

The “Pattern Database” Relaxation

Heavily based on the slides of Jörg Hoffmann (MPI)

Outline

- *General Idea*
- Domain Abstractions
- Pattern Databases
- Disjoint Pattern Databases
- Pattern Databases in Strips

Abstractions: Embeddings and Homomorphisms

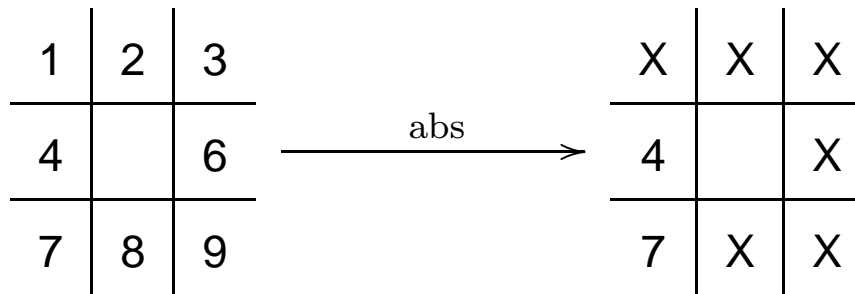
Abstraction: Replacing one state space by another that is easier to search

- **embeddings:** earliest and most commonly studied type of abstraction
 - informally, a problem transformation is an embedding if it **adds edges** to the state-space graph
 - example for planning: dropped preconditions (*other suggestions?*)
- **homomorphism:** another central type of abstraction (of our interest today)
 - informally, homomorphism groups together several states to create a single **abstract state**
 - example for planning: drop a state variable

The Idea of “Pattern Database” Relaxation

1. Define a mapping abs that maps a real problem Π into an **abstract problem** Π^{abs} with a (significantly) smaller state space
 - That is, **ignore** some aspects of the world states
 - That is, take into account only a **pattern** of the real structure
2. Important requirement from the abstraction abs is that paths between states should only get shorter (\Rightarrow **admissibility**)
3. To generate the heuristic function, solve the abstract problem **optimally**
4. Do this just once before search starts for **all** the (possibly relevant) abstract states, and store this complete set of heuristic values in a lookup table (\Rightarrow **database**)

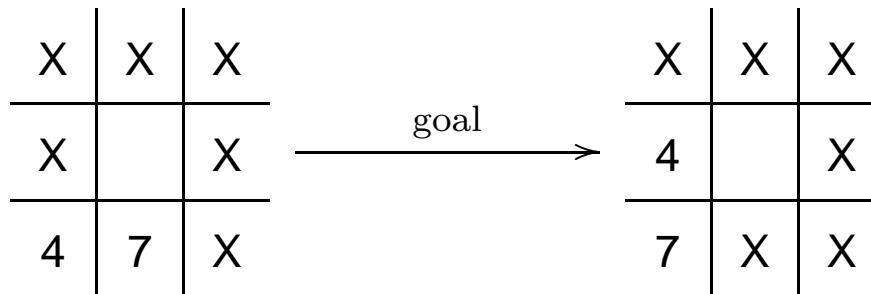
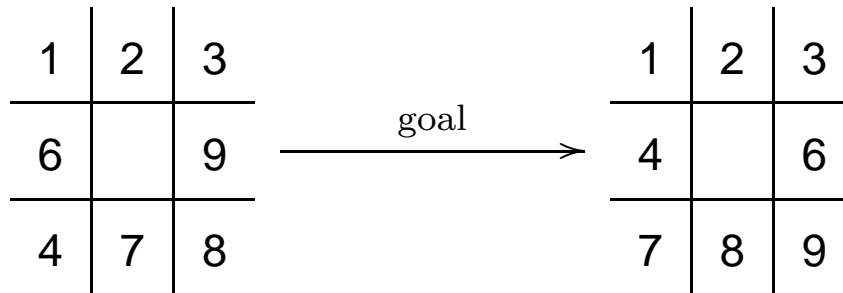
Example for 8-Puzzle



- Replace all but a subset of the tiles with X (that is, do not distinguish between these tiles)
- Every solution is also an abstract solution

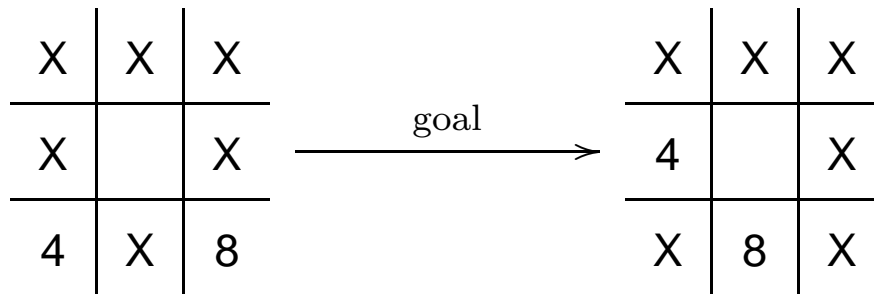
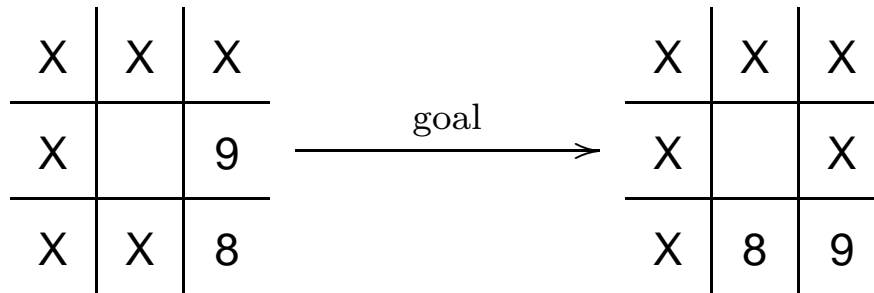
Example for 8-Puzzle

How do we get from left to right?



