

# Principles of AI Planning

## 15. Planning with State-Dependent Action Costs

Albert-Ludwigs-Universität Freiburg



Bernhard Nebel and Robert Mattmüller  
January 22nd, 2020



- Background
- State-Dependent Action Costs
- Edge-Valued Multi-Valued Decision Diagrams
- Compilation
- Relaxations
- Abstractions
- Summary
- References

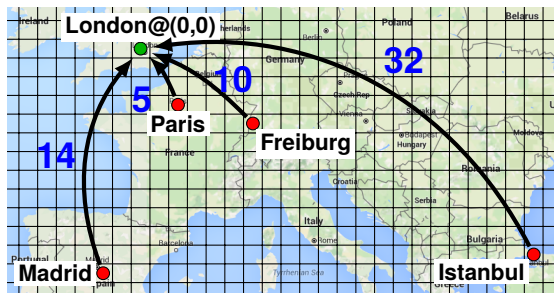
# Background

January 22nd, 2020

B. Nebel, R. Mattmüller – AI Planning

2 / 79

# What are State-Dependent Action Costs?



- Background
- State-Dependent Action Costs
- Edge-Valued Multi-Valued Decision Diagrams
- Compilation
- Relaxations
- Abstractions
- Summary
- References

Action costs: **unit** — **constant** — **state-dependent**

$$\begin{aligned} \text{cost}(\text{fly}(\text{Madrid}, \text{London})) &= 1, & \text{cost}(\text{fly}(\text{Paris}, \text{London})) &= 1, \\ \text{cost}(\text{fly}(\text{Freiburg}, \text{London})) &= 1, & \text{cost}(\text{fly}(\text{Istanbul}, \text{London})) &= 1. \end{aligned}$$

January 22nd, 2020

B. Nebel, R. Mattmüller – AI Planning

3 / 79

# Why Study State-Dependent Action Costs?



- Background
- State-Dependent Action Costs
- Edge-Valued Multi-Valued Decision Diagrams
- Compilation
- Relaxations
- Abstractions
- Summary
- References

- In classical planning: actions have **unit costs**.
  - Each action  $a$  costs 1.
- Simple extension: actions have **constant costs**.
  - Each action  $a$  costs some  $\text{cost}_a \in \mathbb{N}$ .
  - Example: Flying between two cities costs amount proportional to distance.
  - Still easy to handle algorithmically, e. g. when computing  $g$  and  $h$  values.
- Further extension: actions have **state-dependent costs**.
  - Each action  $a$  has **cost function**  $\text{cost}_a : S \rightarrow \mathbb{N}$ .
  - Example: Flying to a destination city costs amount proportional to distance, depending on the current city.

January 22nd, 2020

B. Nebel, R. Mattmüller – AI Planning

4 / 79

# Why Study State-Dependent Action Costs?



- **Human perspective:**
  - “natural”, “elegant”, and “higher-level”
  - **modeler-friendly**  $\rightsquigarrow$  less error-prone?
- **Machine perspective:**
  - more **structured**  $\rightsquigarrow$  exploit structure in algorithms?
  - fewer redundancies, exponentially more **compact**
- **Language support:**
  - numeric **PDDL**, PDDL 3
  - **RDDL**, **MDPs** (state-dependent rewards!)
- **Applications:**
  - modeling **preferences** and **soft goals**
  - application domains such as **PSR**

- Background
- State-Dependent Action Costs
- Edge-Valued Multi-Valued Decision Diagrams
- Compilation
- Relaxations
- Abstractions
- Summary
- References

(Abbreviation: SDAC = state-dependent action costs)

# Handling State-Dependent Action Costs



## Good news:

- Computing  **$g$  values** in forward search still **easy**.  
(When expanding state  $s$  with action  $a$ , we know  $cost_a(s)$ .)

## Challenge:

- But what about **SDAC-aware  $h$  values** (relaxation heuristics, abstraction heuristics)?
- Or can we simply **compile SDAC away**?

## This chapter:

- Proposed **answers** to these challenges.

- Background
- State-Dependent Action Costs
- Edge-Valued Multi-Valued Decision Diagrams
- Compilation
- Relaxations
- Abstractions
- Summary
- References

# Handling State-Dependent Action Costs



## Roadmap:

- 1 Look at **compilations**.
- 2 This leads to **edge-valued multi-valued decision diagrams** (EVMDDs) as data structure to represent cost functions.
- 3 Based on EVMDDs, formalize and discuss:
  - compilations
  - relaxation heuristics
  - abstraction heuristics

- Background
- State-Dependent Action Costs
- Edge-Valued Multi-Valued Decision Diagrams
- Compilation
- Relaxations
- Abstractions
- Summary
- References

# State-Dependent Action Costs



## Definition

A **SAS<sup>+</sup> planning task with state-dependent action costs** or **SDAC planning task** is a tuple  $\Pi = \langle V, I, O, \gamma, (cost_a)_{a \in O} \rangle$  where  $\langle V, I, O, \gamma \rangle$  is a (regular) SAS<sup>+</sup> planning task with state set  $S$  and  $cost_a : S \rightarrow \mathbb{N}$  is the **cost function** of  $a$  for all  $a \in O$ .

**Assumption:** For each  $a \in O$ , the set of variables occurring in the precondition of  $a$  is disjoint from the set of variables on which the cost function  $cost_a$  depends.

(**Question:** Why is this assumption unproblematic?)

Definitions of plans etc. stay as before. A plan is **optimal** if it minimizes the sum of action costs from start to goal.

For the rest of this chapter, we consider the following running example.

- Background
- State-Dependent Action Costs
- Edge-Valued Multi-Valued Decision Diagrams
- Compilation
- Relaxations
- Abstractions
- Summary
- References

# State-Dependent Action Costs

## Running Example



### Example (Household domain)

#### Actions:

vacuumFloor =  $\langle T, \text{floorClean} \rangle$   
 washDishes =  $\langle T, \text{dishesClean} \rangle$   
 doHousework =  $\langle T, \text{floorClean} \wedge \text{dishesClean} \rangle$

#### Cost functions:

$\text{cost}_{\text{vacuumFloor}} = [-\text{floorClean}] \cdot 2$   
 $\text{cost}_{\text{washDishes}} = [-\text{dishesClean}] \cdot (1 + 2 \cdot [-\text{haveDishwasher}])$   
 $\text{cost}_{\text{doHousework}} = \text{cost}_{\text{vacuumFloor}} + \text{cost}_{\text{washDishes}}$

(Question: How much can applying action washDishes cost?)

