Alternatives to Explicit State Space Search
Symbolic Search

Álvaro Torralba & Daniel Gnad

SAARLAND UNIVERSITY

SAARBRÜCKEN
GRADUATE SCHOOL OF
COMPUTER SCIENCE

June 19, 2017
About us

Dr. Álvaro Torralba

Daniel Gnad

Saarland University, Saarbrücken, Germany
Target audience:

Ideally, you are..

- .. familiar with **Classical Planning Formalisms** (FDR/SAS$^+$).
- .. familiar with **Planning as Heuristic Search**.
- .. aware of an important issue in Explicit State Space Search → **State Space Explosion**
Target audience:

Ideally, you are:

- familiar with **Classical Planning Formalisms (FDR/SAS⁺)**.
- familiar with **Planning as Heuristic Search**.
- aware of an important issue in Explicit State Space Search → **State Space Explosion**

Don’t hesitate to ask questions if something is unclear!
Symbolic Search:

There have been many tutorials on the usefulness of Decision Diagrams: → Here: focus on symbolic search algorithms
Agenda

1. About this Tutorial
2. Classical Planning: Models, Approaches
3. Symbolic Representation of Planning Tasks
4. Symbolic Blind Search
5. Heuristic Search
6. Symbolic Abstraction Heuristics
7. Implementation
8. Conclusions and Open Challenges
Agenda

1. About this Tutorial

2. Classical Planning: Models, Approaches

3. Symbolic Representation of Planning Tasks

4. Symbolic Blind Search

5. Heuristic Search

6. Symbolic Abstraction Heuristics

7. Implementation

8. Conclusions and Open Challenges
Definition. A planning task is a 4-tuple $\Pi = (V, A, I, G)$ where:

- $V$ is a set of state variables, each $v \in V$ with a finite domain $D_v$.
- $A$ is a set of actions; each $a \in A$ is a triple $(\text{pre}_a, \text{eff}_a, c_a)$, of precondition and effect (partial assignments), and the action’s cost $c_a \in \mathbb{R}^{0+}$.
- Initial state $I$ (complete assignment), goal $G$ (partial assignment).
**Definition.** A *planning task* is a 4-tuple \( \Pi = (V, A, I, G) \) where:

- **V** is a set of *state variables*, each \( v \in V \) with a finite domain \( D_v \).
- **A** is a set of *actions*; each \( a \in A \) is a triple \((\text{pre}_a, \text{eff}_a, c_a)\), of *precondition* and *effect* (partial assignments), and the action's *cost* \( c_a \in \mathbb{R}^{0+} \).
- **Initial state** \( I \) (complete assignment), **goal** \( G \) (partial assignment).

**Running Example:**

- \( V = \{t, p_1, p_2, p_3, p_4\} \)
  with \( D_t = \{l_1, l_2, l_3\} \) and \( D_{p_i} = \{t, l_1, l_2, l_3\} \).
- \( A = \{\text{load}(p_i, x), \text{unload}(p_i, x), \text{drive}(x, x')\} \)
Semantics – The State Space of a Planning Task

**Definition.** Let $\Pi = (V, A, I, G)$ be an FDR planning task. The *state space of $\Pi$* is the labeled transition system $\Theta_\Pi = (S, L, c, T, I, S^G)$ where:

- The *states* $S$ are the complete variable assignments.
- The *labels* $L = A$ are $\Pi$’s actions; the *cost function* $c$ is that of $\Pi$.
- The *transitions* are $T = \{ s \xrightarrow{a} s' \mid \text{pre}_a \subseteq s, s' = s[a] \}$.
  - If $\text{pre}_a \subseteq s$, then $a$ is *applicable* in $s$ and, for all $v \in V$, $s[a][v] := \text{eff}_a[v]$ if $\text{eff}_a[v]$ is defined and $s[a][v] := s[v]$ otherwise.
  - If $\text{pre}_a \not\subseteq s$, then $s[a]$ is *undefined*.
- The *initial state* $I$ is identical to that of $\Pi$.
- The *goal states* $S^G = \{ s \in S \mid G \subseteq s \}$ are those that satisfy $\Pi$’s goal.
Semantics – The State Space of a Planning Task

Definition. Let \( \Pi = (V, A, I, G) \) be an FDR planning task. The state space of \( \Pi \) is the labeled transition system \( \Theta_\Pi = (S, L, c, T, I, S^G) \) where:

- The states \( S \) are the complete variable assignments.
- The labels \( L = A \) are \( \Pi \)'s actions; the cost function \( c \) is that of \( \Pi \).
- The transitions are \( T = \{ s \xrightarrow{a} s' \mid \text{pre}_a \subseteq s, \ s' = s[a] \} \).
  
  If \( \text{pre}_a \subseteq s \), then \( a \) is applicable in \( s \) and, for all \( v \in V \), \( s[a][v] := \text{eff}_a[v] \) if \( \text{eff}_a[v] \) is defined and \( s[a][v] := s[v] \) otherwise.
  
  If \( \text{pre}_a \not\subseteq s \), then \( s[a] \) is undefined.
- The initial state \( I \) is identical to that of \( \Pi \).
- The goal states \( S^G = \{ s \in S \mid G \subseteq s \} \) are those that satisfy \( \Pi \)'s goal.

→ Solution ("Plan"): Action sequence mapping \( I \) into \( s \in S^G \).
Optimal plan: Minimum summed-up cost.
A successful approach: Heuristic Search

\[ \text{init} \quad \bigcirc \quad \text{goal} \quad \bullet \]
A successful approach: Heuristic Search
A successful approach: Heuristic Search

→ Forward state space search. Heuristic function $h$ maps states $s$ to an estimate $h(s)$ of goal distance.
Alternatives to State Space Search (not covered here)

- **Planning as SAT**: Extensions use, e.g., heuristics, symmetry breaking. [KS92, KS96, EMW97, Rin98, Rin03, Rin12]

- **Property Directed Reachability**
  [Bra11, EMB11, Sud14]

- **Planning via Petri Net Unfolding**
  [GW91, McM92, ERV02, ELL04, HRTW07, BHHT08, BHK+14]

- **Partial-order Planning**
  [Sac75, KKY95, YS03, BGB13]

- **Factored Planning**
  [Kno94, AE03, BD06, KBHT07, BD08, BD13, FJHT10]

- ...
State Space Explosion

Huge branching factor $\rightarrow$ state space explosion
State Space Explosion

Huge branching factor $\rightarrow$ state space \textit{explosion}

BDDs to the rescue!