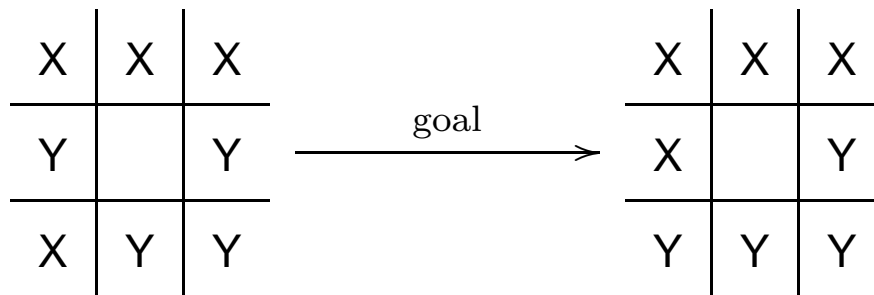
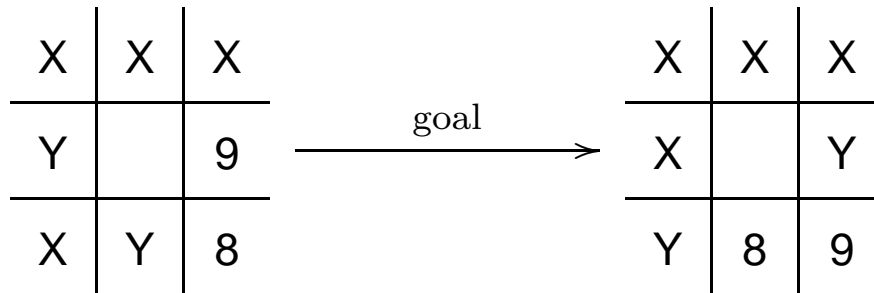
Example for 8-Puzzle

Different abstraction functions are possible:



Example for 8-Puzzle

Abstraction functions can be more complex:



Example of a Pattern Database for 8-Puzzle

Pattern database:

	X	X
X	X	X
X	X	X

2

X		X
X	X	X
X	X	X

1

X	X	
X	X	X
X	X	X

2

X	X	X
	X	X
X	X	X

1

X	X	X
X		X
X	X	X

0

X	X	X
X	X	
X	X	X

1

X	X	X
X	X	X
	X	X

2

X	X	X
X	X	X
X		X

1

X	X	X
X	X	X
X	X	

2

The “Pattern Database” Relaxation: Observations

There are many possible abstraction functions.

- Different abstraction functions yield different heuristics.
- The less coarse the abstraction, the more precise the heuristic
- What happens in the extreme cases:
 - Everything is mapped to the same symbol
 - Nothing abstracted away

A nice and very important property of admissible heuristics:

- ♠ We can combine numerous admissible heuristics by taking the **maximum**, and this combination preserves admissibility

The “Pattern Database” Relaxation: Observations

In PDH we do not map the real problem into a computationally (worst-case) simpler problem, but only reduce the size parameters.

- Useful (yet computationally reasonable) abstract state spaces are typically large
 - No point in solving the abstract problem at each search state
 - Create a complete pattern database off-line, and use it online
 - Q: *when will this turn out not to be a good idea?*
- We can keep several databases in memory
 - Q: *why this can be a good idea?*

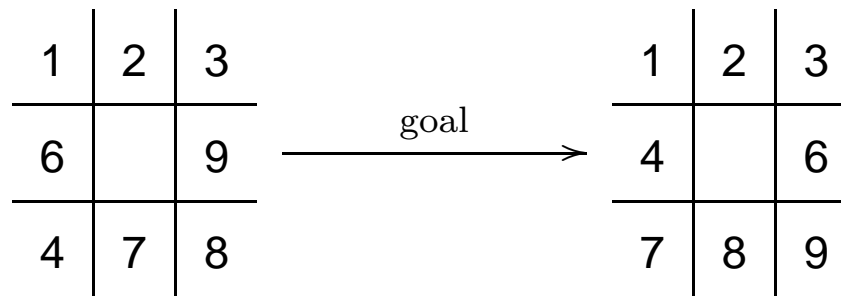
Outline

- General Idea
- *Domain Abstractions*
- Pattern Databases
- Disjoint Pattern Databases
- Pattern Databases in Strips

Multi-valued Classical Planning (MVStrips)

- Set V of variables v_1, \dots, v_n with finite domains $dom(v_i)$
 - For ease of presentation, assume that $dom(v_i) = dom$ for all v_i
- States are complete value assignments to the variables V
- Actions a have
 - a precondition $Pre(a) = \{v_{a_1} = c_{a_1}, \dots, v_{a_k} = c_{a_k}\}$
 - an effect $Eff(a) = \{v_{a_1} = c'_{a_1}, \dots, v_{a_k} = c'_{a_k}\}$
- Goal G : a partial assignment to V

Example 8-Puzzle



Encoding: variables are fields, values are tiles.

- Variables: p_i for $1 \leq i \leq 9$,
 $dom = \{\text{blank}, t_1, t_2, t_3, t_4, t_6, t_7, t_8, t_9\}$
- Actions: $move(t_i, p_j, p_k)$ (for neighbors p_j and p_k):
 $Pre = \{p_j = t_i, p_k = \text{blank}\}$, $Eff = \{p_j = \text{blank}, p_k = t_i\}$
- Initial state, goal: ...

(Variable) Domain Abstractions

Let $\langle V, A, I, G \rangle$ be an MVStrips task with variable domain dom .

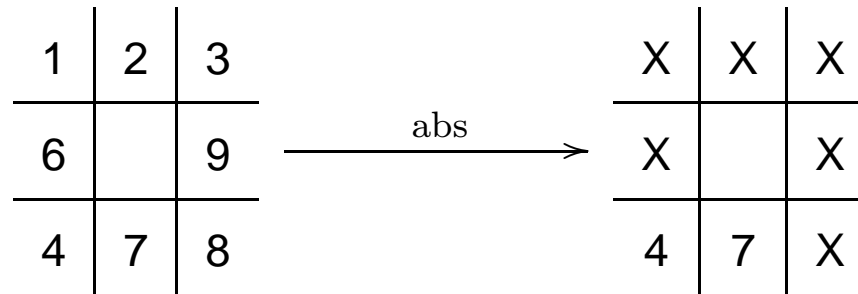
A **domain abstraction** is a function

$$^{abs} : dom \rightarrow absdom$$

where $|absdom| \leq |dom|$.

