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
TU Wien, Institute of Logic and Computation, Austria
CSAIL, MIT, USA

ICAPS 2023

Context: Grounding for Classical Planning

lifted classical planning task

- predicates: $\{\mathit{at}(X), \mathit{connected}(X, Y)\}$
- objects: $\{\text{uni}, \text{bar}\}$
- initial state:
  $\{\mathit{at}(\text{uni}), \mathit{connected}(\text{uni}, \text{bar}), \mathit{connected}(\text{bar}, \text{uni})\}$
- goal: $\{\mathit{at}(\text{bar})\}$
- actions: $\mathit{move}(X, Y)$:
  - $\mathit{pre}(\mathit{move}(X, Y)) = \{\mathit{at}(X), \mathit{connected}(X, Y)\}$
  - $\mathit{eff}(\mathit{move}(X, Y)) = \{\neg\, \mathit{at}(X), \mathit{at}(Y)\}$

Context: Grounding for Classical Planning

lifted classical planning task

- predicates: $\{at(X), connected(X, Y)\}$
- objects: $\{\text{uni}, \text{bar}\}$
- initial state:
  $\{at(\text{uni}), connected(\text{uni}, \text{bar}), connected(\text{bar}, \text{uni})\}$
- goal: $\{at(\text{bar})\}$
- actions: $move(X, Y)$:
  - $pre(move(X, Y)) = \{at(X), connected(X, Y)\}$
  - $eff(move(X, Y)) = \{\neg\, at(X), at(Y)\}$

$\rightsquigarrow$ Most planners translate this to propositional logic.

grounded task

- atoms: *at-uni*, *at-bar*
- actions: *move-uni-bar*, *move-bar-uni*
- ...

Popular Approach to Grounding

idea: compute all relaxed-reachable atoms/actions with Datalog

Example

$$\mathit{at}(\text{uni}).$$
$$\mathit{connected}(\text{uni}, \text{bar}).$$
$$\mathit{connected}(\text{bar}, \text{uni}).$$
$$\mathit{move}(X, Y) \leftarrow \mathit{at}(X), \mathit{connected}(X, Y).$$
$$\mathit{at}(Y) \leftarrow \mathit{move}(X, Y).$$

Popular Approach to Grounding

idea: compute all relaxed-reachable atoms/actions with Datalog

Example

$$\mathit{at}(\mathrm{uni}).$$
$$\mathit{connected}(\mathrm{uni}, \mathrm{bar}).$$
$$\mathit{connected}(\mathrm{bar}, \mathrm{uni}).$$
$$\mathit{move}(X, Y) \leftarrow \mathit{at}(X), \mathit{connected}(X, Y).$$
$$\mathit{at}(Y) \leftarrow \mathit{move}(X, Y).$$

Example (Ground Datalog Rules)

$$\mathit{at}(\mathrm{uni}). \quad \mathit{connected}(\mathrm{uni}, \mathrm{bar}). \quad \mathit{connected}(\mathrm{bar}, \mathrm{uni}).$$
$$\mathit{move}(\mathrm{uni}, \mathrm{bar}) \leftarrow \mathit{at}(\mathrm{uni}), \mathit{connected}(\mathrm{uni}, \mathrm{bar}).$$
$$\mathit{at}(\mathrm{bar}) \leftarrow \mathit{move}(\mathrm{uni}, \mathrm{bar}).$$
$$\mathit{move}(\mathrm{bar}, \mathrm{uni}) \leftarrow \mathit{at}(\mathrm{bar}), \mathit{connected}(\mathrm{bar}, \mathrm{uni}).$$

First Results

Domain	Default Grounding FD^{++}	G
blocksworld (40)	36	**40**
childsnack (144)	**120**	**120**
genome-edit-dist. (312)	**312**	**312**
logistics (40)	**40**	**40**
organic-synthesis (56)	**21**	**21**
pipesworld-tankage (50)	**35**	**35**
rovers (40)	5	**40**
visitall-multidim. (180)	120	**144**
Total (862)	689	**752**

methods:

- FD^{++}: Fast Downward's grounder in C++
- G: gringo (off-the-shelf Datalog solver)

Step 1: Do not ground Actions

problem: not enough memory to ground actions with large arity

Step 1: Do not ground Actions

problem: not enough memory to ground actions with large arity
solution: ignore actions (for now)

original rules

$$move(X, Y) \leftarrow at(X), connected(X, Y).$$
$$at(Y) \leftarrow move(X, Y).$$

