Detecting Unsolvability Based on Parity Functions

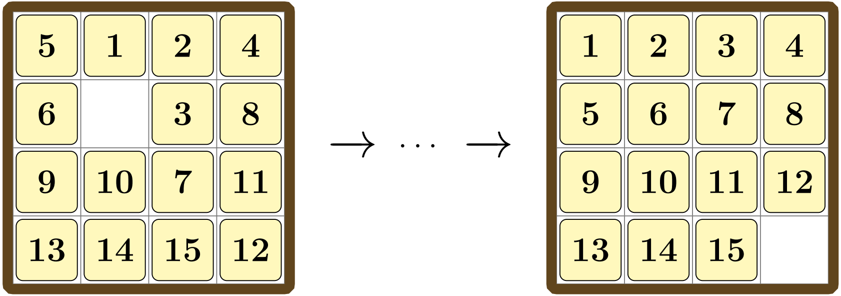
Remo Christen

Department of Mathematics and Computer Science
University of Basel

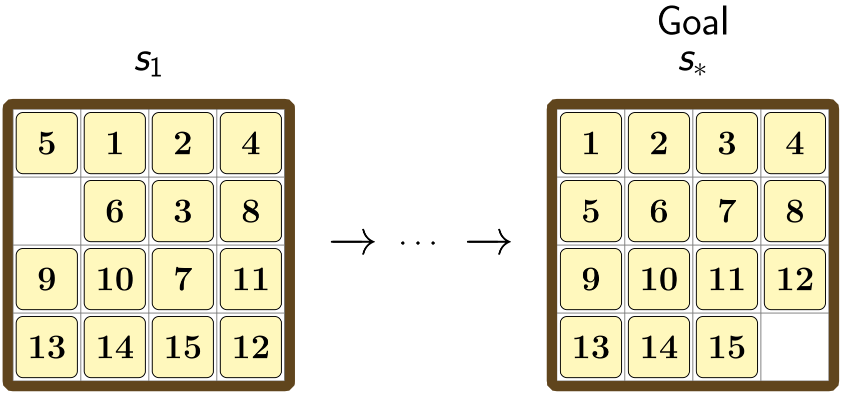
9. April 2021

Classical Planning – Example

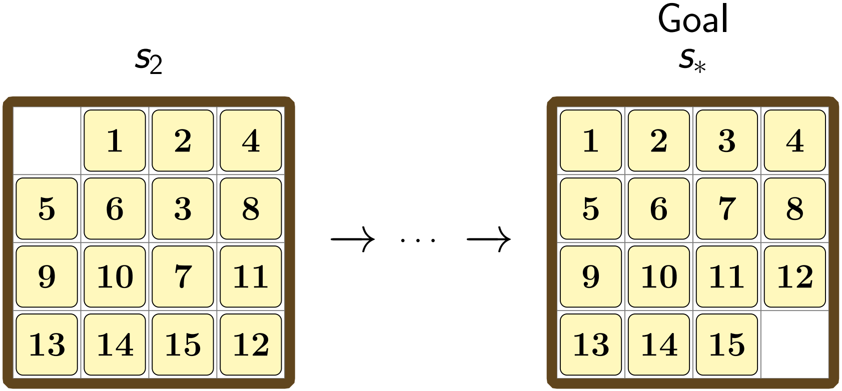
Initial state s_0 Goal s_*



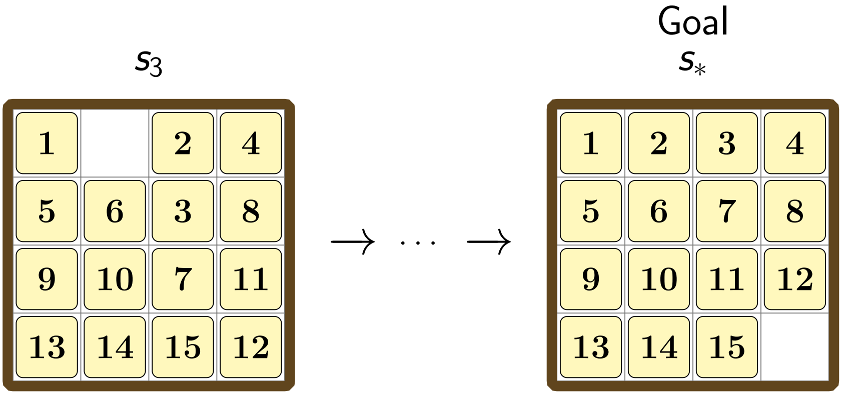
Classical Planning – Example



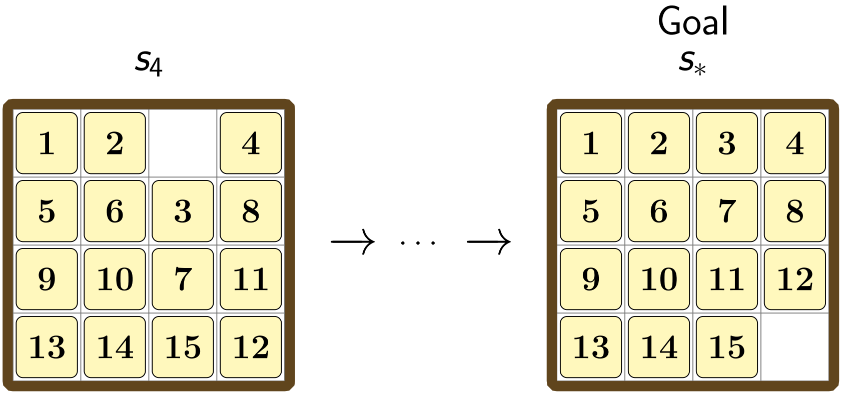
Classical Planning – Example



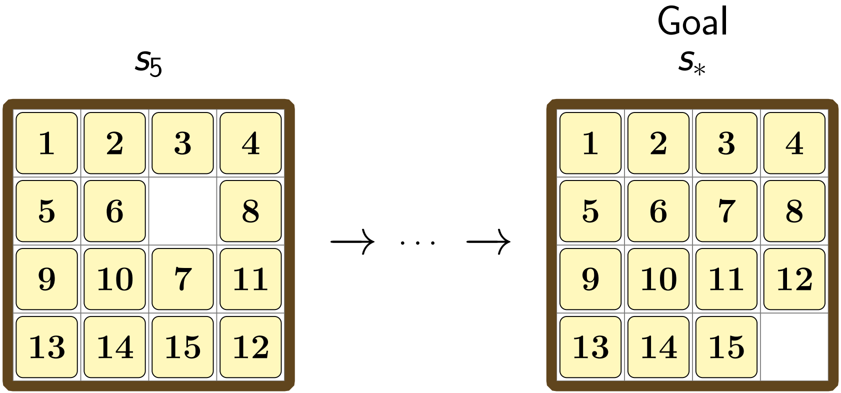
Classical Planning – Example



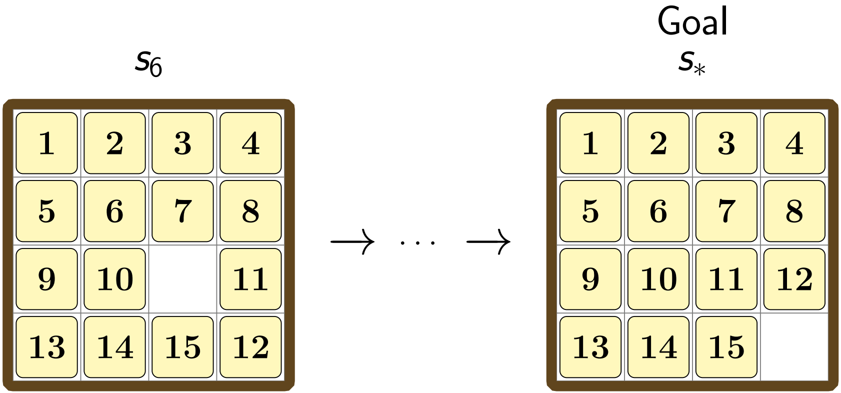
Classical Planning – Example



Classical Planning – Example