- dom is mapped into a set $absdom$ that allows **less distinctions**
- definition of abs is recursively extended to preconditions/effects/actions/tasks

Example: 8-Puzzle

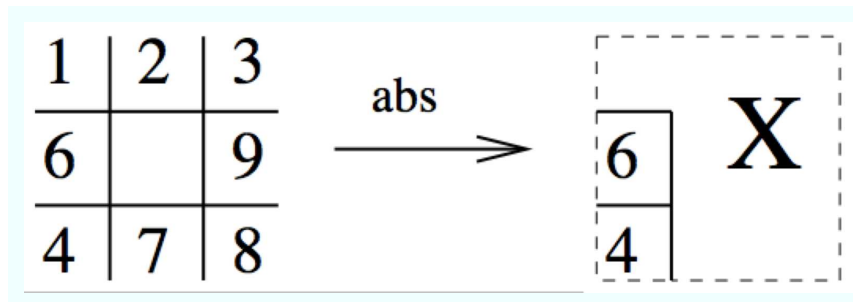


- $dom = \{\text{blank}, t_1, t_2, t_3, t_4, t_6, t_7, t_8, t_9\}$
 $absdom = \{\text{blank}, X, t_4, t_7\}$
- $\{\text{blank}, t_1, t_2, t_3, t_4, t_6, t_7, t_8, t_9\}^{abs} =$
 $\{\text{blank}, X, X, X, t_4, X, t_7, X, X\}$
- Actions: e.g., $move(t_6, p_j, p_k)^{abs} = move(X, p_j, p_k)$:
 $Pre = \{p_j = X, p_k = \text{blank}\}, Eff = \{p_j = \text{blank}, p_k = X\}$

(Different) Example: 8-Puzzle

Encoding variables as tiles and fields (positions) as values

- $dom = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9\}$
 $absdom = \{X, p_4, p_7\}$
- $\{p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9\}^{abs} =$
 $\{X, X, X, p_4, X, X, p_7, X, X\}$



This is a bit less intuitive, but the same formalism.

State Space Homomorphisms

Proposition

Let $\langle V, A, I, G \rangle$ be an MVStrips task, and ${}^{\text{abs}}$ be a domain abstraction. If s is a state in the task, and a an action applicable in s , then a^{abs} is applicable in s^{abs} , and

$$f(s, a)^{\text{abs}} = f(s^{\text{abs}}, a^{\text{abs}})$$

Proof: Blackboard

Domain abstractions as we defined them are **homomorphisms**

State Space Homomorphisms

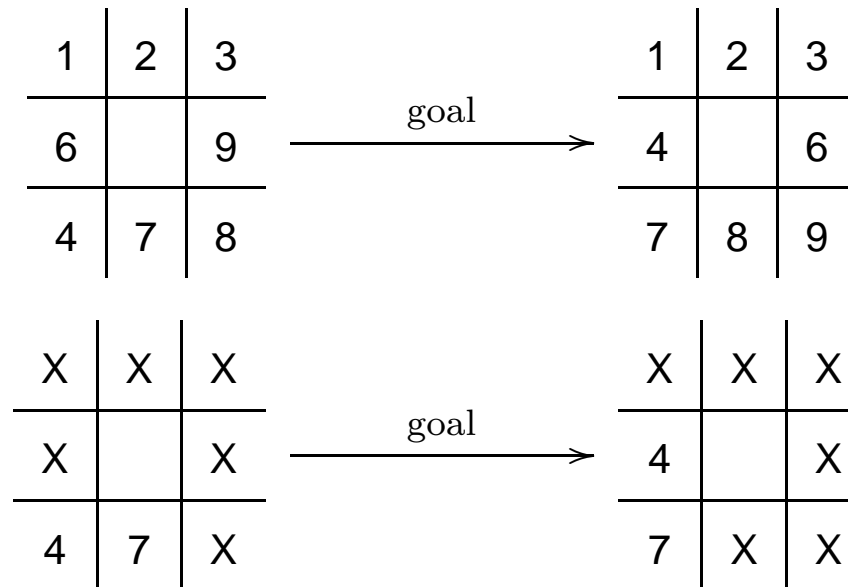
Proposition

Let $\langle V, A, I, G \rangle$ be an MVStrips task, and ${}^{\text{abs}}$ be a domain abstraction. Let s_0 and s_m be a pair of states in the task, m be the length of the shortest path (optimal plan) from s_0 to s_m , and m^{abs} be the length of the shortest path (optimal plan) from s_0^{abs} to s_m^{abs} . Then we have $m^{\text{abs}} \leq m$

Proof: *Suggestions?*

State Space Homomorphisms

- Let $\langle a_1, \dots, a_m \rangle$ be a shortest path from s_0 to s_m . Denote by s_0, \dots, s_m the states on this path.
- It may happen that there is a **shorter** path than $\langle a_1^{\text{abs}}, \dots, a_m^{\text{abs}} \rangle$ from s_0^{abs} to s_m^{abs} .
 - For instance, shortcuts $s_j^{\text{abs}} = f(s_i^{\text{abs}}, a^{\text{abs}})$ for some $j > i + 1$ and an action a not in $\langle a_1, \dots, a_m \rangle$



- *When does this happen here? Other reasons?*

Outline

- General Idea
- Domain Abstractions
- *Pattern Databases*
- Disjoint Pattern Databases
- Pattern Databases in Strips

Pattern Databases

Let $\langle V, A, I, G \rangle$ be an MVStrips task, and ${}^{\text{abs}}$ be a domain abstraction. For a state s , we define $h^{\text{abs}}(s)$ as the (forward) solution distance $h^*(s^{\text{abs}})$ of s^{abs} in $\langle V, A^{\text{abs}}, I^{\text{abs}}, G^{\text{abs}} \rangle$. The set of all pairs $(s^{\text{abs}}, h^*(s^{\text{abs}}))$ is called a **pattern database**.