Step 1: Do not ground Actions

problem: not enough memory to ground actions with large arity
solution: ignore actions (for now)

original rules

$$\mathit{move}(X, Y) \leftarrow \mathit{at}(X), \mathit{connected}(X, Y).$$
$$\mathit{at}(Y) \leftarrow \mathit{move}(X, Y).$$

without action predicate

$$\mathit{at}(Y) \leftarrow \mathit{at}(X), \mathit{connected}(X, Y).$$

Step 1: Do not ground Actions

problem: not enough memory to ground actions with large arity
solution: ignore actions (for now)

original rules

$$\mathit{move}(X, Y) \leftarrow \mathit{at}(X), \mathit{connected}(X, Y).$$
$$\mathit{at}(Y) \leftarrow \mathit{move}(X, Y).$$

without action predicate

$$\mathit{at}(Y) \leftarrow \mathit{at}(X), \mathit{connected}(X, Y).$$

- can no longer ground actions
- can still ground relaxed-reachable atoms

Results without Action Predicates

Domain	Action Predicates FD^{++}	Action Predicates G	No Action Predicates FD^{++}	No Action Predicates G	No Action Predicates G+L
blocksworld (40)	36	**40**	**40**	**40**	**40**
childsnack (144)	**120**	**120**	**144**	**144**	**144**
genome-edit-dist. (312)	**312**	**312**	**312**	**312**	**312**
logistics (40)	**40**	**40**	**40**	**40**	**40**
organic-synthesis (56)	**21**	**21**	**56**	41	**56**
pipesworld-tankage (50)	**35**	**35**	**40**	**50**	**50**
rovers (40)	5	**40**	**40**	**40**	**40**
visitall-multidim. (180)	120	**144**	120	**180**	**180**
Total (862)	689	**752**	802	847	**862**

methods:

- FD^{++}: Fast Downward's grounder in C++
- G: gringo (off-the-shelf Datalog solver)
- G+L: gringo + preprocess rules with lpopt
  - decomposes rules according to tree decomposition

Step 2: Constructing Ground Actions

We have all reachable atoms but no reachable actions

for each action schema

- reconstruct the original rule
- unify with all reachable atoms

Example

$$\mathit{at}(\mathrm{uni}).$$
$$\mathit{connected}(\mathrm{uni}, \mathrm{bar}).$$
$$\mathit{connected}(\mathrm{bar}, \mathrm{uni}).$$
$$\mathit{at}(\mathrm{bar}).$$
$$\mathit{move}(X, Y) \leftarrow \mathit{at}(X), \mathit{connected}(X, Y).$$

$\rightsquigarrow$ Can handle one action schema at a time
but otherwise same issue as before

Step 3: Iterated Solving

New contribution: Grounding via Iterated Solving

- change from Datalog to more expressive ASP with body-decoupled grounding
- meaning of stable models
  - before: one model for all reachable actions
  - here: each model corresponds to a single ground action
- solver can generate one model at a time
  - low memory overhead

Iterated Solving (Example)

Example (Original Action)

$$\textit{move}(X, Y) \leftarrow \textit{at}(X), \textit{connected}(X, Y).$$

$$1\{\textit{first-param}(X) : \textit{object}(X)\}1.$$

$$1\{\textit{second-param}(Y) : \textit{object}(Y)\}1.$$

$$\bot \leftarrow \text{first-param}(X), \neg \textit{at}(X).$$

$$\bot \leftarrow \text{first-param}(X), \text{second-param}(Y), \neg \textit{connected}(X, Y).$$

Iterated Solving (Example)

Example (Original Action)

$$\textit{move}(X, Y) \leftarrow \textit{at}(X), \textit{connected}(X, Y).$$

Example (ASP for Iterated Solving)

$\textit{at}(\text{uni}).\quad \textit{at}(\text{bar}).\quad \textit{object}(\text{uni}).\quad \textit{object}(\text{bar}).$

$\textit{connected}(\text{uni}, \text{bar}).\quad \textit{connected}(\text{bar}, \text{uni}).$

Iterated Solving (Example)