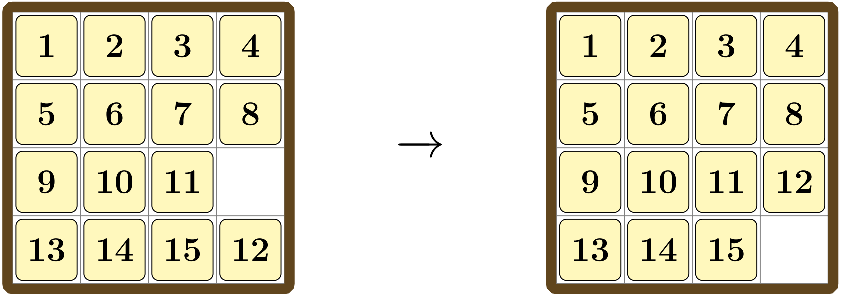


Classical Planning – Example



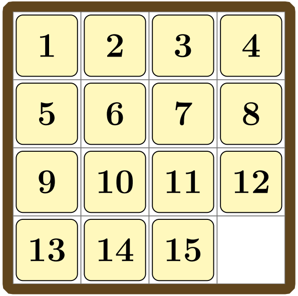
Classical Planning – Example

s_7 → Goal s_*



Classical Planning – Example

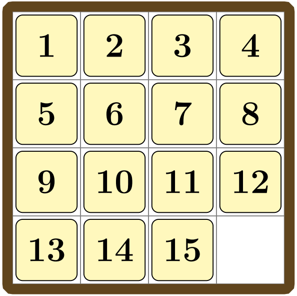
s_8



$=$

Goal

s_*

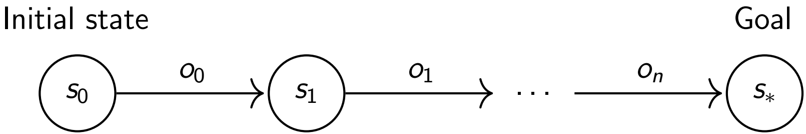


Classical Planning

Planning Task in TNF

$\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, s_* \rangle$ where

- $\mathcal{V}$ is a finite set of variables
- $\mathcal{O}$ is a finite set of operators with $vars(pre) = vars(eff)$
- s_0 is the initial state
- s_* is the goal state



Unsolvability

Ideal Outcomes of a Search

