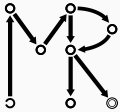
Efficiently Computing Transitions in Cartesian Abstractions

Jendrik Seipp

June 4, 2024

Machine Reasoning Lab, Linköping University



Setting

- optimal classical planning
- A^* + single abstraction heuristic
- focus on abstraction computation

Abstraction Classes

- projections (pattern database heuristics)
- domain abstractions
- Cartesian abstractions
- Merge-and-shrink abstractions

Abstraction Classes

- projections (pattern database heuristics)
- domain abstractions
- Cartesian abstractions
- Merge-and-shrink abstractions

Example task and abstraction

- variables x and y
- $dom(x) = dom(y) = \{0, 1, 2\}$
- abstract state $\{0, 2\} \times \{1\}$ contains concrete states $\{01, 21\}$

Counterexample-Guided Abstraction Refinement

CEGAR

```
compute initial abstraction
until TERMINATE():
    find shortest path in abstraction
    if there is no path:
        return unsolvable
    find flaw in path
    if there is no flaw:
        return plan
    refine abstraction for flaw
return abstraction
```

Cartesian CEGAR Bottlenecks

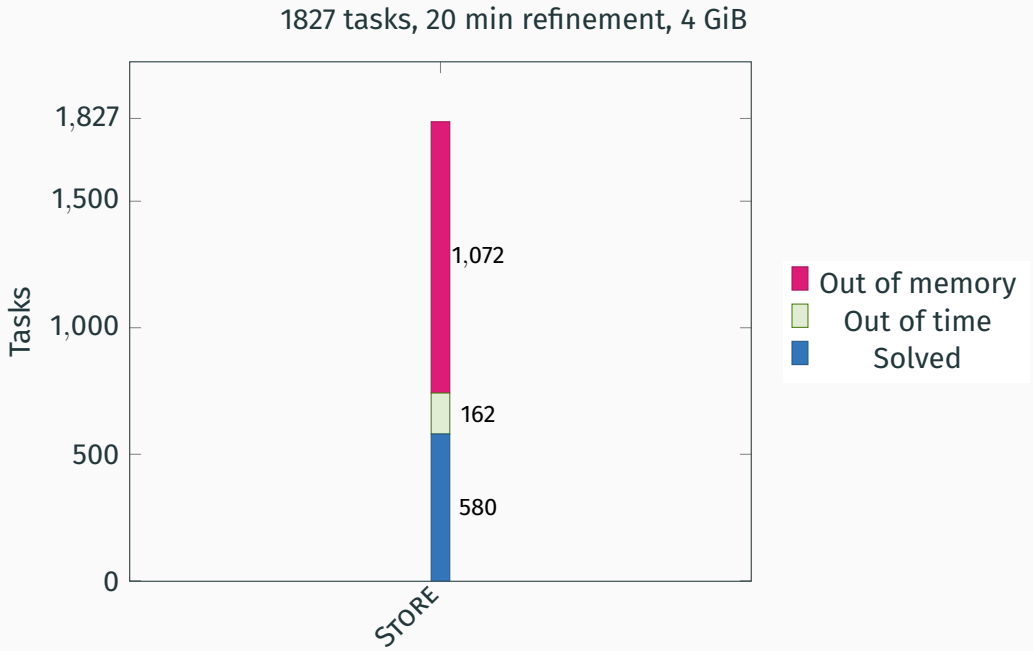
- find next abstract solution with Dijkstra's algorithm
- find next abstract solution with A* (Seipp & Helmert, ICAPS 2013)
- update shortest paths incrementally (Seipp et al., ICAPS 2020)
- find and repair multiple flaws at once (Speck & Seipp, ICAPS 2022)
- trace abstract plans backwards (Pozo et al., AAAI 2024)

Cartesian CEGAR Bottlenecks

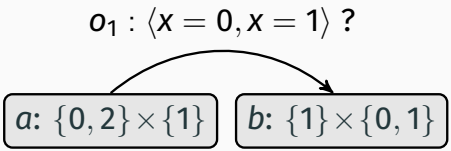
- find next abstract solution with Dijkstra's algorithm
- find next abstract solution with A* (Seipp & Helmert, ICAPS 2013)
- update shortest paths incrementally (Seipp et al., ICAPS 2020)
- find and repair multiple flaws at once (Speck & Seipp, ICAPS 2022)
- trace abstract plans backwards (Pozo et al., AAAI 2024)

