Translating Totally Ordered HTN Planning Problems to Classical Planning Problems Using Regular Approximation of Context-Free Languages

Daniel Höller

Saarland University, Saarland Informatics Campus, Saarbrücken, Germany

hoeller@cs.uni-saarland.de

Motivation

- Many sophisticated techniques available for classical planning
- Interesting for related problems like HTN planning
  - Direct application in HTN planning (RC heuristics)
  - Adapted techniques (reachability, SAT, IP/LP heuristics)
  - Translation (Alford et al. 2016)
- We present a novel translation technique
  - HTN planning more expressive than classical planning
  - Alford et al. (2016) bound the problem
  - We use an approximation instead

Approach

- We focus on Totally Ordered (TO) HTN Planning
- → Decomposition methods resemble a context-free grammar
- Still more expressive than classical planning
- → Instead of bounding, we over-approximate set of solutions
- → We verify generated solutions to only return valid ones
- We build on existing techniques introduced to approximate context-free languages by finite automata

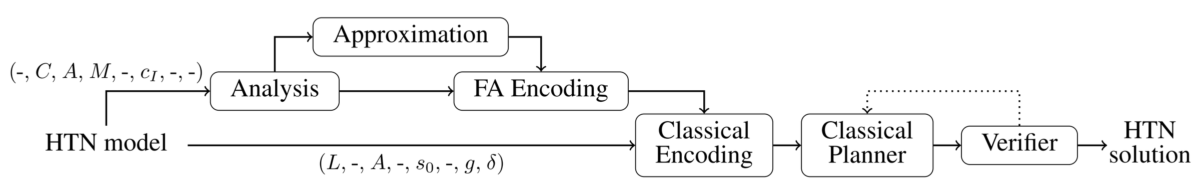


Figure 1: *Overview of the Approach*

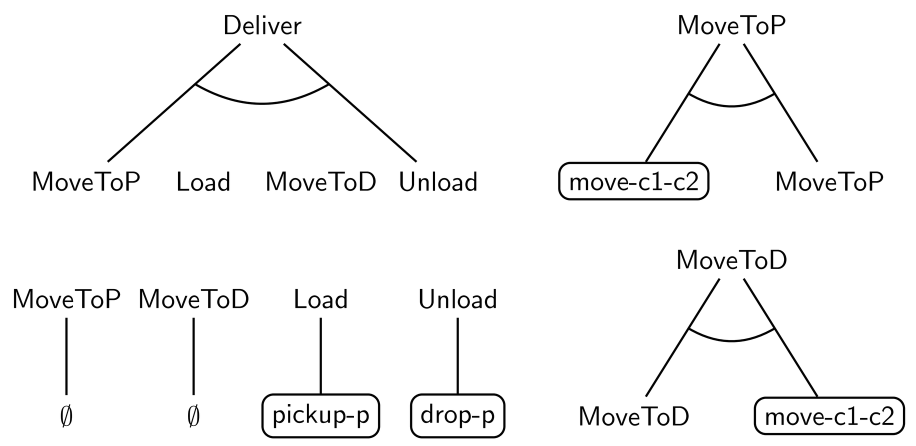
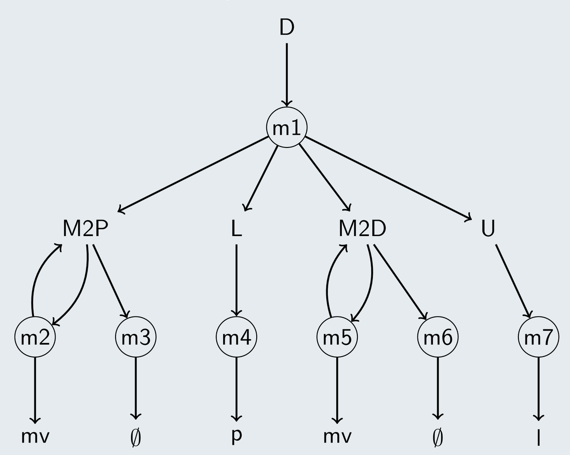