Example (Original Action)

$$\textit{move}(X, Y) \leftarrow \textit{at}(X), \textit{connected}(X, Y).$$

Example (ASP for Iterated Solving)

$$\textit{at}(\text{uni}). \quad \textit{at}(\text{bar}). \quad \textit{object}(\text{uni}). \quad \textit{object}(\text{bar}).$$

$$\textit{connected}(\text{uni}, \text{bar}). \quad \textit{connected}(\text{bar}, \text{uni}).$$

$$1\{\textit{first-param}(X) : \textit{object}(X)\}1.$$

$$1\{\textit{second-param}(Y) : \textit{object}(Y)\}1.$$

Iterated Solving (Example)

Example (Original Action)

$$move(X, Y) \leftarrow at(X), connected(X, Y).$$

Example (ASP for Iterated Solving)

$$at(\text{uni}). \quad at(\text{bar}). \quad object(\text{uni}). \quad object(\text{bar}).$$

$$connected(\text{uni}, \text{bar}). \quad connected(\text{bar}, \text{uni}).$$

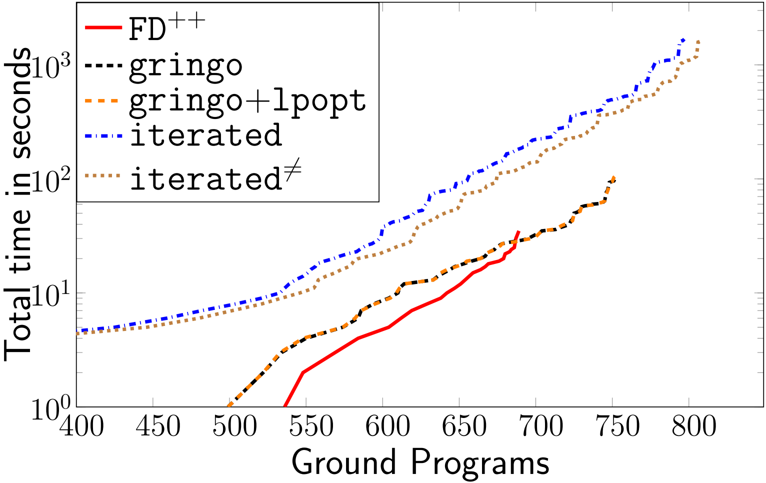
$$1\{first\text{-}param(X) : object(X)\}1.$$

$$1\{second\text{-}param(Y) : object(Y)\}1.$$

$$\bot \leftarrow \text{first-param}(X), \neg at(X).$$

$$\bot \leftarrow \text{first-param}(X), \text{second-param}(Y), \neg connected(X, Y).$$

Results with Iterated Solving



Was it worth it?

Both `gringo` and the iterated approach ground more tasks.
Can we solve these tasks?

- Yes, both variants improve over Fast Downward.
- But the iterated approach performs worse than `gringo`.

We can ground 808 of 862 tasks.
What about the rest?

- Probably out of reach: 10^{10}–10^{34} ground actions

Conclusion

grounding via iterated solving

- ground relaxed-reachable atoms first
- ground action schemas later using ASP guess-and-check

in practice

- grounds more, but is slower
- model counting shows it is close to limit for this benchmark

Extra slides

Can We Solve more Tasks?

Domain	FD	G	iterated$^{\neq}$
blocksworld (40)	**8**	**8**	**8**
childsnack (144)	**103**	**103**	**103**
genome-edit-dist. (312)	**312**	**312**	**312**
logistics (40)	**4**	**4**	**4**
organic-synthesis (56)	18	**27**	23
pipesworld-tankage (50)	**15**	14	14
rovers (40)	4	**40**	20
visitall-multidim. (180)	**84**	79	78
Total (862)	548	**587**	562

Can We Improve?

total number of ground tasks: 808/862

given more time, can we ground more tasks?

Can We Improve?

total number of ground tasks: $808/862$

given more time, can we ground more tasks? probably not!

model counting:

- count number of models without generating them
- instances left are out-of-reach: $> 10^{30}$ actions

Example II

$$
\begin{aligned}
&\textsf{p}(x_1).\\
&\vdots\\
&\textsf{p}(x_{10}).\\
&\textsf{q}(a).\\
&\textsf{t}(V_0) \leftarrow \textsf{q}(V_0), \textsf{p}(V_1), \ldots, \textsf{p}(V_{10}).
\end{aligned}
$$

number of instantiations of the rule: 10^{10}.