$\rightarrow$ latest bottleneck: memory for storing transitions

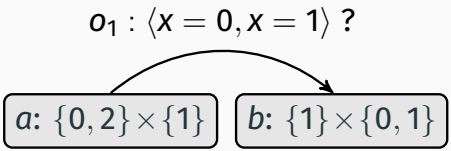
Results for Storing Transitions



Compute Transitions on Demand



Compute Transitions on Demand

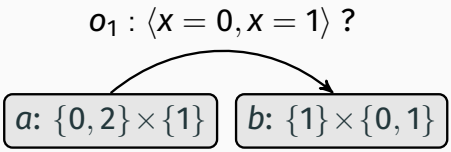


Transition check

$progr(a, o_1) \cap b \neq \emptyset$

- $progr(a, o_1) \cap b = \{1\} \times \{1\} \cap b = \{1\} \times \{1\} \neq \emptyset \rightsquigarrow$ transition exists

Compute Transitions on Demand



Transition check

$progr(a, o_1) \cap b \neq \emptyset$

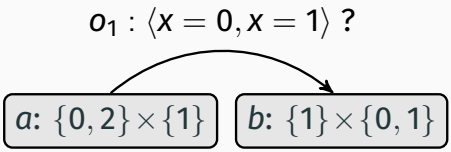
- $progr(a, o_1) \ \cap \ b = \{1\} \times \{1\} \ \cap \ b = \{1\} \times \{1\} \neq \emptyset \rightsquigarrow$ transition exists

Transition check implies

$a \cap \mathcal{C}(pre(o_1)) \neq \emptyset$

- $a \cap \mathcal{C}(pre(o_1)) = \{0,2\} \times \{1\} \ \cap \ \{0\} \times \{0,1,2\} = \{0\} \times \{1\} \neq \emptyset \rightsquigarrow o_1$ applicable

Compute Transitions on Demand



Transition check

$progr(a, o_1) \cap b \neq \emptyset$

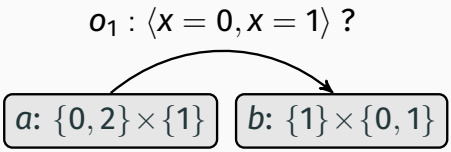
- $progr(a, o_1) \ \cap \ b = \{1\} \times \{1\} \ \cap \ b = \{1\} \times \{1\} \neq \emptyset \rightsquigarrow$ transition exists

Transition check implies

$a \cap \mathcal{C}(pre(o_1)) \neq \emptyset$

- $a \cap \mathcal{C}(pre(o_1)) = \{0,2\} \times \{1\} \ \cap \ \{0\} \times \{0,1,2\} = \{0\} \times \{1\} \neq \emptyset \rightsquigarrow o_1$ applicable
- applicable operators

Compute Transitions on Demand



Transition check

$progr(a, o_1) \cap b \neq \emptyset$

- $progr(a, o_1) \ \cap \ b = \{1\} \times \{1\} \ \cap \ b = \{1\} \times \{1\} \neq \emptyset \rightsquigarrow$ transition exists
- reached states

Transition check implies

$a \cap \mathcal{C}(pre(o_1)) \neq \emptyset$

- $a \cap \mathcal{C}(pre(o_1)) = \{0,2\} \times \{1\} \ \cap \ \{0\} \times \{0,1,2\} = \{0\} \times \{1\} \neq \emptyset \rightsquigarrow o_1$ applicable
- applicable operators

Compute outgoing transitions naively

for all operators o with $a \cap \mathcal{C}(pre(o)) \neq \emptyset$:
    for all states b with $progr(a, o) \cap b \neq \emptyset$:
        $a \xrightarrow{o} b$ exists

Compute outgoing transitions naively

for all operators o with $a \cap \mathcal{C}(pre(o)) \neq \emptyset$:
    for all states b with $progr(a, o) \cap b \neq \emptyset$:
        $a \xrightarrow{o} b$ exists

How to speed this up?

Compute outgoing transitions naively

for all operators o with $a \cap \mathcal{C}(pre(o)) \neq \emptyset$:
    for all states b with $progr(a, o) \cap b \neq \emptyset$:
        $a \xrightarrow{o} b$ exists

How to speed this up?

- applicable operators: successor generator

Compute outgoing transitions naively

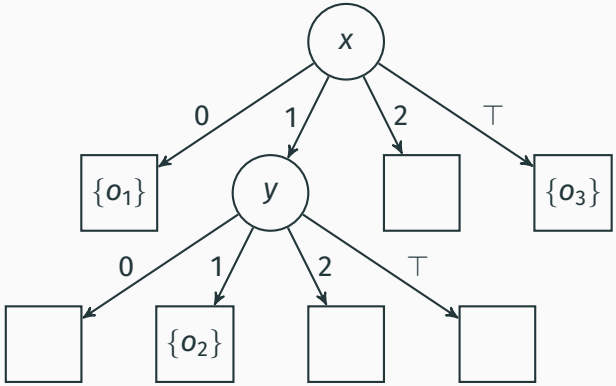
for all operators o with $a \cap \mathcal{C}(pre(o)) \neq \emptyset$:
    for all states b with $progr(a, o) \cap b \neq \emptyset$:
        $a \xrightarrow{o} b$ exists

How to speed this up?