- Background
- State-Dependent Action Costs
- Edge-Valued
- Multi-Valued
- Decision Diagrams
- Compilation
- Relaxations
- Abstractions
- Summary
- References

# State-Dependent Action Costs

## Compilations



Different ways of compiling SDAC away:

- **Compilation I:** “Parallel Action Decomposition”
- **Compilation II:** “Purely Sequential Action Decomposition”
- **Compilation III:** “EVMDD-Based Action Decomposition”  
(combination of Compilations I and II)

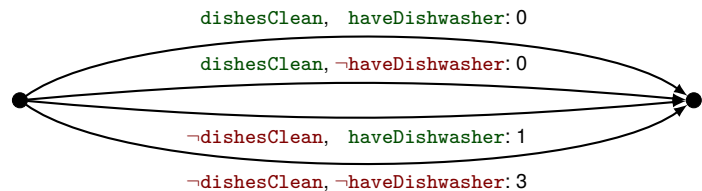
- Background
- State-Dependent Action Costs
- Edge-Valued
- Multi-Valued
- Decision Diagrams
- Compilation
- Relaxations
- Abstractions
- Summary
- References

# State-Dependent Action Costs

## Compilation I: “Parallel Action Decomposition”



### Example



washDishes( dC, hD) =  $\langle dC \wedge hD, dC \rangle$ , cost = 0  
 washDishes( dC, ¬hD) =  $\langle dC \wedge \neg hD, dC \rangle$ , cost = 0  
 washDishes( ¬dC, hD) =  $\langle \neg dC \wedge hD, dC \rangle$ , cost = 1  
 washDishes( ¬dC, ¬hD) =  $\langle \neg dC \wedge \neg hD, dC \rangle$ , cost = 3

- Background
- State-Dependent Action Costs
- Edge-Valued
- Multi-Valued
- Decision Diagrams
- Compilation
- Relaxations
- Abstractions
- Summary
- References

# State-Dependent Action Costs

## Compilation I: “Parallel Action Decomposition”



### Compilation I

Transform each action into multiple actions:

- **one for each partial state** relevant to cost function
- add partial state to **precondition**
- use cost for partial state as **constant cost**

#### Properties:

- ✓ **always possible**
- ✗ **exponential blow-up**

Question: Exponential blow-up **avoidable**?  $\rightsquigarrow$  Compilation II

- Background
- State-Dependent Action Costs
- Edge-Valued
- Multi-Valued
- Decision Diagrams
- Compilation
- Relaxations
- Abstractions
- Summary
- References

# State-Dependent Action Costs

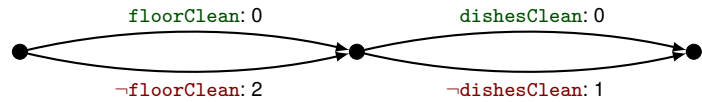
Compilation II: "Purely Sequential Action Decomposition"



## Example

Assume we own a dishwasher:

$$cost_{doHousework} = 2 \cdot [\neg floorClean] + [\neg dishesClean]$$



- $doHousework_1(\ fC) = \langle \ fC, fC \rangle, \ cost = 0$
- $doHousework_1(\neg fC) = \langle \neg fC, fC \rangle, \ cost = 2$
- $doHousework_2(\ dC) = \langle \ dC, dC \rangle, \ cost = 0$
- $doHousework_2(\neg dC) = \langle \neg dC, dC \rangle, \ cost = 1$

- Background
- State-Dependent Action Costs
- Edge-Valued
- Multi-Valued
- Decision Diagrams
- Compilation
- Relaxations
- Abstractions
- Summary
- References

# State-Dependent Action Costs

Compilation II: "Purely Sequential Action Decomposition"



## Compilation II

If costs are **additively decomposable/separable**:

- high-level actions  $\approx$  **macro actions**
- decompose into **sequential micro actions**

- Background
- State-Dependent Action Costs
- Edge-Valued
- Multi-Valued
- Decision Diagrams
- Compilation
- Relaxations
- Abstractions
- Summary
- References

# State-Dependent Action Costs

Compilation II: "Purely Sequential Action Decomposition"



## Properties:

- ✓ only **linear** blow-up
  - ✗ **not always possible**
  - plan lengths not preserved
- E. g., in a state where  $\neg fC$  and  $\neg dC$  hold, an application of

$doHousework$

in the SDAC setting is replaced by an application of the action **sequence**

$doHousework_1(\neg fC), doHousework_2(\neg dC)$

in the compiled setting.

- Background
- State-Dependent Action Costs
- Edge-Valued
- Multi-Valued
- Decision Diagrams
- Compilation
- Relaxations
- Abstractions
- Summary
- References

# State-Dependent Action Costs

Compilation II: "Purely Sequential Action Decomposition"



## Properties (ctd.):

- plan costs preserved
  - blow-up in search space
- E. g., in a state where  $\neg fC$  and  $\neg dC$  hold, should we apply  $doHousework_1(\neg fC)$  or  $doHousework_2(\neg dC)$  first?
- $\rightsquigarrow$  impose action ordering!
- attention: we should apply all partial effects at end!
- Otherwise, an effect of an earlier action in the compilation might affect the cost of a later action in the compilation.

**Question:** Can this **always work** (kind of)?  $\rightsquigarrow$  Compilation III

- Background
- State-Dependent Action Costs
- Edge-Valued
- Multi-Valued
- Decision Diagrams
- Compilation
- Relaxations
- Abstractions
- Summary
- References

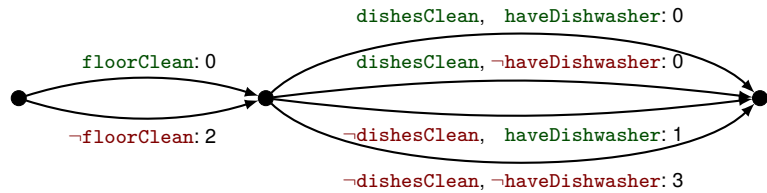
# State-Dependent Action Costs

Compilation III: "EVMD-Based Action Decomposition"



## Example

$$COST_{doHousework} = [\neg floorClean] \cdot 2 + [\neg dishesClean] \cdot (1 + 2 \cdot [\neg haveDishwasher])$$



Simplify right-hand part of diagram:

- Branch over single variable at a time.
- Exploit: haveDishwasher irrelevant if dishesClean is true.

Background  
 State-Dependent Action Costs  
 Edge-Valued Multi-Valued Decision Diagrams  
 Compilation  
 Relaxations  
 Abstractions  
 Summary  
 References

# State-Dependent Action Costs

Compilation III: "EVMD-Based Action Decomposition"



## Example (ctd.)



Later:

- Compiled actions
- Auxiliary variables to enforce action ordering

Background  
 State-Dependent Action Costs  
 Edge-Valued Multi-Valued Decision Diagrams  
 Compilation  
 Relaxations  
 Abstractions  
 Summary  
 References

# State-Dependent Action Costs

Compilation III: "EVMD-Based Action Decomposition"



## Compilation III

- exploit as much additive separability as possible
- multiply out variable domains where inevitable
- Technicalities:
  - fix variable ordering
  - perform Shannon and isomorphism reduction (cf. theory of BDDs)

Properties:

- ✓ always possible
- worst-case exponential blow-up, but as good as it gets
- as with Compilation II: plan lengths not preserved, plan costs preserved
- as with Compilation II: action ordering, all effects at end!

Background  
 State-Dependent Action Costs  
 Edge-Valued Multi-Valued Decision Diagrams  
 Compilation  
 Relaxations  
 Abstractions  
 Summary  
 References

# State-Dependent Action Costs

Compilation III: "EVMD-Based Action Decomposition"



Compilation III provides optimal combination of sequential and parallel action decomposition, given fixed variable ordering.

Question: How to find such decompositions automatically?

Answer: Figure for Compilation III basically a reduced ordered edge-valued multi-valued decision diagram (EVMD)!

