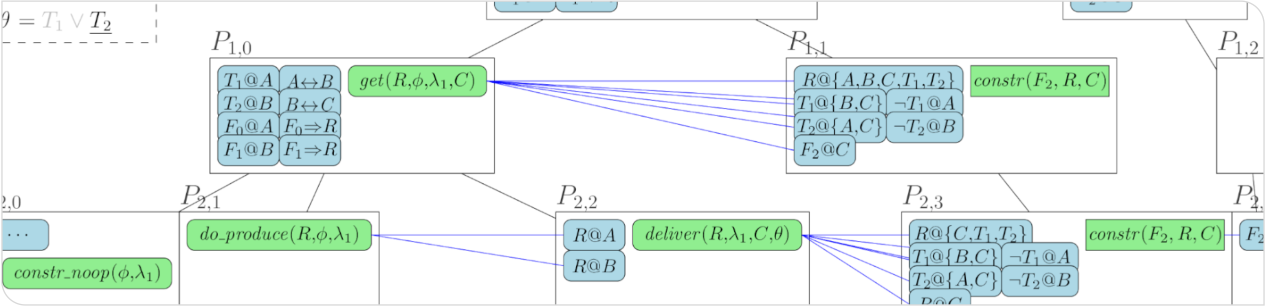
Lilotane: A Lifted SAT-based Approach to Hierarchical Planning

IJCAI 2021 | Presentation of JAIR Article

Dominik Schreiber | July 9, 2021



Total-Order Hierarchical Task Network Planning

Objective

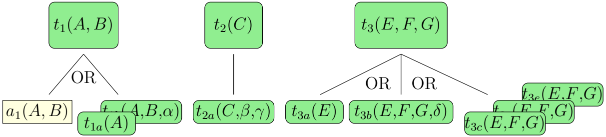
- Achieve a given set of tasks . . .

$t_1(A, B)$ $t_2(C)$ $t_3(E, F, G)$

Total-Order Hierarchical Task Network Planning

Objective

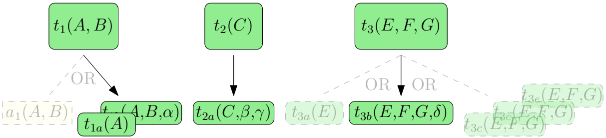
- Achieve a given set of tasks . . .
- by recursively replacing each task with a specific set of subtasks . . .



Total-Order Hierarchical Task Network Planning

Objective

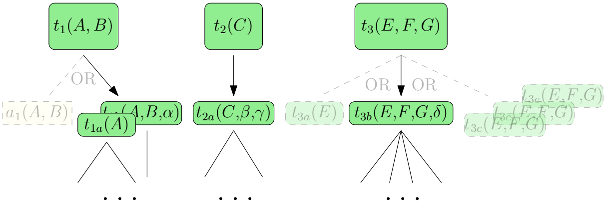
- Achieve a given set of tasks . . .
- by recursively replacing each task with a specific set of subtasks . . .



Total-Order Hierarchical Task Network Planning

Objective

- Achieve a given set of tasks . . .
- by recursively replacing each task with a specific set of subtasks . . .

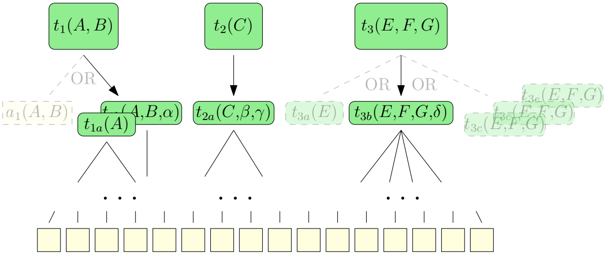


Total-Order Hierarchical Task Network Planning

Karlsruher Institut für Technologie

Objective

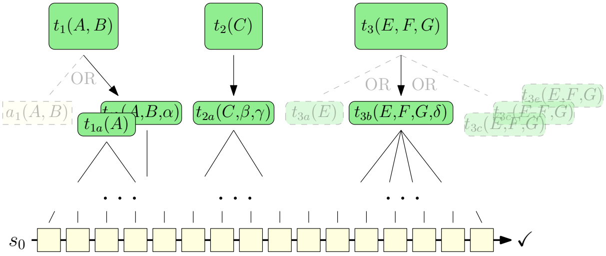
- Achieve a given set of tasks . . .
- by recursively replacing each task with a specific set of subtasks . . .
- until only "atomic" primitive tasks remain . . .



