Classical Planning

Part 1: Foundations and Approaches

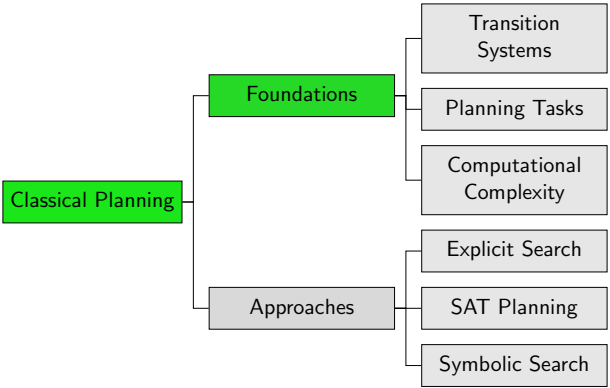
Gabriele Röger

Universität Basel

ICAPS 26 Summer School
June 22, 2026

Thanks to Malte Helmert, David Speck, and Silvan Sievers for sharing their slides.

Content


Classical Planning
Foundations
Transition Systems
Planning Tasks
Computational Complexity
Approaches
Explicit Search
SAT Planning
Symbolic Search

Planning

Planning (pithy definition)

"Planning is the art and practice of thinking before acting."

— Patrik Haslum

Planning

Planning (pithy definition)

"Planning is the art and practice of thinking before acting."

— Patrik Haslum

Planning (more technical definition)

"Selecting a goal-leading course of action
based on a high-level description of the world."

— Jörg Hoffmann

Domain-independent Planning

Image credits: Freecell: GNOME Project (GNU General Public License), Smart greenhouse: photo © LemnaTec GmbH,
Termes robots: Eliza Grinnell, SEAS Communications

Many Flavors of Planning (1)

What happens when we act?

- deterministic planning
- nondeterministic planning
- probabilistic planning

. . . and others (e.g., adversarial and multi-agent)

Many Flavors of Planning (2)

Modelling language complexity:

- classical: finite-domain state variables
- numerical: + real-valued state variables
- temporal: + concurrent and overlapping actions

. . . and many fine-grained modelling features, normal forms, etc.

Many Flavors of Planning (3)

Solution quality requirements:

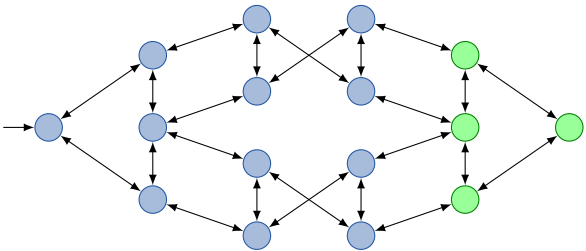
- optimal planning: only best possible solutions will do
- satisficing planning: optimality not mandatory;
  better quality preferred

. . . and many further variations (bounded suboptimal, cost-bounded, anytime, oversubscription, net benefit. . .)

Classical Planning as Reachability in Transition Systems

classical planning:

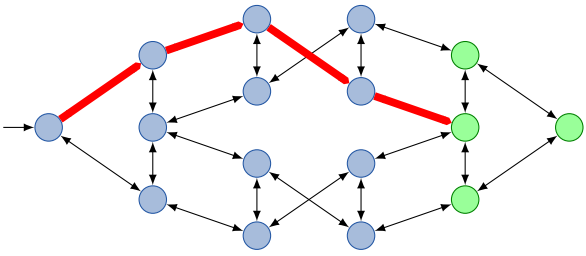
can be seen as finding paths in implicitly defined digraphs



Classical Planning as Reachability in Transition Systems

classical planning:

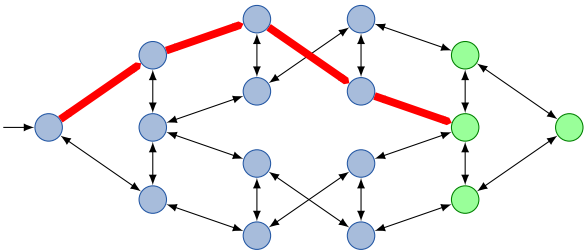
can be seen as finding paths in implicitly defined digraphs



Classical Planning as Reachability in Transition Systems

classical planning:

can be seen as finding paths in large implicitly defined digraphs



Example problem sizes:

- elevator control: $6.92 \cdot 10^{19}$ reachable states
- greenhouse automation: $1.68 \cdot 10^{21}$ reachable states
- transportation logistics: $6.31 \cdot 10^{218}$ reachable states

Classical Planning Tasks

- state variables:
  $x \in \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$, $y \in \{\mathsf{a}, \mathsf{b}\}$, $z \in \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$

Classical Planning Tasks

- state variables:
  $x \in \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$, $y \in \{\mathsf{a}, \mathsf{b}\}$, $z \in \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$
- initial state:
  $\{x \mapsto \mathsf{a}, y \mapsto \mathsf{a}, z \mapsto \mathsf{a}\}$

Classical Planning Tasks

- state variables:
  $x \in \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$, $y \in \{\mathsf{a}, \mathsf{b}\}$, $z \in \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$
- initial state:
  $\{x \mapsto \mathsf{a}, y \mapsto \mathsf{a}, z \mapsto \mathsf{a}\}$
- goal:
  $\{x \mapsto \mathsf{c}, z \mapsto \mathsf{b}\}$

Classical Planning Tasks

- state variables:
  $x \in \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$, $y \in \{\mathsf{a}, \mathsf{b}\}$, $z \in \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$
- initial state:
  $\{x \mapsto \mathsf{a}, y \mapsto \mathsf{a}, z \mapsto \mathsf{a}\}$
- goal:
  $\{x \mapsto \mathsf{c}, z \mapsto \mathsf{b}\}$
- actions a.k.a. operators:
  $a_1 : \quad x \mapsto \mathsf{a},\ y \mapsto \mathsf{a} \quad \xrightarrow{4} \quad y := \mathsf{b},\ z := \mathsf{c}$
  $a_2 : \quad x \mapsto \mathsf{a},\ z \mapsto \mathsf{b} \quad \xrightarrow{3} \quad z := \mathsf{b}$
  $\ldots$

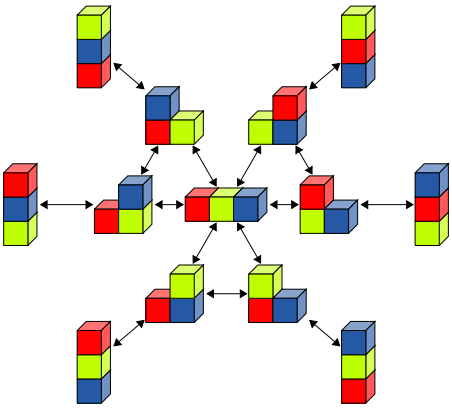
Classical Planning Tasks

- state variables:
  $x \in \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$, $y \in \{\mathsf{a}, \mathsf{b}\}$, $z \in \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$
- initial state:
  $\{x \mapsto \mathsf{a}, y \mapsto \mathsf{a}, z \mapsto \mathsf{a}\}$
- goal:
  $\{x \mapsto \mathsf{c}, z \mapsto \mathsf{b}\}$
- actions a.k.a. operators:
  $a_1 : \quad x \mapsto \mathsf{a},\ y \mapsto \mathsf{a} \quad \xrightarrow{4} \quad y := \mathsf{b},\ z := \mathsf{c}$
  $a_2 : \quad x \mapsto \mathsf{a},\ z \mapsto \mathsf{b} \quad \xrightarrow{3} \quad z := \mathsf{b}$
  $\ldots$

Problem:

- find sequence of actions transforming initial state to state consistent with the goal
- objective function: sum of action costs

Blocks World Transition System for Three Blocks



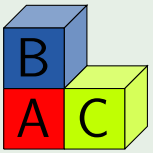
Labels omitted for clarity. Initial/goal states not marked.

Blocks World State with Finite-Domain Variables

Example

- *below-A*: $\{\mathsf{B}, \mathsf{C}, \mathsf{table}\}$
- *below-B*: $\{\mathsf{A}, \mathsf{C}, \mathsf{table}\}$
- *below-C*: $\{\mathsf{A}, \mathsf{B}, \mathsf{table}\}$
- *clear-A*: $\{\mathbf{T}, \mathbf{F}\}$
- *clear-B*: $\{\mathbf{T}, \mathbf{F}\}$
- *clear-C*: $\{\mathbf{T}, \mathbf{F}\}$

$s(\textit{below-A}) \mapsto \mathsf{table}$
$s(\textit{below-B}) \mapsto \mathsf{A}$
$s(\textit{below-C}) \mapsto \mathsf{table}$
$s(\textit{clear-A}) \mapsto \mathbf{F}$
$s(\textit{clear-B}) \mapsto \mathbf{T}$
$s(\textit{clear-C}) \mapsto \mathbf{T}$



Operator: move-A-from-B-to-C

Action: `move-A-from-B-to-C`

Precondition

$$\textit{below-A} \mapsto \mathrm{B}, \textit{clear-A} \mapsto \mathbf{T}, \textit{clear-C} \mapsto \mathbf{T}$$

Effect

$$\textit{below-A} := \mathrm{C}, \textit{clear-B} := \mathbf{T}, \textit{clear-C} := \mathbf{F}$$

Formalism Variants for Classical Planning Tasks (1)

Finite-domain representation a.k.a. SAS$^+$

What we just saw.