Example II

$$\mathit{p}(x_1).$$

$$\vdots$$

$$\mathit{p}(x_{10}).$$

$$\mathit{q}(a).$$

$$\mathit{temp}_1 \leftarrow \mathit{p}(V_1), \mathit{p}(V_2).$$

$$\vdots$$

$$\mathit{temp}_5 \leftarrow \mathit{p}(V_9), \mathit{p}(V_{10}).$$

$$\mathit{t}(V_0) \leftarrow \mathit{q}(V_0), \mathit{temp}_1, \ldots, \mathit{temp}_5.$$

Example II

$$\mathit{p}(x_1).$$

$$\vdots$$

$$\mathit{p}(x_{10}).$$

$$\mathit{q}(a).$$

$$\mathit{temp} \leftarrow \mathit{p}(X), \mathit{p}(Y).$$

$$\mathit{t}(V_0) \leftarrow \mathit{q}(V_0), \mathit{temp}, \ldots, \mathit{temp}.$$

Example II

$$p(x_1).$$
$$\vdots$$
$$p(x_{10}).$$
$$q(a).$$
$$temp \leftarrow p(X), p(Y).$$
$$t(V_0) \leftarrow q(V_0), temp.$$

number of instantiations: $10^2 + 1$.

Example II – Actions...

in planning, actions destroy our idea...

Example II – Actions...

in planning, actions destroy our idea...

$$p(x_1).$$
$$\vdots$$
$$p(x_{10}).$$
$$q(a).$$
$$\mathit{Action}(V_0, V_1, \ldots, V_{10}) \leftarrow q(V_0), p(V_1), \ldots, p(V_{10}).$$
$$t(V_0) \leftarrow \mathit{Action}(V_0, V_1, \ldots, V_{10}).$$

each possible ground action unifies the rule once
no good way to decompose the rules!

Example II – Actions...

$$\mathit{Action}(V_0, V_1, \ldots, V_{10}) \leftarrow q(V_0), p(V_1), \ldots, p(V_{10}).$$
$$t(V_0) \leftarrow \mathit{Action}(V_0, V_1, \ldots, V_{10}).$$

Example II – Actions...

$$\mathit{Action}(V_0, V_1, \ldots, V_{10}) \leftarrow q(V_0), p(V_1), \ldots, p(V_{10}).$$
$$t(V_0) \leftarrow \mathit{Action}(V_0, V_1, \ldots, V_{10}).$$

$$t(V_0) \leftarrow q(V_0), p(V_1), \ldots, p(V_{10}).$$

A Detour in ASP

solution: ASP (answer set programming) choice rule:

$$\begin{aligned} &p(x_1).\\ &\vdots\\ &p(x_{10}).\\ &\{t(X) : p(X)\}. \end{aligned}$$

A Detour in ASP

choice rule: pick one element in the set

$$p(x_1).$$

$$\vdots$$

$$p(x_{10}).$$

$$\{t(x_1), \ldots, t(x_{100})\}.$$

Going back to planning

iterated grounding via solving:

- transform each action rule into an ASP program
- each stable model is an action (and vice-versa)
- you can guess-and-check

$$\mathit{Action}(V_0, V_1, V_2) \leftarrow p(V_0, V_1), p(V_1, V_2), p(V_2, V_0),$$

$$
\begin{aligned}
\mathit{Action}(V_0, V_1, V_2) \leftarrow\, & \mathit{p}(V_0, V_1), \mathit{p}(V_1, V_2), \mathit{p}(V_2, V_0), \\
& \mathit{type}\text{-}T_0(V_0), \mathit{type}\text{-}T_1(V_1), \mathit{type}\text{-}T_2(V_2).
\end{aligned}
$$

$$\mathit{Action}(V_0, V_1, V_2) \leftarrow \mathit{p}(V_0, V_1), \mathit{p}(V_1, V_2), \mathit{p}(V_2, V_0),$$
$$\mathit{type}\text{-}T_0(V_0), \mathit{type}\text{-}T_1(V_1), \mathit{type}\text{-}T_2(V_2).$$

$$\{V_0\text{-}\mathit{assign}(X) : \mathit{type}\text{-}T_0(X)\}.$$
$$\{V_1\text{-}\mathit{assign}(Y) : \mathit{type}\text{-}T_1(Y)\}.$$
$$\{V_2\text{-}\mathit{assign}(Z) : \mathit{type}\text{-}T_2(Z)\}.$$
$$\bot \leftarrow V_0\text{-assign}(X), V_1\text{-assign}(Y), \neg \mathit{p}(X, Y).$$
$$\bot \leftarrow V_1\text{-assign}(Y), V_2\text{-assign}(Z), \neg \mathit{p}(Y, Z).$$
$$\bot \leftarrow V_2\text{-assign}(Z), V_0\text{-assign}(X), \neg \mathit{p}(Z, X).$$

