Certified Unsolvability in Classical Planning

2. Applications

Salomé Eriksson Gabriele Röger Malte Helmert

University of Basel, Switzerland

ICAPS 2020

Certified Unsolvability in Classical Planning

2. Applications

Certificate Structure

First look

on Github: `https://github.com/salome-eriksson/helve`

A certificate consists of the following files:

- task description
  - limited to STRIPS
    → STRIPS with negation coming soon™
  - variable and action IDs (according to occurence in list)
- certificate
  - state sets: `e ID type description`
    `e 0 h p cnf 3 2 2 -1 0`
  - action sets: `a ID type description`
    `a 0 u 2 4 5 6`
  - statements: `k ID type set-ID(s) justification premises k 0 d 7 sd 5 4`
- (BDD descriptions)

(detailed explanation in README.md of github repository)

Certificate File

Example

what we have seen so far:

#	statement	justification
(0)	$\emptyset$ dead	**ED**
(1)	$\{\mathbf{I}\} \sqsubseteq states(\neg a \lor \neg b)$	**B1**
(2)	$S_{\neg a \lor \neg b}[\mathbf{A}] \sqsubseteq S_{\neg a \lor \neg b} \cup \emptyset$	**B2**
(3)	$\overline{S_{\neg a \lor \neg b}}$ dead	**PI** with (3),(1) and (2)

...

the corresponding certificate file:

```
e 0 c e
e 1 c i
a 0 a
k 0 d 0 ed
e 2 h p cnf 2 1 1 2 0
k 1 s 1 2 b1
e 3 p 2 0
e 4 u 2 0
k 2 s 3 4 b2
e 5 n 2
k 3 d 5 pi 2 0 1
...
```

→ Demo

Blind Search

High-Level Certificate

Blind search explores all reachable states $\mathcal{R}^{\Pi}$.
→ Recall completeness proof:

Forward Blind Search Certificate

#	statement	justification
(0)	$\emptyset$ dead	**ED**
(1)	$\mathcal{R}^{\Pi}[\mathbf{A}] \sqsubseteq \mathcal{R}^{\Pi} \cup \emptyset$	**B2**
(2)	$\mathcal{R}^{\Pi} \cap \mathbf{S}_G \sqsubseteq \emptyset$	**B1**
(3)	$\mathcal{R}^{\Pi} \cap \mathbf{S}_G$ dead	**SD** with (0) and (2)
(4)	$\mathcal{R}^{\Pi}$ dead	**PG** with (1), (0) and (3)
(5)	$\{\mathbf{I}\} \sqsubseteq \mathcal{R}^{\Pi}$	**B1**
(6)	$\{\mathbf{I}\}$ dead	**SD** with (4) and (5)
(7)	unsolvable	**CI** with (6)

For backwards search, show $\{\mathbf{I}\} \sqsubseteq \overline{\mathcal{R}^{\Pi}_B}$ and $[\mathbf{A}]\mathcal{R}^{\Pi}_B \sqsubseteq \mathcal{R}^{\Pi}_B \cup \emptyset$, deduce $\mathcal{R}^{\Pi}_B$ dead (rule **RI**), and from $\mathbf{S}_G \sqsubseteq \mathcal{R}^{\Pi}_B$ show $\mathbf{S}_G$ dead.

Translation to Certificate File

e 0 c e		$S_0 = \emptyset$
e 1 c i		$S_1 = \{\mathbf{I}\}$
e 2 c g		$S_2 = \mathbf{S}_G$
a 0 a		$A_0 = \mathbf{A}$
k 0 d 0 ed	(0)	$S_0 (= \emptyset)$ dead
e 3 ...		$S_3 = \mathcal{R}^{\Pi}$
e 4 p 3 0		$S_4 = S_3[A_0] = \mathcal{R}^{\Pi}[\mathbf{A}]$
e 5 u 3 0		$S_5 = S_3 \cup S_0 = \mathcal{R}^{\Pi} \cup \emptyset$
k 1 s 4 5 b2	(1)	$S_4 (= \mathcal{R}^{\Pi}[\mathbf{A}]) \sqsubseteq S_5 (= \mathcal{R}^{\Pi} \cup \emptyset)$
e 6 i 3 2		$S_6 = S_3 \cap S_2 = \mathcal{R}^{\Pi} \cap \mathbf{S}_G$
k 2 s 6 0 b1	(2)	$S_6 (= \mathcal{R}^{\Pi} \cap \mathbf{S}_G) \sqsubseteq S_0 (= \emptyset)$
k 3 d 6 sd 2 0	(3)	$S_6 (= \mathcal{R}^{\Pi} \cap \mathbf{S}_G)$ dead
k 4 d 3 pg 1 0 3	(4)	$S_3 (= \mathcal{R}^{\Pi})$ dead
k 5 s 1 3 b1	(5)	$S_1 (= \{\mathbf{I}\}) \sqsubseteq S_3 (= \mathcal{R}^{\Pi})$
k 6 d 1 sd 5 4	(6)	$S_1 (= \{\mathbf{I}\})$ dead
k 7 u ci 6	(7)	unsolvable