[Lai et al., 1996; Ciardo and Siminiceanu, 2002]

Background  
 State-Dependent Action Costs  
 Edge-Valued Multi-Valued Decision Diagrams  
 Compilation  
 Relaxations  
 Abstractions  
 Summary  
 References

## EVMDDs:

- Decision diagrams for arithmetic functions
- Decision nodes with associated decision variables
- Edge weights: partial costs contributed by facts
- Size of EVMDD **compact** in many “typical”, well-behaved cases (**Question**: For example?)

## Properties:

- ✓ **satisfy all requirements** for Compilation III, even (almost) uniquely determined by them
- ✓ already have **well-established theory and tool support**
- ✓ **detect and exhibit additive structure** in arithmetic functions

- Background
- State-Dependent Action Costs
- Edge-Valued Multi-Valued Decision Diagrams
- Compilation
- Relaxations
- Abstractions
- Summary
- References

## Consequence:

- represent cost functions as **EVMDDs**
- exploit** additive structure exhibited by them
- draw on theory and tool support for EVMDDs

## Two perspectives on EVMDDs:

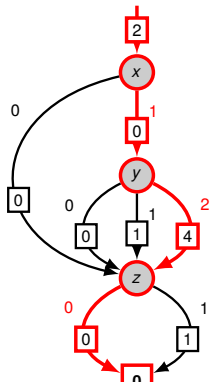
- graphs specifying how to **decompose** action costs
- data structures **encoding** action costs (used **independently from compilations**)

- Background
- State-Dependent Action Costs
- Edge-Valued Multi-Valued Decision Diagrams
- Compilation
- Relaxations
- Abstractions
- Summary
- References

## Example (EVMDD Evaluation)

$$cost_a = xy^2 + z + 2$$

$$\mathcal{D}_x = \mathcal{D}_z = \{0, 1\}, \mathcal{D}_y = \{0, 1, 2\}$$



- Directed acyclic graph
- Dangling incoming edge
- Single **terminal node 0**
- Decision nodes** with:
  - decision variables
  - edge label
  - edge weights

We see:  $z$  independent from rest,  $y$  only matters if  $x \neq 0$ .

$$s = \{x \mapsto 1, y \mapsto 2, z \mapsto 0\}$$

- Background
- State-Dependent Action Costs
- Edge-Valued Multi-Valued Decision Diagrams
- Compilation
- Relaxations
- Abstractions
- Summary
- References

## Properties of EVMDDs:

- ✓ **Existence** for finitely many finite-domain variables
- ✓ **Uniqueness/canonicity** if reduced and ordered
- ✓ **Basic arithmetic operations** supported

(Lai et al., 1996; Ciardo and Siminiceanu, 2002)

- Background
- State-Dependent Action Costs
- Edge-Valued Multi-Valued Decision Diagrams
- Compilation
- Relaxations
- Abstractions
- Summary
- References

Given arithmetic operator  $\otimes \in \{+, -, \cdot, \dots\}$ , EVMDDs  $\mathcal{E}_1, \mathcal{E}_2$ .  
 Compute EVMDD  $\mathcal{E} = \mathcal{E}_1 \otimes \mathcal{E}_2$ .

Implementation: procedure **apply**( $\otimes, \mathcal{E}_1, \mathcal{E}_2$ ):

- **Base case:** single-node EVMDDs encoding constants
- **Inductive case:** apply  $\otimes$  recursively:
  - push down edge weights
  - recursively apply  $\otimes$  to corresponding children
  - pull up excess edge weights from children

Time complexity [Lai et al., 1996]:

- **additive operations:** product of input EVMDD sizes
- **in general:** exponential

# Compilation

## EVMDD-Based Action Compilation

**Idea:** each edge in the EVMDD becomes a new micro action with constant cost corresponding to the edge constraint, precondition that we are currently at its start EVMDD node, and effect that we are currently at its target EVMDD node.

**Example (EVMDD-based action compilation)**

Let  $a = \langle \chi, e \rangle$ ,  $cost_a = xy^2 + z + 2$ .

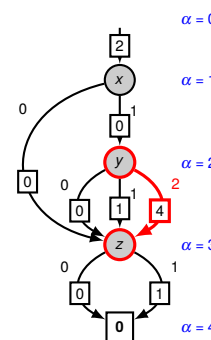
Auxiliary variables:

- One **semaphore variable**  $\sigma$  with  $\mathcal{D}_\sigma = \{0, 1\}$  for entire planning task.
- One **auxiliary variable**  $\alpha = \alpha_a$  with  $\mathcal{D}_{\alpha_a} = \{0, 1, 2, 3, 4\}$  for action  $a$ .

Replace  $a$  by new auxiliary actions (similarly for other actions).

## EVMDD-Based Action Compilation

**Example (EVMDD-based action compilation, ctd.)**



$a^x = \langle \chi \wedge \sigma = 0 \wedge \alpha = 0, \sigma := 1 \wedge \alpha := 1 \rangle,$	$cost = 2$
$a^{1,x=0} = \langle \alpha = 1 \wedge x = 0, \alpha := 3 \rangle,$	$cost = 0$
$a^{1,x=1} = \langle \alpha = 1 \wedge x = 1, \alpha := 2 \rangle,$	$cost = 0$
$a^{2,y=0} = \langle \alpha = 2 \wedge y = 0, \alpha := 3 \rangle,$	$cost = 0$
$a^{2,y=1} = \langle \alpha = 2 \wedge y = 1, \alpha := 3 \rangle,$	$cost = 1$
$a^{2,y=2} = \langle \alpha = 2 \wedge y = 2, \alpha := 3 \rangle,$	$cost = 4$
$a^{3,z=0} = \langle \alpha = 3 \wedge z = 0, \alpha := 4 \rangle,$	$cost = 0$
$a^{3,z=1} = \langle \alpha = 3 \wedge z = 1, \alpha := 4 \rangle,$	$cost = 1$
$a^e = \langle \alpha = 4, e \wedge \sigma := 0 \wedge \alpha := 0 \rangle,$	$cost = 0$



## Definition (EVMDD-based action compilation)

Let  $\Pi = \langle V, I, O, \gamma, (cost_a)_{a \in O} \rangle$  be an SDAC planning task, and for each action  $a \in O$ , let  $\mathcal{E}_a$  be an EVMDD that encodes the cost function  $cost_a$ .

Let  $EAC(a)$  be the set of actions created from  $a$  using  $\mathcal{E}_a$  similar to the previous example. Then the **EVMDD-based action compilation** of  $\Pi$  using  $\mathcal{E}_a, a \in O$ , is the task  $\Pi' = EAC(\Pi) = \langle V', I', O', \gamma' \rangle$ , where

- $V' = V \cup \{\sigma\} \cup \{\alpha_a \mid a \in O\}$ ,
- $I' = I \cup \{\sigma \mapsto 0\} \cup \{\alpha_a \mapsto 0 \mid a \in O\}$ ,
- $O' = \bigcup_{a \in O} EAC(a)$ , and
- $\gamma' = \gamma \wedge (\sigma = 0) \wedge \bigwedge_{a \in O} (\alpha_a = 0)$ .

- Background
- Compilation
- Relaxations
- Abstractions
- Summary
- References

Let  $\Pi$  be an SDAC task and  $\Pi' = EAC(\Pi)$  its EVMDD-based action compilation (for appropriate EVMDDs  $\mathcal{E}_a$ ).

## Proposition

$\Pi'$  has only state-independent costs.