$\textsf{p}(a, b).$

$\textsf{p}(b, c).$

$\textsf{p}(c, a).$

$\textit{type-}T_0(a).$ $\textit{type-}T_0(a').$

$\textit{type-}T_1(b).$ $\textit{type-}T_1(b').$

$\textit{type-}T_2(c).$ $\textit{type-}T_2(c').$

$$\mathsf{p}(a,b).$$
$$\mathsf{p}(b,c).$$
$$\mathsf{p}(c,a).$$
$$\textit{type-}T_0(a). \qquad \textit{type-}T_0(a').$$
$$\textit{type-}T_1(b). \qquad \textit{type-}T_1(b').$$
$$\textit{type-}T_2(c). \qquad \textit{type-}T_2(c').$$
$$\{V_0\textit{-assign}(a), V_0\textit{-assign}(a')\}.$$
$$\{V_1\textit{-assign}(b), V_1\textit{-assign}(b')\}.$$
$$\{V_2\textit{-assign}(c), V_2\textit{-assign}(c')\}.$$
$$\bot \leftarrow V_0\text{-assign}(X), V_1\text{-assign}(Y), \neg \mathsf{p}(X,Y).$$
$$\bot \leftarrow V_1\text{-assign}(Y), V_2\text{-assign}(Z), \neg \mathsf{p}(Y,Z).$$
$$\bot \leftarrow V_2\text{-assign}(Z), V_0\text{-assign}(X), \neg \mathsf{p}(Z,X).$$

$$\mathsf{p}(a, b).$$
$$\mathsf{p}(b, c).$$
$$\mathsf{p}(c, a).$$
$$\mathit{type}\text{-}T_0(a). \qquad \mathit{type}\text{-}T_0(a').$$
$$\mathit{type}\text{-}T_1(b). \qquad \mathit{type}\text{-}T_1(b').$$
$$\mathit{type}\text{-}T_2(c). \qquad \mathit{type}\text{-}T_2(c').$$
$$\{V_0\text{-}\mathit{assign}(a), V_0\text{-}\mathit{assign}(a')\}.$$
$$\{V_1\text{-}\mathit{assign}(b), V_1\text{-}\mathit{assign}(b')\}.$$
$$\{V_2\text{-}\mathit{assign}(c), V_2\text{-}\mathit{assign}(c')\}.$$
$$\bot \leftarrow V_0\text{-assign}(a), V_1\text{-assign}(b), \neg \mathsf{p}(a, b).$$
$$\bot \leftarrow V_1\text{-assign}(b), V_2\text{-assign}(c), \neg \mathsf{p}(b, c).$$
$$\bot \leftarrow V_2\text{-assign}(c), V_0\text{-assign}(a), \neg \mathsf{p}(c, a).$$

$$p(a, b).$$
$$p(b, c).$$
$$p(c, a).$$
$$\textit{type-}T_0(a). \qquad \textit{type-}T_0(a').$$
$$\textit{type-}T_1(b). \qquad \textit{type-}T_1(b').$$
$$\textit{type-}T_2(c). \qquad \textit{type-}T_2(c').$$
$$\{V_0\textit{-assign}(a), V_0\textit{-assign}(a')\}.$$
$$\{V_1\textit{-assign}(b), V_1\textit{-assign}(b')\}.$$
$$\{V_2\textit{-assign}(c), V_2\textit{-assign}(c')\}.$$
$$\bot \leftarrow V_0\text{-assign}(a), V_1\text{-assign}(b), \neg p(a, b).$$
$$\bot \leftarrow V_1\text{-assign}(b), V_2\text{-assign}(c'), \neg p(b, c').$$
$$\bot \leftarrow V_2\text{-assign}(c'), V_0\text{-assign}(a), \neg p(c', a).$$

Advantages

why is this better?

body-decoupled grounding

- check each precondition locally
- avoid combinatorial blow-up

drawback: overhead