- We have seen that h^{abs} is admissible
- A (forward) database can be re-used for several initial states.
- Backward PDs can be defined very similarly. A backward database can be re-used for several goals.

Building Pattern Databases

Exhaustively solve $\langle V, A^{\text{abs}}, I^{\text{abs}}, G^{\text{abs}} \rangle$

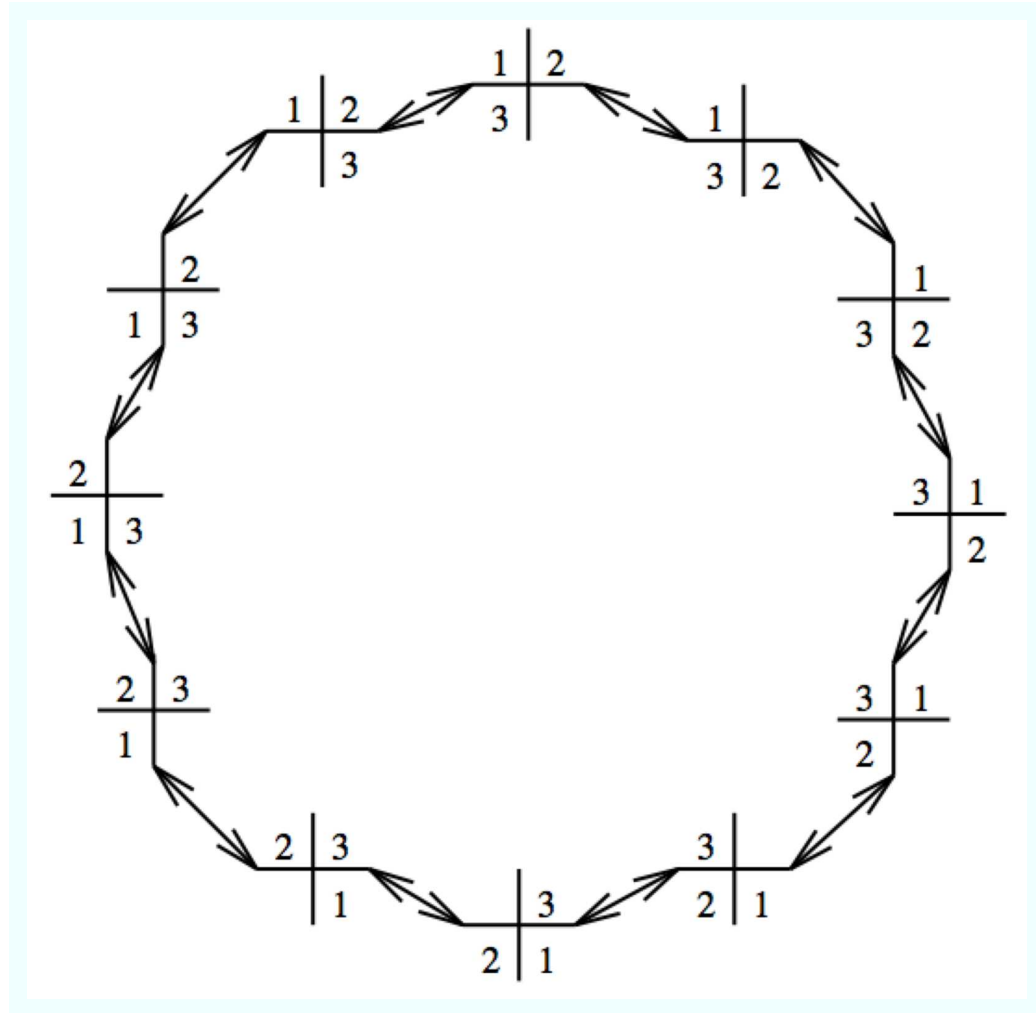
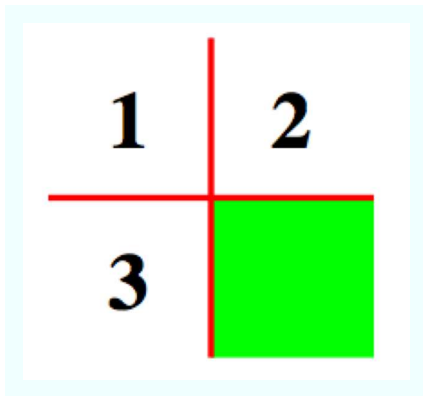
- Do an exhaustive BrFS from I^{abs}
- Generate each state only once (closed list)
- When explicit state-space graph is built, compute the solution distances (a simple generalization of Dijkstra)

Surjectivity

Let $\langle V, A, I, G \rangle$ be an MVStrips task. A homomorphism domain abstraction ${}^{\text{abs}}$ is **surjective** iff for every abstract state σ there is a state s such that $\sigma = s^{\text{abs}}$.