### Proof.

By construction. □

## Proposition

The size  $\|\Pi'\|$  is in the order  $O(\|\Pi\| \cdot \max_{a \in O} \|\mathcal{E}_a\|)$ , i. e. **polynomial** in the size of  $\Pi$  and the largest used EVMDD.

### Proof.

By construction. □

- Background
- Compilation
- Relaxations
- Abstractions
- Summary
- References

Let  $\Pi$  be an SDAC task and  $\Pi' = EAC(\Pi)$  its EVMDD-based action compilation (for appropriate EVMDDs  $\mathcal{E}_a$ ).

## Proposition

$\Pi$  and  $\Pi'$  admit the same plans (up to replacement of actions by action **sequences**). Optimal plan costs are preserved.

### Proof.

Let  $\pi = a_1, \dots, a_n$  be a plan for  $\Pi$ , and let  $s_0, \dots, s_n$  be the corresponding state sequence such that  $a_i$  is applicable in  $s_{i-1}$  and leads to  $s_i$  for all  $i = 1, \dots, n$ .

For each  $i = 1, \dots, n$ , let  $\mathcal{E}_{a_i}$  be the EVMDD used to compile  $a_i$ . State  $s_{i-1}$  determines a unique path through the EVMDD  $\mathcal{E}_{a_i}$ , which uniquely corresponds to an action sequence  $a_i^0, \dots, a_i^{k_i}$  (for some  $k_i \in \mathbb{N}$ ; including  $a_i^x$  and  $a_i^e$ ).

- Background
- Compilation
- Relaxations
- Abstractions
- Summary
- References

## Proof (ctd.)

By construction,  $cost(a_i^0) + \dots + cost(a_i^{k_i}) = cost_{a_i}(s_{i-1})$ .

Moreover, the sequence  $a_i^0, \dots, a_i^{k_i}$  is applicable in  $s_{i-1} \cup \{\sigma \mapsto 0\} \cup \{\alpha_a \mapsto 0 \mid a \in O\}$  and leads to  $s_i \cup \{\sigma \mapsto 0\} \cup \{\alpha_a \mapsto 0 \mid a \in O\}$ .

Therefore, by induction,  $\pi' = a_1^0, \dots, a_1^{k_1}, \dots, a_n^0, \dots, a_n^{k_n}$  is applicable in  $s_0 \cup \{\sigma \mapsto 0\} \cup \{\alpha_a \mapsto 0 \mid a \in O\}$  (and leads to a goal state). Moreover,  
 $cost(\pi') = cost(a_1^0) + \dots + cost(a_1^{k_1}) + \dots + cost(a_n^0) + \dots + cost(a_n^{k_n}) = cost_{a_1}(s_0) + \dots + cost_{a_n}(s_{n-1}) = cost(\pi)$ .

Still to show:  $\Pi'$  admits no other plans. It suffices to see that the semaphore  $\sigma$  prohibits interleaving more than one EVMDD evaluation, and that each  $\alpha_a$  makes sure that the EVMDD for  $a$  is traversed in the unique correct order. □

- Background
- Compilation
- Relaxations
- Abstractions
- Summary
- References



## Example

Let  $\Pi = \langle V, I, O, \gamma \rangle$  with  $V = \{x, y, z, u\}$ ,  $\mathcal{D}_x = \mathcal{D}_z = \{0, 1\}$ ,  $\mathcal{D}_y = \mathcal{D}_u = \{0, 1, 2\}$ ,  $I = \{x \mapsto 1, y \mapsto 2, z \mapsto 0, u \mapsto 0\}$ ,  $O = \{a, b\}$ , and  $\gamma = (u = 2)$  with

$$a = \langle u = 0, u := 1 \rangle, \quad cost_a = xy^2 + z + 2,$$

$$b = \langle u = 1, u := 2 \rangle, \quad cost_b = z + 1.$$

Optimal plan for  $\Pi$ :

$$\pi = a, b \text{ with } cost(\pi) = 6 + 1 = 7.$$

Background  
Compilation  
Relaxations  
Abstractions  
Summary  
References

## Example (Ctd.)

Compilation of  $a$ :

$a^x = \langle u = 0 \wedge \sigma = 0 \wedge \alpha_a = 0, \sigma := 1 \wedge \alpha_a := 1 \rangle,$	$cost = 2$
$a^{1,x=0} = \langle \alpha_a = 1 \wedge x = 0, \alpha_a := 3 \rangle,$	$cost = 0$
$a^{1,x=1} = \langle \alpha_a = 1 \wedge x = 1, \alpha_a := 2 \rangle,$	$cost = 0$
$a^{2,y=0} = \langle \alpha_a = 2 \wedge y = 0, \alpha_a := 3 \rangle,$	$cost = 0$
$a^{2,y=1} = \langle \alpha_a = 2 \wedge y = 1, \alpha_a := 3 \rangle,$	$cost = 1$
$a^{2,y=2} = \langle \alpha_a = 2 \wedge y = 2, \alpha_a := 3 \rangle,$	$cost = 4$
$a^{3,z=0} = \langle \alpha_a = 3 \wedge z = 0, \alpha_a := 4 \rangle,$	$cost = 0$
$a^{3,z=1} = \langle \alpha_a = 3 \wedge z = 1, \alpha_a := 4 \rangle,$	$cost = 1$
$a^e = \langle \alpha_a = 4, u := 1 \wedge \sigma := 0 \wedge \alpha_a := 0 \rangle,$	$cost = 0$

Background  
Compilation  
Relaxations  
Abstractions  
Summary  
References

## Example (Ctd.)

Compilation of  $b$ :

$b^x = \langle u = 1 \wedge \sigma = 0 \wedge \alpha_b = 0, \sigma := 1 \wedge \alpha_b := 1 \rangle,$	$cost = 1$
$b^{1,z=0} = \langle \alpha_b = 1 \wedge z = 0, \alpha_b := 2 \rangle,$	$cost = 0$
$b^{1,z=1} = \langle \alpha_b = 1 \wedge z = 1, \alpha_b := 2 \rangle,$	$cost = 1$
$b^e = \langle \alpha_b = 2, u := 2 \wedge \sigma := 0 \wedge \alpha_b := 0 \rangle,$	$cost = 0$

Optimal plan for  $\Pi'$  (with  $cost(\pi') = 6 + 1 = 7 = cost(\pi)$ ):

$$\pi' = a^x, \underbrace{a^{1,x=1}, a^{2,y=2}, a^{3,z=0}}_{cost=2+0+4+0+0=6}, \underbrace{a^e, b^x, b^{1,z=0}, b^e}_{cost=1+0+0=1}$$

Background  
Compilation  
Relaxations  
Abstractions  
Summary  
References

- Okay. We can compile SDAC away somewhat efficiently. Is this the end of the story?
- No! Why not?
  - Tighter integration of SDAC into planning process might be beneficial.
  - Analysis of heuristics for SDAC might improve our understanding.
- Consequence: Let's study heuristics for SDAC in uncompiled setting.