- applicable operators: successor generator
- reached states: refinement hierarchy

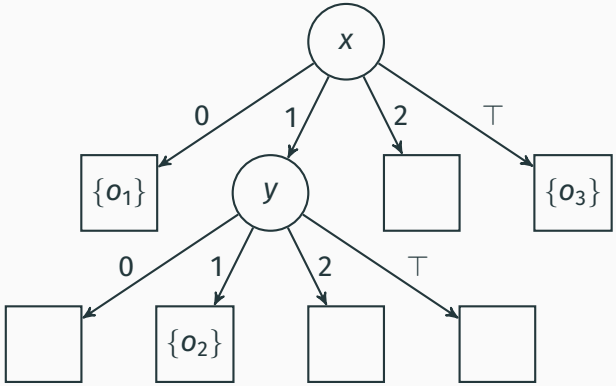
Compute Applicable Operators with Successor Generator

- $o_1 : \langle x = 0,\ x = 1 \rangle$
- $o_2 : \langle x = 1 \wedge y = 1,\ y = 0 \rangle$
- $o_3 : \langle \top,\ y = 2 \rangle$



Compute Applicable Operators with Successor Generator

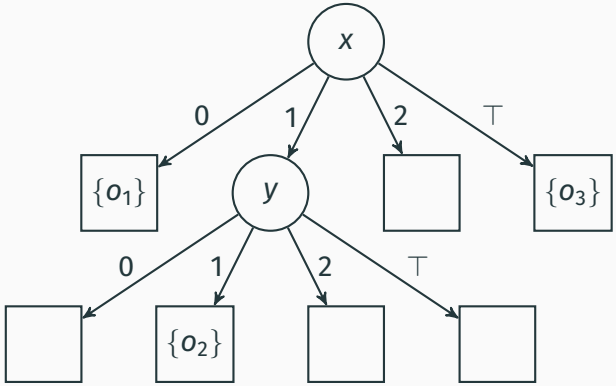
- $o_1 : \langle x = 0,\ x = 1 \rangle$
- $o_2 : \langle x = 1 \wedge y = 1,\ y = 0 \rangle$
- $o_3 : \langle \top,\ y = 2 \rangle$



- $s : \{x \mapsto 2, y \mapsto 1\} \rightsquigarrow \{o_3\}$

Compute Applicable Operators with Successor Generator

- $o_1 : \langle x = 0,\ x = 1 \rangle$
- $o_2 : \langle x = 1 \wedge y = 1,\ y = 0 \rangle$
- $o_3 : \langle \top,\ y = 2 \rangle$



- $s : \{x \mapsto 2, y \mapsto 1\} \rightsquigarrow \{o_3\}$
- $a : \{0, 2\} \times \{1\} \rightsquigarrow \{0_1, o_3\}$

Compute Reached States with Refinement Hierarchy

Applicable operators: $\{o_1, o_3\}$

$progr(a, o_1) \cap b \neq \emptyset$
$\rightsquigarrow a \xrightarrow{o_1} b$ exists

Compute Reached States with Refinement Hierarchy

Applicable operators: $\{o_1, o_3\}$

$progr(a, o_1) \cap b \neq \emptyset$
$\rightsquigarrow a \xrightarrow{o_1} b$ exists

Reached states with o_3?

$progr(a, o_3) =$
$progr(\{0,2\} \times \{1\}, \langle \top, y = 2 \rangle) =$
$\{0,2\} \times \{2\}$

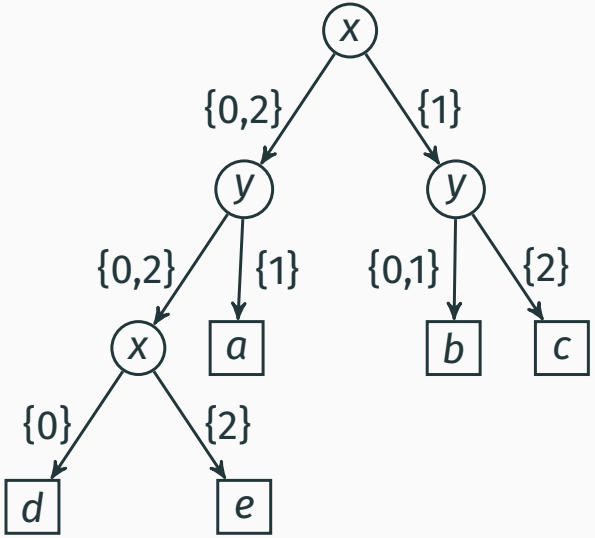
Compute Reached States with Refinement Hierarchy

Applicable operators: $\{o_1, o_3\}$

$progr(a, o_1) \cap b \neq \emptyset$
$\rightsquigarrow a \xrightarrow{o_1} b$ exists

Reached states with o_3?