Derives from SAS = Simplified Action Structures (Bäckström et al., 1991)

Formalism Variants for Classical Planning Tasks (1)

Finite-domain representation a.k.a. SAS$^+$

What we just saw.

Derives from SAS = Simplified Action Structures (Bäckström et al., 1991)

STRIPS

- all variable domains are $\{\mathbf{T}, \mathbf{F}\}$
- only $v \mapsto \mathbf{T}$ in action preconditions and goal
- set-based notations:

  $a: \quad x \mapsto \mathbf{T},\ y \mapsto \mathbf{T} \quad \xrightarrow{5} \quad w := \mathbf{F},\ y := \mathbf{F},\ z := \mathbf{T}$

  written as

  $pre(a) = \{x, y\},\ add(a) = \{z\},\ del(a) = \{w, y\},\ cost(a) = 5$

Blocks World State with Propositional Variables

Example

$s(\textit{A-on-B}) \mapsto \mathbf{F}$ $s(\textit{A-on-C}) \mapsto \mathbf{F}$

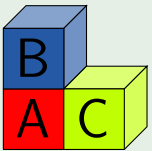
$s(\textit{A-on-table}) \mapsto \mathbf{T}$ $s(\textit{clear-A}) \mapsto \mathbf{F}$

$s(\textit{B-on-A}) \mapsto \mathbf{T}$ $s(\textit{B-on-C}) \mapsto \mathbf{F}$

$s(\textit{B-on-table}) \mapsto \mathbf{F}$ $s(\textit{clear-B}) \mapsto \mathbf{T}$

$s(\textit{C-on-A}) \mapsto \mathbf{F}$ $s(\textit{C-on-B}) \mapsto \mathbf{F}$

$s(\textit{C-on-table}) \mapsto \mathbf{T}$ $s(\textit{clear-C}) \mapsto \mathbf{T}$



Operator: move-A-from-B-to-C

Action: move-A-from-B-to-C

Preconditions

$$A\text{-}on\text{-}B \mapsto \mathbf{T}, clear\text{-}A \mapsto \mathbf{T}, clear\text{-}C \mapsto \mathbf{T}$$

Effects

$$A\text{-}on\text{-}C := \mathbf{T}, clear\text{-}B := \mathbf{T}, A\text{-}on\text{-}B := \mathbf{F}, clear\text{-}C := \mathbf{F}$$

Same Operator in Set Notation

$$pre(\texttt{move-A-from-B-to-C}) = \{A\text{-}on\text{-}B, clear\text{-}A, clear\text{-}C\}$$
$$add(\texttt{move-A-from-B-to-C}) = \{A\text{-}on\text{-}C, clear\text{-}B, clear\text{-}B\}$$
$$del(\texttt{move-A-from-B-to-C}) = \{A\text{-}on\text{-}B, clear\text{-}C\}$$

Formalism Variants for Classical Planning Tasks (1)

ADL

- all variable domains are $\{\mathbf{T}, \mathbf{F}\}$
- preconditions and goal are logical formulae over state variables
  - precondition $x \mapsto \mathbf{T},\ y \mapsto \mathbf{T}$ becomes $x \wedge y$
  - but not limited to conjunctions: $(x \vee \neg(y \vee \neg z))$ etc.
- conditional (state-dependent) effects
  - $(y \vee z) \rhd (x := \mathbf{T}),\ (\neg x) \rhd (y := \mathbf{F})$

$\rightsquigarrow$ variants with and without action costs

PDDL

- Planning Domain Definition Language
- Standard input language used in the International Planning Competitions
- Introduced by McDermott et al., 1998, occasionally updated and extended
  (Edelkamp and Hoffmann, 2004; Fox et al., 2003; Gerevini et al., 2005)
- Only Boolean state variables in basic version
- Schematic actions

```
(:action move
  :parameters (?x ?y ?z)
  :precondition (and (on ?x ?y) (clear ?x) (clear ?z))
  :effect (and (on ?x ?z) (clear ?y) (not (on ?x ?y)) (not (clear ?z)))
)
```

Satisficing or Optimal Planning?

must carefully distinguish:

- satisficing planning: any plan is OK (cheaper ones preferred)
- optimal planning: plans must have minimum cost

solved by similar techniques, but:

- details very different
- almost no overlap between best techniques for satisficing planning and best techniques for optimal planning
- many tasks that are trivial for satisficing planners are impossibly hard for optimal planners

Decision Problems for Planning

Definition (Plan Existence)

GIVEN: planning task Π
QUESTION: Is there a plan for Π?

$\rightsquigarrow$ decision problem analogue of satisficing planning

Definition (Bounded-Cost Plan Existence)

GIVEN: planning task Π, cost bound $K \in \mathbb{N}_0$
QUESTION: Is there a plan for Π with cost at most K?

$\rightsquigarrow$ decision problem analogue of optimal planning

Decision Problems for Planning

Definition (Plan Existence)

GIVEN: planning task Π
QUESTION: Is there a plan for Π?

$\rightsquigarrow$ decision problem analogue of satisficing planning

Definition (Bounded-Cost Plan Existence)

GIVEN: planning task Π, cost bound $K \in \mathbb{N}_0$
QUESTION: Is there a plan for Π with cost at most K?

$\rightsquigarrow$ decision problem analogue of optimal planning

Theorem (PSPACE-Completeness)

Plan existence and bounded plan existence are PSPACE*-complete.*
This is true even if only STRIPS tasks are allowed. *(Bylander, 1994)*

The Big Three

Of the many planning approaches, three techniques stand out:

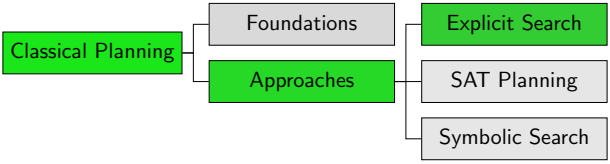
- explicit search
- SAT planning
- symbolic search

Many other approaches:

- Property Directed Reachability (Bradley, 2011; Suda, 2014)
- Planning via Petri Net Unfolding (Bonet et al., 2014; Hickmott et al., 2007; McMillan, 1992)
- Partial-order Planning (Bercher et al., n.d.; Olz et al., 2019; Sacerdoti, n.d.; Younes et al., 2003)
- Factored Planning (Amir et al., 2003; Brafman et al., 2006; Fabre et al., 2010; Kelareva et al., 2007; Knoblock, 1994)
- ...

also: many algorithm portfolios

Content


Classical Planning
Foundations
Approaches
Explicit Search
SAT Planning
Symbolic Search

Explicit Search

Explicit search in the state space (= transition system).

Major Topics

- heuristics for informed search algorithms
- pruning techniques: invariants, symmetry elimination, partial-order reduction, helpful actions pruning, ...

How do we find good heuristics in a domain-independent way?

$\rightsquigarrow$ one of the main focus areas of classical planning research
(and of part 2 of this presentation)

Planning by Forward Search: Progression

Progression: Computing the successor state $s[\![o]\!]$ of a state s with respect to an operator o.

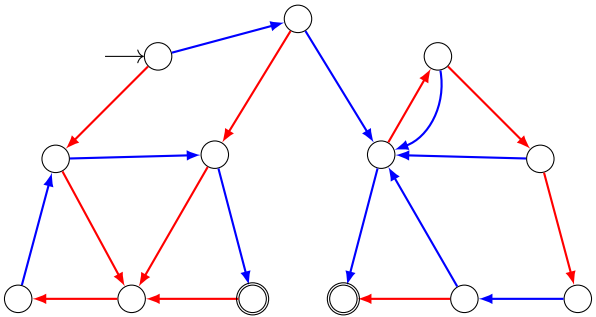
Progression planners find solutions by forward search:

- start from initial state
- iteratively pick a previously generated state and progress it through an operator, generating a new state
- solution found when a goal state generated

pro: very easy and efficient to implement

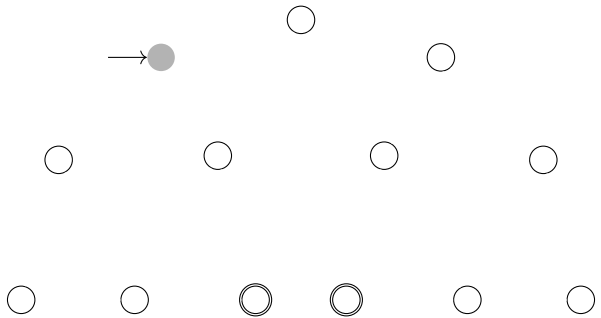
Progression Planning Example

Example of a progression search



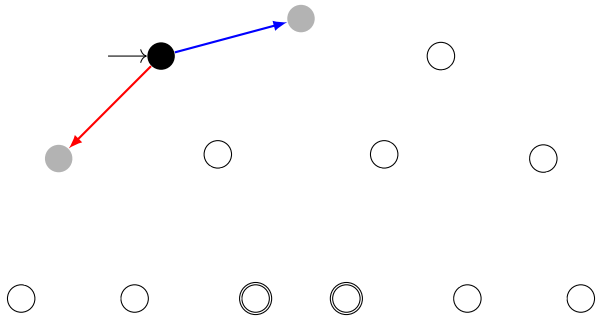
Progression Planning Example

Example of a progression search



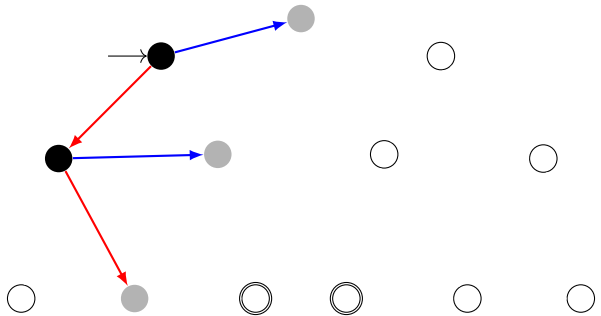
Progression Planning Example

Example of a progression search



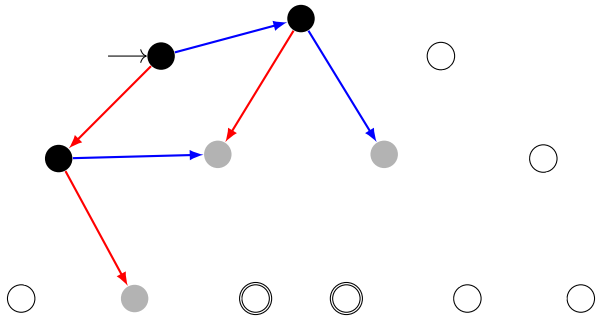
Progression Planning Example

Example of a progression search



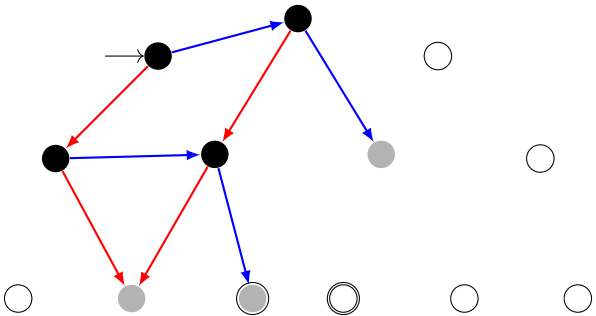
Progression Planning Example

Example of a progression search



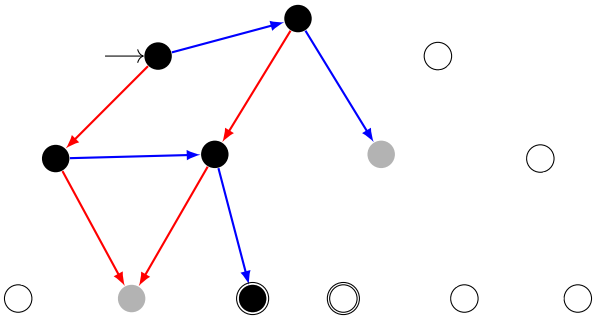
Progression Planning Example

Example of a progression search



Progression Planning Example

Example of a progression search



Backward Search

Can also search in backward direction from goal states to initial state.
Not symmetric to forward search:

- Forward search starts from a single initial state;
  backward search starts from a set of goal states
- When applying an operator o in a state s in forward direction,
  there is a unique successor state s';
  if we just applied operator o and ended up in state s',
  there can be several possible predecessor states s

Backward Search

Can also search in backward direction from goal states to initial state.
Not symmetric to forward search:

- Forward search starts from a single initial state;
  backward search starts from a set of goal states
- When applying an operator o in a state s in forward direction,
  there is a unique successor state s';
  if we just applied operator o and ended up in state s',
  there can be several possible predecessor states s

$\rightsquigarrow$ In most natural representation for backward search in planning,
each search state corresponds to a set of world states

$\rightsquigarrow$ Represent sets of world states as formulas: φ represents $\{s \in S \mid s \models \varphi\}$

$\rightsquigarrow$ Many basic search operations like detecting duplicates
are NP-complete or coNP-complete

Key Concept from Backward Search: Regression

Regression: Computing the possible predecessor states $regr(S', o)$
of a set of states S' ("subgoal") given the last operator o that was applied.

Regression Property

For all states s, opertors o and sets of states described by a conjunction of atoms φ,

$$s \models regr(\varphi, o) \quad \text{iff} \quad s[\![o]\!] \models \varphi.$$

Key Concept from Backward Search: Regression

Regression: Computing the possible predecessor states $regr(S', o)$ of a set of states S' ("subgoal") given the last operator o that was applied.

Regression Property

For all states s, opertors o and sets of states described by a conjunction of atoms φ,

$$s \models regr(\varphi, o) \quad \text{iff} \quad s[\![o]\!] \models \varphi.$$

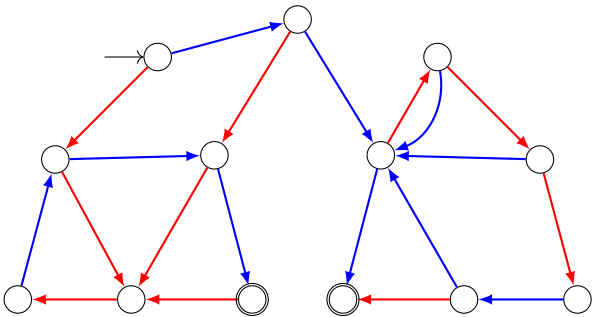
Definition (STRIPS Regression)

Let $\varphi = v_1 \wedge \cdots \wedge v_n$ be a conjunction of atoms, and let o be a STRIPS operator with $add(o) = \{a_1, \ldots, a_k\}$ and $del(o) = \{d_1, \ldots, d_l\}$ (w.l.o.g. $del(o) \cap add(o) = \emptyset$).

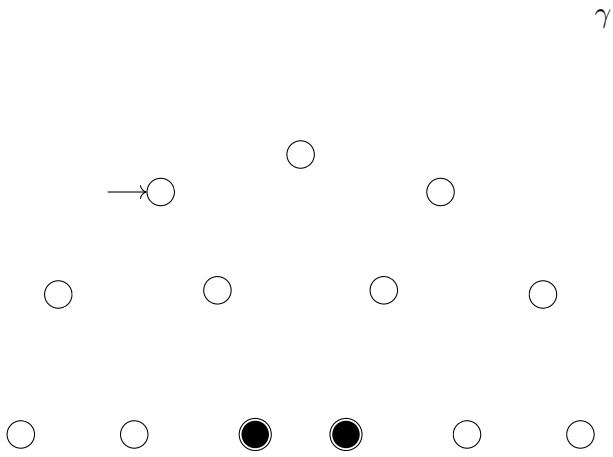
$$regr(\varphi, o) := \begin{cases} \bot & \text{if } v_i = d_j \text{ for some } i, j \\ \bigwedge(\{pre(o)\} \cup (\{v_1, \ldots, v_n\} \setminus \{a_1, \ldots, a_k\})) & \text{else} \end{cases}$$

Regression beyond STRIPS: (Rintanen, 2008)

Regression Planning Example (Depth-first Search)



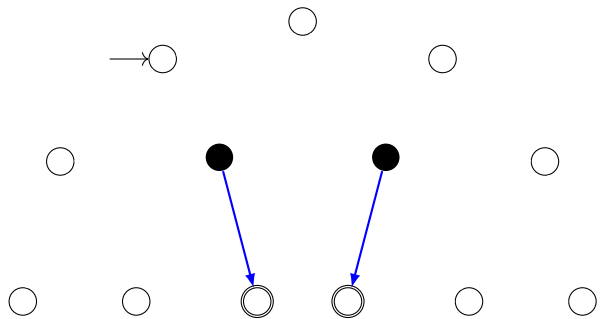
Regression Planning Example (Depth-first Search)



Regression Planning Example (Depth-first Search)

$\varphi_1 = \mathit{regr}(\gamma, \longrightarrow)$

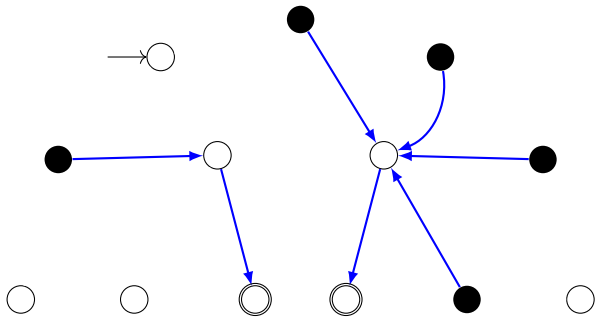
$\varphi_1 \longrightarrow \gamma$



Regression Planning Example (Depth-first Search)