$g = 0$

$g = 1$

$g = 2$

goal
About this Tutorial

Classical Planning: Models, Approaches

Symbolic Representation of Planning Tasks

Symbolic Blind Search

Heuristic Search

Symbolic Abstraction Heuristics

Implementation

Conclusions and Open Challenges
Sets of States as Logical Formulas

\[ l_1 \quad \quad l_2 \quad \quad l_3 \]

\[ \langle t, l_1 \rangle \land \langle p_1, l_1 \rangle \land \langle p_2, l_1 \rangle \]

Disclaimer: In propositional logic there is no closed-world assumption. In our examples, we ignore state invariants: \[ \langle t, l_1 \rangle \leftrightarrow (\neg \langle t, l_2 \rangle \land \neg \langle t, l_3 \rangle), \ldots \]
Sets of States as Logical Formulas

Disclaimer: In propositional logic there is no closed-world assumption. In our examples, we ignore state invariants: \( \langle t, l_1 \rangle \leftrightarrow (\neg \langle t, l_2 \rangle \land \neg \langle t, l_3 \rangle) \), ...
Sets of States as Logical Formulas

Disclaimer: In propositional logic there is no closed-world assumption. In our examples, we ignore state invariants: $\langle t, l_1 \rangle \leftrightarrow (\neg \langle t, l_2 \rangle \land \neg \langle t, l_3 \rangle)$, ...
Sets of States as Logical Formulas

\[ \langle t, l_1 \rangle \land \langle p_1, l_1 \rangle \land \langle p_2, l_1 \rangle \lor \langle t, l_1 \rangle \land \langle p_1, l_2 \rangle \land \langle p_2, l_1 \rangle \lor \langle t, l_1 \rangle \land \langle p_1, l_1 \rangle \land \langle p_2, l_2 \rangle \lor \langle t, l_1 \rangle \land \langle p_1, l_2 \rangle \land \langle p_2, l_2 \rangle \]

\[ \langle t, l_1 \rangle \land (\langle p_1, l_1 \rangle \lor \langle p_1, l_2 \rangle) \land (\langle p_2, l_1 \rangle \lor \langle p_2, l_2 \rangle) \]

Disclaimer: In propositional logic there is no closed-world assumption. In our examples, we ignore state invariants: \[ \langle t, l_1 \rangle \leftrightarrow (\neg \langle t, l_2 \rangle \land \neg \langle t, l_3 \rangle), \ldots \]
## Operating with Sets of States as Logical Formulas

<table>
<thead>
<tr>
<th>Sets</th>
<th>Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty set</td>
<td>( \bot )</td>
</tr>
<tr>
<td>All states</td>
<td>( \top )</td>
</tr>
<tr>
<td>All states in which the truck is at ( l_1 )</td>
<td>( \langle t, l_1 \rangle )</td>
</tr>
</tbody>
</table>
### Operating with Sets of States as Logical Formulas

<table>
<thead>
<tr>
<th>Sets</th>
<th>Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Empty set</td>
<td>$\bot$</td>
</tr>
<tr>
<td>All states</td>
<td>$\top$</td>
</tr>
<tr>
<td>All states in which the truck is at $l_1$</td>
<td>$\langle t, l_1 \rangle$</td>
</tr>
<tr>
<td>Union ($\cup$)</td>
<td>Disjunction ($\lor$)</td>
</tr>
<tr>
<td>Intersection ($\cap$)</td>
<td>Conjunction ($\land$)</td>
</tr>
<tr>
<td>Complement</td>
<td>Negation ($\neg$)</td>
</tr>
</tbody>
</table>
How to Represent Logical Formulas in Practice?

<table>
<thead>
<tr>
<th>Normal Form/Decision Diagram</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Negation NF (NNF)</td>
<td></td>
</tr>
<tr>
<td>Disjunctive NF (DNF)</td>
<td></td>
</tr>
<tr>
<td>Conjunctive NF (CNF)</td>
<td></td>
</tr>
<tr>
<td>Binary DD (BDD)</td>
<td>[Bry86]</td>
</tr>
<tr>
<td>Zero-sup DD (ZDD)</td>
<td>[Min93]</td>
</tr>
<tr>
<td>Sentential DD (SDD)</td>
<td>[Dar11]</td>
</tr>
<tr>
<td>Determ. DNNF (d-DNNF)</td>
<td>[Dar02]</td>
</tr>
<tr>
<td>Decomp. NNF (DNNF)</td>
<td>[Dar01]</td>
</tr>
</tbody>
</table>
## How to Represent Logical Formulas in Practice?

<table>
<thead>
<tr>
<th>Normal Form/Decision Diagram</th>
<th>∨</th>
<th>∧</th>
<th>¬</th>
<th>σ ≡ ⊤</th>
<th>σ ≡ ⊥</th>
<th>σ ≡ σ'</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negation NF (NNF)</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>co-NP</td>
<td>NP</td>
<td>co-NP</td>
</tr>
<tr>
<td>Disjunctive NF (DNF)</td>
<td>P</td>
<td>E</td>
<td>E</td>
<td>co-NP</td>
<td>P</td>
<td>co-NP</td>
</tr>
<tr>
<td>Conjunctive NF (CNF)</td>
<td>E</td>
<td>P</td>
<td>E</td>
<td>P</td>
<td>NP</td>
<td>co-NP</td>
</tr>
<tr>
<td>Binary DD (BDD) [Bry86]</td>
<td>E/P</td>
<td>E/P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>Zero-sup DD (ZDD) [Min93]</td>
<td>E/P</td>
<td>E/P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>Sentential DD (SDD) [Dar11]</td>
<td>E/P*</td>
<td>E/P*</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>Determ. DNNF (d-DNNF) [Dar02]</td>
<td>P</td>
<td>E</td>
<td>E</td>
<td>co-NP</td>
<td>P</td>
<td>co-NP</td>
</tr>
<tr>
<td>Decomp. NNF (DNNF) [Dar01]</td>
<td>P</td>
<td>E</td>
<td>E</td>
<td>co-NP</td>
<td>P</td>
<td>co-NP</td>
</tr>
</tbody>
</table>

*: In SDDs, ∨ and ∧ with compression is not polynomial.

**P**: polynomial in the size of the representation

---

Á. Torralba, D. Gnad

Symbolic State Space Search

15/54
### How to Represent Logical Formulas in Practice?

<table>
<thead>
<tr>
<th>Normal Form/Decision Diagram</th>
<th>$\lor$</th>
<th>$\land$</th>
<th>$\neg$</th>
<th>$\sigma \equiv \top$</th>
<th>$\sigma \equiv \bot$</th>
<th>$\sigma \equiv \sigma'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Negation NF (NNF)</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>co-NP</td>
<td>NP</td>
<td>co-NP</td>
</tr>
<tr>
<td>Disjunctive NF (DNF)</td>
<td>P</td>
<td>E</td>
<td>E</td>
<td>co-NP</td>
<td>P</td>
<td>co-NP</td>
</tr>
<tr>
<td>Conjunctive NF (CNF)</td>
<td>E</td>
<td>P</td>
<td>E</td>
<td>P</td>
<td>NP</td>
<td>co-NP</td>
</tr>
<tr>
<td>Binary DD (BDD) [Bry86]</td>
<td>E/P</td>
<td>E/P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>Zero-sup DD (ZDD) [Min93]</td>
<td>E/P</td>
<td>E/P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>Sentential DD (SDD) [Dar11]</td>
<td>E/P*</td>
<td>E/P*</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>Determ. DNNF (d-DNNF) [Dar02]</td>
<td>P</td>
<td>E</td>
<td>E</td>
<td>co-NP</td>
<td>P</td>
<td>co-NP</td>
</tr>
<tr>
<td>Decomp. NNF (DNNF) [Dar01]</td>
<td>P</td>
<td>E</td>
<td>E</td>
<td>co-NP</td>
<td>P</td>
<td>co-NP</td>
</tr>
</tbody>
</table>

*: In SDDs, $\lor$ and $\land$ with compression is not polynomial.

**P**: polynomial in the size of the representation
Multi-valued Decision Diagram (MDD)

DAG with a **fixed variable ordering**.