$progr(a, o_3) =$
$progr(\{0,2\} \times \{1\}, \langle \top, y = 2 \rangle) =$
$\{0,2\} \times \{2\}$



Compute Reached States with Refinement Hierarchy

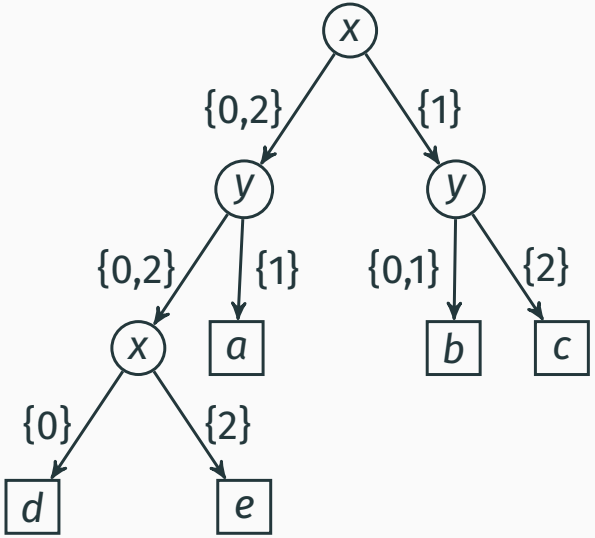
Applicable operators: $\{o_1, o_3\}$

$progr(a, o_1) \cap b \neq \emptyset$
$\rightsquigarrow a \xrightarrow{o_1} b$ exists

Reached states with o_3?

$progr(a, o_3) =$
$progr(\{0,2\} \times \{1\}, \langle \top, y = 2 \rangle) =$
$\{0,2\} \times \{2\}$

$\rightsquigarrow \{d, e\}$

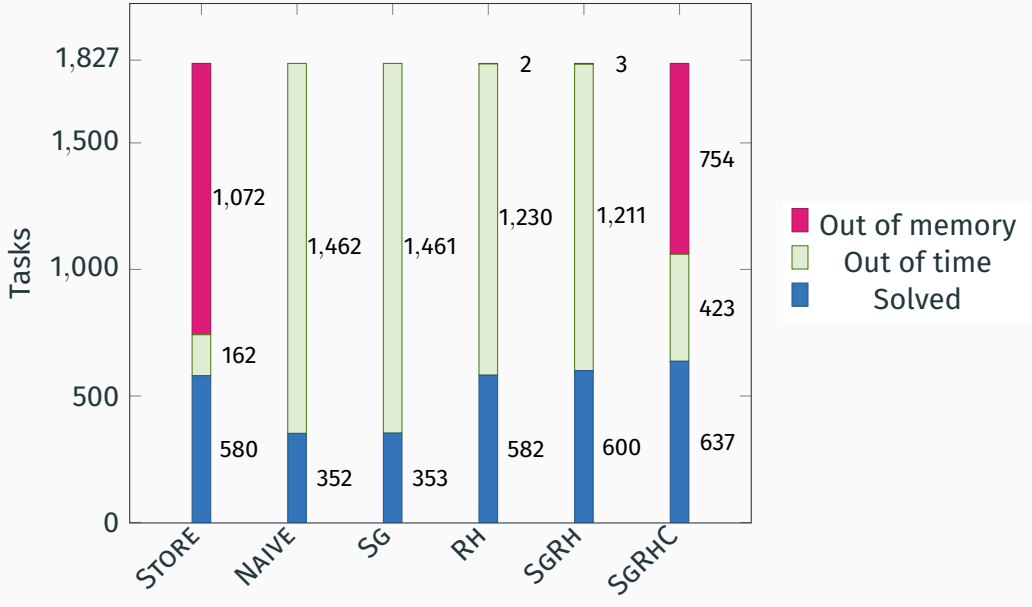


Compute Incoming Transitions

Cartesian progression and regression are symmetric

$progr(a, o) \cap b \neq \emptyset$ iff $regr(b, o) \cap a \neq \emptyset$

Results


1,827
1,500
1,000
500
0
Tasks
1,072
162
580
1,462
352
1,461
353
2
1,230
582
3
1,211
600
754
423
637
STORE
NAIVE
SG
RH
SGRH
SGRHC
Out of memory
Out of time
Solved

Summary

- we can efficiently compute transitions on demand for Cartesian abstractions
- split into computing applicable operators and reached states
- use sucessor generator and refinement hierarchy
- caching optimal transitions yields good trade-off between memory and time