$\varphi_1 = \mathit{regr}(\gamma, \longrightarrow)$
$\varphi_2 = \mathit{regr}(\varphi_1, \longrightarrow)$

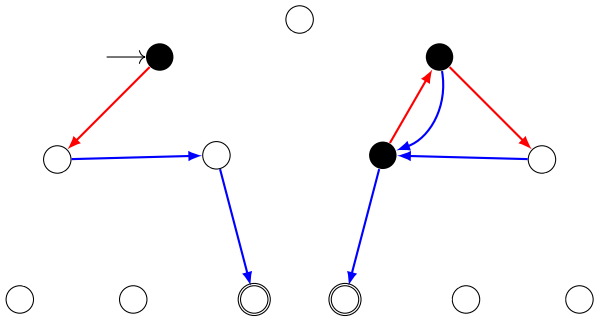
$\varphi_2 \longrightarrow \varphi_1 \longrightarrow \gamma$



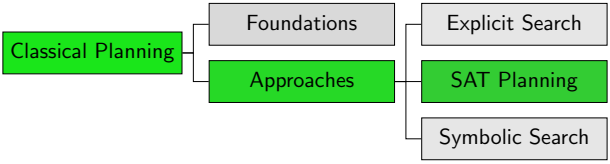
Regression Planning Example (Depth-first Search)

$\varphi_1 = \mathit{regr}(\gamma, \longrightarrow)$
$\varphi_2 = \mathit{regr}(\varphi_1, \longrightarrow)$
$\varphi_3 = \mathit{regr}(\varphi_2, \longrightarrow), I \models \varphi_3$

$\varphi_3 \longrightarrow \varphi_2 \longrightarrow \varphi_1 \longrightarrow \gamma$



Content


Classical Planning
Foundations
Approaches
Explicit Search
SAT Planning
Symbolic Search

SAT Planning

- SAT solvers (algorithms that find satisfying assignments to propositional logic formulas) are one of the major success stories in solving hard combinatorial problems.
- SAT planning (a.k.a. planning as satisfiability) leverages them for classical planning.
- Proposed by Kautz et al. (1992)

Complexity Mismatch

- The SAT problem is NP-complete,
  while plan existence is PSPACE-complete.

$\rightsquigarrow$ one-shot polynomial reduction from plan existence to SAT not possible (unless NP $=$ PSPACE)

Solution: Iterative Deepening

- We can generate a propositional formula that tests if task Π has a plan with horizon (length bound) T in time $O(\|\Pi\|^k \cdot T)$ ($\rightsquigarrow$ pseudo-polynomial reduction).
- Use as building block of algorithm that probes increasing horizons.

Solution: Iterative Deepening

- We can generate a propositional formula that tests if task Π has a plan with horizon (length bound) T in time $O(\|\Pi\|^k \cdot T)$ ($\rightsquigarrow$ pseudo-polynomial reduction).
- Use as building block of algorithm that probes increasing horizons.

basic SAT planning algorithm:

SAT Planning

def satplan(Π):
    for $T \in \{0, 1, 2, \dots\}$:
        $\varphi :=$ build_sat_formula(Π, T)
        $M =$ sat_solver(φ) ▷ returns a model or **none**
        if M **is not none**:
            return extract_plan(Π, T, M)

SAT Formula: Variables

- given propositional planning task Π with variables V and operators O
- given horizon $T \in \mathbb{N}_0$

Variables of the SAT Formula

- propositional variables v^t for all $v \in V$, $0 \leq t \leq T$
  encode state after t steps of the plan
- propositional variables o^t for all $o \in O$, $1 \leq t \leq T$
  encode operator(s) applied in t-th step of the plan

Formulas with Time Steps

Definition (Time-Stamped Formulas)

Let φ be a propositional logic formula over the variables V.
Let $0 \leq t \leq T$.

We write φ^t for the formula obtained from φ by replacing each $v \in V$ with v^t.

Example: $((a \wedge b) \vee \neg c)^3 = (a^3 \wedge b^3) \vee \neg c^3$

Sequential Encoding I

Transition formula τ_o^t for operator o states that o is applicable at time step t and expresses the values of the state variables at time step t (after applying o) in terms of their values at time step $t-1$:

$$\tau_o^t = pre(o)^{t-1} \wedge \bigwedge_{v \in V} [v^t \leftrightarrow (effcond(v, eff(o))^{t-1} \vee (v^{t-1} \wedge \neg effcond(\neg v, eff(o))^{t-1}))]$$

Sequential Encoding II

Formula $\mathcal{T}(t)$ states that from $t-1$ to t the state changes as implied by one of the operators.

$$\mathcal{T}(t) = \bigvee_{o \in O} \tau_o^t$$

Then $I^0 \wedge \mathcal{T}(1) \wedge \cdots \wedge \mathcal{T}(T) \wedge G^T$ is satisfiable iff there is a plan of length T.

Many SAT solvers require input in CNF; use an efficient transformation (Jackson et al., 2004; Manolios et al., 2007; Tseitin, 1968).

Parallel Encodings

Parallel plans permit multiple operators in a time step.

- $\forall$-step plans: A set of actions can be executed in the same time step t only if every ordering of those actions is applicable in s^{t-1} and leads to the same state s^t (Blum et al., 1997).
- $\exists$-step plans: A set of actions can be executed in the same time step t if there exists at least one valid ordering of those actions that is executable in s^{t-1} and results in state s^t (Rintanen et al., 2006).

- More efficient: Smaller necessary horizon T leads to smaller formulas (faster SAT solving) and fewer calls of the SAT solver.
- But cannot be used for optimal (sequential) planning.

Base Encoding

Base Encoding

precondition clauses:

- $o^t \rightarrow pre(o)^{t-1}$ for all $1 \leq t \leq T$, $o \in O$

positive and negative effect formulas:

- $[\bigvee_{o \in O}(o^t \wedge \alpha_o^{t-1})] \rightarrow v^t$ for all $1 \leq t \leq T$, $v \in V$
- $[\bigvee_{o \in O}(o^t \wedge \neg\alpha_o^{t-1} \wedge \delta_o^{t-1})] \rightarrow \neg v^t$ for all $1 \leq t \leq T$, $v \in V$

positive and negative frame clauses:

- $(v^{t-1} \wedge \neg v^t) \rightarrow \bigvee_{o \in O}(o^t \wedge \delta_o^{t-1})$ for all $1 \leq t \leq T$, $v \in V$
- $(\neg v^{t-1} \wedge v^t) \rightarrow \bigvee_{o \in O}(o^t \wedge \alpha_o^{t-1})$ for all $1 \leq t \leq T$, $v \in V$

where $\alpha_o = effcond(v, eff(o))$, $\delta_o = effcond(\neg v, eff(o))$.

(Still permits parallel operators that cannot be sequenced.)

$\forall$-step Plans: Encoding

Definition (Affected STRIPS Operator)

Operator o affects operator o' of a STRIPS task if o deletes a precondition of o'.

$\rightarrow$ Extend base encoding with $\neg o^t \vee \neg o'^t$
for all time steps t and all $o \neq o'$, where o affects o' or vice versa.
(Kautz et al., 1996).

$\rightsquigarrow$ quadratic in number of operators;
there also is a linear encoding of conflicts (Rintanen et al., 2006)

$\exists$-step Plans: Encoding

- Fix an ordering $o_1, \ldots, o_n$ of the operators.
- Extend base encoding with

$$o_i^t \rightarrow \neg o_j^t$$

for all time steps t and all o_i, o_j, where o_i affects o_j and $i < j$.

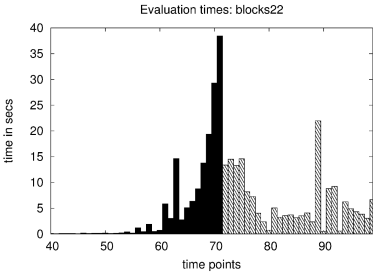
$\rightsquigarrow$ only approximative: there are parallel plans that do not satisfy this encoding.

$\rightsquigarrow$ quadratic in number of operators;
there also is a linear encoding of conflicts (Rintanen et al., 2006)

Search Strategy

Sequential Search Strategy

def satplan(Π):
  for $T \in \{0, 1, 2, \dots\}$:
    $\varphi :=$ build_sat_formula(Π, T)
    $M =$ sat_solver(φ)
    if M **is not none**:
      return extract_plan(Π, T, M)



from Rintanen et al., 2006

Other strategies (Rintanen et al., 2006):

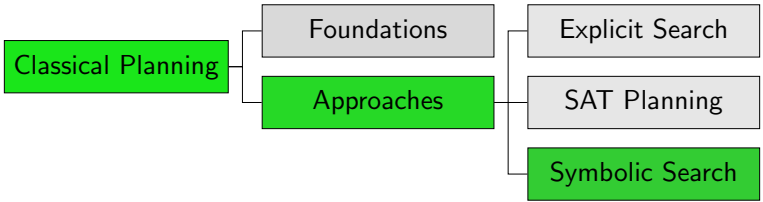
- use multiple concurrent processes and stop at first solution
- interleaved evaluation with a geometric division of CPU use

Do not guarantee shortest possible horizon length but can be arbitrarily much faster (and at most a constant factor slower).