Implementation Details

Depending on the concrete algorithm, some implementation details affect performance:

- formalism for $\mathcal{R}^\Pi$
  - symbolic search: BDD
    $\rightarrow$ overhead if no singular closed BDD!
  - explicit search: explicit enumeration or BDD
    $\rightarrow$ fast generation vs fast verification
- when to build the certificate
  - during search: unnecessary overhead for solvable problems
  - at the end: more overhead (iterate over entire closed list), but also more localized

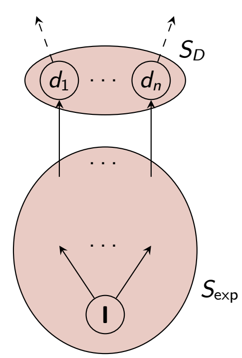
Heuristic Search

Idea

1. Each dead-end is dead.
2. The set of expanded states contains no goal state.
3. The set of expanded states can only reach itself and dead-ends.
4. $\rightarrow$ the set of expanded states is dead.
5. The initial state is in the set of expanded states.
6. $\rightarrow$ The initial state is dead and the task unsolvable.

High-Level Certificate

(1) $\emptyset$ dead
(2) $\{d_1\}$ dead (3) $\{d_2\}$ dead ... (4) $\{d_n\}$ dead



#	statement	justification
(5)	$\{d_1\} \cup \{d_2\}$ dead	**UD** with (2) and (3)
	...	
(6)	$\bigcup_{1 \le i \le n} d_i$ dead	**UD** with ...
(7)	$S_D \sqsubseteq \bigcup_{1 \le i \le n} d_i$	**B1**
(8)	S_D dead	**SD** with (6) and (7)
(9)	$S_{exp}[\mathbf{A}] \sqsubseteq S_{exp} \cup S_D$	**B2**
(10)	$S_{exp} \cap \mathbf{S}_G \sqsubseteq \emptyset$	**B1**
(11)	$S_{exp} \cap \mathbf{S}_G$ dead	**SD** with (1) and (10)
(12)	S_{exp} dead	**PG** with (9), (8) and (11)
(13)	$\{\mathbf{I}\} \sqsubseteq S_{exp}$	**B1**
(14)	$\{\mathbf{I}\}$ dead	**SD** with (12) and (13)
(15)	task unsolvable	**CI** with (14)

Bridging Representations

Statements "$\{d_i\}$ dead" might use different representations.

1 Show $\{d_i\}_{\textbf{explicit}} \sqsubseteq \{d_i\}_{\textbf{R}}$ (basic statement **B4**)

and then either

2a build $(S_{\text{exp}})_{\textbf{explicit}}$, and

3a show $(S_{\text{exp}})_{\textbf{explicit}} \sqsubseteq (S_{\text{exp}})_{\textbf{explicit}} \cup \bigcup \{d_i\}_{\textbf{explicit}}$ (**B2**).

or

2b build $(S_D)_{\textbf{explicit}}$, $(S_D)_{\textbf{BDD}}$ and $(S_{\text{exp}})_{\textbf{BDD}}$,

3b show $(S_D)_{\textbf{explicit}} \sqsubseteq \bigcup \{d_i\}_{\textbf{explicit}}$ (**B2**),

4b show $(S_D)_{\textbf{BDD}} \sqsubseteq (S_D)_{\textbf{explicit}}$ (**B4**), and

5b show $(S_{\text{exp}})_{\textbf{BDD}} \sqsubseteq (S_{\text{exp}})_{\textbf{BDD}} \cup (S_D)_{\textbf{BDD}}$ (**B2**).

$\rightarrow$ tradeoff efficient generation vs efficient verification

Delete Relaxation

$h^{\max}$ dead-end

$h^{\max}(s) = \infty \leftrightarrow$ some $g \in \mathbf{G}$ relaxed unreachable

Consider $R_u^+(s) = \{v \mid v$ relaxed unreachable from $s\}$ and $\varphi = \bigwedge_{v \in R_u^+(s)} \neg v$.

- We can't reach any s' containing any $v \in R_u^+(s)$: $S_\varphi[\mathbf{A}] \subseteq S_\varphi$
- All states satisfying φ do not satisfy g: $S_\varphi \cap \mathbf{S}_G = \emptyset$
- State s satisfies φ: $\{s\} \subseteq S_\varphi$

$\rightarrow$ Show that S_φ is dead (**PG**) and thus s is dead (**SD**).

We can choose between different representations:
BDD, Horn formula, 2CNF formula, explicit (over $R_u^+(s)$)

h^{m} & Clause-Learning State Space Search

h^{m} dead-ends:

- same concept as $h^{\max}$: $\varphi = \bigwedge_{t \in R_u^m(s)} \bigvee_{v \in t} \neg v$, where $R_u^m(s)$ are the tuples unreachable from s.
- representation: Horn formulas (or 2CNF formulas for $m = 2$)
  $\rightarrow$ BDDs not suited [Edelkamp & Kissmann 2011]

Clause-Learning State Space Search [Steinmetz & Hoffmann (2017)]:

- uses h^{C} $\rightarrow$ same concept (again)
- can be refined to detect **I** as dead-end $\rightarrow$ compact certificate
- uses additional source for mutexes
  $\rightarrow$ Integrate additional information into certificate!