<table>
<thead>
<tr>
<th></th>
<th>$t$</th>
<th>$p_1$</th>
<th>$p_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$t$</td>
<td></td>
</tr>
<tr>
<td>$l_2$</td>
<td>$l_1$</td>
<td>$l_1$</td>
<td></td>
</tr>
<tr>
<td>$l_1$</td>
<td>$t$</td>
<td>$l_1$</td>
<td></td>
</tr>
</tbody>
</table>

Binary Decision Diagrams: MDDs where variables are all binary

→ Compilation that uses $\log_2 |Dv|$ binary variables per FDR variable
**Binary Decision Diagrams (BDDs)**

---

**Multi-valued Decision Diagram (MDD)**

DAG with a **fixed variable ordering**.

Reduction rules:

1. Each node represented only once
2. Nodes whose children are all the same are ommited

```
<table>
<thead>
<tr>
<th>t</th>
<th>p1</th>
<th>p2</th>
</tr>
</thead>
<tbody>
<tr>
<td>l1</td>
<td>l1</td>
<td>t</td>
</tr>
<tr>
<td>l2</td>
<td>l1</td>
<td>l1</td>
</tr>
<tr>
<td>l1</td>
<td>t</td>
<td>l1</td>
</tr>
</tbody>
</table>
```

---

Compilation that uses $\log_2 |D_v|$ binary variables per FDR variable.
Binary Decision Diagrams (BDDs)

Multi-valued Decision Diagram (MDD)

DAG with a fixed variable ordering.
Reduction rules:
1. Each node represented only once
2. Nodes whose children are all the same are omitted

<table>
<thead>
<tr>
<th>$t$</th>
<th>$p_1$</th>
<th>$p_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$t$</td>
</tr>
<tr>
<td>$l_2$</td>
<td>$l_1$</td>
<td>$l_1$</td>
</tr>
<tr>
<td>$l_1$</td>
<td>$t$</td>
<td>$l_1$</td>
</tr>
</tbody>
</table>

$T$ $F$
Binary Decision Diagrams (BDDs)

Multi-valued Decision Diagram (MDD)

DAG with a fixed variable ordering.
Reduction rules:
1. Each node represented only once
2. Nodes whose children are all the same are omitted

\[
\begin{array}{ccc}
  t & p_1 & p_2 \\
  l_1 & l_1 & t \\
  l_2 & l_1 & l_1 \\
  l_1 & t & l_1 \\
\end{array}
\]

Binary Decision Diagrams: MDDs where variables are all binary
→ Compilation that uses \( \log_2 |D_v| \) binary variables per FDR variable \( v \)
Multi-valued Decision Diagram (MDD)

DAG with a fixed variable ordering.

Reduction rules:

1. Each node represented only once
2. Nodes whose children are all the same are omitted

<table>
<thead>
<tr>
<th>t</th>
<th>p₁</th>
<th>p₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>l₁</td>
<td>l₁</td>
<td>t</td>
</tr>
<tr>
<td>l₂</td>
<td>l₁</td>
<td>l₁</td>
</tr>
<tr>
<td>l₁</td>
<td>t</td>
<td>l₁</td>
</tr>
</tbody>
</table>

Binary Decision Diagrams: MDDs where variables are all binary

→ Compilation that uses $\log_2 |D_v|$ binary variables per FDR variable $v$
BDD Variable Ordering

\[(x_1 \neq y_1) \lor (x_2 \neq y_2) \lor (x_3 \neq y_3)\]

Exponential \((> 2^{n+1})\)

Polynomial \((3n + 2)\)
Practical Strategies for a Good Variable Ordering

**Static Variable Ordering:** Put causally-related variables close [KE11]

Choose ordering \( o \) that minimizes

\[
\sum_{v_i, v_j \in CG} d_o(v_i, v_j)^2
\]

→ No strong theoretical guarantees [KH13] but compares well against other alternatives [BRKM91, CHP93, Mai09, MWBSV88, MIY90]
Practical Strategies for a Good Variable Ordering

**Static Variable Ordering:** Put causally-related variables close [KE11]

Choose ordering $o$ that minimizes $\sum_{v_i, v_j \in CG} d_o(v_i, v_j)^2$

→ No strong theoretical guarantees [KH13] but compares well against other alternatives [BRKM91, CHP93, Mai09, MWBSV88, MIY90]

**Dynamic Variable Ordering:** variable re-ordering to minimize the size of the BDDs generated so far

- Finding the optimal BDD ordering is NP-hard [Bry86]
- But practical approximations (based on local-search) exist [Rud93].
- Applied in planning with good results by dynamic-Gamer [KH14]
**Complexity Results**

### BDD Representation of Interesting Sets of States [EK11]

<table>
<thead>
<tr>
<th>Goal States/Reachable states</th>
<th>Polynomial</th>
<th>Exponential</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polynomial</td>
<td>Gripper</td>
<td>Blocksworld</td>
</tr>
<tr>
<td></td>
<td></td>
<td>N-puzzle</td>
</tr>
<tr>
<td>Exponential</td>
<td>Connect-4</td>
<td>Tic-tac-toe</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Gomoku</td>
</tr>
</tbody>
</table>
### Complexity Results

**BDD Representation of Interesting Sets of States [EK11]**

<table>
<thead>
<tr>
<th>Goal States/Reachable states</th>
<th>Polynomial</th>
<th>Exponential</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polynomial</td>
<td>Gripper</td>
<td>Blocksworld</td>
</tr>
<tr>
<td>Exponential</td>
<td>Connect-4</td>
<td>N-puzzle</td>
</tr>
<tr>
<td></td>
<td>Tic-tac-toe</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Gomoku</td>
<td></td>
</tr>
</tbody>
</table>

Can variable orderings schemas based on the causal graph give us theoretical guarantees for the size of BDDs in the search? [KH14] → Mostly not.
Planning Actions as Logical Formulas

Transition Relation: represents an action $a$ as the relation (set of pairs of states) containing $(s, s')$ where $a$ is applicable in $s$ resulting in $s'$. 

$\langle p_1, l_1 \rangle$: pre: $\{\langle t, l_1 \rangle, \langle p_1, l_1 \rangle\}$ and eff: $\{\langle p_1, t \rangle\}$ (prevail: $\{\langle t, l_1 \rangle\}$)
Planning Actions as Logical Formulas

Transition Relation: represents an action \( a \) as the relation (set of pairs of states) containing \((s, s')\) where \( a \) is applicable in \( s \) resulting in \( s' \).

\[
\text{load}(p_1, l_1): \quad \text{pre} : \{\langle t, l_1 \rangle, \langle p_1, l_1 \rangle\} \quad \text{and} \quad \text{eff} : \{\langle p_1, t \rangle\} \quad (\text{prevail:} \quad \{\langle t, l_1 \rangle\})
\]

\[
l_1 - l_2 - l_3 \quad \text{and} \quad l_1 - l_2 - l_3 \quad \langle t, l_1 \rangle \land \langle p_1, l_1 \rangle \land \langle p_2, l_1 \rangle \land \langle t, l_1 \rangle' \land \langle p_1, t \rangle' \land \langle p_2, l_1 \rangle'
\]
Planning Actions as Logical Formulas

Transition Relation: represents an action \( a \) as the relation (set of pairs of states) containing \((s, s')\) where \( a \) is applicable in \( s \) resulting in \( s' \).