Total-Order Hierarchical Task Network Planning

Objective

- Achieve a given set of tasks . . .
- by recursively replacing each task with a specific set of subtasks . . .
- until only “atomic” primitive tasks remain . . .
- which form a plan, a sequence of executable actions from the given initial state

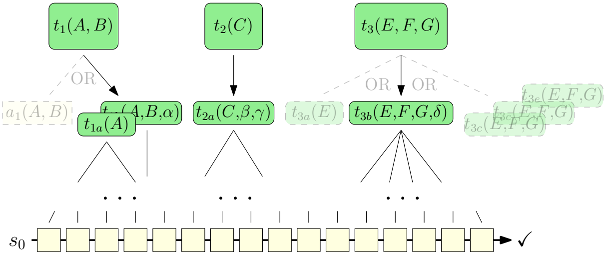


Total-Order Hierarchical Task Network Planning

Karlsruher Institut für Technologie

Objective

- Achieve a given set of tasks . . .
- by recursively replacing each task with a specific set of subtasks . . .
- until only "atomic" primitive tasks remain . . .
- which form a plan, a sequence of executable actions from the given initial state



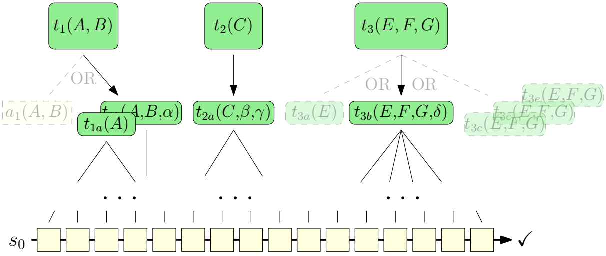
Structures

- Fact: Boolean feature of world state

Total-Order Hierarchical Task Network Planning

Objective

- Achieve a given set of tasks . . .
- by recursively replacing each task with a specific set of subtasks . . .
- until only “atomic” primitive tasks remain . . .
- which form a plan, a sequence of executable actions from the given initial state



Structures

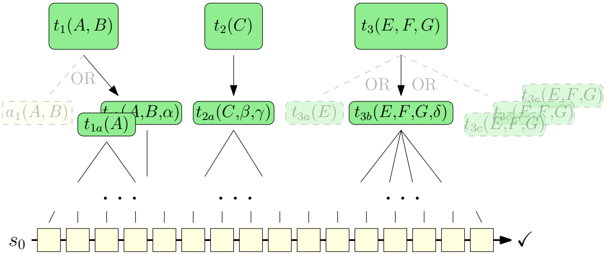
- Fact: Boolean feature of world state
- Task: Footprint of sth. to achieve

Total-Order Hierarchical Task Network Planning

Karlsruher Institut für Technologie

Objective

- Achieve a given set of tasks . . .
- by recursively replacing each task with a specific set of subtasks . . .
- until only "atomic" primitive tasks remain . . .
- which form a plan, a sequence of executable actions from the given initial state

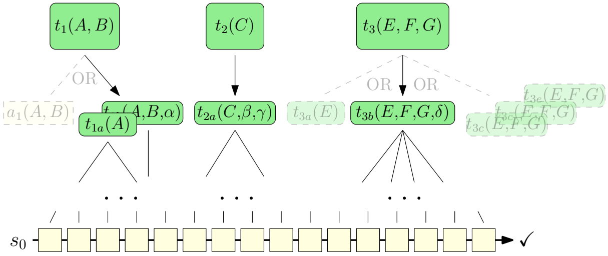


Structures

- Fact: Boolean feature of world state
- Task: Footprint of sth. to achieve
- Method: Recipe to achieve certain compound task ⇒ preconditions, subtasks