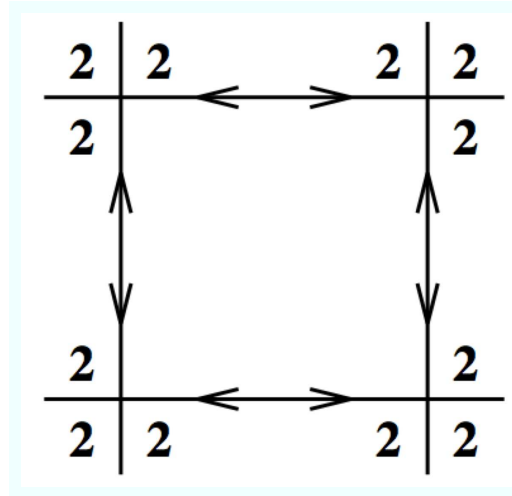
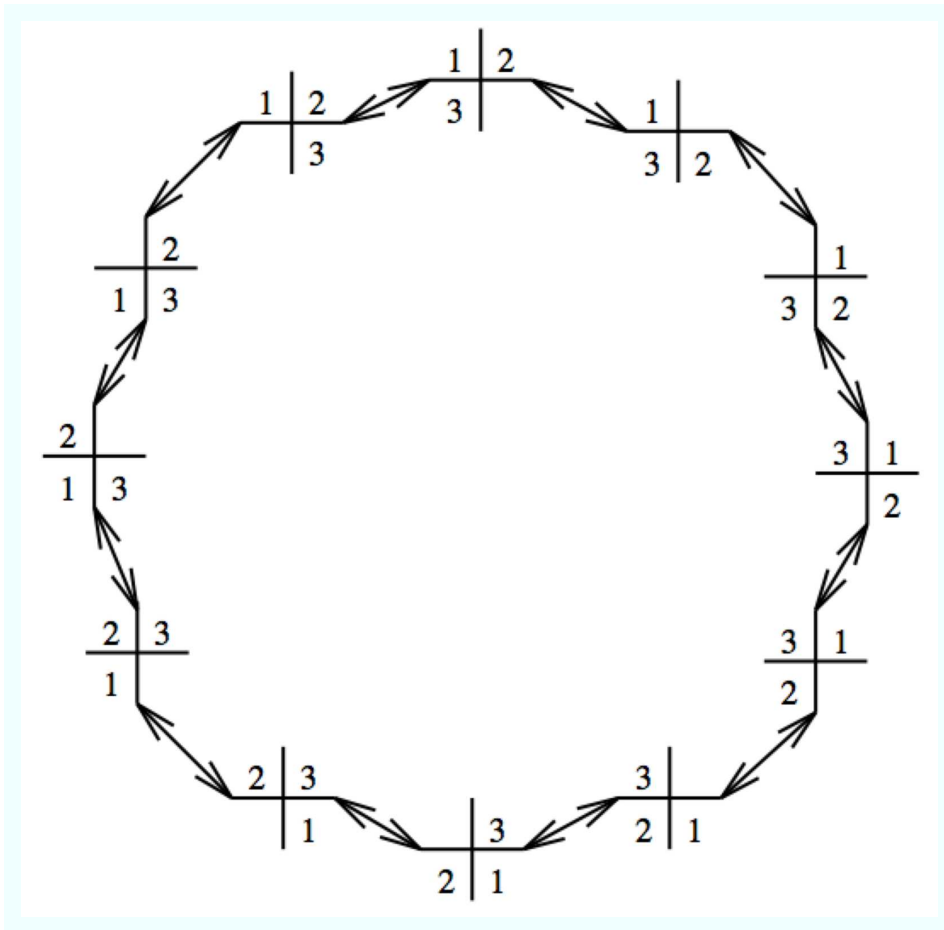
- Non-surjective homomorphisms arise quite often
- *Why do we care?* When ${}^{\text{abs}}$ is not surjective, the database contains entries which will never be mapped to by ${}^{\text{abs}}$

Example: 4-puzzle



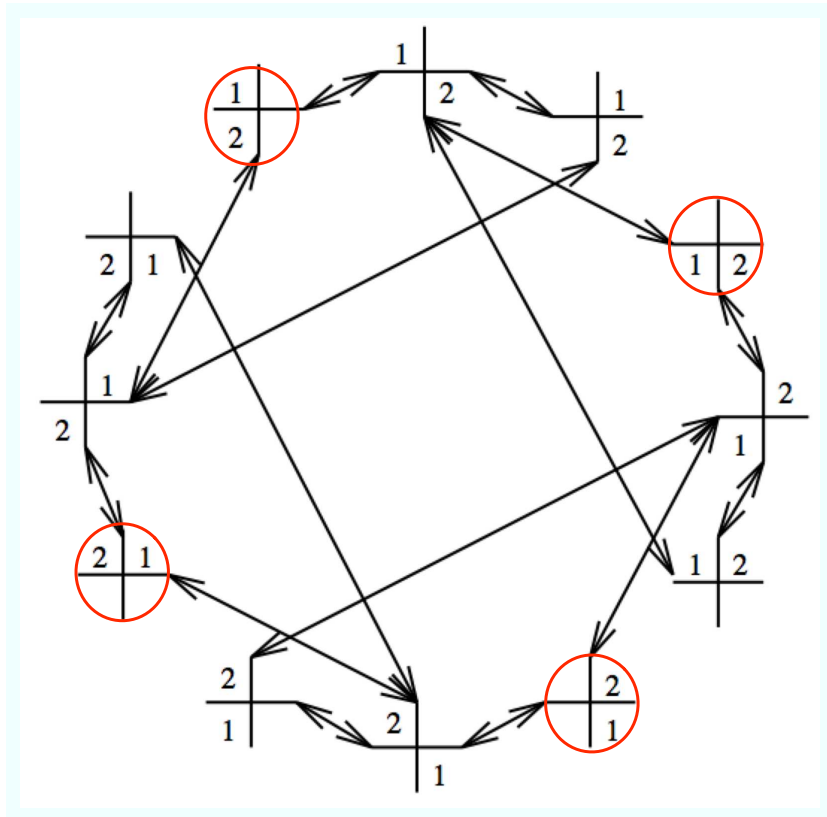
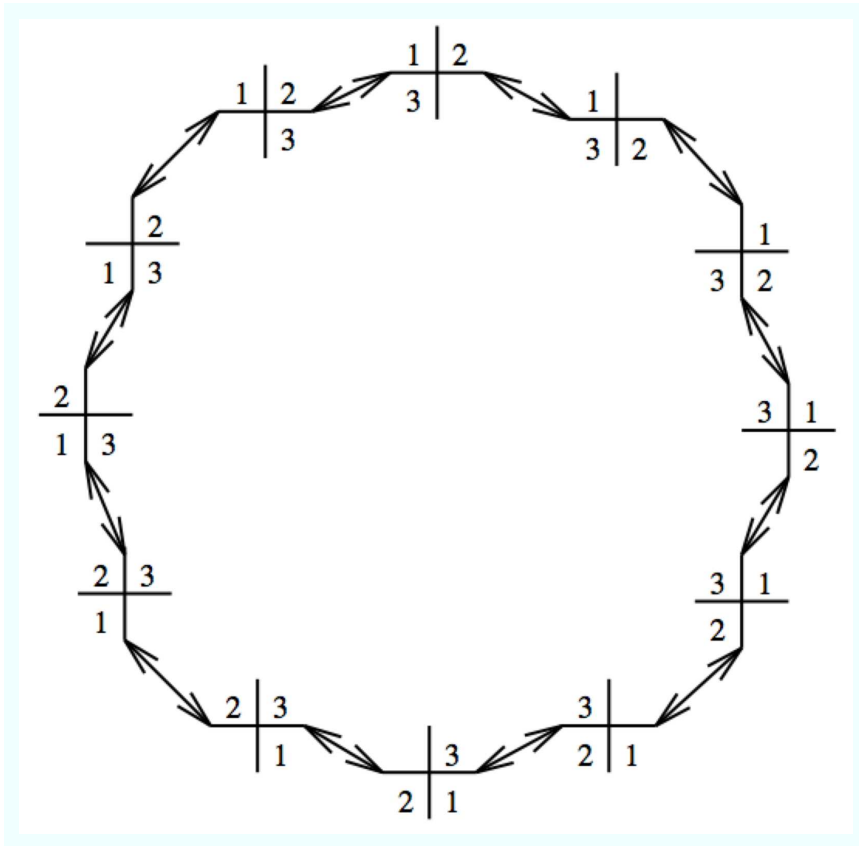
Surjective homomorphism for 4-puzzle

