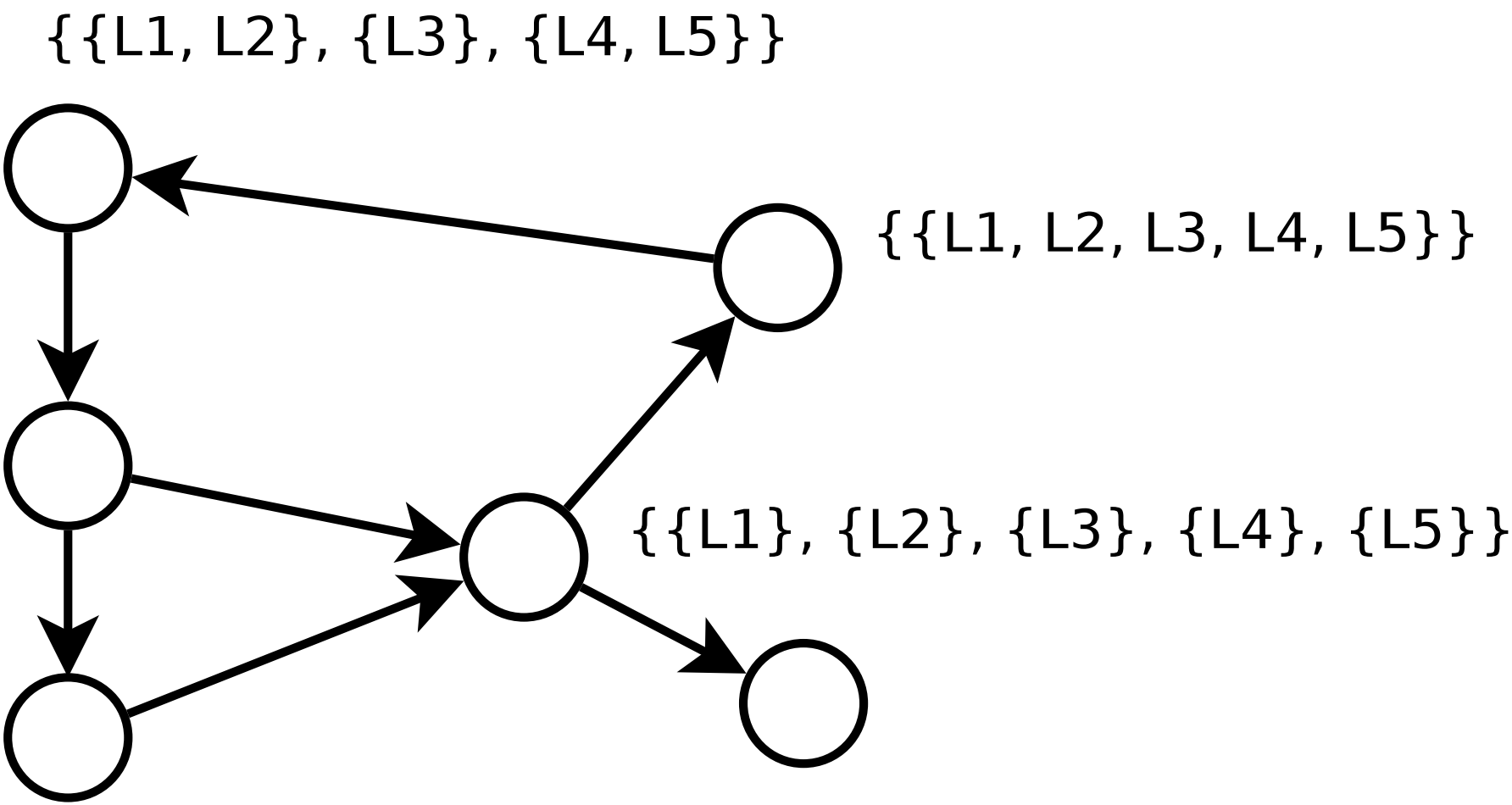
## Definition (Lattice partitioning)

Records how the current splitting/partitioning of lattice elements is for each state:

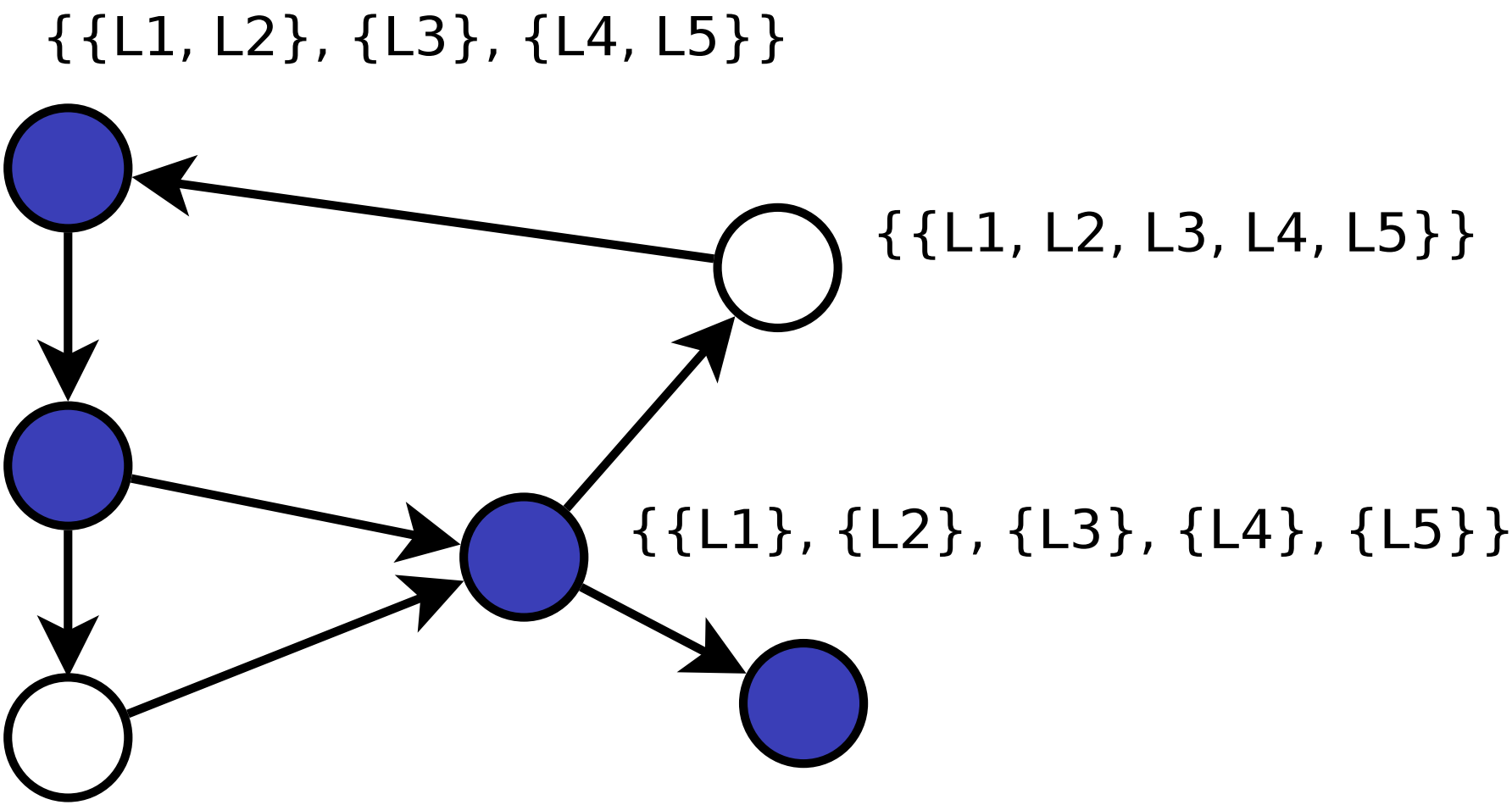
$$partitioning : S \rightarrow 2^{2^L}$$

such that at each state no overlap between partitions occur.

# Lattice Partitioning Example



# Lattice Partitioning Example




Path: $(s_1, \bigsqcup\{L1, L2\}) \longrightarrow (s_2, \top) \longrightarrow (s_3, L2) \longrightarrow (s_4, \top)$

# Infeasible path repartitioning

If path infeasible, need to re-partition:

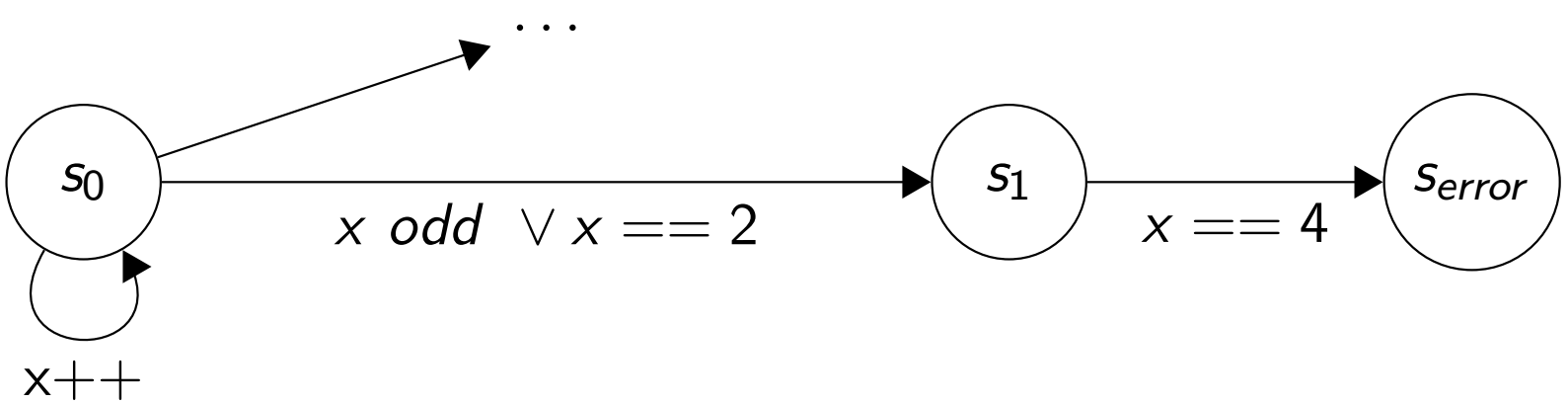
## Definition (Lattice repartitioning)