Further Topics

- Further encodings (Huang et al., 2012; Wehrle et al., 2007)
- Tayloring SAT solvers towards planning (Rintanen, 2012)

Content


Classical Planning
Foundations
Approaches
Explicit Search
SAT Planning
Symbolic Search

Symbolic Search

Planning as symbolic search

(Edelkamp and Kissmann, 2008; Speck, 2022; Torralba, Alcázar, et al., 2017)

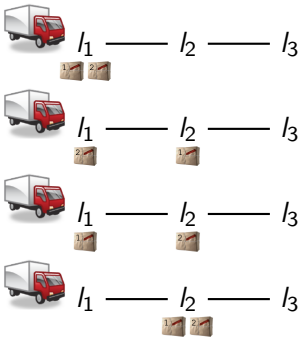
- express sets of states and transition relation as formulas
- planning algorithms based on relational operations
- suitable data structure: BDDs
- can process many states at once

Sets of States as Logical Formulas

$\langle t, l_1 \rangle \wedge \langle p_1, l_1 \rangle \wedge \langle p_2, l_1 \rangle$

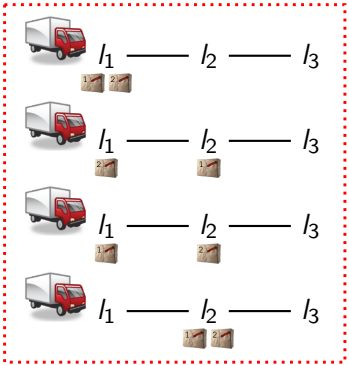
Disclaimer: Here, we ignore state invariants: $\langle t, l_1 \rangle \leftrightarrow (\neg\langle t, l_2 \rangle \wedge \neg\langle t, l_3 \rangle), \ldots$

Sets of States as Logical Formulas



Disclaimer: Here, we ignore state invariants: $\langle t, l_1 \rangle \leftrightarrow (\neg\langle t, l_2 \rangle \wedge \neg\langle t, l_3 \rangle), \ldots$

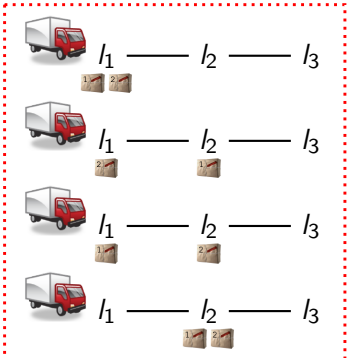
Sets of States as Logical Formulas



$$\langle t, l_1\rangle \wedge \langle p_1, l_1\rangle \wedge \langle p_2, l_1\rangle$$
$$\vee$$
$$\langle t, l_1\rangle \wedge \langle p_1, l_2\rangle \wedge \langle p_2, l_1\rangle$$
$$\vee$$
$$\langle t, l_1\rangle \wedge \langle p_1, l_1\rangle \wedge \langle p_2, l_2\rangle$$
$$\vee$$
$$\langle t, l_1\rangle \wedge \langle p_1, l_2\rangle \wedge \langle p_2, l_2\rangle$$

Disclaimer: Here, we ignore state invariants: $\langle t, l_1\rangle \leftrightarrow (\neg\langle t, l_2\rangle \wedge \neg\langle t, l_3\rangle), \ldots$

Sets of States as Logical Formulas



$$\langle t, l_1\rangle \wedge \langle p_1, l_1\rangle \wedge \langle p_2, l_1\rangle$$
$$\vee$$
$$\langle t, l_1\rangle \wedge \langle p_1, l_2\rangle \wedge \langle p_2, l_1\rangle$$
$$\vee$$
$$\langle t, l_1\rangle \wedge \langle p_1, l_1\rangle \wedge \langle p_2, l_2\rangle$$
$$\vee$$
$$\langle t, l_1\rangle \wedge \langle p_1, l_2\rangle \wedge \langle p_2, l_2\rangle$$

$$\langle \mathbf{t}, \mathbf{l_1}\rangle \wedge (\langle \mathbf{p_1}, \mathbf{l_1}\rangle \vee \langle \mathbf{p_1}, \mathbf{l_2}\rangle) \wedge (\langle \mathbf{p_2}, \mathbf{l_1}\rangle \vee \langle \mathbf{p_2}, \mathbf{l_2}\rangle)$$

Disclaimer: Here, we ignore state invariants: $\langle t, l_1\rangle \leftrightarrow (\neg\langle t, l_2\rangle \wedge \neg\langle t, l_3\rangle), \ldots$

Operating with Sets of States as Logical Formulas

Sets	Logic
Empty set	$\bot$
All states	$\top$
All states in which the truck is at l_1	$\langle t, l_1 \rangle$
Union ($\cup$)	Disjunction ($\vee$)
Inersection ($\cap$)	Conjunction ($\wedge$)
Complement	Negation ($\neg$)

How to Represent Logical Formulas in Practice?

- k: number of state variables
- $\|S\|$: representation size of S

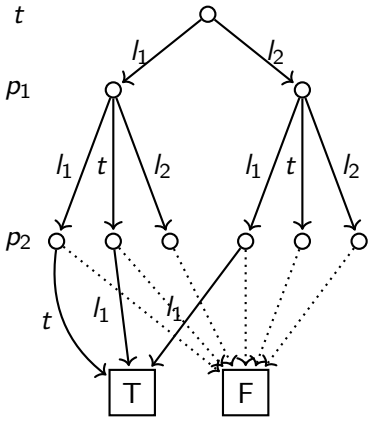
	Formula	BDD
$s \in S$?	$O(\|S\|)$	$O(k)$
$S := S \cup \{s\}$	$O(k)$	$O(k)$
$S := S \setminus \{s\}$	$O(k)$	$O(k)$
$S \cup S'$	$O(1)$	$O(\|S\|\|S'\|)$
$S \cap S'$	$O(1)$	$O(\|S\|\|S'\|)$
$S \setminus S'$	$O(1)$	$O(\|S\|\|S'\|)$
$\overline{S}$	$O(1)$	$O(1)$
$\{s \mid s(v) = 1\}$	$O(1)$	$O(1)$
$S = \emptyset$?	co-NP-complete	$O(1)$
$S = S'$?	co-NP-complete	$O(1)$
$\|S\|$	#P-complete	$O(\|S\|)$

Binary Decision Diagrams (BDDs)

Multi-valued Decision Diagram (MDD)

DAG with a fixed variable ordering.

t	p_1	p_2
l_1	l_1	t
l_2	l_1	l_1
l_1	t	l_1



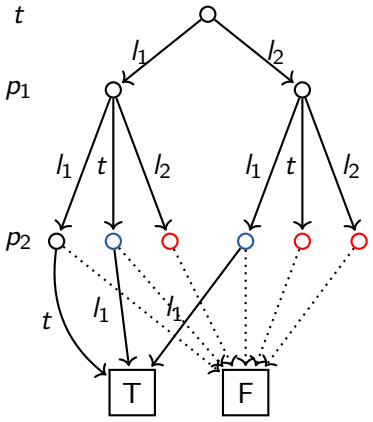
Binary Decision Diagrams (BDDs)

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

t	p_1	p_2
l_1	l_1	t
l_2	l_1	l_1
l_1	t	l_1



Binary Decision Diagrams (BDDs)

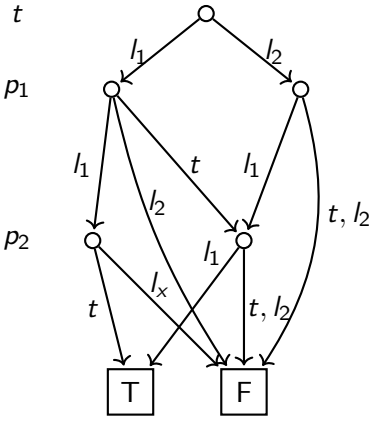
Multi-valued Decision Diagram (MDD)

DAG with a fixed variable ordering.