- Task is **solvable**, return a (optimal) plan.
- Task is provably **unsolvable**.

Our Outcomes

- Task is provably **unsolvable**.
- We don't know.

Unsolvability

Ideal Outcomes of a Search

- Task is **solvable**, return a (optimal) plan.
- Task is provably **unsolvable**.

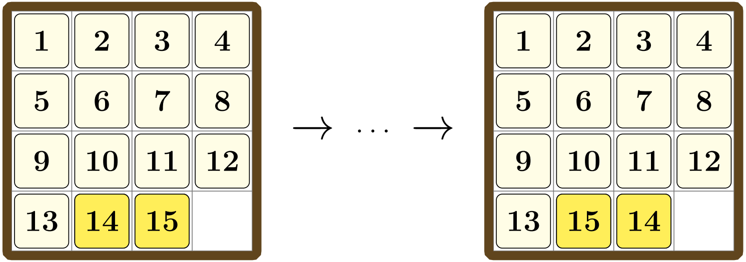
Our Outcomes

- Task is provably **unsolvable**.
- We don't know.

Q Can the 15 puzzle be unsolvable?

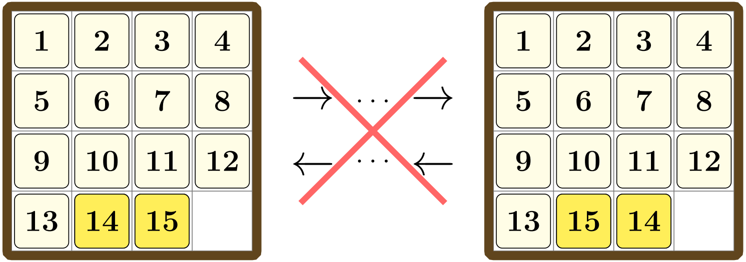
Unsolvability – Example

The 14-15 puzzle



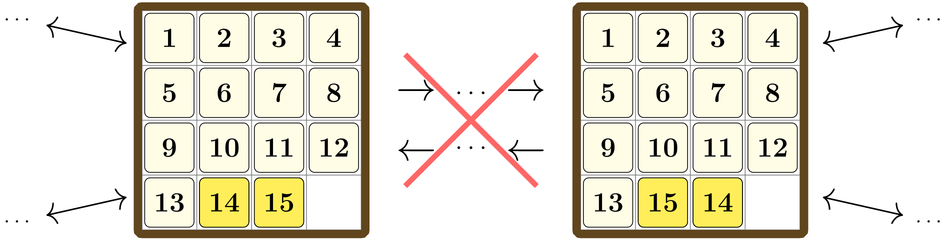
Unsolvability – Example

The 14-15 puzzle is **unsolvable**.



Unsolvability – Example

The 14-15 puzzle is **unsolvable**.