Figure 2: *Transport Domain*

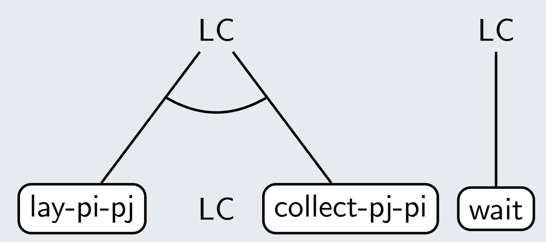
Analysis – Non-Self-Embedding Model

- We test if the instance is self-embedding. If it is not, this is a sufficient criterion to proof that a context-free grammar describes a regular language



- {M2D, M2D mv, M2D mv mv, M2D mv mv mv, . . . }
- {M2P, mv M2P, mv mv M2P, mv mv mv M2P, . . . }

Analysis – Self-Embedding Model



- {LC, lay-pi-pj LC collect-pj-pi, lay-pi-pj lay-pk-pl LC collect-pl-pk collect-pj-pi, . . . }

$$
\begin{array}{l}
\textbf{procedure } \texttt{make\_fa}\,(q_0, \alpha, q_1) \\
\quad \textbf{if } \alpha = \varepsilon \textbf{ then } \Delta = \Delta \cup \{(q_0, \varepsilon, q_1)\} \\
\quad \textbf{else if } \alpha = a, a \in A \textbf{ then } \Delta = \Delta \cup \{(q_0, a, q_1)\} \\
\quad \textbf{else if } \alpha = x\beta, x \in \mathcal{T}, \beta \in \mathcal{T}^*, |\beta| > 0 \textbf{ then} \\
\quad\quad q = \textit{fresh\_state} \\
\quad\quad \texttt{make\_fa}\,(q_0, x, q) \\
\quad\quad \texttt{make\_fa}\,(q, \beta, q_1) \\
\quad \textbf{else} \\
\quad\quad c_a = \alpha \qquad \texttt{/* } \alpha \texttt{ is abstr. task */} \\
\quad\quad \textbf{if } \exists i : c_a \in N_i \textbf{ then} \\
\quad\quad\quad \textbf{for } c_b \in N_i \textbf{ do } q_{c_b} = \textit{fresh\_state} \\
\quad\quad\quad \textbf{if } \textit{recursive}(N_i) = \textit{left} \textbf{ then} \\
\quad\quad\quad\quad \textbf{for } (c_c, x_1 \dots x_m) \in M \textit{ s.t. } c_c \in N_i \\
\quad\quad\quad\quad\quad \wedge\, x_1, \dots, x_m \notin N_i \textbf{ do} \\
\quad\quad\quad\quad\quad \texttt{make\_fa}\,(q_0, x_1 \dots x_m, q_{c_c}) \\
\quad\quad\quad\quad \textbf{for } (c_c, c_d x_1 \dots x_m) \in M \textit{ s.t.} \\
\quad\quad\quad\quad\quad c_c, c_d \in N_i \wedge x_1, \dots, x_m \notin N_i \textbf{ do} \\
\quad\quad\quad\quad\quad \texttt{make\_fa}\,(q_{c_d}, x_1 \dots x_m, q_{c_c}) \\
\quad\quad\quad\quad \Delta = \Delta \cup \{(q_{c_a}, \varepsilon, q_1)\} \\
\quad\quad\quad \textbf{else} \\
\quad\quad\quad\quad \textbf{for } (c_c, x_1 \dots x_m) \in M \textit{ s.t.} \\
\quad\quad\quad\quad\quad c_c \in N_i \wedge x_1, \dots, x_m \notin N_i \textbf{ do} \\
\quad\quad\quad\quad\quad \texttt{make\_fa}\,(q_{c_c}, x_1 \dots x_m, q_1) \\
\quad\quad\quad\quad \textbf{for } (c_c, x_1 \dots x_m c_d) \in M \textit{ s.t.} \\
\quad\quad\quad\quad\quad c_c, c_d \in N_i \wedge x_1, \dots, x_m \notin N_i \textbf{ do} \\
\quad\quad\quad\quad\quad \texttt{make\_fa}\,(q_{c_c}, x_1 \dots x_m, q_{c_d}) \\
\quad\quad\quad\quad \Delta = \Delta \cup \{(q_0, \varepsilon, q_{c_a})\} \\
\quad\quad \textbf{else} \\
\quad\quad\quad \textbf{for } (c_a, \beta) \in M \textbf{ do} \\
\quad\quad\quad\quad \texttt{make\_fa}\,(q_0, \beta, q_1)
\end{array}
$$