Reduction rules:

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

t	p_1	p_2
l_1	l_1	t
l_2	l_1	l_1
l_1	t	l_1



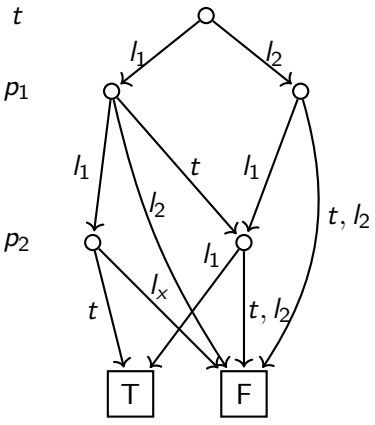
Binary Decision Diagrams (BDDs)

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

t	p_1	p_2
l_1	l_1	t
l_2	l_1	l_1
l_1	t	l_1



Binary Decision Diagrams: MDDs, where variables are all binary
$\rightsquigarrow$ Use $\lceil \log_2 |\mathrm{dom}(v)| \rceil$ binary variables per finite-domain variable v

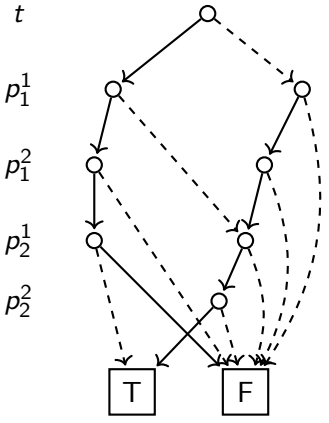
Binary Decision Diagrams (BDDs)

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

t	p_1	p_2
l_1	l_1	t
l_2	l_1	l_1
l_1	t	l_1

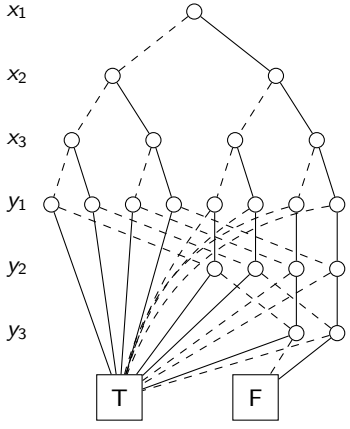


Binary Decision Diagrams: MDDs, where variables are all binary
$\rightsquigarrow$ Use $\lceil \log_2 |\mathrm{dom}(v)| \rceil$ binary variables per finite-domain variable v

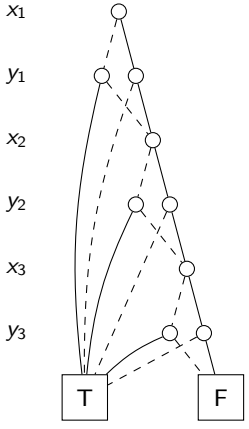
BDD Variable Ordering

$$(x_1 \neq y_1) \vee (x_2 \neq y_2) \vee (x_3 \neq y_3)$$

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



Polynomial ($3n + 2$)



Strategies for a Good Variable Ordering

- Finding the optimal BDD ordering is NP-hard (Bryant, 1986)
- Almost impossible to predict BDDs sizes before creating them

Strategies for a Good Variable Ordering

- Finding the optimal BDD ordering is NP-hard (Bryant, 1986)
- Almost impossible to predict BDDs sizes before creating them

In practice:

Static Variable Ordering

- put causally related variables close (Kissmann and Edelkamp, 2011)
- no strong theoretical guarantees (Kissmann and Hoffmann, 2013)

Strategies for a Good Variable Ordering

- Finding the optimal BDD ordering is NP-hard (Bryant, 1986)
- Almost impossible to predict BDDs sizes before creating them

In practice:

Static Variable Ordering

- put causally related variables close (Kissmann and Edelkamp, 2011)
- no strong theoretical guarantees (Kissmann and Hoffmann, 2013)

Dynamic Variable Ordering

- re-ordering to minimize the size of the BDDs generated so far
- practical approximations (based on local-search) (Rudell, 1993)
- applied in planning with good results (Kissmann and Hoffmann, 2014)

Planning Actions as Logical Formulas

Transition Relation

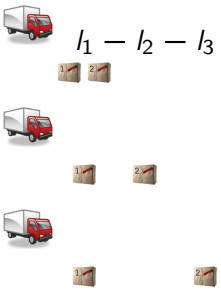
Represents an action a as the relation (set of pairs of states) containing (s, s') where a is applicable in s resulting in s'.

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'.

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



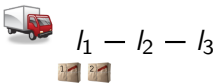
$\langle t, l_1\rangle \wedge \langle p_1, l_1\rangle \wedge \langle p_2, l_1\rangle \wedge$
$\langle t, l_1\rangle' \wedge \langle p_1, t\rangle' \wedge \langle p_2, l_1\rangle'$

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'.

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



$\langle t, l_1\rangle \wedge \langle p_1, l_1\rangle \wedge \langle p_2, l_1\rangle \wedge$
$\langle t, l_1\rangle' \wedge \langle p_1, t\rangle' \wedge \langle p_2, l_1\rangle'$

$\langle t, l_1\rangle \wedge \langle p_1, l_1\rangle \wedge \langle p_2, l_2\rangle \wedge$
$\langle t, l_1\rangle' \wedge \langle p_1, t\rangle' \wedge \langle p_2, l_2\rangle'$

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'.

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

$l_1 - l_2 - l_3$

$l_1 - l_2 - l_3$

$\langle t, l_1\rangle \wedge \langle p_1, l_1\rangle \wedge \langle p_2, l_1\rangle \wedge$
$\langle t, l_1\rangle' \wedge \langle p_1, t\rangle' \wedge \langle p_2, l_1\rangle'$

$l_1 - l_2 - l_3$

$l_1 - l_2 - l_3$

$\langle t, l_1\rangle \wedge \langle p_1, l_1\rangle \wedge \langle p_2, l_2\rangle \wedge$
$\langle t, l_1\rangle' \wedge \langle p_1, t\rangle' \wedge \langle p_2, l_2\rangle'$

$l_1 - l_2 - l_3$

$l_1 - l_2 - l_3$

$\langle t, l_1\rangle \wedge \langle p_1, l_1\rangle \wedge \langle p_2, l_3\rangle \wedge$
$\langle t, l_1\rangle' \wedge \langle p_1, t\rangle' \wedge \langle p_2, l_3\rangle'$

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'.

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

$\langle t, l_1\rangle \wedge \langle p_1, l_1\rangle \wedge \langle p_2, l_1\rangle \wedge$
$\langle t, l_1\rangle' \wedge \langle p_1, t\rangle' \wedge \langle p_2, l_1\rangle'$

$\langle t, l_1\rangle \wedge \langle p_1, l_1\rangle \wedge \langle p_2, l_2\rangle \wedge$
$\langle t, l_1\rangle' \wedge \langle p_1, t\rangle' \wedge \langle p_2, l_2\rangle'$

$\langle t, l_1\rangle \wedge \langle p_1, l_1\rangle \wedge \langle p_2, l_3\rangle \wedge$
$\langle t, l_1\rangle' \wedge \langle p_1, t\rangle' \wedge \langle p_2, l_3\rangle'$

$$\langle p_1, l_1\rangle \wedge \langle t, l_1\rangle \wedge \langle p_1, t\rangle' \wedge \langle t, l_1\rangle' \wedge (\langle p_2, l_i\rangle \leftrightarrow \langle p_2, l_i\rangle')$$

Computing the Successors (Image)

Image

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

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

Computing the Successors (Image)

Image

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

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

$S(x)$:

t	p_1	p_2
l_1	l_1	l_1
l_1	l_1	l_2
l_2	l_1	l_3
l_1	l_3	l_1

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

t	p_1	p_2	t'	p_1'	p_2'
l_1	l_1	l_1	l_1	t	l_1
l_1	l_1	l_2	l_1	t	l_2
l_1	l_1	l_3	l_1	t	l_3

Computing the Successors (Image)

Image

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

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

$S(x)$:

t	p_1	p_2
l_1	l_1	l_1
l_1	l_1	l_2
l_2	l_1	l_3
l_1	l_3	l_1

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

t	p_1	p_2	t'	p_1'	p_2'
l_1	l_1	l_1	l_1	t	l_1
l_1	l_1	l_2	l_1	t	l_2
l_1	l_1	l_3	l_1	t	l_3

Result:

t	p_1	p_2	t'	p_1'	p_2'
l_1	l_1	l_1	l_1	t	l_1
l_1	l_1	l_2	l_1	t	l_2

Computing the Successors (Image)

Image

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

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

$S(x)$:

t	p_1	p_2
l_1	l_1	l_1
l_1	l_1	l_2
l_2	l_1	l_3
l_1	l_3	l_1

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

t	p_1	p_2	t'	p_1'	p_2'
l_1	l_1	l_1	l_1	t	l_1
l_1	l_1	l_2	l_1	t	l_2
l_1	l_1	l_3	l_1	t	l_3

Result:

t	p_1	p_2	t'	p_1'	p_2'
			l_1	t	l_1
			l_1	t	l_2

Computing the Successors (Image)

Image

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

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

$S(x)$:

t	p_1	p_2
l_1	l_1	l_1
l_1	l_1	l_2
l_2	l_1	l_3
l_1	l_3	l_1

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

t	p_1	p_2	t'	p_1'	p_2'
l_1	l_1	l_1	l_1	t	l_1
l_1	l_1	l_2	l_1	t	l_2
l_1	l_1	l_3	l_1	t	l_3

Result:

t	p_1	p_2	t'	p_1'	p_2'
l_1	t	l_1			
l_1	t	l_2			

Computing the Predecessors (Preimage)

Preimage

Given a state set $S(x)$ and a TR $T(x, x')$ generate the predecessor states.