Total-Order Hierarchical Task Network Planning

Karlsruher Institut für Technologie



Objective

- Achieve a given set of tasks . . .
- by recursively replacing each task with a specific set of subtasks . . .
- until only “atomic” primitive tasks remain . . .
- which form a plan, a sequence of executable actions from the given initial state

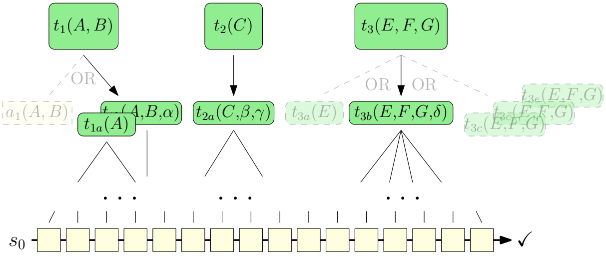
Structures

- Fact: Boolean feature of world state
- Task: Footprint of sth. to achieve
- Method: Recipe to achieve certain compound task ⇒ preconditions, subtasks
- Operator: Recipe to execute primitive task ⇒ preconditions, effects

Total-Order Hierarchical Task Network Planning

Objective

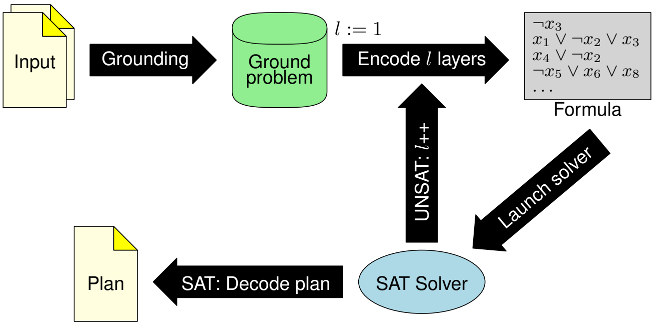
- Achieve a given set of tasks . . .
- by recursively replacing each task with a specific set of subtasks . . .
- until only "atomic" primitive tasks remain . . .
- which form a plan, a sequence of executable actions from the given initial state



Structures

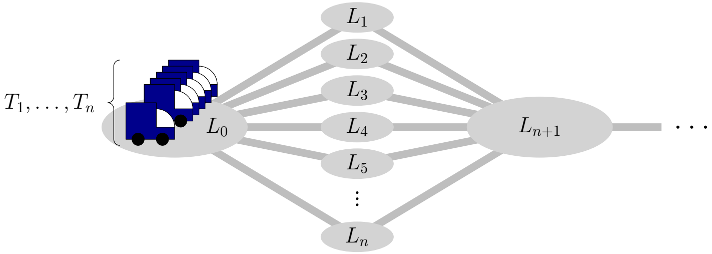
- Fact: Boolean feature of world state
- Task: Footprint of sth. to achieve
- Method: Recipe to achieve certain compound task ⇒ preconditions, subtasks
- Operator: Recipe to execute primitive task ⇒ preconditions, effects
- Action (Reduction): *ground* operator (method) – no free variables

SAT-based TOHTN Planning



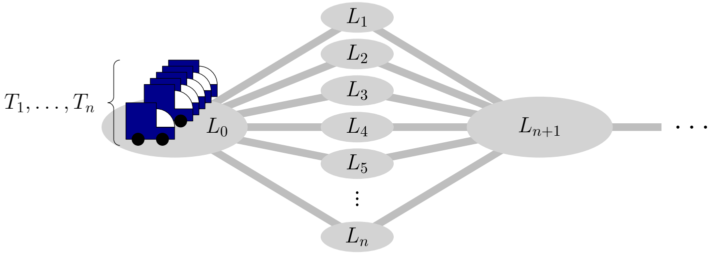
Behnke et al. (AAAI'18) with separate formulae & solver instances,
Schreiber et al. (ICAPS'19) with incremental SAT solving