\[
\text{load}(p_1, l_1): \quad \text{pre} : \{\langle t, l_1 \rangle, \langle p_1, l_1 \rangle\} \quad \text{and} \quad \text{eff} : \{\langle p_1, t \rangle\} \quad (\text{prevail}: \{\langle t, l_1 \rangle\})
\]

\[
\begin{align*}
&l_1 - l_2 - l_3 &\langle t, l_1 \rangle \land \langle p_1, l_1 \rangle \land \langle p_2, l_1 \rangle \land \\
&l_1 - l_2 - l_3 &\langle t, l_1 \rangle' \land \langle p_1, t \rangle' \land \langle p_2, l_1 \rangle'
\end{align*}
\]
Planning Actions as Logical Formulas

Transition Relation: represents an action $a$ as the relation (set of pairs of states) containing $(s, s')$ where $a$ is applicable in $s$ resulting in $s'$.

```latex
load(p_1, l_1): \text{pre} : \{\langle t, l_1 \rangle, \langle p_1, l_1 \rangle\} \text{ and } \text{eff} : \{\langle p_1, t \rangle\} \text{ (prevail: } \{\langle t, l_1 \rangle\})
```

\[
\begin{align*}
l_1 - l_2 - l_3 & \quad \langle t, l_1 \rangle \land \langle p_1, l_1 \rangle \land \langle p_2, l_1 \rangle \land \langle t, l_1 \rangle' \land \langle p_1, t \rangle' \land \langle p_2, l_1 \rangle' \\
l_1 - l_2 - l_3 & \quad \langle t, l_1 \rangle \land \langle p_1, l_1 \rangle \land \langle p_2, l_2 \rangle \land \langle t, l_1 \rangle' \land \langle p_1, t \rangle' \land \langle p_2, l_2 \rangle' \\
l_1 - l_2 - l_3 & \quad \langle t, l_1 \rangle \land \langle p_1, l_1 \rangle \land \langle p_2, l_3 \rangle \land \langle t, l_1 \rangle' \land \langle p_1, t \rangle' \land \langle p_2, l_3 \rangle'
\end{align*}
\]

\[
\langle p_1, l_1 \rangle \land \langle t, l_1 \rangle \land \langle p_1, t \rangle' \land \langle t, l_1 \rangle' \land (\langle p_2, l_1 \rangle \leftrightarrow \langle p_2, l_1 \rangle') \land (\langle p_2, l_2 \rangle \leftrightarrow \langle p_2, l_2 \rangle') \ldots
\]
Computing the Successors (Image Computation)

Image: Given a set of states $S(x)$ and a TR $T(x, x')$ generate the successor states

\[
\text{image}(S(x), T(x, x')) = \exists x . S(x) \land T(x, x')[x' \leftrightarrow x]
\]
Computing the Successors (Image Computation)

Image: Given a set of states $S(x)$ and a TR $T(x, x')$ generate the successor states

\[
\text{image}(S(x), T(x, x')) = \exists x . S(x) \land T(x, x')[x' \leftrightarrow x]
\]

<table>
<thead>
<tr>
<th>$S(x)$:</th>
<th>$t$</th>
<th>$p_1$</th>
<th>$p_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_1$</td>
</tr>
<tr>
<td></td>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_2$</td>
</tr>
<tr>
<td></td>
<td>$l_2$</td>
<td>$l_1$</td>
<td>$l_3$</td>
</tr>
<tr>
<td></td>
<td>$l_1$</td>
<td>$l_3$</td>
<td>$l_1$</td>
</tr>
</tbody>
</table>

$T(x, x')$: (load($p_1, l_1$))

<table>
<thead>
<tr>
<th>$T(x, x')$:</th>
<th>$t$</th>
<th>$p_1$</th>
<th>$p_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_1$</td>
</tr>
<tr>
<td></td>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_2$</td>
</tr>
<tr>
<td></td>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_3$</td>
</tr>
<tr>
<td></td>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_3$</td>
</tr>
</tbody>
</table>

Result: $\exists$-quantification: exponential in the number of variables
Computing the Successors (Image Computation)

Image: Given a set of states $S(x)$ and a TR $T(x, x')$ generate the successor states

$$\text{image}(S(x), T(x, x')) = \exists x . S(x) \land T(x, x')[x' \leftrightarrow x]$$

**$S(x)$:**

<table>
<thead>
<tr>
<th></th>
<th>$t$</th>
<th>$p_1$</th>
<th>$p_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_1$</td>
</tr>
<tr>
<td></td>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_2$</td>
</tr>
<tr>
<td></td>
<td>$l_2$</td>
<td>$l_1$</td>
<td>$l_3$</td>
</tr>
<tr>
<td></td>
<td>$l_1$</td>
<td>$l_3$</td>
<td>$l_1$</td>
</tr>
</tbody>
</table>

**$T(x, x')$:**

- $(\text{load}(p_1, l_1))$

<table>
<thead>
<tr>
<th></th>
<th>$t$</th>
<th>$p_1$</th>
<th>$p_2$</th>
<th></th>
<th>$t'$</th>
<th>$p'_1$</th>
<th>$p'_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_1$</td>
<td></td>
<td>$l_1$</td>
<td>$t$</td>
<td>$l_1$</td>
</tr>
<tr>
<td></td>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_2$</td>
<td></td>
<td>$l_1$</td>
<td>$t$</td>
<td>$l_2$</td>
</tr>
<tr>
<td></td>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_3$</td>
<td></td>
<td>$l_1$</td>
<td>$t$</td>
<td>$l_3$</td>
</tr>
</tbody>
</table>

**Result:**

<table>
<thead>
<tr>
<th></th>
<th>$t$</th>
<th>$p_1$</th>
<th>$p_2$</th>
<th></th>
<th>$t'$</th>
<th>$p'_1$</th>
<th>$p'_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_1$</td>
<td></td>
<td>$l_1$</td>
<td>$t$</td>
<td>$l_1$</td>
</tr>
<tr>
<td></td>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_2$</td>
<td></td>
<td>$l_1$</td>
<td>$t$</td>
<td>$l_2$</td>
</tr>
</tbody>
</table>
Computing the Successors (Image Computation)

Image: Given a set of states $S(x)$ and a TR $T(x, x')$ generate the successor states

\[
\text{image}(S(x), T(x, x')) = \exists x . \ S(x) \land T(x, x') [x' \leftrightarrow x]
\]

$S(x) :$

<table>
<thead>
<tr>
<th>$t$</th>
<th>$p_1$</th>
<th>$p_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_1$</td>
</tr>
<tr>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_2$</td>
</tr>
<tr>
<td>$l_2$</td>
<td>$l_1$</td>
<td>$l_3$</td>
</tr>
<tr>
<td>$l_1$</td>
<td>$l_3$</td>
<td>$l_1$</td>
</tr>
</tbody>
</table>

$T(x, x') :$

<table>
<thead>
<tr>
<th>$t$</th>
<th>$p_1$</th>
<th>$p_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_1$</td>
</tr>
<tr>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_2$</td>
</tr>
<tr>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_3$</td>
</tr>
</tbody>
</table>

(load($p_1, l_1$))

Result:

<table>
<thead>
<tr>
<th>$t$</th>
<th>$p_1$</th>
<th>$p_2$</th>
<th>$t'$</th>
<th>$p'_1$</th>
<th>$p'_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_1$</td>
<td>$t$</td>
<td>$l_1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$l_1$</td>
<td>$t$</td>
<td>$l_2$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Computing the Successors (Image Computation)