$$\mathit{repartition} : S \to 2^{2^L} \to 2^{2^L}$$

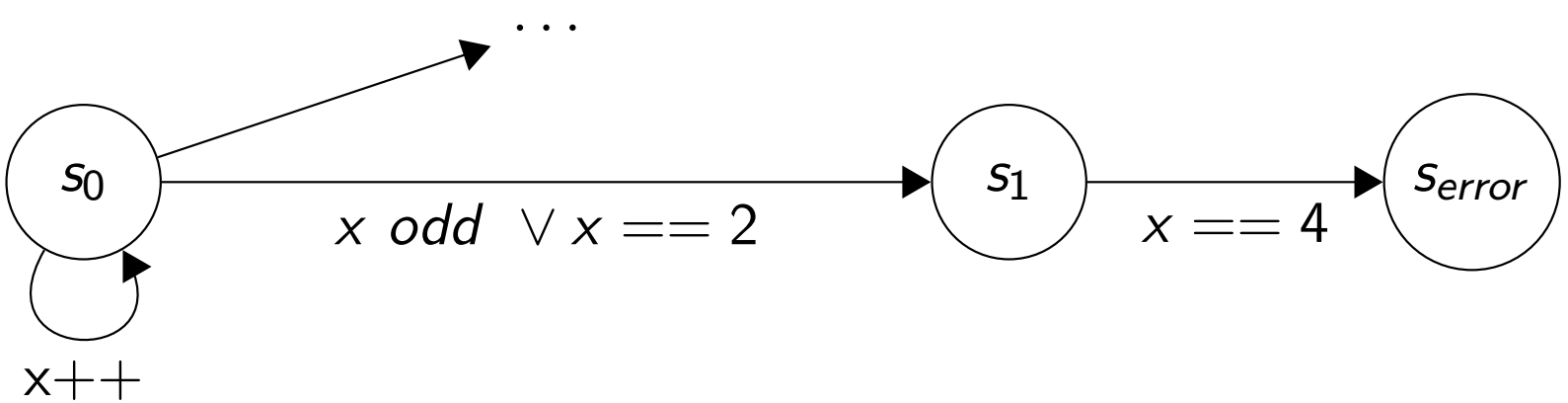
One strategy to guarantee termination:

- Repartitioning must respect previous choices
- Cannot move lattice element from one partition to another
- Only split partitions further