The Problem with Grounding



$drive(T_1,L_0,L_1)$	$drive(T_1,L_0,L_2)$	$drive(T_1,L_0,L_3)$	$\cdots$	$drive(T_n,L_0,L_n)$
$drive(T_2,L_0,L_1)$	$drive(T_2,L_0,L_2)$	$drive(T_2,L_0,L_3)$		
$drive(T_3,L_0,L_1)$	$drive(T_3,L_0,L_2)$			
$drive(T_4,L_0,L_1)$		$\ddots$		$\vdots$
$drive(T_5,L_0,L_1)$				
$\vdots$				
$drive(T_n,L_0,L_1)$		$\cdots$		$drive(T_n,L_0,L_n)$

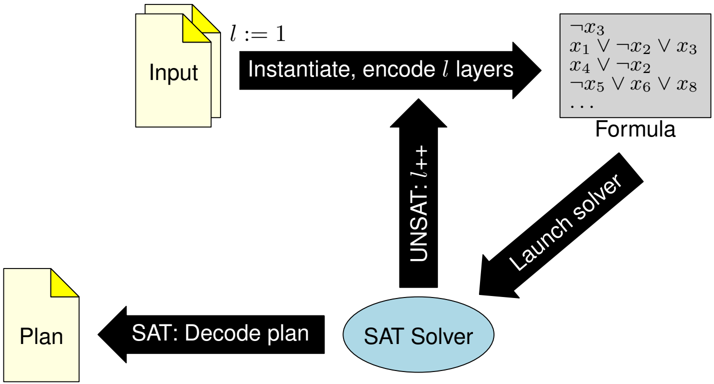
The Problem with Grounding



$drive(T_1,L_0,L_1)$ $drive(T_1,L_0,L_2)$ $drive(T_1,L_0,L_3)$ $\cdots$ $drive(T_n,L_0,L_n)$
$drive(T_2,L_0,L_1)$ $drive(T_2,L_0,L_2)$ $drive(T_2,L_0,L_3)$
$drive(T_3,L_0,L_1)$ $drive(T_3,L_0,L_2)$
$drive(T_4,L_0,L_1)$ $\ddots$ $\vdots$
$drive(T_5,L_0,L_1)$
$\vdots$
$drive(T_n,L_0,L_1)$ $\cdots$ $drive(T_n,L_0,L_n)$

$\Rightarrow$ Combinatorial blowup!

Our Approach

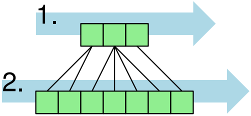


Instantiation

- Layer 0: Instantiate operations matching initial tasks

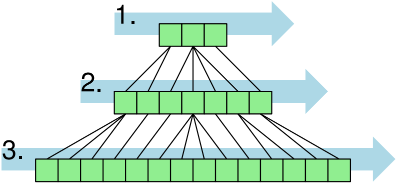
Instantiation

- Layer 0: Instantiate operations matching initial tasks
- Layer l: Instantiate possible children of op. at layer $l - 1$



Instantiation

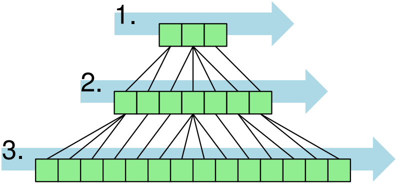
- Layer 0: Instantiate operations matching initial tasks
- Layer l: Instantiate possible children of op. at layer $l - 1$
- Keep variables, do not ground into separate operations
- Layer fully instantiated: Encode, attempt to solve