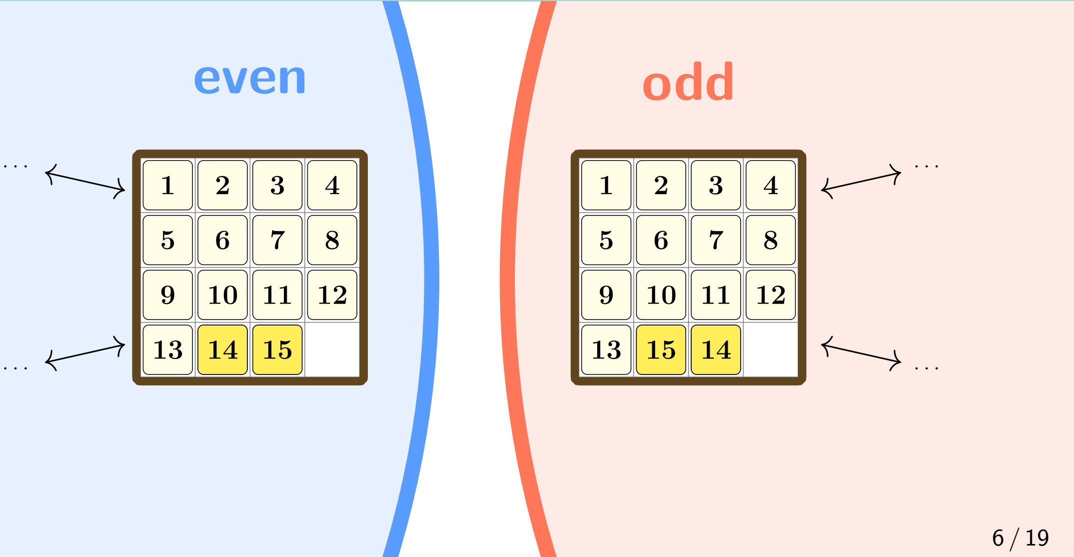
Q How can this be proven?

Parity Argument

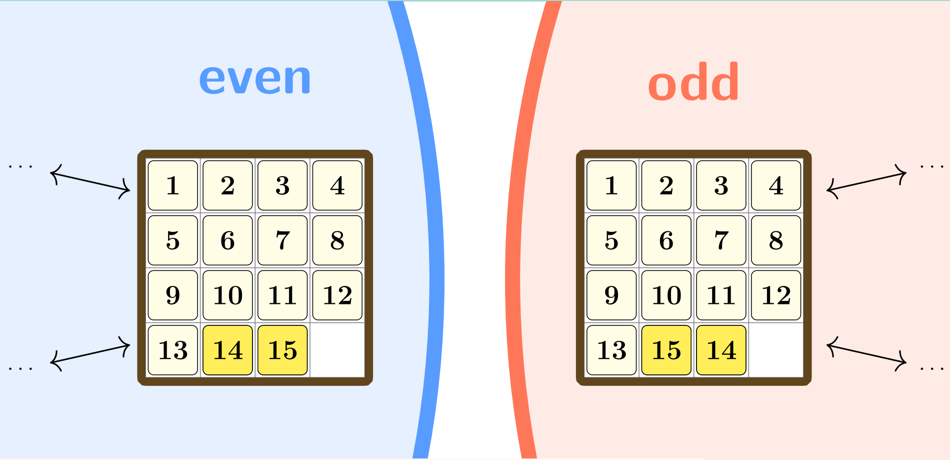
Parity arguments can be expressed as follows for a given task Π:

- Define parity function f with domain $\{0,1\}$ (conceptually $\{\text{even, odd}\}$).
- Ensure that f satisfies the following conditions:
  - $f(s_0) \neq f(s_*)$
  - $f(s) = f(s')$ for all transitions $s \to s'$
- Existence of f proves Π unsolvable.

Parity Argument – Example



Parity Argument – Example



Q How can we construct parity functions automatically?

Field F_2

We construct parity functions as potential functions over F_2.

- F_2 is the **smallest finite field** with two elements $\{0, 1\}$.
- F_2 captures the **parity property** of **integer arithmetic** over...

...addition, subtraction and multiplication.

even	$\pm$	even	$=$	even		even	$\times$	even	$=$	even
even	$\pm$	odd	$=$	odd		even	$\times$	odd	$=$	even
odd	$\pm$	odd	$=$	even		odd	$\times$	odd	$=$	odd

Field F_2

We construct parity functions as potential functions over F_2.

- F_2 is the **smallest finite field** with two elements $\{0, 1\}$.
- F_2 captures the **parity property** of **integer arithmetic** over...

...addition, subtraction and multiplication.

$$\underbrace{\begin{array}{ccccc} 0 & \pm & 0 & = & 0 \\ 0 & \pm & 1 & = & 1 \\ 1 & \pm & 1 & = & 0 \end{array}}_{\text{logical XOR}} \qquad \underbrace{\begin{array}{ccccc} 0 & \times & 0 & = & 0 \\ 0 & \times & 1 & = & 0 \\ 1 & \times & 1 & = & 1 \end{array}}_{\text{logical AND}}$$

Potential Functions over F_2

Potential Functions over $\mathbb{R}$

$\varphi(s) = \sum_{f \in \mathcal{F}} w(f) \cdot [s \models f]$ where

- s is a state
- $\mathcal{F}$ is a set of features (conjunctions of atoms)
- w is a weight function: $\mathcal{F} \mapsto \mathbb{R}$

Potential Functions over F_2

Potential Functions over ~~$\mathbb{R}$~~ F_2

~~$\varphi(s) = \sum_{f \in \mathcal{F}} w(f) \cdot [s \models f]$~~

$\varphi(s) = \bigoplus_{f \in \mathcal{F}} w(f) \wedge [s \models f]$ where

- s is a state
- $\mathcal{F}$ is a set of features (conjunctions of atoms)
- w is a weight function: ~~$\mathcal{F} \mapsto \mathbb{R}$~~ $\mathcal{F} \mapsto F_2$

To define a potential function, we must choose $\mathcal{F}$ and w.