$$\forall x \in \{\text{blank}, 1, 2, 3\} : x^{\text{abs}} = \begin{cases} \text{blank}, & x = \text{blank} \\ 2, & \text{otherwise,} \end{cases}$$



Non-surjective homomorphism for 4-puzzle

$$\forall x \in \{\text{blank}, 1, 2, 3\} : x^{\text{abs}} = \begin{cases} \text{blank}, & x = 3 \\ x, & \text{otherwise,} \end{cases}$$



Outline

- General Idea
- Domain Abstractions
- Pattern Databases
- *Disjoint Pattern Databases*
- Pattern Databases in Strips

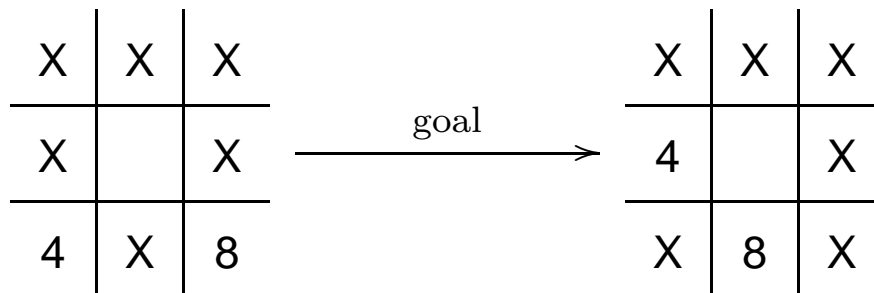
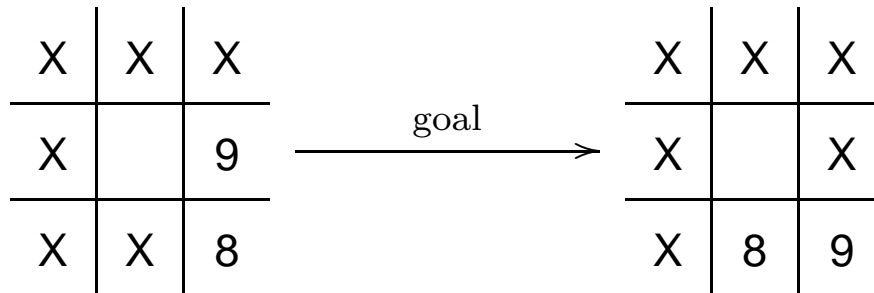
Adding vs. Choosing Maximum

- We can keep several databases, and **max** over the induced h values
- It would be much better to **add** the values (*Why?*)

- In general, adding h values is **not admissible**
- Can we ...
 - identify cases where it *is* admissible?
 - do something about the other cases?

Example: 8-puzzle

Is adding h values from these two abstractions admissible?



Disjoint Pattern Databases

Let $\langle V, A, I, G \rangle$ be an MVStrips task, and abs1 and abs2 be some domain abstractions. abs1 and abs2 are said to be **disjoint** if, for every state s , we have

$$h^{abs1}(s) + h^{abs2}(s) \leq h^*(s)$$

- The pattern databases of two disjoint abstractions are also called disjoint
- In general, checking disjointness is hard ...
- But some simple *sufficient* conditions do exist

Void Actions

Let $\langle V, A, I, G \rangle$ be an MVStrips task. An action $a \in A$ is said to be **void** if

$$\forall (v := c) \in Eff(a) : \exists (v := c) \in Pre(a)$$

- Void actions do not change the state
- You wouldn't expect to find a void action in a real task, but in an abstracted task things are different!
- Example:
 - If the abstraction of 8-puzzle maps both blank and some t_i to a symbol X , then moving t_i is void
 - $move(t_i, p_j, p_k)^{abs}$:
 $Pre = \{p_j = X, p_k = X\}, Eff = \{p_j = X, p_k = X\}$

Void Actions

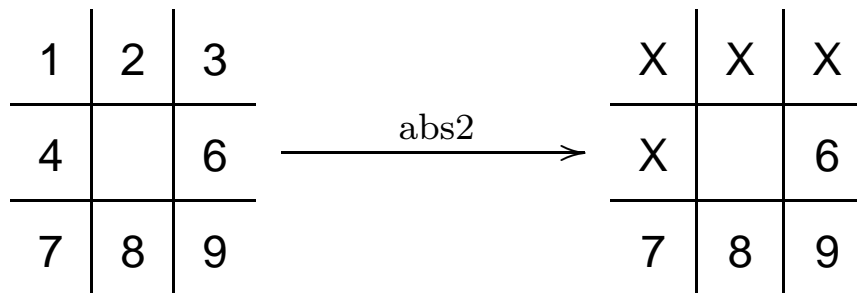
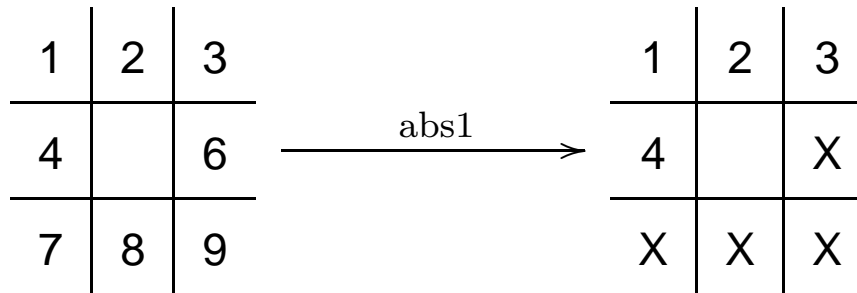
Proposition