# Example



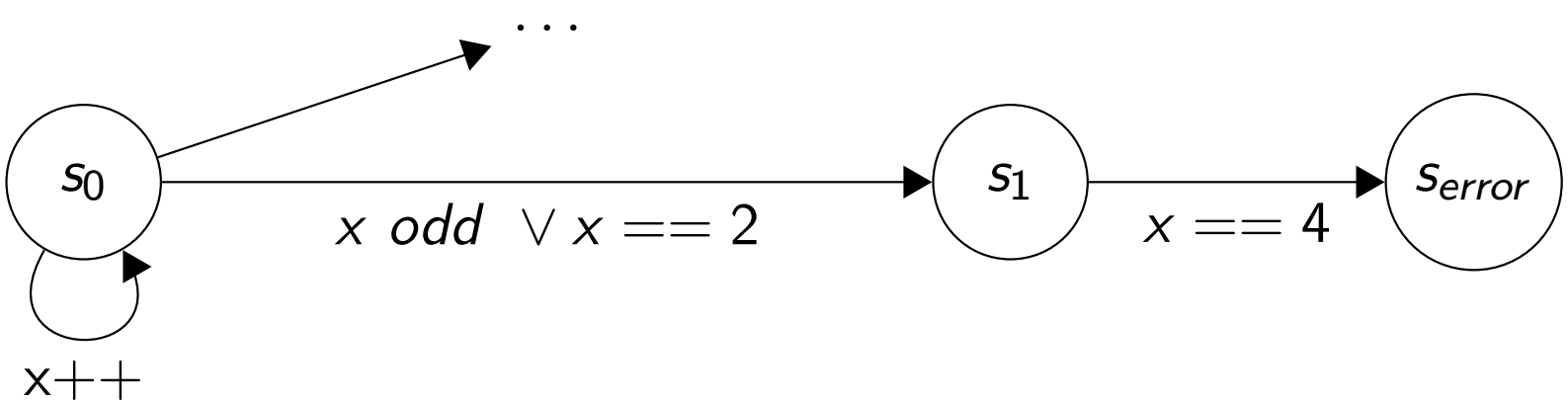
# Example




Initial: $partitioning(*) = \{L\}$

Error path: $(s_0, \top) \longrightarrow (s_1, \top) \longrightarrow (s_{error}, \top)$

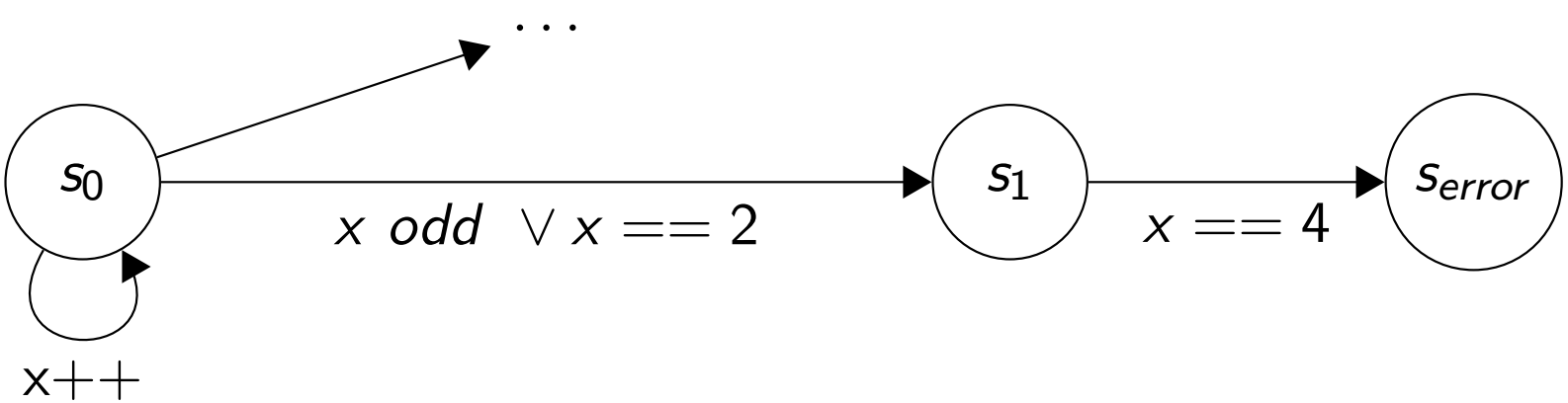
# Example




Initial: $\mathit{partitioning}(*) = \{L\}$
Error path: $(s_0, \top) \longrightarrow (s_1, \top) \longrightarrow (s_{error}, \top)$

Repartition: $\mathit{partitioning}(*) = \{\{\ell | \ell \sqsubseteq \mathit{Even}\}, \{\ell | \ell \sqsubseteq \mathit{Odd}\}, \{\top\}\}$
Error path: $(s_0, \mathit{Even}) \longrightarrow (s_1, \mathit{Even}) \longrightarrow (s_{error}, \mathit{Even})$

# Example




Initial: $\mathit{partitioning}(*) = \{L\}$
Error path: $(s_0, \top) \longrightarrow (s_1, \top) \longrightarrow (s_{error}, \top)$

Repartition: $\mathit{partitioning}(*) = \{\{\ell | \ell \sqsubseteq \mathit{Even}\}, \{\ell | \ell \sqsubseteq \mathit{Odd}\}, \{\top\}\}$
Error path: $(s_0, \mathit{Even}) \longrightarrow (s_1, \mathit{Even}) \longrightarrow (s_{error}, \mathit{Even})$

Repartition: $\mathit{partitioning}(*) =$
$\{\{\ell | \ell \sqsubseteq \mathit{Even} \wedge \ell \neq 2\}, \{2\}, \{\ell | \ell \sqsubseteq \mathit{Odd}\}, \{\top\}\}$
Possible lattice values at $s_1$: $\{\mathit{Odd}, 2\}$

# Current Status

- Working out the precise theory
- Proving soundness
- Criteria for termination (depends on heuristics)

Future Work:

- Implementing tool
- Making heuristics for boolean programs, timed automata
- Improving tool (Parallel, distributed)

Thank you for your attention!

Questions?