Potential Functions over F_2

Potential Functions over ~~$\mathbb{R}$~~ F_2

~~$\varphi(s) = \sum_{f \in \mathcal{F}} w(f) \cdot [s \models f]$~~

$\varphi(s) = \bigoplus_{f \in \mathcal{F}} w(f) \wedge [s \models f]$ where

- s is a state
- $\mathcal{F}$ is a set of features (conjunctions of atoms)
- w is a weight function: ~~$\mathcal{F} \mapsto \mathbb{R}$~~ $\mathcal{F} \mapsto F_2$

To define a potential function, we must choose $\mathcal{F}$ and w.

Q How can we find potential functions that encode parity arguments?

Separation Constraints

Idea

Given a feature set $\mathcal{F}$, construct constraints such that a satisfying weight function results in a potential function encoding a parity argument.

Separation Constraints

$$\varphi(s_0) \neq \varphi(s_*)$$
$$\varphi(s) = \varphi(s') \qquad \text{for all transitions } s \to s'$$

Problem: solving constraints in the number of transitions is generally not feasible.

Separation Constraints

Idea

Given a feature set $\mathcal{F}$, construct constraints such that a satisfying weight function results in a potential function encoding a parity argument.

Separation Constraints

$$\varphi(s_0) \neq \varphi(s_*)$$
$$\varphi(s) = \varphi(s') \qquad \text{for all transitions } s \to s'$$

Problem: solving constraints in the number of transitions is generally not feasible.

Q Can we compute parity functions efficiently?

Theoretical Result

Result to be shown

Efficient constraints exist for up to **two-dimensional** features.

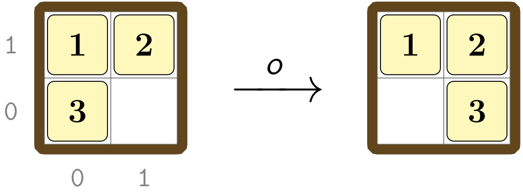
Approach

- Construct constraints for all **operators** instead of transitions.
- Difference in parity must be **independent** of s and s'.

Let Δ_t be $\varphi(s) \oplus \varphi(s')$ across transition $t = s \xrightarrow{o} s'$.

One-dimensional Features

- Only the atoms/features **mentioned** in operator o are **affected** by o.
- Δ_t is **fully determined** by o for all transitions $t = s \xrightarrow{o} s'$ (in TNF).

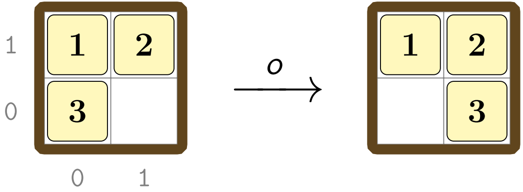


$$pre(o) = \{V_{0,0} \mapsto 3, V_{1,0} \mapsto \square\}$$

$$eff(o) = \{V_{0,0} \mapsto \square, V_{1,0} \mapsto 3\}$$

One-dimensional Features

- Only the atoms/features **mentioned** in operator o are **affected** by o.
- Δ_t is **fully determined** by o for all transitions $t = s \xrightarrow{o} s'$ (in TNF).



$$pre(o) = \{V_{0,0} \mapsto 3, V_{1,0} \mapsto \square\}$$
$$eff(o) = \{V_{0,0} \mapsto \square, V_{1,0} \mapsto 3\}$$

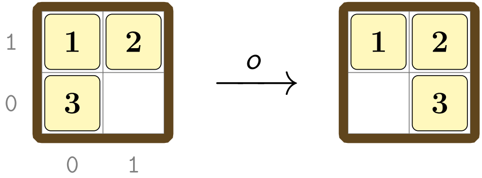
No other atoms/features are changed across any transition $s \xrightarrow{o} s'$.

Two-dimensional Features (I)

Given...