Background  
Compilation  
Relaxations  
Abstractions  
Summary  
References

# Relaxations

- Background
- Compilation
- Relaxations
  - Delete Relaxations in SAS\*
  - Costs in Relaxed States
  - Additive Heuristic
  - Relaxed Planning Graph
- Abstractions
- Summary
- References

# Relaxation Heuristics

- Background
- Compilation
- Relaxations
  - Delete Relaxations in SAS\*
  - Costs in Relaxed States
  - Additive Heuristic
  - Relaxed Planning Graph
- Abstractions
- Summary
- References

**We know:** Delete-relaxation heuristics informative in classical planning.

**Question:** Are they also informative in SDAC planning?

# Relaxation Heuristics

- Background
- Compilation
- Relaxations
  - Delete Relaxations in SAS\*
  - Costs in Relaxed States
  - Additive Heuristic
  - Relaxed Planning Graph
- Abstractions
- Summary
- References

- Assume we want to compute the **additive heuristic  $h^{add}$**  in a task with state-dependent action costs.
- But what does an action  $a$  cost in a relaxed state  $s^+$ ?
- And how to compute that cost?

# Relaxed SAS<sup>+</sup> Tasks

- Background
- Compilation
- Relaxations
  - Delete Relaxations in SAS\*
  - Costs in Relaxed States
  - Additive Heuristic
  - Relaxed Planning Graph
- Abstractions
- Summary
- References

Delete relaxation in SAS<sup>+</sup> tasks works as follows:

- Operators are already in effect normal form.
- We do not need to impose a positive normal form, because all conditions are conjunctions of facts, and facts are just variable-value pairs and hence always positive.
- Hence  $a^+ = a$  for any operator  $a$ , and  $\Pi^+ = \Pi$ .
- For simplicity, we identify relaxed states  $s^+$  with their on-sets  $on(s^+)$ .
- Then, a relaxed state  $s^+$  is a set of facts  $(v, d)$  with  $v \in V$  and  $d \in \mathcal{D}_v$  including at least one fact  $(v, d)$  for each  $v \in V$  (but possibly more than one, which is what makes it a **relaxed** state).

# Relaxed SAS<sup>+</sup> Tasks



- A relaxed operator  $a$  is applicable in a relaxed state  $s^+$  if all precondition facts of  $a$  are contained in  $s^+$ .
- Relaxed states **accumulate** facts reached so far.
- Applying a relaxed operator  $a$  to a relaxed state  $s^+$  adds to  $s^+$  those facts made true by  $a$ .

- Background
- Compilation
- Relaxations
- Delete Relaxations in SAS<sup>+</sup>
- Costs in Relaxed States
- Additive Heuristic Relaxed Planning Graph
- Abstractions
- Summary
- References

## Example

Relaxed operator  $a^+ = \langle x = 2, y := 1 \wedge z := 0 \rangle$  is applicable in relaxed state  $s^+ = \{(x, 0), (x, 2), (y, 0), (z, 1)\}$ , because precondition  $(x, 2) \in s^+$ , and leads to successor  $(s^+)' = s^+ \cup \{(y, 1), (z, 0)\}$ .

Relaxed plans, dominance, monotonicity etc. as before. The above definition generalizes the one for propositional tasks.

# Action Costs in Relaxed States

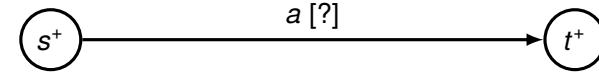


## Example

Assume  $s^+$  is the relaxed state with

$$s^+ = \{(x, 0), (x, 1), (y, 1), (y, 2), (z, 0)\}.$$

What should action  $a$  with  $cost_a = xy^2 + z + 2$  cost in  $s^+$ ?



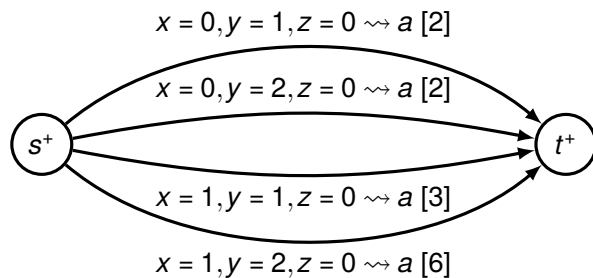
- Background
- Compilation
- Relaxations
- Delete Relaxations in SAS<sup>+</sup>
- Costs in Relaxed States
- Additive Heuristic Relaxed Planning Graph
- Abstractions
- Summary
- References

# Action Costs in Relaxed States



**Idea:** We should assume the **cheapest** way of applying  $a^+$  in  $s^+$  to guarantee admissibility of  $h^+$ . (Allow at least the behavior of the unrelaxed setting at no higher cost.)

## Example



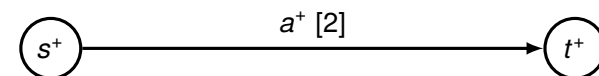
- Background
- Compilation
- Relaxations
- Delete Relaxations in SAS<sup>+</sup>
- Costs in Relaxed States
- Additive Heuristic Relaxed Planning Graph
- Abstractions
- Summary
- References

# Action Costs in Relaxed States



**Idea:** We should assume the **cheapest** way of applying  $a^+$  in  $s^+$  to guarantee admissibility of  $h^+$ . (Allow at least the behavior of the unrelaxed setting at no higher cost.)

## Example



- Background
- Compilation
- Relaxations
- Delete Relaxations in SAS<sup>+</sup>
- Costs in Relaxed States
- Additive Heuristic Relaxed Planning Graph
- Abstractions
- Summary
- References

## Definition

Let  $V$  be a set of FDR variables,  $s : V \rightarrow \bigcup_{v \in V} \mathcal{D}_v$  an unrelaxed state over  $V$ , and  $s^+ \subseteq \{(v, d) \mid v \in V, d \in \mathcal{D}_v\}$  a relaxed state over  $V$ . We call  $s$  **consistent** with  $s^+$  if  $\{(v, s(v)) \mid v \in V\} \subseteq s^+$ .

## Definition

Let  $a \in O$  be an action with cost function  $cost_a$ , and  $s^+$  a relaxed state. Then the **relaxed cost** of  $a$  in  $s^+$  is defined as

$$cost_a(s^+) = \min_{s \in S \text{ consistent with } s^+} cost_a(s).$$

(Question: How many states  $s$  are consistent with  $s^+$ ?)

- Background
- Compilation
- Relaxations
  - Delete Relaxations in SAS\*
  - Costs in Relaxed States
  - Additive Heuristic
  - Relaxed Planning Graph
- Abstractions
- Summary
- References

**Problem with this definition:** There are generally **exponentially** many states  $s$  consistent with  $s^+$  to minimize over.

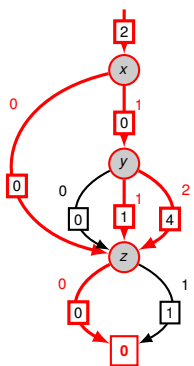
**Central question:** Can we still do this minimization efficiently?

**Answer:** Yes, at least efficiently in the size of an EVMDD encoding  $cost_a$ .

- Background
- Compilation
- Relaxations
  - Delete Relaxations in SAS\*
  - Costs in Relaxed States
  - Additive Heuristic
  - Relaxed Planning Graph
- Abstractions
- Summary
- References

## Example

Relaxed state  $s^+ = \{(x, 0), (x, 1), (y, 1), (y, 2), (z, 0)\}$ .