Let $\langle V, A, I, G \rangle$ be an MVStrips task, and ${}^{\text{abs1}}$ and ${}^{\text{abs2}}$ be two domain abstractions. If there is no $a \in A$ that is non-void in both $\langle V, A^{\text{abs1}}, I^{\text{abs1}}, G^{\text{abs1}} \rangle$ and $\langle V, A^{\text{abs2}}, I^{\text{abs2}}, G^{\text{abs2}} \rangle$, then ${}^{\text{abs1}}$ and ${}^{\text{abs2}}$ are disjoint.

Proof: Blackboard.

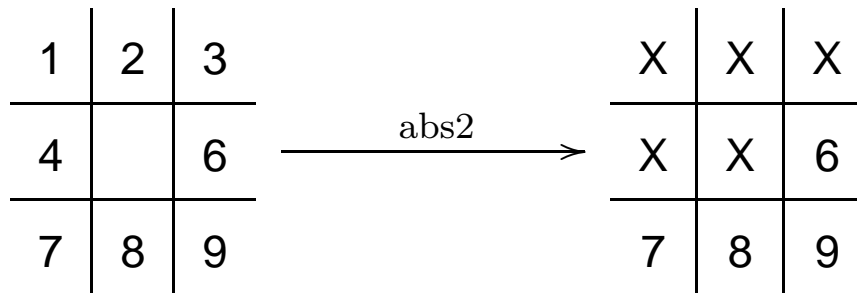
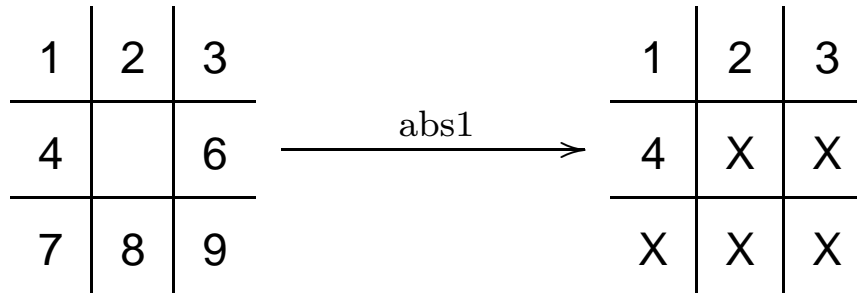
Example: 8-puzzle

Are these two abstractions disjoint?



Example: 8-puzzle

Are these two?



“Disjoining” Pattern Databases

- If we have two PDs that share the non-void actions $A_{1,2} \subset A$, then an option is to **count** the cost of $a \in A_{1,2}$ only in one of the pattern databases
 - When building the other database, set edge costs for non-counted actions to 0
- The resulting h values can be added safely. *Proof?*

Multiple Pattern Databases

Given a budget of m memory bytes for PDs,
how should one exploit this memory budget?

- Observation: *max*-ing over numerous small databases **reduces** the number of expanded search nodes with respect to using just a single large PD. (*Intuitive?*)
- Example 8-puzzle: 20 PDs of size 252 perform less state expansions (318) than 1 PD of size 5040 (2160 state expansions)

Multiple Pattern Databases

Given a budget of m memory bytes for PDs,
how should one exploit this memory budget?

- Observation: *max*-ing over numerous small databases **reduces** the number of expanded search nodes with respect to using just a single large PD. (*Intuitive?*)
 - Example 8-puzzle: 20 PDs of size 252 perform less state expansions (318) than 1 PD of size 5040 (2160 state expansions)
1. Adopting a single large PD \Rightarrow increase the number of patterns with **high h values**
 2. Adopting a set of smaller PDs \Rightarrow reduce the number of patterns with **low h values**
 3. Eliminating low h -values is **more important** for improving search performance than retaining large h -values

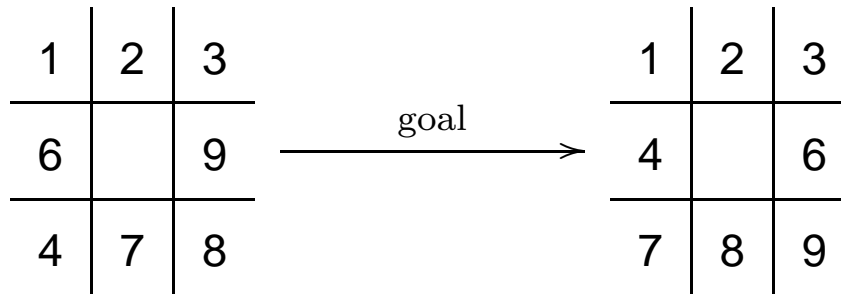
Outline

- General Idea
- Domain Abstractions
- Pattern Databases
- Disjoint Pattern Databases
- *Pattern Databases in Strips*