Figure 3: *Algorithm by Nederhof (2000) to create FA from grammar*

FA Encoding

- We use an algorithm by Nederhof (2000)
- Algorithm goes down the hierarchy
- Collects all sequences of actions that can be generated
- Transitions in resulting FA are labeled with actions

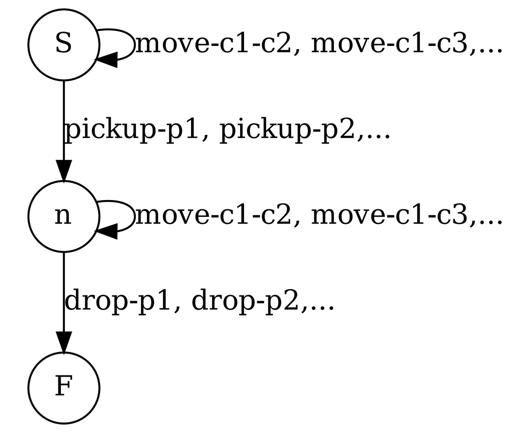


Figure 4: *FA encoding of the transport example*

1. Add new non-terminals $a_b^{\uparrow}\ a_b^{\downarrow}\ a_b^{\leftarrow}\ a_b^{\rightarrow}$ with $a, b \in N_i$
2. Add the following methods with $a, b, c, d, e \in N_i$

$$(a, a_b^{\uparrow}) \tag{1}$$
$$(a_b^{\uparrow}, a_c^{\leftarrow} x_1 \dots x_m c_b^{\downarrow}), \forall (c, x_1 \dots x_m) \in M \tag{2}$$
$$(a_b^{\downarrow}, c_a^{\rightarrow} x_1 \dots x_m e_b^{\uparrow}), \forall (d, \alpha c x_1 \dots x_m e \beta) \in M \tag{3}$$
$$(a_b^{\downarrow}, b_a^{\rightarrow}) \tag{4}$$
$$(a_b^{\leftarrow}, x_1 \dots x_m c_b^{\leftarrow}), \forall (a, x_1 \dots x_m c \beta) \in M \tag{5}$$
$$(a_a^{\leftarrow}, \varepsilon) \tag{6}$$
$$(a_b^{\rightarrow}, c_b^{\rightarrow} x_1 \dots x_m), \forall (a, \alpha c x_1 \dots x_m) \in M \tag{7}$$
$$(a_a^{\rightarrow}, \varepsilon) \tag{8}$$

3. Remove (a, α) from M

Figure 5: *Definition of approximation by Nederhof (2000)*

Approximation

- We use an approach by Nederhof (2000) to compile a self-embedding grammar into a non-self-embedding grammar
- It is an over-approximation, i.e., the language described by the resulting grammar is a superset of the original one
- The compilation decouples the parts generated to the left from the part generated to the right
- → After the compilation, we can use the same FA encoding as before

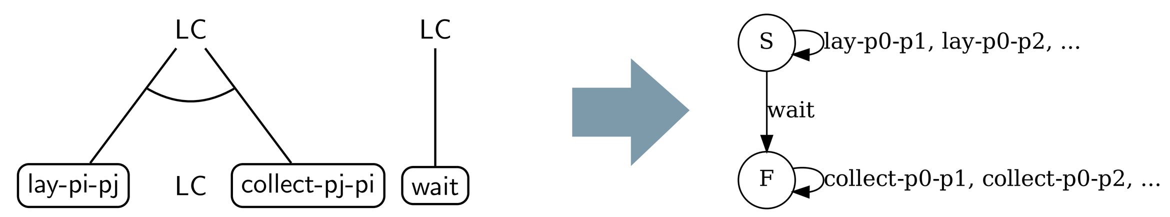


Figure 6: *Result of approximation and FA encoding*

Discussion

- We show that the approach is sound and complete
- However, in the current implementation, only a single solution is generated and verified
- When it is not a valid solution, the instance is treated as unsolved
- → Implementation is incomplete
- → We need a planning system that generates more than one solution, and eventually returns *every* solution (problem with FD: graph search)