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

$\rightsquigarrow$ corresponds to regression (Rintanen, 2008)

Efficient Image Computation

Transition Relation Partitioning

- Given a set of actions with the same cost
- Replace $T_i(x, x')$ and $T_j(x, x')$ by $T_i(x, x') \vee T_j(x, x')$
  (Burch et al., 1991; Jensen et al., 2008)

Efficient Image Computation

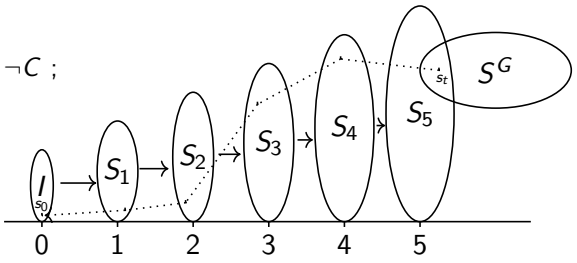
Transition Relation Partitioning

- Given a set of actions with the same cost
- Replace $T_i(x, x')$ and $T_j(x, x')$ by $T_i(x, x') \vee T_j(x, x')$
  (Burch et al., 1991; Jensen et al., 2008)

$\rightsquigarrow$ Merge as many TRs as possible given the available memory
(Torralba, Alcázar, et al., 2017; Torralba, Edelkamp, et al., 2013)

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 $S_i \wedge G$ **then**
    return Plan ;
  end
  $C \leftarrow C \vee S_i$;
  $S_{i+1} \leftarrow image(S_i, TR) \wedge \neg C$;
  $i \leftarrow i + 1$;
end
return Unsolvable ;



Symbolic Uniform-Cost Search

Expand set of states S_i with minimum g-value i

- Zero-cost breadth-first search $\rightsquigarrow$ 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

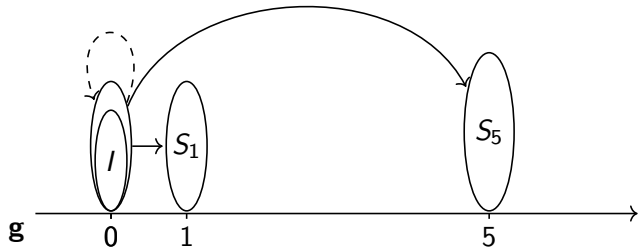
Expand set of states S_i with minimum g-value i

- Zero-cost breadth-first search $\rightsquigarrow$ 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

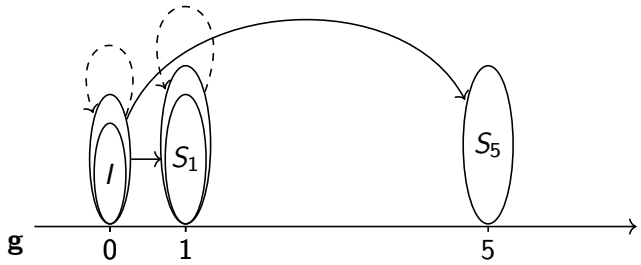
- Zero-cost breadth-first search $\rightsquigarrow$ all states reachable with $g = i$
- For each TR with action cost c:
  - Use image to compute states reachable with $i + c$
  - Insert the result in the corresponding bucket (disjunction)



Symbolic Uniform-Cost Search

Expand set of states S_i with minimum g-value i

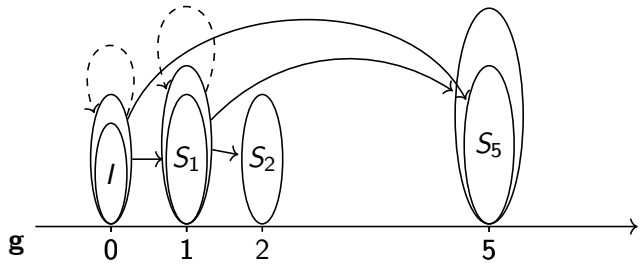
- Zero-cost breadth-first search $\rightsquigarrow$ all states reachable with $g = i$
- For each TR with action cost c:
  - Use image to compute states reachable with $i + c$
  - Insert the result in the corresponding bucket (disjunction)



Symbolic Uniform-Cost Search

Expand set of states S_i with minimum g-value i

- Zero-cost breadth-first search $\rightsquigarrow$ 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 preimage instead of image operation

Challenges:

1. Multiple goal states
2. 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 preimage instead of image operation

Challenges:

1. Multiple goal states $\rightsquigarrow$ Not a problem in symbolic search!
2. Partial states $\rightsquigarrow$ 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 preimage instead of image operation

Challenges:

1. Multiple goal states $\rightsquigarrow$ Not a problem in symbolic search!
2. Partial states $\rightsquigarrow$ Not a problem in symbolic search!
3. Spurious states $\rightsquigarrow$ Solution: state-invariant pruning (Torralba, Alcázar, et al., 2017)

Symbolic Bidirectional Uniform-Cost Search

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

$g_f = 0$ $g_b = 0$

Symbolic Bidirectional Uniform-Cost Search

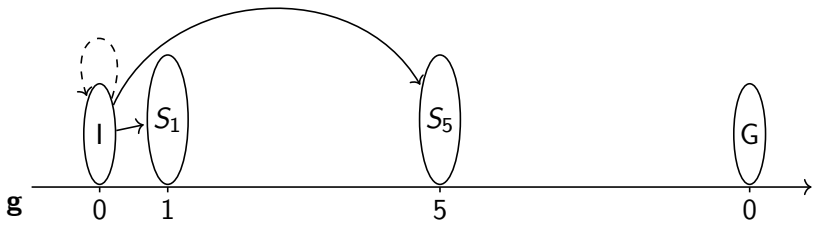
- Do forward and backward search at the same time
- Decide forward or backward direction at each step
- Stop when $g_f + g_b + min_{a \in A} c(a) \geq Sol$

$g_f = 0$ $g_b = 0$

Symbolic Bidirectional Uniform-Cost Search

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

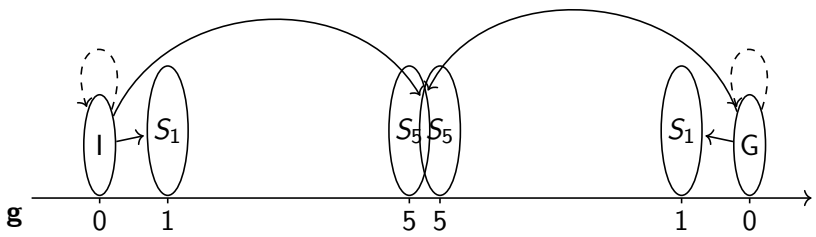
$g_f = 1$ $g_b = 0$



Symbolic Bidirectional Uniform-Cost Search

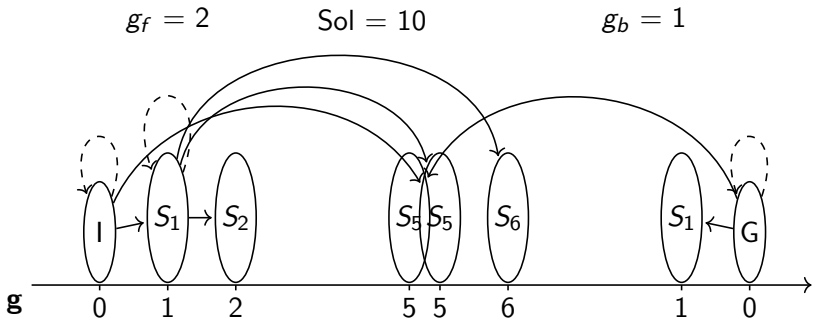
- Do forward and backward search at the same time
- Decide forward or backward direction at each step
- Stop when $g_f + g_b + min_{a \in A} c(a) \geq Sol$

$g_f = 1$ $\quad$ Sol $= 10$ $\quad$ $g_b = 1$



Symbolic Bidirectional Uniform-Cost Search

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



Informed Symbolic Search

- Informed symbolic search algorithms BDDA* (Edelkamp, 2002) and GHSETA* (Jensen et al., 2008)
- Symbolic heuristics, e.g. based on abstraction (Edelkamp, 2002; Torralba, Linares López, et al., 2013, 2016)
  or potential heuristics (Fišer et al., 2022)
- depending on the heuristic, state evaluation can be expensive (Jensen et al., 2008)
- can increase or decrease the sizes of the BDDs (Speck et al., 2020)
  - in the worst case exponentially
  - even with the perfect heuristic h^*

Summary

- Planning tasks are compact encodings of (exponentially larger) transition systems.
- Three major approaches:
  - Explicit search solves planning problems by systematic exploration of the transition system.
  - SAT planning encodes plans for a given horizon T as propositional logic formula.
  - Symbolic search uses BDDs (or related representations) to process entire sets of states and transitions at once.