- atoms a and $\bar{a}$,
- variables $V = var(a)$ and $\bar{V} = var(\bar{a})$,
- feature $f = a \wedge \bar{a}$,
- and operator o,

there are **two relevant cases**:



1. $V, \bar{V} \in vars(o)$ e.g. $f = V_{0,0} \mapsto 3 \wedge V_{1,0} \mapsto \square$

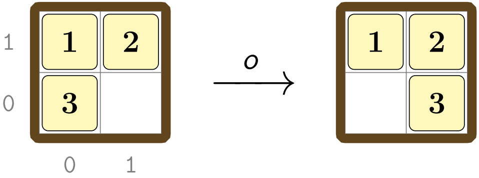
   ⟶ same as one-dimensional features

2. $V \in vars(o), \bar{V} \notin vars(o)$ e.g. $f = V_{0,0} \mapsto 3 \wedge V_{0,1} \mapsto 1$

   ⟶ more complex, see next slide

Two-dimensional Features (II)

Reminder: $f = a \wedge \bar{a}$,
$V = var(a) \in vars(o)$,
$\bar{V} = var(\bar{a}) \notin vars(o)$.



Operator o **cannot** determine Δ_t across $t = s \xrightarrow{o} s'$ because $[s \models \bar{a}]$ is **unknown**:

- $[s \models \bar{a}] \;\Rightarrow\; \Delta_t$ determined by a e.g. $f = V_{0,0} \mapsto 3 \wedge V_{0,1} \mapsto 1$
- $[s \not\models \bar{a}] \;\Rightarrow\; [s \not\models f]$ and $[s' \not\models f]$ e.g. $f = V_{0,0} \mapsto 3 \wedge V_{0,1} \mapsto 2$

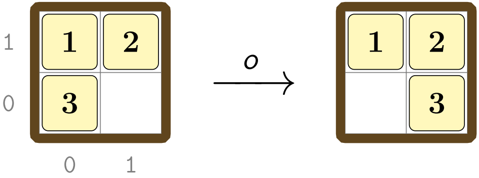
Solution

Ensure that both cases lead to the same contribution to Δ_t.

Two-dimensional Features (III)

For every variable $V \notin vars(o)$:

$$C = \bigoplus_{\substack{f = a \wedge \bar{a}, \\ a \in produced(o) \cup consumed(o)}} w(f) \quad \text{for all atoms } \bar{a} \in V$$

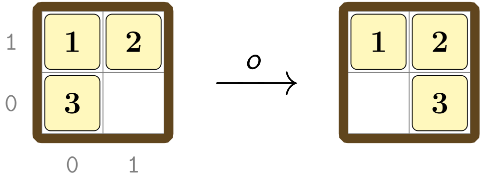


$$\begin{array}{l} V_{0,1} \mapsto 1 \wedge V_{0,0} \mapsto 3 \;\oplus \\ V_{0,1} \mapsto 1 \wedge V_{0,0} \mapsto \Box \;\oplus \\ V_{0,1} \mapsto 1 \wedge V_{1,0} \mapsto 3 \;\oplus \\ V_{0,1} \mapsto 1 \wedge V_{1,0} \mapsto \Box \end{array} = V_{0,1} \mapsto 2 \wedge \dots = V_{0,1} \mapsto 3 \wedge \dots = V_{0,1} \mapsto \Box \wedge \dots$$

Two-dimensional Features (III)

For every variable $V \notin vars(o)$:

$$C = \bigoplus_{\substack{f = a \wedge \bar{a}, \\ a \in produced(o) \cup consumed(o)}} w(f) \quad \text{for all atoms } \bar{a} \in V$$



Mutex Optimization

Skip atoms that are h^2-**mutex** with $pre(o)$ or $eff(o)$.

$$\begin{array}{l} V_{0,1} \mapsto 1 \wedge V_{0,0} \mapsto 3 \; \oplus \\ V_{0,1} \mapsto 1 \wedge V_{0,0} \mapsto \square \; \oplus \\ V_{0,1} \mapsto 1 \wedge V_{1,0} \mapsto 3 \; \oplus \\ V_{0,1} \mapsto 1 \wedge V_{1,0} \mapsto \square \end{array} = V_{0,1} \mapsto 2 \wedge \ldots = V_{0,1} \mapsto 3 \wedge \ldots = V_{0,1} \mapsto \square \wedge \ldots$$

Experimental Results

- Unsolvable **benchmark** with **19 domains**
  - `3unsat`, `bag-barman`, `bag-gripper`, `bag-transport`, `bottleneck`, `cave-diving`, `chessboard-pebbling`, `document-transfer`, `mystery`, `over-nomystery`, `over-rovers`, `over-tpp`, `pegsol`, `pegsol-row5`, `sliding-tiles`, `tetris`, `unsat-nomystery`, `unsat-rovers`, `unsat-tpp`
- Unsolvability **proven** by parity in **2 domains**
  - `pegsol`: 22/24 instances
  - `sliding-tiles`: 20/20 instances

