Grounding Planning Tasks Using Tree Decompositions and Iterated Solving

Augusto B. Corrêa, Markus Hecher, Malte Helmert,
Davide Mario Longo, Florian Pommerening, Stefan Woltran

University of Basel, Switzerland
Massachusetts Institute of Technology, USA
TU Wien, Institute of Logic and Computation, Austria

KR 2023

Originally published at ICAPS 2023

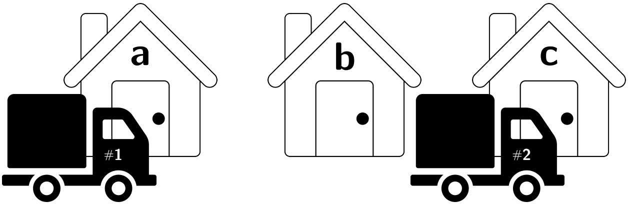
Classical Planning

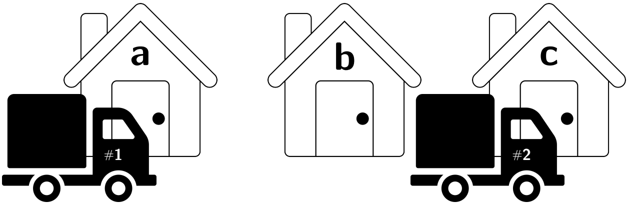
classical planning:

- input: initial state, goal, possible actions
- output: action sequence achieving the goal (plan)

properties: deterministic, fully-observable

A Simple Domain

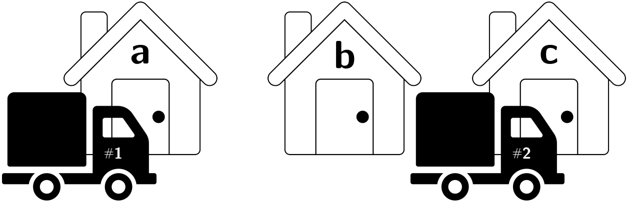




at	
#1	*a*
#2	*c*

adj	
a	*b*
b	*a*
b	*c*
c	*b*

A Simple Domain

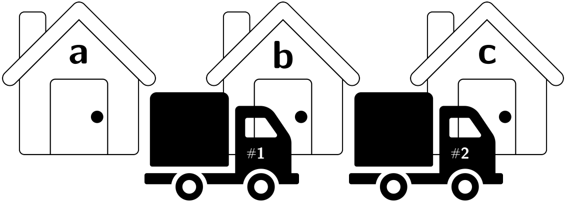


`drive(#1, a, b)`

precondition: $\texttt{at}(\#1, a), \texttt{adj}(a, b)$

effects: $\texttt{at}(\#1, b), \neg\texttt{at}(\#1, a)$

A Simple Domain

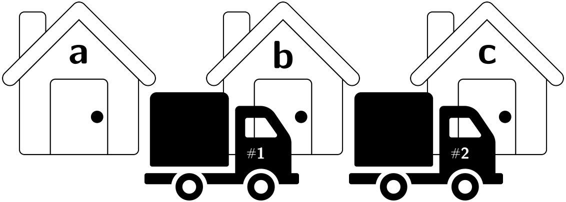


drive(#1, a, b)

precondition: $\texttt{at}(\#1, a), \texttt{adj}(a, b)$

effects: $\texttt{at}(\#1, b), \neg\texttt{at}(\#1, a)$

A Simple Domain



$\texttt{drive(}T\texttt{, }L_1\texttt{, }L_2\texttt{)}$

precondition: $\texttt{at}(T, L_1), \texttt{adj}(L_1, L_2)$

effects: $\texttt{at}(T, L_2), \neg\texttt{at}(T, L_1)$

Grounding

planners use propositional tasks

- solution: ground actions that can be reached from initial state
- very hard to compute this exact set
- overapproximate ground actions

Grounding

planners use propositional tasks

- solution: ground actions that can be reached from initial state
- very hard to compute this exact set
- overapproximate ground actions

`drive(`T`, `L_1`, `L_2`)`

precondition: $\mathtt{at}(T, L_1), \mathtt{adj}(L_1, L_2)$

effects: $\mathtt{at}(T, L_2), \neg\mathtt{at}(T, L_1)$

Grounding

planners use propositional tasks

- solution: ground actions that can be reached from initial state
- very hard to compute this exact set
- overapproximate ground actions

$\texttt{drive}(T,\ L_1,\ L_2)$

precondition: $\texttt{at}(T, L_1), \texttt{adj}(L_1, L_2)$

effects: $\texttt{at}(T, L_2)$

Grounding Datalog

$$\texttt{drive}(T,\ L_1,\ L_2) \leftarrow \texttt{at}(T, L_1), \texttt{adj}(L_1, L_2).$$
$$\texttt{at}(T, L_2) \leftarrow \texttt{drive}(T,\ L_1,\ L_2).$$

Grounding Datalog

$$
\begin{aligned}
\texttt{adj}(a, b). \\
\texttt{adj}(b, a). \\
\texttt{adj}(b, c). \\
\texttt{adj}(c, b). \\
\texttt{at}(\#1, b). \\
\texttt{at}(\#2, c). \\
\texttt{drive}(T,\ L_1,\ L_2) &\leftarrow \texttt{at}(T, L_1), \texttt{adj}(L_1, L_2). \\
\texttt{at}(T, L_2) &\leftarrow \texttt{drive}(T,\ L_1,\ L_2).
\end{aligned}
$$