References I

- Eyal Amir and Barbara Engelhardt (2003). "Factored Planning". In: *Proc. IJCAI 2003*, pp. 929–935.
- Christer Bäckström and Inger Klein (1991). "Planning in polynomial time: the SAS-PUBS class". In: *Computational Intelligence* 7.3, pp. 181–197.
- Pascal Bercher, Thomas Geier, and Susanne Biundo (n.d.). "Using State-Based Planning Heuristics for Partial-Order Causal-Link Planning". In: pp. 1–12.
- Avrim Blum and Merrick L. Furst (1997). "Fast Planning Through Planning Graph Analysis". In: *Artificial Intelligence* 90.1–2, pp. 281–300.
- Blai Bonet, Patrik Haslum, Victor Khomenko, Sylvie Thiébaux, and Walter Vogler (2014). "Recent advances in unfolding technique". In: *Theoretical Computer Science* 551, pp. 84–101.
- Aaron R. Bradley (2011). "SAT-Based Model Checking without Unrolling". In: *Proc. VMCAI 2011*, pp. 70–87.

References II

- Ronen I. Brafman and Carmel Domshlak (2006). "Factored Planning: How, When and When Not". In: *Proc. AAAI 2006*, pp. 809–814.
- Randal E. Bryant (1986). "Graph-Based Algorithms for Boolean Function Manipulation". In: *IEEE Transactions on Computers* 35.8, pp. 677–691.
- Jerry R. Burch, Edmund M. Clarke, and David E. Long (1991). "Symbolic Model Checking with Partitioned Transition Relations". In: *Proc. VLSI 1991*, pp. 49–58.
- Tom Bylander (1994). "The Computational Complexity of Propositional STRIPS Planning". In: *Artificial Intelligence* 69.1–2, pp. 165–204.
- Stefan Edelkamp (2002). "Symbolic Pattern Databases in Heuristic Search Planning". In: *Proc. AIPS 2002*, pp. 274–283.
- Stefan Edelkamp and Jörg Hoffmann (2004). *PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition*. Tech. rep. 195. University of Freiburg, Department of Computer Science.

References III

- Stefan Edelkamp and Peter Kissmann (2008). "Limits and Possibilities of BDDs in State Space Search". In: *Proc. AAAI 2008*, pp. 1452–1453.
- Eric Fabre, Loïg Jezequel, Patrik Haslum, and Sylvie Thiébaux (2010). "Cost-Optimal Factored Planning: Promises and Pitfalls". In: *Proc. ICAPS 2010*, pp. 65–72.
- Daniel Fišer, Álvaro Torralba, and Jörg Hoffmann (2022). "Operator-Potential Heuristics for Symbolic Search". In: *Proc. AAAI 2022*, pp. 9750–9757.
- Maria Fox and Derek Long (2003). "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains". In: *Journal of Artificial Intelligence Research* 20, pp. 61–124.
- Alfonso E. Gerevini and Derek Long (2005). *Plan Constraints and Preferences in PDDL3*. Tech. rep. R. T. 2005-08-47. University of Brescia, Department of Electronics for Automation.

References IV

- Sarah L. Hickmott, Jussi Rintanen, Sylvie Thiébaux, and Langford B. White (2007). "Planning via Petri Net Unfolding". In: *Proc. IJCAI 2007*, pp. 1904–1911.
- Ruoyun Huang, Yixin Chen, and Weixiong Zhang (2012). "SAS$^+$ Planning as Satisfiability". In: *Journal of Artificial Intelligence Research* 43, pp. 293–328.
- Paul B. Jackson and Daniel Sheridan (2004). "Clause Form Conversions for Boolean Circuits". In: *Theory and Applications of Satisfiability Testing, SAT 2004, 2004, Revised Selected Papers*. Vol. 3542. LNCS. Springer, pp. 183–198.
- Rune M. Jensen, Manuela M. Veloso, and Randal E. Bryant (2008). "State-set branching: Leveraging BDDs for heuristic search". In: *Artificial Intelligence* 172.2–3, pp. 103–139.
- Henry Kautz and Bart Selman (1992). "Planning as Satisfiability". In: *Proc. ECAI 1992*, pp. 359–363.
- Henry Kautz and Bart Selman (1996). "Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search". In: *Proc. AAAI 1996*, pp. 1194–1201.

References V

- Elena Kelareva, Olivier Buffet, Jinbo Huang, and Sylvie Thiébaux (2007). "Factored Planning Using Decomposition Trees". In: *Proc. IJCAI 2007*, pp. 1942–1947.
- Peter Kissmann and Stefan Edelkamp (2011). "Improving Cost-Optimal Domain-Independent Symbolic Planning". In: *Proc. AAAI 2011*, pp. 992–997.
- Peter Kissmann and Jörg Hoffmann (2013). "What's in It for My BDD? On Causal Graphs and Variable Orders in Planning". In: *Proc. ICAPS 2013*, pp. 327–331.
- Peter Kissmann and Jörg Hoffmann (2014). "BDD Ordering Heuristics for Classical Planning". In: *Journal of Artificial Intelligence Research* 51, pp. 779–804.
- Craig A. Knoblock (1994). "Automatically Generating Abstractions for Planning". In: *Artificial Intelligence* 68.2, pp. 243–302.
- Panagiotis Manolios and Daron Vroon (2007). "Efficient Circuit to CNF Conversion". In: *Theory and Applications of Satisfiability Testing, SAT 2007, 2007*. Ed. by João Marques-Silva and Karem A. Sakallah. Vol. 4501. LNCS. Springer, pp. 4–9.

References VI

- Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins (1998). *PDDL – The Planning Domain Definition Language – Version 1.2*. Tech. rep. CVC TR-98-003/DCS TR-1165. Yale University: Yale Center for Computational Vision and Control.
- Kenneth L. McMillan (1992). "Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits". In: *Proc. CAV 1992*, pp. 164–177.
- Conny Olz and Pascal Bercher (2019). "Eliminating Redundant Actions in Partially Ordered Plans – A Complexity Analysis". In: *Proc. ICAPS 2019*, pp. 310–319.
- Jussi Rintanen (2008). "Regression for Classical and Nondeterministic Planning". In: *Proc. ECAI 2008*, pp. 568–572.
- Jussi Rintanen (2012). "Planning as Satisfiability: Heuristics". In: *Artificial Intelligence* 193, pp. 45–86.

References VII

- Jussi Rintanen, Keijo Heljanko, and Ilkka Niemelä (2006). "Planning as satisfiability: parallel plans and algorithms for plan search". In: *Artificial Intelligence* 170.12–13, pp. 1031–1080.
- Richard Rudell (1993). "Dynamic Variable Ordering for Ordered Binary Decision Diagrams". In: *Proc. ICCAD 1993*, pp. 42–47.
- Earl D. Sacerdoti (n.d.). "The Nonlinear Nature of Plans". In: pp. 206–214.
- David Speck (2022). "Symbolic Search for Optimal Planning with Expressive Extensions". PhD thesis. University of Freiburg.
- David Speck, Florian Geißer, and Robert Mattmüller (2020). "When Perfect Is Not Good Enough: On the Search Behaviour of Symbolic Heuristic Search". In: *Proc. ICAPS 2020*, pp. 263–271.
- Martin Suda (2014). "Property Directed Reachability for Automated Planning". In: *Journal of Artificial Intelligence Research* 50, pp. 265–319.

References VIII

- Álvaro Torralba, Vidal Alcázar, Peter Kissmann, and Stefan Edelkamp (2017). "Efficient Symbolic Search for Cost-optimal Planning". In: *Artificial Intelligence* 242, pp. 52–79.
- Álvaro Torralba, Stefan Edelkamp, and Peter Kissmann (2013). "Transition Trees for Cost-Optimal Symbolic Planning". In: *Proc. ICAPS 2013*, pp. 206–214.
- Álvaro Torralba, Carlos Linares López, and Daniel Borrajo (2013). "Symbolic Merge-and-Shrink for Cost-Optimal Planning". In: *Proc. IJCAI 2013*, pp. 2394–2400.
- Álvaro Torralba, Carlos Linares López, and Daniel Borrajo (2016). "Abstraction Heuristics for Symbolic Bidirectional Search". In: *Proc. IJCAI 2016*, pp. 3272–3278.
- Grigori Tseitin (1968). "On the Complexity of Derivation in the Propositional Calculus". In: *Studies in Constructive Mathematics and Mathematical Logic, Part II*. English Translation. Consultants Bureau, New York, pp. 115–125.

References IX

- Martin Wehrle and Jussi Rintanen (2007). "Planning as Satisfiability with Relaxed ∃-Step Plans". In: *AI 2007: Advances in Artificial Intelligence, 20th Australian Joint Conference on Artificial Intelligence*. Vol. 4830. LNCS. Springer, pp. 244–253.
- Håkan L. S. Younes and Reid G. Simmons (2003). "VHPOP: Versatile Heuristic Partial Order Planner". In: *Journal of Artificial Intelligence Research* 20, pp. 405–430.