Concise finite-domain representations for PDDL planning tasks

Ethan Coots

PDDL

- Objects
- Predicates
- Operators
- Initial State
- Goal Specification

Finite-Domain Representations

- Nonbinary representation of variables
- PDDL representation overestimates feasible world states
  - Example: only one proposition of the form at-p1-x can be true at once (PDDL ignores this)

Current goals: cargo_msy in plane_atl, plane_atl in msy

Fly plane_atl msy msy

Load cargo_msy plane_atl atl

Propositions:

at-p1-a, at-p1-b, at-p1-c, at-p1-d, at-p1-e, at-p1-f, at-p1-g,
at-p2-a, at-p2-b, at-p2-c, at-p2-d, at-p2-e, at-p2-f, at-p2-g,
at-c1-a, at-c1-b, at-c1-c, at-c1-d,
at-c2-a, at-c2-b, at-c2-c, at-c2-d,
at-c3-e, at-c3-f, at-c3-g,
at-t-d, at-t-e,
in-p1-c1, in-p1-c2, in-p1-c3, in-p1-t,
in-p2-c1, in-p2-c2, in-p2-c3, in-p2-t

$$\downarrow$$

Variables:

$$\begin{array}{lcl}
\texttt{p1}, \texttt{p2} & \in & \{\texttt{at-a}, \texttt{at-b}, \texttt{at-c}, \texttt{at-d}, \texttt{at-e}, \texttt{at-f}, \texttt{at-g}, \\
 & & \ \texttt{in-c1}, \texttt{in-c2}, \texttt{in-c3}, \texttt{in-t}\} \\
\texttt{c1}, \texttt{c2} & \in & \{\texttt{at-a}, \texttt{at-b}, \texttt{at-c}, \texttt{at-d}\} \\
\texttt{c3} & \in & \{\texttt{at-e}, \texttt{at-f}, \texttt{at-g}\} \\
\texttt{t} & \in & \{\texttt{at-d}, \texttt{at-e}\}
\end{array}$$

Why?

- SAT representations disallowing inconsistent values of a single finite domain variable
- Binary Decision Diagram encodings contain fewer variables
- Less memory used with heuristic planning
- Overall, more concise representation of planning problems

How?

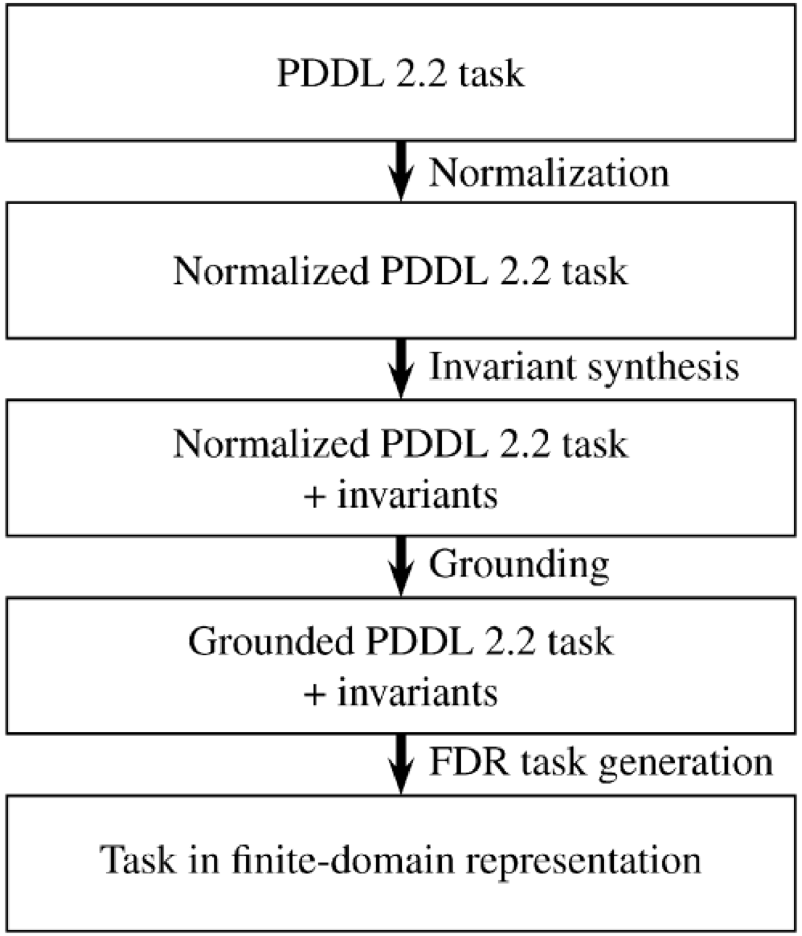

Fig. 6. Overview of the translation algorithm.

Normalization

- Compiles away types
- Simplifies conditions
- Simplifies effects
- This produces a normalized PDDL task

Invariant Synthesis

- Invariant: property satisfied by all world states that are reachable from the initial state
- “Balanced”: Must show that no operator can increase the weight of any of its instances.

$$
\begin{aligned}
\text{Precondition:} \quad & \texttt{at}(x, l_1) \wedge \texttt{at}(x, l_2) \\
\text{Add effects:} \quad & \texttt{at}(x, l_3) \wedge \texttt{at}(x, l_4) \\
\text{Delete effects:} \quad & \texttt{at}(x, l_1) \wedge \texttt{at}(x, l_2)
\end{aligned}
$$

Grounding

- We can improve the naïve “permute-like” grounding algorithm by eliminating ground operators that are not applicable in any reachable state:
  - Exploiting type information
  - Checking static preconditions
  - Checking relaxed reachability

Datalog Exploration

- Encodes atom reachability problem for relaxed planning tasks as a set of logical facts and rules (a logic program)
- Computes *canonical model* of the logic program, consisting of the set of ground atoms that it logically implies.

Implementation

algorithm compute-mutex-groups(*invariants*, $\mathcal{P}_f$, s_0):
    for each invariant $I \in \mathit{invariants}$:
        for each instance α of I:
            if weight$(\alpha, s_0) = 1$:
                Create a mutex group containing all atoms in $\mathcal{P}_f$
                  covered by α.

algorithm choose-variables($\mathcal{P}_f$, *mutex-groups*):
    $\mathit{uncovered} := \mathcal{P}_f$
    while $\mathit{mutex\text{-}groups} \neq \emptyset$:
        Pick a mutex group P of maximal cardinality.
        Create a variable v with domain $\mathcal{D}_v = P \cup \{\perp\}$.
        $\mathit{uncovered} := \mathit{uncovered} \setminus P$
        $\mathit{mutex\text{-}groups} := \{\ P' \setminus P \mid P' \in \mathit{mutex\text{-}groups}\ \}$
        $\mathit{mutex\text{-}groups} := \{\ P' \mid P' \in \mathit{mutex\text{-}groups} \wedge |P'| \geq 2\ \}$
    Create a variable v with domain $\{p, \perp\}$ for each remaining
      element of *uncovered*.

Fig. 17. Greedy algorithm for computing the FDR variables and variable domains.

Conclusion

- FDR is a much more concise representation of planning problems than previously discussed binary representations.