Sliding Tiles Domain

	Size	#States	Time	Memory
`sliding-tiles` (20)	$3 \times 3/4$	$10^5/10^8$	2.5 s	59 931 KiB
15 Puzzles (100)	4×4	10^{13}	2.7 min	91 798 KiB
24 Puzzles (50)	5×5	10^{25}	2.0 h	637 952 KiB

Aidos

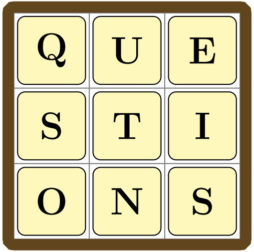
	Parity Arguments	Dead-End Potentials	Aidos 1
pegsol (24)	22	4	24
sliding-tiles (20)	20	–	10

- pegsol
  - Cyclic mod 2 property may be essential.
- sliding-tiles
  - Mutexes seem crucial.
  - Instances of size 3×4 and larger are hard.

Summary

- Parity arguments can **prove unsolvability**.
- They can be **automatically computed** as potential functions over F_2.
- **Compact constraints** exist for up to two-dimensional features.
- Parity arguments are useful for very **few domains**.
- When **suitable**, they can be **powerful**.

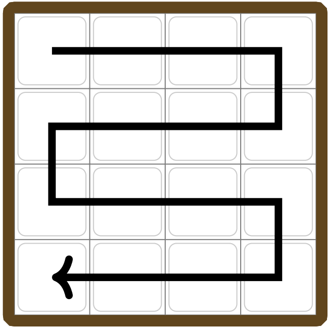
Questions



15 Puzzle Parity Argument

Parity argument for 15 puzzle instances:

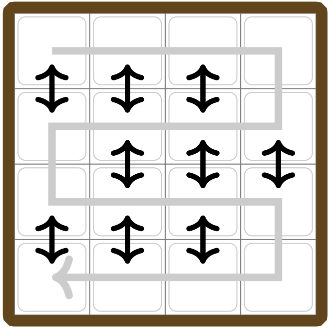
- → Define a **total order** over cells, ignoring blank.
- → Define s_0 to have no misorderings.
  - There are 18 moves that affect $\#\mathit{misorderings}$.
  - Difference in $\#\mathit{misorderings}$ is **always even**.
  - $\#\mathit{misorderings} \bmod 2$ is a **parity function** for the 14-15 puzzle $(s_0 \mapsto 0, s_* \mapsto 1)$.



15 Puzzle Parity Argument

Parity argument for 15 puzzle instances:

- Define a **total order** over cells, ignoring blank.
- Define s_0 to have no misorderings.
- → There are 18 moves that affect $\#\mathit{misorderings}$.
- Difference in $\#\mathit{misorderings}$ is **always even**.
- $\#\mathit{misorderings} \bmod 2$ is a **parity function** for the 14-15 puzzle $(s_0 \mapsto 0, s_* \mapsto 1)$.



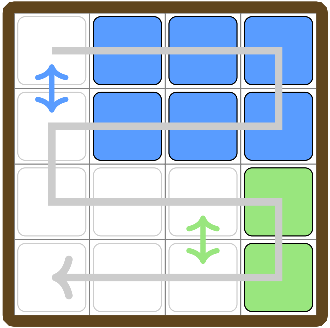
15 Puzzle Parity Argument

Parity argument for 15 puzzle instances:

- Define a **total order** over cells, ignoring blank.
- Define s_0 to have no misorderings.
- There are 18 moves that affect $\#$*misorderings*.

→ Difference in $\#$*misorderings* is **always even**.

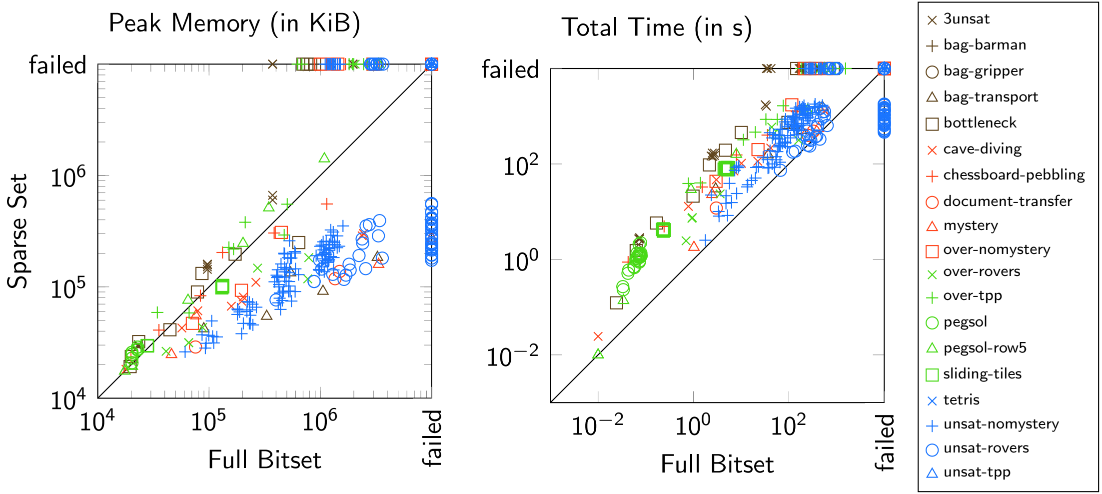
→ $\#$*misorderings* `mod` 2 is a **parity function** for the 14-15 puzzle $(s_0 \mapsto 0, s_* \mapsto 1)$.