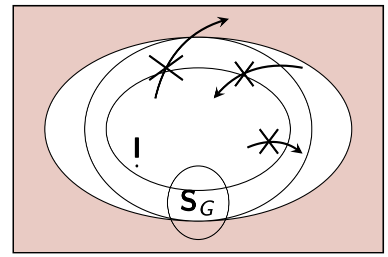
Other heuristics

- $h^{\max}$ approach covers all delete-relaxation heuristics ($h^{\text{LM-Cut}}$, landmarks based on delete relaxation, ...)
- Merge & Shrink:
  - transformation from Merge & Shrink representation to ADD [Helmert et al. 2014] and extract ∞-paths to BDD
    $\rightarrow$ limited to linear merge strategies [Helmert et al. 2015]
  - one set for all dead-ends
    $\rightarrow$ certificate more compact
  - implementation detail: unreachable and dead-end states merged
    $\rightarrow$ disable for certificate generation

h^2 Preprocessor

Algorithm Overview

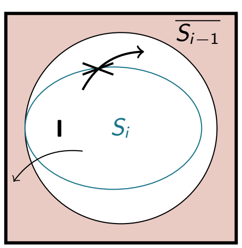
- introduced in [Alcázar & Torralba (2015)]
- preprocessing step that simplifies planning task
- used in many IPC planners
- incremental h^2 reachability analysis, alternating between forward and backward
  $\rightarrow$ remove unreachable facts and actions



High-Level Certificate

- D_i: set of literal pairs shown dead before or in iteration i
- $S_i = \{s \mid \{p,q\} \not\subseteq s \text{ for all } \{p,q\} \in D_i\}$
- start with iteration 1 and $D_0 = \{\} \rightarrow \overline{S_0}(=\{\})$ dead

Forward iteration i



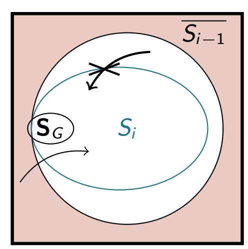
given: (1) $\overline{S_{i-1}}$ dead

#	statement	justification
(2)	$\{\mathbf{I}\} \sqsubseteq S_i$	**B1**
(3)	$S_i[\mathbf{A}] \sqsubseteq S_i \cup \overline{S_{i-1}}$	**B2**
(4)	$\overline{S_i}$ dead	**PI** with (3), (1) and (2)

High-Level Certificate

- D_i: set of literal pairs shown dead before or in iteration i
- $S_i = \{s \mid \{p,q\} \not\subseteq s \text{ for all } \{p,q\} \in D_i\}$
- start with iteration 1 and $D_0 = \{\} \rightarrow \overline{S_0}(=\{\})$ dead

Backward iteration i



given: (1) $\overline{S_{i-1}}$ dead

#	statement	justification
(2)	$\overline{S_i} \cap \mathbf{S}_G \sqsubseteq \overline{S_{i-1}}$	**B1**
(3)	$\overline{S_i} \cap \mathbf{S}_G$ dead	**SD** with (1) and (2)
(4)	$[\mathbf{A}]S_i \sqsubseteq S_i \cup \overline{S_{i-1}}$	**B2**
(5)	$\overline{S_i}$ dead	**RG** with (4), (1) and (3)

Remarks

- representation of S_i: $\bigwedge_{\{p,q\}\in D_i} \neg p \vee \neg q$
  $\rightarrow$ 2CNF (Horn not suitable since p and q can be negative)
- If the h^2 preprocessor detects the task unsolvable, we can extract a full proof:
  - ends in forward iteration: $\mathbf{S}_G \subseteq \overline{S_n}$ (all goal states dead)
  - ends in backward iteration: $\{\mathbf{I}\} \subseteq \overline{S_n}$ (initial state dead)
- Otherwise, we can use the statement "$\overline{S_n}$ dead" to explain why we pruned certain states.
- We can also extract more fine-grained statements such as "$S_{p\wedge q}$ dead" within the proof system.

Recap

Take-Home Messages

- We can verify algorithms on different levels (unit tests, certifying algorithms, theorem provers).
- The unsolvability proof system incrementally deduces knowledge about dead states.
- Its modularity enables us to combine different sources of information.
- Efficient verification depends on the representation of state sets, i.e. which operations are efficiently supported.
- Different representations can offer tradeoffs between efficient generation and verification.
- Generating certificates often involves reachability arguments.

Future Work

- cover more planning techniques, e.g.
  - dead-end potentials
  - partial order reduction
  - task transformations
- extend the verifier
  - more representations
    $\rightarrow$ talk in Session E2 next week about CNF
  - more inference rules
- verify the verifier!