Image: Given a set of states $S(x)$ and a TR $T(x, x')$ generate the successor states

\[
\text{image}(S(x), T(x, x')) = \exists x . S(x) \land T(x, x')[x' \leftrightarrow x]
\]

<table>
<thead>
<tr>
<th>$t$</th>
<th>$p_1$</th>
<th>$p_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_1$</td>
</tr>
<tr>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_2$</td>
</tr>
<tr>
<td>$l_2$</td>
<td>$l_1$</td>
<td>$l_3$</td>
</tr>
<tr>
<td>$l_1$</td>
<td>$l_3$</td>
<td>$l_1$</td>
</tr>
</tbody>
</table>

$S(x)$:

<table>
<thead>
<tr>
<th>$t$</th>
<th>$p_1$</th>
<th>$p_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_1$</td>
</tr>
<tr>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_2$</td>
</tr>
<tr>
<td>$l_2$</td>
<td>$l_1$</td>
<td>$l_3$</td>
</tr>
<tr>
<td>$l_1$</td>
<td>$l_3$</td>
<td>$l_1$</td>
</tr>
</tbody>
</table>

$T(x, x')$:

\[
T(x, x') : (\text{load}(p_1, l_1))
\]

<table>
<thead>
<tr>
<th>$t$</th>
<th>$p_1$</th>
<th>$p_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_1$</td>
</tr>
<tr>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_2$</td>
</tr>
<tr>
<td>$l_1$</td>
<td>$l_1$</td>
<td>$l_3$</td>
</tr>
<tr>
<td>$l_1$</td>
<td>$l_3$</td>
<td>$l_1$</td>
</tr>
</tbody>
</table>

Result:

<table>
<thead>
<tr>
<th>$t$</th>
<th>$p_1$</th>
<th>$p_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_1$</td>
<td>$t$</td>
<td>$l_1$</td>
</tr>
<tr>
<td>$l_1$</td>
<td>$t$</td>
<td>$l_2$</td>
</tr>
</tbody>
</table>
Computing the Successors (Image Computation)

**Image:** Given a set of states $S(x)$ and a TR $T(x, x')$ generate the successor states

<table>
<thead>
<tr>
<th>$S(x)$</th>
<th>$T(x, x')$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$ $p_1$ $p_2$</td>
<td>$T(x, x')$: $(load(p_1, l_1))$</td>
</tr>
<tr>
<td>$l_1$ $l_1$ $l_1$</td>
<td>$l_1$ $l_1$ $l_1$</td>
</tr>
<tr>
<td>$l_1$ $l_1$ $l_2$</td>
<td>$l_1$ $t$ $l_1$</td>
</tr>
<tr>
<td>$l_2$ $l_1$ $l_3$</td>
<td>$l_1$ $l_1$ $l_2$</td>
</tr>
<tr>
<td>$l_1$ $l_3$ $l_1$</td>
<td>$l_1$ $t$ $l_2$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Result:</th>
<th>$\exists$-quantification: exponential in the number of variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t$ $p_1$ $p_2$</td>
<td>$t'$ $p'_1$ $p'_2$</td>
</tr>
<tr>
<td>$l_1$ $t$ $l_1$</td>
<td>$l_1$ $t$ $l_3$</td>
</tr>
<tr>
<td>$l_1$ $t$ $l_2$</td>
<td>$l_1$ $t$ $l_3$</td>
</tr>
</tbody>
</table>
Computing the Predecessors (Pre-Image Computation)

Image: Given a set of states $S(x)$ and a TR $T(x, x')$ generate the predecessor states

\[
\text{pre-image}(S(x), T(x, x')) = \exists x' . S(x)[x' \leftrightarrow x] \land T(x, x')
\]

- Corresponds to regression [Rin08]
Efficient Image Computation: Variable Ordering

Variable Ordering: Interleave variables $x$ and $x'$

→ The TR of an action has linear size on the number of variables

Á. Torralba, D. Gnad

Symbolic State Space Search
Efficient Image Computation

Transition Relation Partitioning [BCL91, JVB08]

Given a set of $K$ actions with the same cost, replace $T_i(x, x')$ and $T_j(x, x')$ by $T_i(x, x') \lor T_j(x, x')$
Efficient Image Computation

Transition Relation Partitioning [BCL91, JVB08]

Given a set of $K$ actions with the same cost, replace $T_i(x, x')$ and $T_j(x, x')$ by $T_i(x, x') \lor T_j(x, x')$
Transition Relation Partitioning [BCL91, JVB08]

Given a set of $K$ actions with the same cost, replace $T_i(x, x')$ and $T_j(x, x')$ by $T_i(x, x') \lor T_j(x, x')$

→ Merge as many TRs as possible given the available memory [TEK13, TAKE17]
Uses of Decision Diagrams in Classical Planning

- Representation of state-dependent action costs [GKM15]
- Subsumption of partial states [AFB14]
- Dominance pruning [TH15]
Uses of Decision Diagrams in Classical Planning

- Representation of state-dependent action costs [GKM15]
- Subsumption of partial states [AFB14]
- Dominance pruning [TH15]

→ Here: symbolic search
Agenda

1 About this Tutorial
2 Classical Planning: Models, Approaches
3 Symbolic Representation of Planning Tasks
4 Symbolic Blind Search
5 Heuristic Search
6 Symbolic Abstraction Heuristics
7 Implementation
8 Conclusions and Open Challenges
Symbolic Breadth-First Search

**Input:** Planning Task $\Pi = (V, A, I, G)$

$S_0 \leftarrow I$ ;

$C \leftarrow \emptyset$ ;

$i \leftarrow 0$ ;

while $S_i \neq \emptyset$ do

  if $S_i \land G$ then

    return Plan ;

  end

  $C \leftarrow C \lor S_i$ ;

  $S_{i+1} \leftarrow \text{image}(S_i, TR) \land \neg C$ ;

  $i \leftarrow i + 1$ ;

end

return Unsolvable ;

---

Á. Torralba, D. Gnad

Symbolic State Space Search
Symbolic Uniform-Cost Search

Expand set of states $S_i$ with minimum $g$-value $i$

- Zero-cost breadth-first search to obtain all states reachable with $g = i$
- For each TR with action cost $c$:
  - Use image to compute states reachable with $i + c$
  - Insert the result in the corresponding bucket (disjunction)
Symbolic Uniform-Cost Search

Expand set of states $S_i$ with minimum $g$-value $i$

- Zero-cost breadth-first search to obtain all states reachable with $g = i$
- For each TR with action cost $c$:
  - Use image to compute states reachable with $i + c$
  - Insert the result in the corresponding bucket (disjunction)
Symbolic Uniform-Cost Search

Expand set of states $S_i$ with minimum $g$-value $i$

- Zero-cost breadth-first search to obtain all states reachable with $g = i$
- For each TR with action cost $c$:
  - Use image to compute states reachable with $i + c$
  - Insert the result in the corresponding bucket (disjunction)
Symbolic Uniform-Cost Search

Expand set of states $S_i$ with minimum $g$-value $i$

- Zero-cost breadth-first search to obtain all states reachable with $g = i$
- For each TR with action cost $c$:
  - Use image to compute states reachable with $i + c$
  - Insert the result in the corresponding bucket (disjunction)
Symbolic Uniform-Cost Search