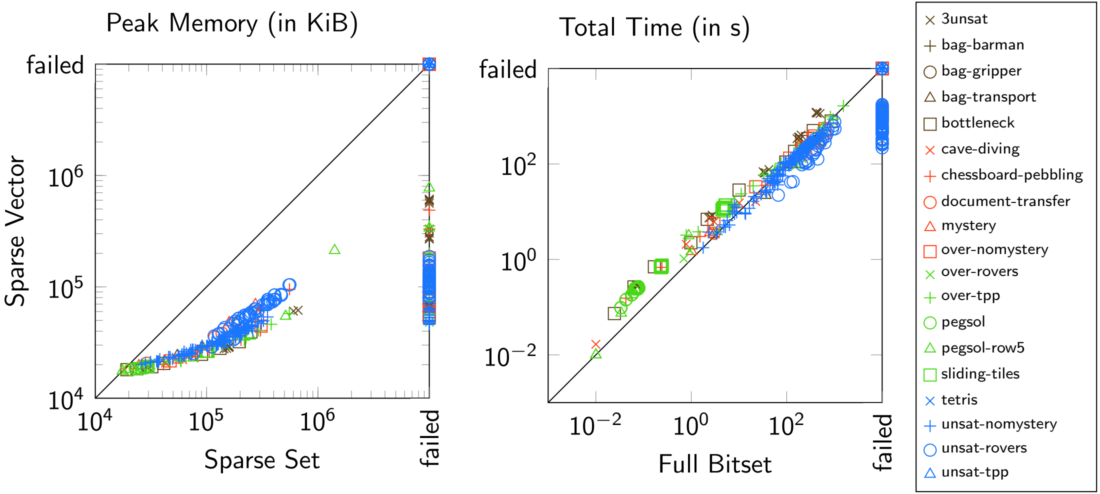
Outcomes of Best Implementation (Sparse Vector)

	By Parity	By h^2	Not			Time		Memory		Error
3unsat (30)	–	–	25		(+13)	5		–		–
bag-barman (20)	–	–	–			20		–		–
bag-gripper (25)	–	–	–			16		–		9
bag-transport (29)	–	15	2	(+1)	(+1)	5		7		–
bottleneck (25)	–	10	4		(+4)	11		–		–
cave-diving (25)	–	1	10	(+2)	(+2)	14		–		–
chessboard-pebbling (23)	–	–	9	(+2)	(+3)	13		1		–
document-transfer (20)	–	2	2			16		–		–
mystery (9)	–	9	–			–		–		–
over-nomystery (24)	–	2	9	(+2)	(+8)	13		–		–
over-rovers (20)	–	3	8	(+3)	(+4)	9		–		–
over-tpp (30)	–	1	13		(+7)	16		–		–
pegsol (24)	22	–	2			–		–		–
pegsol-row5 (15)	1	2	6	(+1)	(+2)	6		–		–
sliding-tiles (20)	20	–	–			–		–		–
tetris (20)	–	–	–			20		–		–
unsat-nomystery (150)	–	32	101	(+8)	(+30)	17		–		–
unsat-rovers (150)	–	62	40	(+32)	(+34)	48		–		–
unsat-tpp (25)	–	1	–			24		–		–
Sum (684)	43	140	231	(+51)	(+108)	253	(+215) (−74)	8	(−266) (−34)	9

Implementation Comparison – Sparse Set



Implementation Comparison – Sparse Vector



Sliding-tiles Domain – Full Results

	Size	#States	Impl.	Total Time	Peak Memory (in KiB)
`sliding-tiles` (geometric)	$3 \times 3/4$	$10^5/10^8$	Full	1.0 s	105 019
			Sparse	2.5 s	59 931
15 Puzzles (arithmetic)	4×4	10^{13}	Full	3.0 min	1 388 748
			Sparse	2.7 min	91 798
24 Puzzles (arithmetic)	5×5	10^{25}	Sparse	2.0 h	637 952

Pegsol Case Study