domain	#inst	¬rec	¬self embedding			s.e.	ukn
			l rec	r rec	both		
Assembly Hierarchical	30	-	-	30	-	-	-
Barman BDI	20	20	-	-	-	-	-
Blocksworld (GTOHP)	30	1	-	-	-	27	2
Blocksworld (HPDDL)	30	-	-	30	-	-	-
Childsnack	30	26	-	-	-	-	4
Depots	30	20	-	-	-	10	-
Elevator (L)	147	-	-	147	-	-	-
Entertainment	12	5	4	-	3	-	-
Factories	20	-	-	20	-	-	-
Freecell (L)	60	-	-	-	-	60	-
Hiking	30	-	-	-	26	-	4
Logistics (L)	80	-	-	80	-	-	-
Minecraft Player	20	-	-	4	-	-	16
Minecraft Regular	59	44	-	-	-	-	15
Monroe (FO)	20	-	-	-	-	20	-
Monroe (PO)	20	-	-	-	-	20	-
Multiarm Blocksworld	74	-	-	74	-	-	-
Robot	20	-	-	20	-	-	-
Rover (GTOHP)	30	2	-	-	-	28	-
Satellite (GTOHP)	20	-	-	-	-	20	-
Snake	20	-	-	20	-	-	-
Towers	20	-	-	20	-	-	-
Transport	40	-	40	-	-	-	-
Woodworking	30	30	-	-	-	-	-
	892	148	44	445	29	185	41

Figure 7: *Empirical Evaluation – Domain Properties*

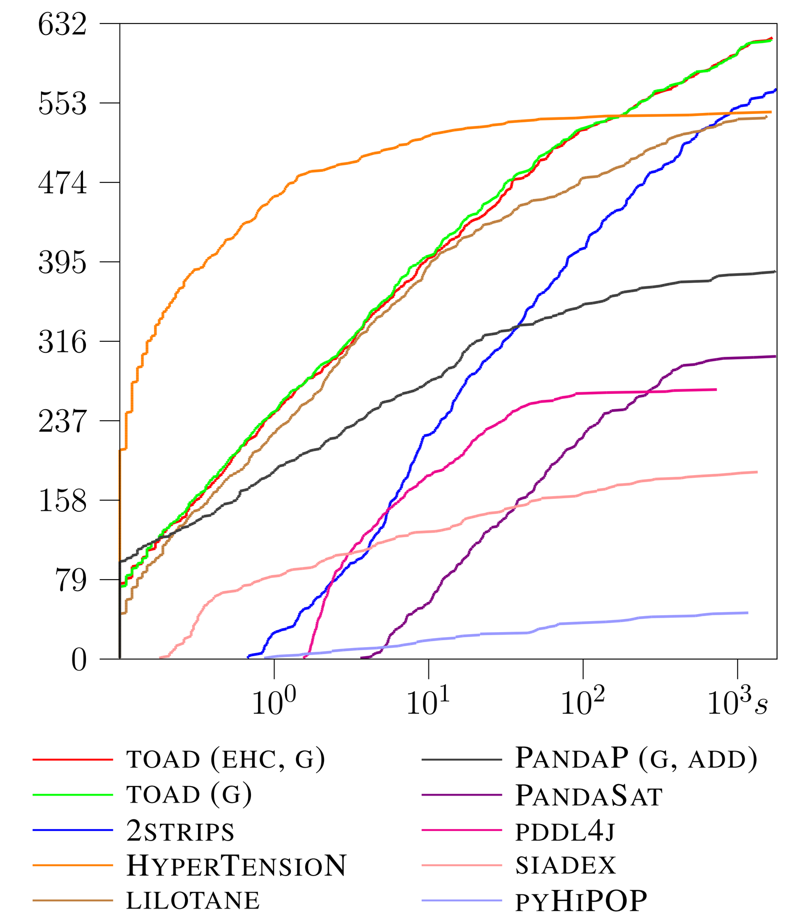


Figure 8: *Empirical Evaluation – Runtime*

Conclusion

- We have realized a novel, compilation-based planning system for TO HTN planning
- Instead of bounded the input problem, the set of solutions is over-approximated
- We exploited techniques from the literature originally introduced
  - to encode a CFG as a FA and
  - to compile a (potentially) non-regular grammar to a regular one
- We have shown that the overall approach is sound and complete (while our implementation currently is not complete)
- In combination with FD, the resulting overall system outperforms the planners from the IPC in terms of coverage

UNIVERSITÄT DES SAARLANDES