Expand set of states $S_i$ with minimum $g$-value $i$

- Zero-cost breadth-first search to obtain all states reachable with $g = i$
- For each TR with action cost $c$:
  - Use image to compute states reachable with $i + c$
  - Insert the result in the corresponding bucket (disjunction)
Symbolic Backward Uniform-Cost Search

We can perform the search in backward direction:

- Start with the set of goal states
- Use pre-image instead of image operation

Challenges:

1. Multiple goal states
2. Subsumption of partial states
3. Spurious states
Symbolic Backward Uniform-Cost Search

We can perform the search in backward direction:
- Start with the set of goal states
- Use pre-image instead of image operation

Challenges:
1. Multiple goal states → Not a problem in symbolic search!
2. Subsumption of partial states → Not a problem in symbolic search!
3. Spurious states
Symbolic Backward Uniform-Cost Search

We can perform the search in backward direction:

- Start with the set of goal states
- Use pre-image instead of image operation

Challenges:

1. Multiple goal states → Not a problem in symbolic search!
2. Subsumption of partial states → Not a problem in symbolic search!
   - Compute state invariants, e.g., $h^2$ mutexes
   - Encode the set of spurious states as a BDD
   - Remove spurious states from the goal and the TRs
Symbolic Uniform-Cost Search: Results

[Graphs showing symbolic and explicit forward and backward runtimes]

Forward

Backward
Symbolic Bidirectional Uniform-Cost Search

- Do forward and backward search at the same time
- Decide forward or backward direction at each step

\[ g_f = 0 \quad \text{and} \quad g_b = 0 \]
Symbolic Bidirectional Uniform-Cost Search

- Do forward and backward search at the same time
- Decide forward or backward direction at each step

\[
g_f = 0 \quad g_b = 0
\]
Symbolic Bidirectional Uniform-Cost Search

- Do forward and backward search at the same time
- Decide forward or backward direction at each step
- Stop when $g_f + g_b + \min_{a \in A} c(a) \geq Sol$

$$g_f = 1 \quad \quad \quad \quad \quad \quad g_b = 0$$
Symbolic Bidirectional Uniform-Cost Search

- Do forward and backward search at the same time
- Decide forward or backward direction at each step
- Stop when $g_f + g_b + \min_{a \in A} c(a) \geq Sol$

$g_f = 1$  
$g_b = 1$  
$Sol = 10$
Symbolic Bidirectional Uniform-Cost Search

- Do forward and backward search at the same time
- Decide forward or backward direction at each step
- Stop when $g_f + g_b + \min_{a \in A} c(a) \geq Sol$

$g_f = 2 \quad \text{Sol} = 10 \quad g_b = 1$
Symbolic Bidirectional Uniform-Cost Search: Results

vs Symbolic Forward

vs A* with LM-cut
Agenda

1. About this Tutorial
2. Classical Planning: Models, Approaches
3. Symbolic Representation of Planning Tasks
4. Symbolic Blind Search
5. Heuristic Search
6. Symbolic Abstraction Heuristics
7. Implementation
8. Conclusions and Open Challenges
Planning Heuristics

**Definition** A *heuristic* $h$ is a function $h : S \rightarrow \mathbb{R}_0^+ \cup \{\infty\}$. Its value $h(s)$ for a state $s$ is referred to as the state’s *heuristic value*, or *$h$ value*. 
Planning Heuristics

**Definition** A *heuristic* $h$ is a function $h : S \mapsto \mathbb{R}_0^+ \cup \{\infty\}$. Its value $h(s)$ for a state $s$ is referred to as the state’s *heuristic value*, or $h$ *value*.

**Definition** For a state $s \in S$, the *perfect heuristic value* $h^*$ of $s$ is the cost of an optimal plan for $s$, or $\infty$ if there exists no plan for $s$.

→ Heuristic functions $h$ estimate the remaining cost $h^*$. 
How to Exploit Heuristics in Symbolic Search?

Split a BDD into subsets of states according to their $h$-value!

Heuristic computation: how to evaluate a set of states?

Does the heuristic improve the search performance?
How to Exploit Heuristics in Symbolic Search?

Split a BDD into subsets of states according to their \( h \)-value!
How to Exploit Heuristics in Symbolic Search?

Split a BDD into subsets of states according to their $h$-value!
How to Exploit Heuristics in Symbolic Search?

Split a BDD into subsets of states according to their $h$-value!
How to Exploit Heuristics in Symbolic Search?

Split a BDD into subsets of states according to their $h$-value!
How to Exploit Heuristics in Symbolic Search?

Split a BDD into subsets of states according to their $h$-value!
How to Exploit Heuristics in Symbolic Search?

Split a BDD into subsets of states according to their $h$-value!
How to Exploit Heuristics in Symbolic Search?

Split a BDD into subsets of states according to their $h$-value!

- Heuristic computation: how to evaluate a set of states?
- Does the heuristic improve the search performance?
Heuristic Computation: How to Evaluate a Set of States?

1. Iterate over all states in the BDD, computing $h(s)$
Heuristic Computation: How to Evaluate a Set of States?

1. Iterate over all states in the BDD, computing \( h(s) \)
2. Precompute the heuristic in form of BDDs: A BDD \( H_i \) for each possible \( h \)-value representing the set of states with \( h(s) = i \)

\[
H_0 \quad H_1 \quad H_2 \quad H_3 \quad H_4
\]
Heuristic Computation: How to Evaluate a Set of States?

1. Iterate over all states in the BDD, computing $h(s)$

2. Precompute the heuristic in form of BDDs: A BDD $H_i$ for each possible $h$-value representing the set of states with $h(s) = i$

Given a set of states $S$, split it according to their $h$-value

$$S = S_0 \lor S_1 \lor S_2 \lor S_3 \lor S_4$$
Heuristic Computation: How to Evaluate a Set of States?

1. Iterate over all states in the BDD, computing $h(s)$

2. Precompute the heuristic in form of BDDs: A BDD $H_i$ for each possible $h$-value representing the set of states with $h(s) = i$

Given a set of states $S$, split it according to their $h$-value: $S_i = S \land B_i$
# Heuristic Computation: How to Evaluate a Set of States?

1. Iterate over all states in the BDD, computing $h(s)$
2. Precompute the heuristic in form of BDDs: A BDD $H_i$ for each possible $h$-value representing the set of states with $h(s) = i$

Given a set of states $S$, split it according to their $h$-value: $S_i = S \land B_i$

<table>
<thead>
<tr>
<th></th>
<th>$H_0$</th>
<th>$H_1$</th>
<th>$H_2$</th>
<th>$H_3$</th>
<th>$H_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S$</td>
<td>$S_0$</td>
<td>$S_1$</td>
<td>$S_2$</td>
<td>$S_3$</td>
<td>$S_4$</td>
</tr>
</tbody>
</table>

→ Can we efficiently precompute a heuristic into BDDs?
   - Yes, for some types of abstraction heuristics
   - Not in the general case. Finding tractable cases is an open research question!
Agenda

1. About this Tutorial
2. Classical Planning: Models, Approaches
3. Symbolic Representation of Planning Tasks
4. Symbolic Blind Search
5. Heuristic Search
6. Symbolic Abstraction Heuristics
7. Implementation
8. Conclusions and Open Challenges
**Abstraction**: function $\alpha : S \mapsto S^\alpha$. Induces an abstract state space s.t.: 

(i) $I^\alpha = \alpha(I)$.