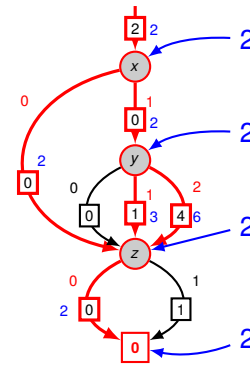


- Computing  $cost_a(s^+) =$  minimizing over  $cost_a(s)$  for all  $s$  consistent with  $s^+ =$  minimizing over all start-end-paths in EVMDD following only edges consistent with  $s^+$ .
- **Observation:** Minimization over exponentially many paths can be replaced by **top-sort traversal of EVMDD**, minimizing over incoming arcs consistent with  $s^+$  at all nodes!

- Background
- Compilation
- Relaxations
  - Delete Relaxations in SAS\*
  - Costs in Relaxed States
  - Additive Heuristic
  - Relaxed Planning Graph
- Abstractions
- Summary
- References

## Example

Relaxed state  $s^+ = \{(x, 0), (x, 1), (y, 1), (y, 2), (z, 0)\}$ .



- $cost_a(s^+) = 2$
- Cost-minimizing  $s$  consistent with  $s^+$ :  $s(x) = s(z) = 0, s(y) \in \{1, 2\}$ .

- Background
- Compilation
- Relaxations
  - Delete Relaxations in SAS\*
  - Costs in Relaxed States
  - Additive Heuristic
  - Relaxed Planning Graph
- Abstractions
- Summary
- References

## Theorem

A top-sort traversal of the EVMDD for  $cost_a$ , adding edge weights and minimizing over incoming arcs consistent with  $s^+$  at all nodes, computes  $cost_a(s^+)$  and takes time in the order of the size of the EVMDD.

Proof.

Homework? □

The following definition is equivalent to the RPG-based one.

## Definition (Classical additive heuristic $h^{add}$ )

$$h_s^{add}(s) = h_s^{add}(GoalFacts)$$

$$h_s^{add}(Facts) = \sum_{fact \in Facts} h_s^{add}(fact)$$

$$h_s^{add}(fact) = \begin{cases} 0 & \text{if } fact \in s \\ \min_{\text{achiever } a \text{ of } fact} [h_s^{add}(pre(a)) + cost_a] & \text{otherwise} \end{cases}$$

Question: How to generalize  $h^{add}$  to SDAC?

## Example

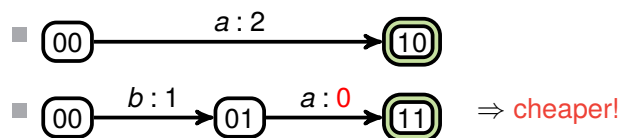
$$a = \langle T, x = 1 \rangle \quad cost_a = 2 - 2y$$

$$b = \langle T, y = 1 \rangle \quad cost_b = 1$$

$$s = \{x \mapsto 0, y \mapsto 0\}$$

$$h_s^{add}(y = 1) = 1$$

$$h_s^{add}(x = 1) = ?$$



(Here, we need the assumption that no variable occurs both in the cost function and the precondition of the same action):

## Definition (Additive heuristic $h^{add}$ for SDAC)

$$h_s^{add}(fact) = \begin{cases} 0 & \text{if } fact \in s \\ \min_{\text{achiever } a \text{ of } fact} [h_s^{add}(pre(a)) + cost_a] & \text{otherwise} \end{cases}$$

$$h_s^{add}(fact) = \begin{cases} 0 & \\ \min_{\text{achiever } a \text{ of } fact} [h_s^{add}(fact)] & \end{cases}$$

$$Cost_a^s = \min_{\hat{s} \in S_a} [cost_a(\hat{s}) + h_s^{add}(\hat{s})]$$

$S_a$ : set of partial states over variables in cost function

$|S_a|$  exponential in number of variables in cost function

## Theorem

Let  $\Pi$  be an SDAC planning task, let  $\Pi'$  be an EVMDD-based action compilation of  $\Pi$ , and let  $s$  be a state of  $\Pi$ . Then the classical  $h^{add}$  heuristic in  $\Pi'$  gives the same value for  $s \cup \{\sigma \mapsto 0\} \cup \{\alpha_a \mapsto 0 \mid a \in O\}$  as the generalization of  $h^{add}$  to SDAC tasks defined above gives for  $s$  in  $\Pi$ . □

## Computing $h^{add}$ for SDAC:

- Option 1: Compute classical  $h^{add}$  on compiled task.
- Option 2: Compute  $Cost_a^s$  directly. How?
  - Plug EVMDDs as subgraphs into RPG
  - $\rightsquigarrow$  efficient computation of  $h^{add}$

- Background
- Compilation
- Relaxations
  - Delete Relaxations in SAS\*
  - Costs in Relaxed States
  - Additive Heuristic
  - Relaxed Planning Graph
- Abstractions
- Summary
- References

**Remark:** We can use EVMDDs to compute  $C_s^a$  and hence the generalized additive heuristic directly, by embedding them into the relaxed planning task.

We just briefly show the example, without going into too much detail.

**Idea:** Augment EVMDD with input nodes representing  $h^{add}$  values from the previous RPG layer.

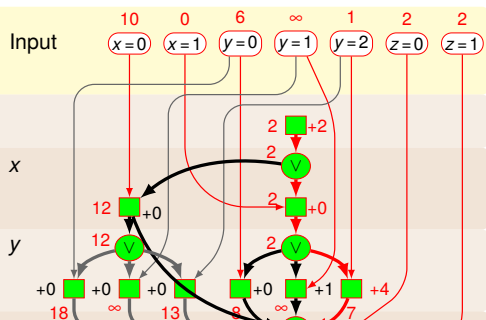
- Use augmented diagrams as RPG subgraphs.
- Allows efficient computation of  $h^{add}$ .

- Background
- Compilation
- Relaxations
  - Delete Relaxations in SAS\*
  - Costs in Relaxed States
  - Additive Heuristic
  - Relaxed Planning Graph
- Abstractions
- Summary
- References

# Option 2: RPG Compilation Option 2: Computing $Cost_a^s$

Evaluate nodes:

- $cost_a = xy^2 + z + 2$
- variable nodes become  $\vee$ -nodes
- weights become  $\wedge$ -nodes
- Augment with input nodes
- Ensure complete evaluation
- Insert  $h^{add}$  values
- $\wedge$ :  $\sum(\text{parents}) + \text{weight}$
- $\vee$ :  $\min(\text{parents})$
- $Cost_a^s =$



- Background
- Compilation
- Relaxations
  - Delete Relaxations in SAS\*
  - Costs in Relaxed States
  - Additive Heuristic
  - Relaxed Planning Graph
- Abstractions
- Summary
- References

# Additive Heuristic

- Use above construction as subgraph of RPG in each layer, for each action (as operator subgraphs).
- Add AND nodes conjoining these subgraphs with operator precondition graphs.
- Link EVMDD outputs to next proposition layer.