Common Issue

in general, action schemas have too many parameters

- more than 30 in many domains
- hard to ground all at the same time

Common Issue

in general, action schemas have too many parameters

- more than 30 in many domains
- hard to ground all at the same time

idea: split grounding of atoms and actions

Removing "Action Predicates"

$$\texttt{drive}(T,\ L_1,\ L_2) \leftarrow \texttt{at}(T, L_1), \texttt{adj}(L_1, L_2).$$
$$\texttt{at}(T, L_2) \leftarrow \texttt{drive}(T,\ L_1,\ L_2).$$

Removing "Action Predicates"

$$\texttt{at}(T, L_2) \leftarrow \texttt{at}(T, L_1), \texttt{adj}(L_1, L_2).$$

Reconstructing “Action Predicates” – Iterated Solving

facts:

$$\texttt{adj}(a, b).\ \texttt{adj}(b, a).\ \texttt{adj}(b, c).\ \texttt{adj}(c, b).$$
$$\texttt{at}(\#1, a).\ \texttt{at}(\#1, b).\ \texttt{at}(\#1, c).$$
$$\texttt{at}(\#2, a).\ \texttt{at}(\#2, b).\ \texttt{at}(\#2, c).$$

Reconstructing "Action Predicates" – Iterated Solving

facts:

$$\texttt{adj}(a,b).\ \texttt{adj}(b,a).\ \texttt{adj}(b,c).\ \texttt{adj}(c,b).$$
$$\texttt{at}(\#1,a).\ \texttt{at}(\#1,b).\ \texttt{at}(\#1,c).$$
$$\texttt{at}(\#2,a).\ \texttt{at}(\#2,b).\ \texttt{at}(\#2,c).$$

rules:

$$1\ \{\texttt{first-param}(X) : \texttt{at}(X, L_1)\}\ 1.$$
$$1\ \{\texttt{second-param}(Y) : \texttt{at}(T, Y), \texttt{adj}(Y, L_2)\}\ 1.$$
$$1\ \{\texttt{third-param}(Z) : \texttt{adj}(L_1, Z)\}\ 1.$$
$$\bot \leftarrow \texttt{first-param}(X), \texttt{second-param}(Y), \neg \texttt{at}(X, Y).$$
$$\bot \leftarrow \texttt{second-param}(Y), \texttt{third-param}(Z), \neg \texttt{adj}(Y, Z).$$

Reconstructing "Action Predicates" – Iterated Solving

facts:

$$\texttt{adj}(a,b).\ \texttt{adj}(b,a).\ \texttt{adj}(b,c).\ \texttt{adj}(c,b).$$
$$\texttt{at}(\#1,a).\ \texttt{at}(\#1,b).\ \texttt{at}(\#1,c).$$
$$\texttt{at}(\#2,a).\ \texttt{at}(\#2,b).\ \texttt{at}(\#2,c).$$

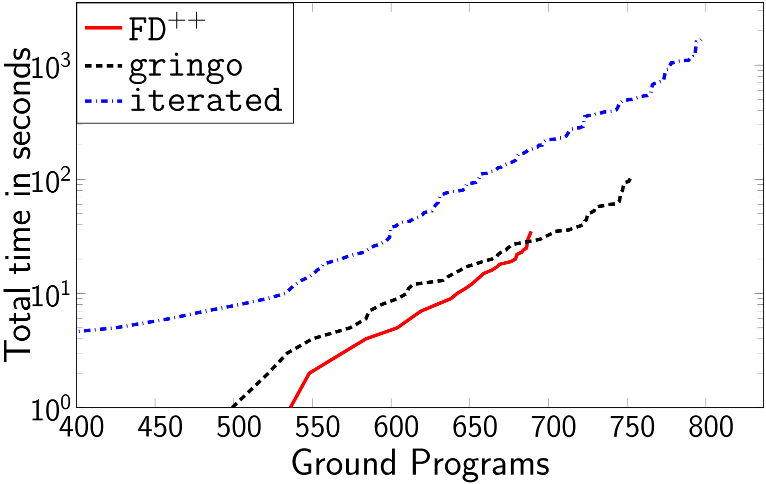
rules:

$$1\ \{\texttt{first-param}(X) : \texttt{at}(X, L_1)\}\ 1.$$
$$1\ \{\texttt{second-param}(Y) : \texttt{at}(T, Y), \texttt{adj}(Y, L_2)\}\ 1.$$
$$1\ \{\texttt{third-param}(Z) : \texttt{adj}(L_1, Z)\}\ 1.$$
$$\bot \leftarrow \texttt{first-param}(X), \texttt{second-param}(Y), \neg\texttt{at}(X, Y).$$
$$\bot \leftarrow \texttt{second-param}(Y), \texttt{third-param}(Z), \neg\texttt{adj}(Y, Z).$$

grounding via iterated solving:

- for each action schema, create an ASP program
- each stable model is an instantiation of the action schema

Results with Iterated Solving



Conclusion

summary:

- grounding planning tasks $\rightarrow$ grounding Datalog
- improved by decoupling action predicates
- better performance than off-the-shelf grounders