(ii) $S^{\alpha G} = \{\alpha(s) \mid s \in S^G\}$. /* preserve goal states */

(iii) $T^\alpha = \{(\alpha(s), l, \alpha(t)) \mid (s, l, t) \in T\}$. /* preserve transitions */
**Abstraction:** function \( \alpha : S \mapsto S^\alpha \). Induces an abstract state space s.t.:

(i) \( I^\alpha = \alpha(I) \).

(ii) \( S^\alpha_G = \{ \alpha(s) \mid s \in S^G \} \). /* preserve goal states */

(iii) \( T^\alpha = \{ (\alpha(s), l, \alpha(t)) \mid (s, l, t) \in T \} \)./* preserve transitions */
Pattern Databases: Select a subset of variables $V^\alpha \subseteq V$ (pattern). The mapping $\alpha$ is defined as the projection onto $V^\alpha$. 
Use the **optimal goal-distance in the abstract state space** as an (admissible) estimate for the distance in the concrete state space:

\[ h(s) = h^*(\alpha(s)) \]
Use the **optimal goal-distance in the abstract state space** as an (admissible) estimate for the distance in the concrete state space:

\[ h(s) = h^*(\alpha(s)) \]

1. Precompute \( h^* \) for all \( \alpha(s) \in S^\alpha \) by performing a backward uniform-cost search in the abstract state space

2. Store them in a look-up table
Abstraction Heuristics

Use the **optimal goal-distance in the abstract state space** as an (admissible) estimate for the distance in the concrete state space:

\[ h(s) = h^*(\alpha(s)) \]

1. Precompute \( h^* \) for all \( \alpha(s) \in S^\alpha \) by performing a **backward uniform-cost search** in the abstract state space
   → Searching the entire abstract state space? That’s what **symbolic search** is good for!

2. Store them in a **look-up table**
Abstraction Heuristics

Use the **optimal goal-distance in the abstract state space** as an (admissible) estimate for the distance in the concrete state space:

\[
h(s) = h^*(\alpha(s))
\]

1. **Precompute** \( h^* \) for all \( \alpha(s) \in S^\alpha \) by performing a **backward uniform-cost search** in the abstract state space
   - Searching the entire abstract state space? That’s what **symbolic search** is good for!

2. **Store them in a look-up table**
   - In the form of **BDDs**, so we can use them in BDDA*
Symbolic Pattern Databases

Do symbolic backward uniform-cost search with only a subset of variables
Symbolic Pattern Databases

Do symbolic backward uniform-cost search with only a subset of variables

- Do not have a limit on the number of variables to consider
- Truncate the search if it takes too much time or memory [AHS07]
- Using all variables: Symbolic Perimeter
Symbolic Pattern Databases

Do symbolic backward uniform-cost search with only a subset of variables

- Do not have a limit on the number of variables to consider
- Truncate the search if it takes too much time or memory [AHS07]
- Using all variables: Symbolic Perimeter

→ Really strong for heuristic search planners too [FTLB17]!
Abstraction Heuristics: Merge-and-Shrink [HHHN14, SWH14]
Abstraction Heuristics: Merge-and-Shrink [HHHN14, SWH14]
Abstraction Heuristics:
Merge-and-Shrink [HHHN14, SWH14]
Abstraction Heuristics: Merge-and-Shrink [HHHN14, SWH14]

- Shrink strategy: What abstraction to apply to reduce the abstract state space size?
- Merge strategy: What two abstractions to merge next?
Abstraction Heuristics: Merge-and-Shrink [HHHN14, SWH14]

- Shrink strategy: What abstraction to apply to reduce the abstract state space size?
- Merge strategy: What two abstractions to merge next?

\[ t, p_1, p_2, p_3, p_4 \]

\[ \alpha_1, \alpha_2 \]
Abstraction Heuristics: Merge-and-Shrink [HHHN14, SWH14]

- Shrink strategy: What abstraction to apply to reduce the abstract state space size?
- Merge strategy: What two abstractions to merge next?

\[ t \]
\[ p_1 \]
\[ p_2 \]
\[ p_3 \]
\[ p_4 \]

\[ \alpha_1 \]
\[ \alpha_2 \]
\[ \alpha_3 \]
Abstraction Heuristics: Merge-and-Shrink [HHHN14, SWH14]

- Shrink strategy: What abstraction to apply to reduce the abstract state space size?
- Merge strategy: What two abstractions to merge next?

$t$ $p_1$ $p_2$ $p_3$ $p_4$

\[ \begin{align*}
\alpha_1 & \rightarrow \alpha_2 & \rightarrow \alpha_3 \\
\end{align*} \]
Abstraction Heuristics: Merge-and-Shrink [HHHN14, SWH14]

- Shrink strategy: What abstraction to apply to reduce the abstract state space size?
- Merge strategy: What two abstractions to merge next?

![Diagram of abstraction heuristics]

- $t$
- $p_1$
- $p_2$
- $p_3$
- $p_4$

- $\alpha_1$
- $\alpha_2$
- $\alpha_3$
- $\alpha_4$
**Abstraction Heuristics:**

**Merge-and-Shrink [HHHN14, SWH14]**

- **Shrink strategy:** What abstraction to apply to reduce the abstract state space size?
- **Merge strategy:** What two abstractions to merge next?

```
t   p1  p2  p3  p4
   \alpha_1
   \alpha_2
   \alpha_3
   \alpha_4
```

```latex
\alpha_1, \alpha_2, \alpha_3, \alpha_4
```
Abstraction Heuristics: Merge-and-Shrink [HHHN14, SWH14]

- Shrink strategy: What abstraction to apply to reduce the abstract state space size?
- Merge strategy: What two abstractions to merge next?

Linear Merge

Non-linear merge
Are $M&S$ heuristics efficiently representable by BDDs?
Symbolic Merge-and-Shrink

Are $M&S$ heuristics efficiently representable by BDDs?

- Yes, if a linear merge strategy is used following the BDD variable ordering [EKT12, Tor15]
Are $M&S$ heuristics efficiently representable by BDDs?

- Yes, if a linear merge strategy is used following the BDD variable ordering [EKT12, Tor15]
- No, if a non-linear merge strategy is used (though the exponential overhead can be bounded on the non-linearity of the merge tree) [HRS15]
Symbolic Merge-and-Shrink

Are $M&S$ heuristics efficiently representable by BDDs?