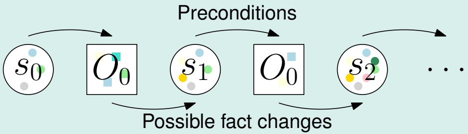
Instantiation

Karlsruher Institut für Technologie

- Layer 0: Instantiate operations matching initial tasks
- Layer l: Instantiate possible children of op. at layer $l-1$
- Keep variables, do not ground into separate operations
- Layer fully instantiated: Encode, attempt to solve

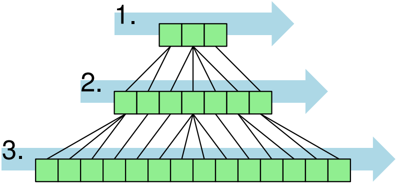


Reachability Analysis

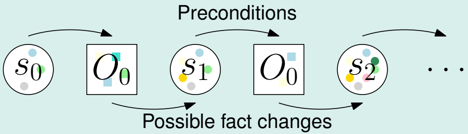


Instantiation

- Layer 0: Instantiate operations matching initial tasks
- Layer l: Instantiate possible children of op. at layer $l-1$
- Keep variables, do not ground into separate operations
- Layer fully instantiated: Encode, attempt to solve



Reachability Analysis



$\Rightarrow$ (Over-)approximate fact changes of instantiated operations, add to possible facts

$\Rightarrow$ Discard operations with impossible preconditions

SAT Encoding

	Tree-REX (ICAPS'19)	Lilotane (ours)
Operation variables (per position)	$drive(T_1, L_0, L_1)$, $drive(T_1, L_0, L_2)$, $drive(T_2, L_0, L_1)$, $drive(T_2, L_0, L_2)$, $\dots$	$drive(\alpha, \beta, \gamma)$

SAT Encoding

	Tree-REX (ICAPS'19)	Lilotane (ours)
Operation variables (per position)	$\mathit{drive}(T_1, L_0, L_1)$, $\mathit{drive}(T_1, L_0, L_2)$, $\mathit{drive}(T_2, L_0, L_1)$, $\mathit{drive}(T_2, L_0, L_2)$, $\ldots$	$\mathit{drive}(\alpha, \beta, \gamma)$
Fact variables (per position)	$\mathit{at}(T_1, L_0), \ldots, \mathit{road}(L_0, L_1), \ldots$	$\mathit{at}(T_1, L_0), \ldots, \mathit{road}(L_0, L_1), \ldots$ $\mathit{at}(\alpha, \beta)$, $\mathit{at}(\alpha, \gamma)$, $\mathit{road}(\beta, \gamma)$

SAT Encoding

	Tree-REX (ICAPS'19)	Lilotane (ours)
Operation variables (per position)	$drive(T_1, L_0, L_1)$, $drive(T_1, L_0, L_2)$, $drive(T_2, L_0, L_1)$, $drive(T_2, L_0, L_2)$, ...	$drive(\alpha, \beta, \gamma)$
Fact variables (per position)	$at(T_1, L_0)$, ..., $road(L_0, L_1)$, ...	$at(T_1, L_0)$, ..., $road(L_0, L_1)$, ... $at(\alpha, \beta)$, $at(\alpha, \gamma)$, $road(\beta, \gamma)$
Substitution variables (only once)	—	$[\alpha/T_1]$, $[\alpha/T_2]$, ... $[\beta/L_0]$, $[\beta/L_1]$, ...