## Theorem

Let  $\Pi$  be an SDAC planning task. Then the classical additive RPG evaluation of the RPG constructed using EVMDDs as above computes the generalized additive heuristic  $h^{add}$  defined before. □

- Background
- Compilation
- Relaxations
  - Delete Relaxations in SAS\*
  - Costs in Relaxed States
  - Additive Heuristic
  - Relaxed Planning Graph
- Abstractions
- Summary
- References

# Abstractions

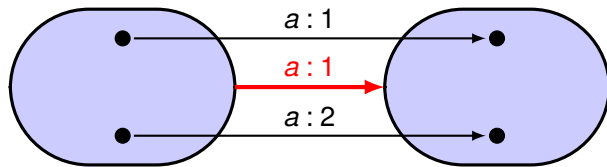
## Abstraction Heuristics for SDAC

**Question:** Why consider **abstraction heuristics**?

**Answer:**

- **admissibility**
- $\rightsquigarrow$  **optimality**

## Abstraction Heuristics for SDAC



**Question:** What are the **abstract action costs**?

**Answer:** For **admissibility**, abstract cost of  $a$  should be

$$cost_a(s^{abs}) = \min_{\substack{\text{concrete state } s \\ \text{abstracted to } s^{abs}}} cost_a(s).$$

**Problem:** exponentially many states in minimization

**Aim:** Compute  $cost_a(s^{abs})$  efficiently  
(given EVMDD for  $cost_a(s)$ ).

## Cartesian Abstractions

We will see: possible if the abstraction is **Cartesian** or coarser.

(Includes projections and domain abstractions.)

**Definition (Cartesian abstraction)**

A set of states  $s^{abs}$  is **Cartesian** if it is of the form

$$D_1 \times \dots \times D_n,$$

where  $D_i \subseteq \mathcal{D}_i$  for all  $i = 1, \dots, n$ .

An abstraction is Cartesian if all abstract states are Cartesian sets.

[Seipp and Helmert, 2013]

**Intuition:** Variables are abstracted **independently**.

$\rightsquigarrow$  **exploit independence** when computing abstract costs!

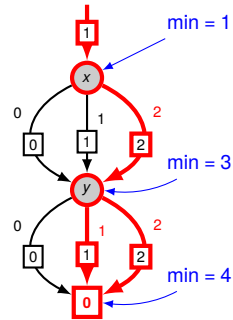
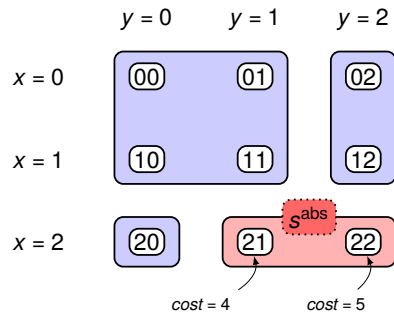


# Cartesian Abstractions

## Example (Cartesian abstraction)

Cartesian abstraction over  $x, y$

Cost  $x + y + 1$   
(edges consistent with  $s^{abs}$ )



- Background
- Compilation
- Relaxations
- Abstractions
- Cartesian Abstractions
- CEGAR
- Summary
- References

# Cartesian Abstractions

Why does the topsort EVMDD traversal (cheapest path computation) correctly compute  $cost_a(s^{abs})$ ?

Short answer: The exact same thing as with relaxed states, because **relaxed states are Cartesian sets!**

Longer answer:

- 1 For each Cartesian state  $s^{abs}$  and each variable  $v$ , each value  $d \in \mathcal{D}_v$  is either **consistent** with  $s^{abs}$  or not.
- 2 This implies: at all decision nodes associated with variable  $v$ , some outgoing edges are **enabled**, others are **disabled**. This is **independent** from all other decision nodes.
- 3 This allows **local minimizations** over linearly many **edges** instead of **global minimization** over exponentially many **paths** in the EVMDD when computing minimum costs.

~> **polynomial** in EVMDD size!

- Background
- Compilation
- Relaxations
- Abstractions
- Cartesian Abstractions
- CEGAR
- Summary
- References

# Cartesian Abstractions

Not Cartesian!

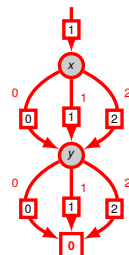
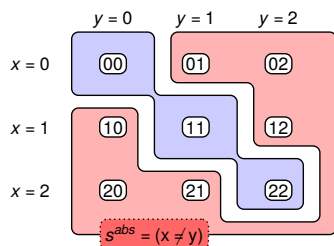
If abstraction **not Cartesian**: two variables can be

- **independent** in cost function (~> compact EVMDD), but
- **dependent** in abstraction.

~> cannot consider independent parts of EVMDD separately.

## Example (Non-Cartesian abstraction)

$cost : x + y + 1$ ,  $cost(s^{abs}) = 2$ , local minim.: 1 ~> underestimate!



- Background
- Compilation
- Relaxations
- Abstractions
- Cartesian Abstractions
- CEGAR
- Summary
- References

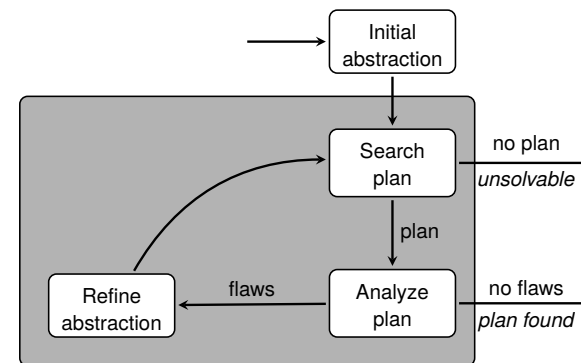
# Counterexample-Guided Abstraction Refinement

Wanted: principled way of **computing Cartesian abstractions**.

~> **Counterexample-Guided Abstraction Refinement (CEGAR)**

[Clarke et al., 2000] [Seipp and Helmert, 2013]

- Background
- Compilation
- Relaxations
- Abstractions
- Cartesian Abstractions
- CEGAR
- Summary
- References



Assume the following:

- **Initial abstraction** is one-state abstraction with single abstract state  $\mathcal{D}_1 \times \dots \times \mathcal{D}_n$ .  
 $\rightsquigarrow$  **Cartesian abstraction**
- Each **refinement step** takes one abstract state  $s^{\text{abs}} = D_1 \times \dots \times D_n$ , one variable  $v_i$ , and splits  $s^{\text{abs}}$  into
  - $D_1 \times \dots \times D_{i-1} \times D'_i \times D_{i+1} \times \dots \times D_n$
  - $D_1 \times \dots \times D_{i-1} \times D''_i \times D_{i+1} \times \dots \times D_n$
 such that  $D'_i \cap D''_i = \emptyset$  and  $D'_i \cup D''_i = D_i$ .  
 $\rightsquigarrow$  still a **Cartesian abstraction**

So, inductively:

- Initial abstraction is Cartesian.
- Each refinement step preserves being Cartesian.
- $\rightsquigarrow$  **All generated abstractions are Cartesian.**

- Background
- Compilation
- Relaxations
- Abstractions
- Cartesian Abstractions
- CEGAR
- Summary
- References

Some questions:

- **Q: When** to split abstract states?  
**A:** When first **flaw** is identified. (Details below.)
- **Q: How** to split abstract states?  
**A:** So as to **resolve that flaw**. (Details below.)

- Background
- Compilation
- Relaxations
- Abstractions
- Cartesian Abstractions
- CEGAR
- Summary
- References

Some questions:

- **Q: How long** to stay in refinement loop?  
**A:** Until one of the following **termination criteria** is met:
  - **No abstract plan** exists.  
 $\rightsquigarrow$  Terminate with result “unsolvable”.
  - Abstract plan  $\pi$  is **concretizable** (= has no flaw).  
 $\rightsquigarrow$  Return  $\pi$  as concrete plan.
  - Available **resources** (time, memory, abstraction size bound, ...) exhausted.  
 $\rightsquigarrow$  Use current abstraction as basis for **abstraction heuristic** for concrete planning task (i. e., compute abstract goal distances, store in lookup table, ...).