- Yes, if a linear merge strategy is used following the BDD variable ordering [EKT12, Tor15]
- No, if a non-linear merge strategy is used (though the exponential overhead can be bounded on the non-linearity of the merge tree) [HRS15]

→ We can use M&S heuristics in symbolic search under some restrictions
We have good symbolic abstraction heuristics

But, bidirectional blind search is often the best symbolic search algorithm

Can we use heuristics in symbolic bidirectional search?
We have good symbolic abstraction heuristics

But, bidirectional blind search is often the best symbolic search algorithm

Can we use heuristics in symbolic bidirectional search? Yes! [TGDH16]

1. Start symbolic bidirectional uniform-cost search
   - If it succeeds → done!

2. Detect when it is going to fail and activate heuristics
   - Perimeters abstractions take advantage of the search already performed in the concrete state space [TLB13]
**Agenda**

1. About this Tutorial
2. Classical Planning: Models, Approaches
3. Symbolic Representation of Planning Tasks
4. Symbolic Blind Search
5. Heuristic Search
6. Symbolic Abstraction Heuristics
7. Implementation
8. Conclusions and Open Challenges
Planners using Symbolic Search

- **MIPS**: Stefan Edelkamp and Malte Helmert
  http://www.tzi.de/~edelkamp/mips/mips-bdd.html

- **MIPS-XXL**: Stefan Edelkamp, Shahid Jabbar, and Mohammed Nazih. Extension to net-benefit with external planning
  http://sjabbar.com/mips-xxl-planner

- **BDDPlan**: Hans-Peter Strr http://www.stoerr.net/bddplan.html

- **Gamer** (IPC08-IPC11): Peter Kissmann and Stefan Edelkamp
  https://fai.cs.uni-saarland.de/kissmann/planning/downloads/

  Extensions (IPC14):
  1. cGamer: improved image computation and state invariant pruning
  2. dynamic Gamer: dynamic variable reordering

- **SymBA** (IPC14): Based on Fast Downward
  http://fai.cs.uni-saarland.de/torralba/software.html
BDD Packages

<table>
<thead>
<tr>
<th>Library</th>
<th>Language</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDD</td>
<td>C/C++</td>
<td>[Som]</td>
</tr>
<tr>
<td>CacBDD</td>
<td>C++</td>
<td>[LSX13]</td>
</tr>
<tr>
<td>BuDDy</td>
<td>C</td>
<td>[CWWG]</td>
</tr>
<tr>
<td>CAL</td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>Sylvan</td>
<td>C</td>
<td>[vDvdP15]</td>
</tr>
<tr>
<td>JDD</td>
<td>Java</td>
<td>[Vah]</td>
</tr>
<tr>
<td>BeeDeeDee</td>
<td>Java</td>
<td>[LMS14]</td>
</tr>
</tbody>
</table>

→ Not clear best performer. CUDD, BuDDy, CacBDD have good results in symbolic model-checking [vDHJ⁺15].

→ There are interfaces that adapt these libraries for other languages like Java, Python, Haskell, ...
Managing BDDs in CUDD in C++

- Supports:
  - BDDs
  - ZDDs
  - ADDs

- Really easy to perform operations:
  - Disjunction (A + B)
  - Conjunction (A * B).

- Possible to set time and memory limits for BDD operations
  → Critical to avoid failure due to an exponential-time BDD operation

- Integrated many other potentially useful algorithms
  - BDD minimization
  - Variable re-ordering
  - ...
Symbolic Search in Fast Downward

- **SymVariables**: BDD representation of FD variables
  - Obtain the BDD that represents a (partial-)state
  - Check if a BDD contains a given state
- **SymStateSpaceManager**: Search-related structures to a state space
  - Retrieve initial state/goal states
  - Transition relation and image computation
  - State invariants
- **Search algorithms**: breadth-first, uniform-cost, A*, bidirectional, ...

http://fai.cs.uni-saarland.de/torralba/software.html
Agenda

1 About this Tutorial
2 Classical Planning: Models, Approaches
3 Symbolic Representation of Planning Tasks
4 Symbolic Blind Search
5 Heuristic Search
6 Symbolic Abstraction Heuristics
7 Implementation
8 Conclusions and Open Challenges
Symbolic search is useful for classical planning

→ Takes advantage of the structure of the planning task **implicitly**
Symbolic search is useful for classical planning

→ Takes advantage of the structure of the planning task implicitly

Advantages:

- Compact representation of sets of states
- Time/memory efficient exploration of state spaces
Conclusions

- Symbolic search is useful for classical planning
  - Takes advantage of the structure of the planning task implicitly
- Advantages:
  - Compact representation of sets of states
  - Time/memory efficient exploration of state spaces
- Disadvantages:
  - Heuristics/pruning methods must be adapted to leverage the symbolic representation
**Explicit vs. Symbolic Search in Cost-Optimal Planning**

### Algorithms

| A* | Forward |
| Uniform-Cost | Bidirectional |

### Heuristics

- **Delete-relaxation:** $h^{max}$, $h^{+}$
- **Landmarks:** $h^{LA}$, LM-cut
- **Abstractions:** PDBs, M&S, CEGAR
- **Critical paths:** $h^{m}$
- **Flow**
- **Potentials**

### Pruning techniques

- State invariants
- Symmetries
- Partial-order pruning
- Dominance pruning

---

**Critical paths:** $h^{m}$

**Potentials:**

- Disjoint sum
- Cost-partitioning

**Heuristics**

<table>
<thead>
<tr>
<th>Delete-relaxation: $h^{max}$, $h^{+}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Landmarks: $h^{LA}$, LM-cut</td>
</tr>
<tr>
<td>Abstractions: PDBs, M&amp;S, CEGAR</td>
</tr>
<tr>
<td>Critical paths: $h^{m}$</td>
</tr>
<tr>
<td>Flow</td>
</tr>
<tr>
<td>Potentials</td>
</tr>
</tbody>
</table>

**Pruning techniques**

- State invariants
- Symmetries
- Partial-order pruning
- Dominance pruning

**Algorithms**

| A* | Forward |
| Uniform-Cost | Bidirectional |

**Heuristics**

- Delete-relaxation: $h^{max}$, $h^{+}$
- Landmarks: $h^{LA}$, LM-cut
- Abstractions: PDBs, M&S, CEGAR
- Critical paths: $h^{m}$
- Flow
- Potentials

**Pruning techniques**

- State invariants
- Symmetries
- Partial-order pruning
- Dominance pruning

---

**Symbolic State Space Search**
Explicit vs. Symbolic Search in Cost-Optimal Planning

### Algorithms

<table>
<thead>
<tr>
<th>A*</th>
<th>Forward</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uniform-Cost</td>
<td>Bidirectional</td>
</tr>
</tbody>
</table>

### Heuristics

- **Delete-relaxation:** \(h^{\text{max}}, h^+\)
- **Landmarks:** \(h^{LA}\), LM-cut
- **Abstractions:** PDBs, M&S, CEGAR
- **Critical paths:** \(h^m\)
- **Flow**
- **Potentials**

### Pruning techniques

- **State invariants**
- **Symmetries**
- **Partial-order pruning**
- **Dominance pruning**
Explicit vs. Symbolic Search in Cost-Optimal Planning

### Algorithms
- $A^*$
- Uniform-Cost
- Forward
- Bidirectional

### Heuristics
- Delete-relaxation: $h_{\text{max}}$, $h^+$
- Landmarks: $h^{LA}$, LM-cut
- Abstractions: PDBs, M&S, CEGAR
- Critical paths: $h^m$
- Flow
- Potentials

### Pruning techniques
- State invariants
- Symmetries
- Partial-order pruning
- Dominance pruning
And What can YOU do?

Still lots of open questions:

- Are there any alternatives to BDDs?
- Can BDDs be split for a more efficient exploration of the state space?
- Keep optimizing operations
  - How to do efficient image operations [TEK13, TAKE17]?
  - How to choose good BDD variable orderings [KE11, KH13]?

BDDs can be useful for you to represent and operate with sets of states even if you are not planning to use symbolic search!

- Partial-state regression search: Keep set of all states expanded so far [AFB14]
- Dominance Pruning: Keep set of all states dominated by any expanded state [TH15]