SAT Encoding

	Tree-REX (ICAPS'19)	Lilotane (ours)
Operation variables (per position)	$drive(T_1, L_0, L_1)$, $drive(T_1, L_0, L_2)$, $drive(T_2, L_0, L_1)$, $drive(T_2, L_0, L_2)$, ...	$drive(\alpha, \beta, \gamma)$
Fact variables (per position)	$at(T_1, L_0)$, ..., $road(L_0, L_1)$, ...	$at(T_1, L_0)$, ..., $road(L_0, L_1)$, ... $at(\alpha, \beta)$, $at(\alpha, \gamma)$, $road(\beta, \gamma)$
Substitution variables (only once)	—	$[\alpha/T_1]$, $[\alpha/T_2]$, ... $[\beta/L_0]$, $[\beta/L_1]$, ...
Selected clause schemes	$drive(T_1, L_0, L_1) \Rightarrow at(T_1, L_0)$	$drive(\alpha, \beta, \gamma) \Rightarrow at(\alpha, \beta)$ $exactly\text{-}one([\alpha/T_1], [\alpha/T_2], \dots, [\alpha/T_n])$ $[\alpha/T_1] \wedge [\beta/L_0] \Rightarrow (at(\alpha, \beta) \Leftrightarrow at(T_1, L_0))$

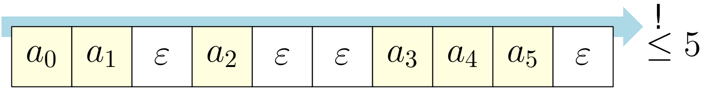
Anytime Plan Improvement

Karlsruher Institut für Technologie

a_0	a_1	ε	a_2	ε	ε	a_3	a_4	a_5	ε

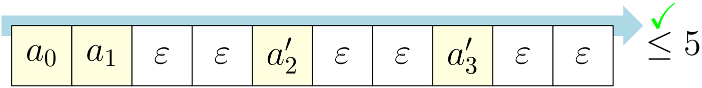
- Maximize number of ε-actions (no-ops): Successively forbid current plan length (ICAPS'19)
- Leads to shortest possible plan at current layer (not globally optimal!)

Anytime Plan Improvement



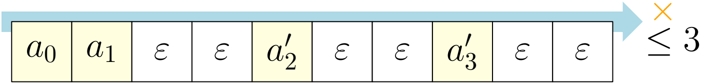
- Maximize number of ε-actions (no-ops): Successively forbid current plan length (ICAPS'19)
- Leads to shortest possible plan at current layer (not globally optimal!)

Anytime Plan Improvement



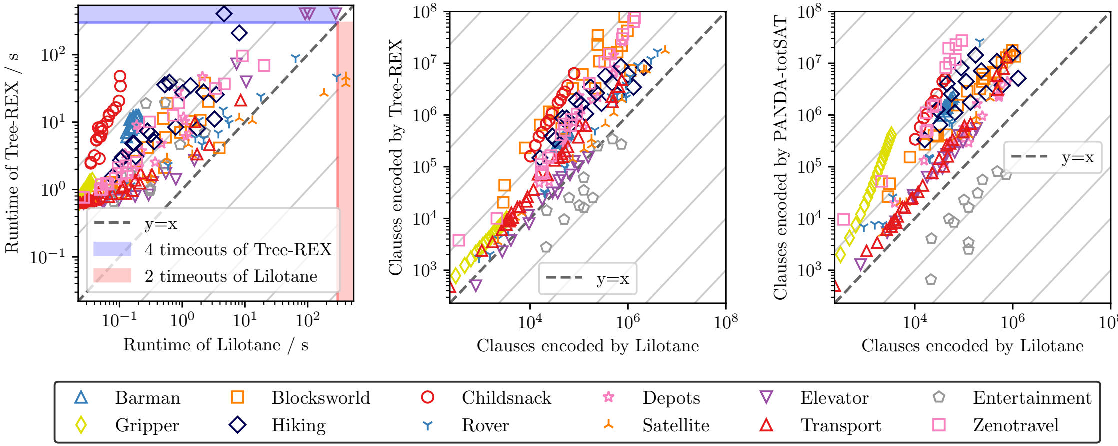
- Maximize number of ε-actions (no-ops): Successively forbid current plan length (ICAPS'19)
- Leads to shortest possible plan at current layer (not globally optimal!)