- Background
- Compilation
- Relaxations
- Abstractions
- Cartesian Abstractions
- CEGAR
- Summary
- References

Example (one package, one truck)

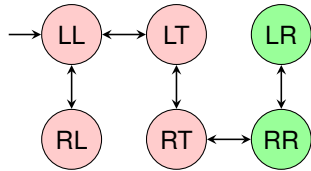
Consider the following FDR planning task  $\langle V, I, O, \gamma \rangle$ :

- $V = \{t, p\}$  with
  - $\mathcal{D}_t = \{L, R\}$
  - $\mathcal{D}_p = \{L, T, R\}$
- $I = \{t \mapsto L, p \mapsto L\}$
- $O = \{pick\text{-}in_i \mid i \in \{L, R\}\} \cup \{drop\text{-}in_i \mid i \in \{L, R\}\} \cup \{move_{i,j} \mid i, j \in \{L, R\}, i \neq j\}$ , where
  - $pick\text{-}in_i = \langle t = i \wedge p = i, p := T \rangle$
  - $drop\text{-}in_i = \langle t = i \wedge p = T, p := i \rangle$
  - $move_{i,j} = \langle t = i, t := j \rangle$
- $\gamma = (p = R)$ .

- Background
- Compilation
- Relaxations
- Abstractions
- Cartesian Abstractions
- CEGAR
- Summary
- References

## Example (Ctd.)

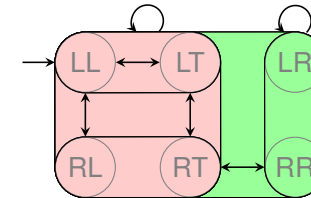
Before we look at CEGAR applied to this task, here is the concrete transition system (just for reference):



- Background
- Compilation
- Relaxations
- Abstractions
- Cartesian Abstractions
- CEGAR
- Summary
- References

## Example (Ctd.)

Refinement step 0 (initial abstraction): Refinement step 1:  
 Refinement step 2: Refinement step 3: Refinement step 4:



### Abstract plan:

$$\pi_0 = \langle \rangle \pi_1 = \langle drop-in_R \rangle \pi_2 = \langle move_{L,R}, drop-in_R \rangle \pi_3 = \langle move_{L,R}, drop-in_R \rangle \pi_4 = \langle pick-in_L, move_{L,R}, drop-in_R \rangle$$

Flaw:  $s_0 = LL$  is not a goal state.

Preconditions ( $\tau = H$ ) and ( $p = I$ ) of  $drop-in_R$  not satisfied in  $s_0 = LL$ . Precondition ( $p = T$ ) of  $drop-in_R$  not satisfied in  $s_1 = RL$ .

CEGAR for unit-cost tasks. Three kinds of flaws:

- Abstract plan works in concrete transition system, but ends in **non-goal state**. (Step 0 in example.)
- Some step of abstract plan fails in concrete transition system, because **operator precondition is violated**. (Steps 1 and 2 in example.)
- Concrete and abstract **paths diverge** at some point, because abstract transition system is nondeterministic. (Step 3 in example.)

- Background
- Compilation
- Relaxations
- Abstractions
- Cartesian Abstractions
- CEGAR
- Summary
- References

Flaw 1: Abstract plan terminates in concrete non-goal state.

Resolution: Split abstraction of last state  $s_n$  of concrete trace into (a) part containing  $s_n$ , but containing no concrete goal state, and (b) rest.

- Background
- Compilation
- Relaxations
- Abstractions
- Cartesian Abstractions
- CEGAR
- Summary
- References

**Flaw 2:** Abstract plan fails because some operator precondition is violated.

**Resolution:** Split abstraction of state  $s_{i-1}$  of concrete trace, where operator precondition  $\chi$  is violated, into (a) part containing  $s_{i-1}$ , but no concrete state in which precondition  $\chi$  is satisfied, and (b) rest.

**Flaw 3:** Concrete and abstract paths diverge.

**Resolution:** Split abstraction of state  $s_{i-1}$  of concrete trace, after which paths diverge when applying operator  $o$ , into (a) part containing  $s_{i-1}$  where applying  $o$  always leads to the “wrong” abstract successor state, and (b), rest.

**Remark:** In tasks with state-dependent action costs, there is a fourth type of flaws, so-called **cost-mismatch** flaws.

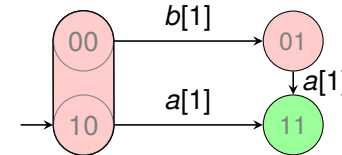
**Flaw 4:** Action is **more costly** in concrete state than in abstract state.

**Resolution:** Split abstraction of violating concrete state into two parts that differ on the value of a variable that is relevant to the cost function of the operator in question, such that we have different cost values in the two parts.

**Example (Cost-mismatch flaw)**

$$a = \langle T, x \wedge y \rangle, \text{ cost}_a = 2x + 1 \quad s_0 = 10$$

$$b = \langle T, \neg x \wedge y \rangle, \text{ cost}_b = 1 \quad s_* = x \wedge y$$



- **Optimal abstract plan:**  $\langle a \rangle$  (abstract cost 1)
- This is also a **concrete plan** (concrete cost  $3 \neq 1$ )  
 $\rightsquigarrow$  split  $\{0, 1\} \times \{0\}$
- **Cf. optimal concrete plan:**  $\langle b, a \rangle$  (concr. and abstr. cost 2)

# Summary


### Summary:


- State-dependent actions costs practically relevant.
- EVMDDs exhibit and exploit structure in cost functions.
- Graph-based representations of arithmetic functions.
- Edge values express partial cost contributed by facts.
- Size of EVMDD is **compact** in many “typical” cases.
- Can be used to compile tasks with state-dependent costs to tasks with state-independent costs.
- Alternatively, can be embedded into the RPG to compute forward-cost heuristics directly.
- For  $h^{add}$ , both approaches give the same heuristic values.
- Abstraction heuristics can also be generalized to state-dependent action costs.


### Future Work and Work in Progress:

- Investigation of other delete-relaxation heuristics for tasks with state-dependent action costs.
- Investigation of static and dynamic EVMDD variable orders.
- Application to cost partitioning, to planning with preferences, ...
- Better integration of SDAC in PDDL.
- Tool support.
- Benchmarks.

# References

 *Ciardo and Siminiceanu*, **Using edge-valued decision diagrams for symbolic generation of shortest paths**, in Proc. 4th Intl. Conference on Formal Methods in Computer-Aided Design (FMCAD 2002), pp. 256–273, 2002.

 *Geißer, Keller, and Mattmüller*, **Delete relaxations for planning with state-dependent action costs**, in Proc. 24th Intl. Joint Conference on Artificial Intelligence (IJCAI 2015), pp. 1573–1579, 2015.

 *Geißer, Keller, and Mattmüller*, **Abstractions for planning with state-dependent action costs**, in Proc. 26th Intl. Conference on Automated Planning and Scheduling (ICAPS 2016), pp. 140–148, 2016.

Background

Compilation

Relaxations

Abstractions

Summary

References