Pattern Databases in Strips

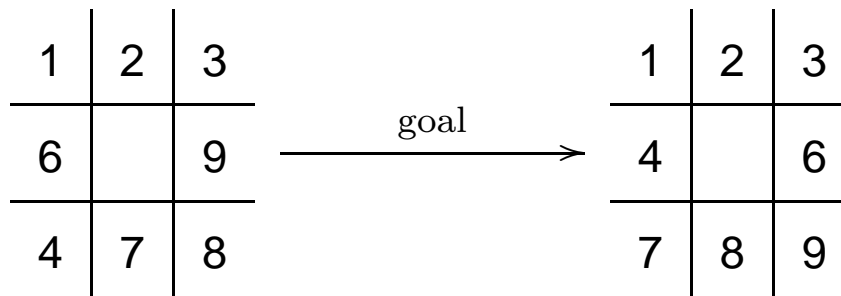
- In Strips we have only Boolean variables
- There are just two different domain abstractions:
 - *What are they?*
 - *What heuristics do they induce?*
- We will have to use a different abstraction technique

Example: 8-puzzle



- Recall: Variables: p_i for $1 \leq i \leq 9$, $dom = \{\text{blank}, t_1, t_2, t_3, t_4, t_6, t_7, t_8, t_9\}$
- Boolean variables: $[p_i, t]$ for $1 \leq i \leq 9$, $t \in \{\text{blank}, t_1, t_2, t_3, t_4, t_6, t_7, t_8, t_9\}$
- Actions: $move(t_i, p_j, p_k)$ (for neighbors p_j and p_k):
 - $Pre = \{[p_j, t_i], [p_k, \text{blank}]\},$
 - $Eff = \{[p_j, \text{blank}], [p_k, t_i], \neg[p_j, t_i], \neg[p_k, \text{blank}]\}$
- Initial state, goal: ...

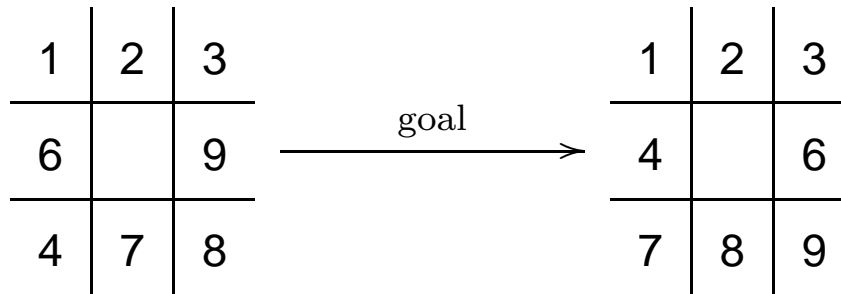
Example: 8-puzzle



- Recall: Variables: p_i for $1 \leq i \leq 9$, $dom = \{\text{blank}, t_1, t_2, t_3, t_4, t_6, t_7, t_8, t_9\}$
- Boolean variables: $[p_i, t]$ for $1 \leq i \leq 9$, $t \in \{\text{blank}, t_1, t_2, t_3, t_4, t_6, t_7, t_8, t_9\}$
- Actions: $move(t_i, p_j, p_k)$ (for neighbors p_j and p_k):
 $Pre = \{[p_j, t_i], [p_k, \text{blank}]\},$
 $Eff = \{[p_j, \text{blank}], [p_k, t_i], \neg[p_j, t_i], \neg[p_k, \text{blank}]\}$
- Initial state, goal: ...

What happens here if we replace tile t_9 with X ?

Example: 8-puzzle



- Recall: Variables: p_i for $1 \leq i \leq 9$, $dom = \{\text{blank}, t_1, t_2, t_3, t_4, t_6, t_7, t_8, t_9\}$
- Boolean variables: $[p_i, t]$ for $1 \leq i \leq 9$, $t \in \{\text{blank}, t_1, t_2, t_3, t_4, t_6, t_7, t_8, t_9\}$
- Actions: $move(t_9, p_6, p_5)^{abs} = move(X, p_6, p_5)$:
 $Pre = \{[p_6, X], [p_5, \text{blank}]\},$
 $Eff = \{[p_6, \text{blank}], [p_5, X], \neg[p_6, X], \neg[p_5, \text{blank}]\}$
- Initial state, goal: ...

We replace all the nine facts $[p_i, t_9]$ with $[p_i, X]$!

Domain Abstraction in Strips

Let $\langle P, A, I, G \rangle$ be a Strips task so that $Del(a) \subseteq Pre(a)$ for all $a \in A$. Let $p, p' \in P$ be *inconsistent*, and $x \notin P$. Then

$$(\xi(P), \{\xi(a) | a \in A\}, \xi(I), \xi(P))$$

is called a **domain abstraction** of $\langle P, A, I, G \rangle$ if

1. For any fact set F ,

$$\xi(F) = \begin{cases} F, & F \cap \{p, p'\} = \emptyset \\ F \setminus \{p, p'\} \cup \{x\}, & \text{otherwise} \end{cases}$$

2. For an action $a = (Pre, Add, Del)$,

$$\xi(a) = \begin{cases} (\xi(Pre), \xi(Add), \xi(Del)), & x \notin \xi(Add) \vee x \notin \xi(Del) \\ (\xi(Pre), \xi(Add) \setminus \{x\}, \xi(Del) \setminus \{x\}), & \text{otherwise} \end{cases}$$

Domain Abstraction in Strips (in English)

- Replace p and p' with x everywhere. If x appears in *add* **and** *del* of an action, remove it from both
- “Inconsistent”: imagine that p and p' correspond to different values of a natural multi-valued variable
 - *Easy to identify?*
- To abstract more we iterate the definition!

Literature

- J. Culberson and J. Schaeffer, *Pattern Databases*, Computational Intelligence, 1998 (first message in 1995)
- I. Hernadvölgyi and R. Holte, *PSVN: A Vector Representation for Production Systems*, Technical Report, 1999
- R. Korf and A. Felner, *Disjoint Pattern Database Heuristics*, Artificial Intelligence, 2002
- A. Felner, R. Korf, and S. Hanan, *Additive Pattern Database Heuristics*, JAIR, 2004
- R. Holte, J. Newton, A. Felner, R. Meshulam, D. Furcy, *Multiple Pattern Databases*, ICAPS 2004
- S. Edelkamp, *Planning with Pattern Databases*, ECP 2001
- J. Hoffmann, A. Sabharwal, C. Domshlak, *Friends or Foes? An AI Planning Perspective on Abstraction and Search*, ICAPS 2006 (tari-tari :-)