Anytime Plan Improvement

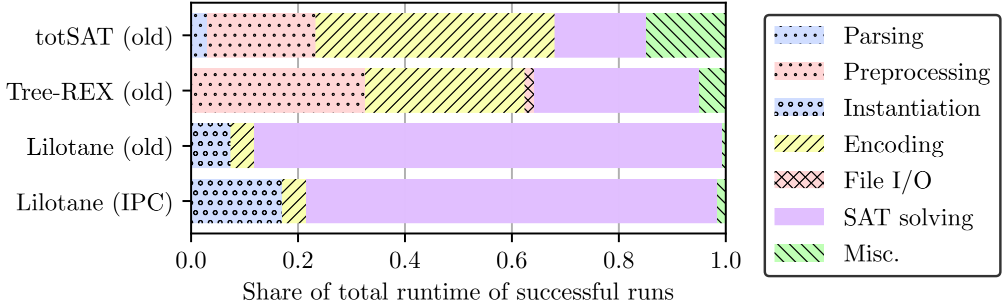


- Maximize number of ε-actions (no-ops): Successively forbid current plan length (ICAPS'19)
- Leads to shortest possible plan at current layer (not globally optimal!)
- Improved encoding exploiting incremental SAT
- Anytime procedure: Cancellable at any time, outputs best plan found

Comparing SAT-based Planners

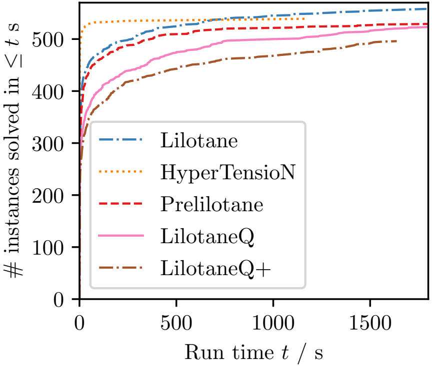


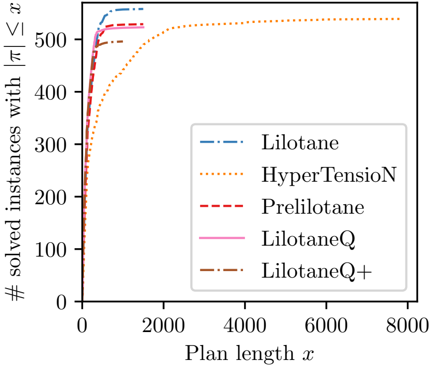
Tree-REX, totSAT, Lilotane: Share of Runtimes



(old): pre-IPC benchmark set (Behnke ’18; Schreiber ’19) **(IPC)**: large IPC benchmark set

Evaluation on IPC Benchmarks





Discussion & Conclusion

- Lilotane: Lifted Logic for Task Networks
- Much more compact formulae, faster planning in most domains
- High-quality plans, even without plan improvement
- Novel encoding techniques may apply to other fields

https://github.com/domschrei/lilotane

Our Encoding

Karlsruher Institut für Technologie

Boolean variables

- f_x^l: "At the l-th layer, fact f holds before the x-th operation" — *f* may contain variables (*pseudo-constants*)
- o_x^l: "At the l-th layer, the x-th operation is o" — *o* may contain pseudo-constants
- $prim_x^l$: "At the l-th layer, the x-th operation is primitive"
- $[\alpha/c]$: "Pseudo-constant α is substituted with constant c"

Sparse Encoding

- Only encode variables which are not trivially true or trivially false
- Reachability analysis from top to bottom, left to right: Filter impossible operations, facts
- Retroactively prune subtrees which turned out to be impossible

Our Encoding

Karlsruher Institut für Technologie

Clauses (1/2)

- Initial state s_0: $\forall f \in s_0 : f_0^0$, $\forall f \notin s_0 : \neg f_0^0$

Our Encoding

Karlsruher Institut für Technologie

Clauses (1/2)

- Initial state s_0: $\forall f \in s_0 : f_0^0$, $\forall f \notin s_0 : \neg f_0^0$
- At-most-one constraints over operations at each position

Our Encoding

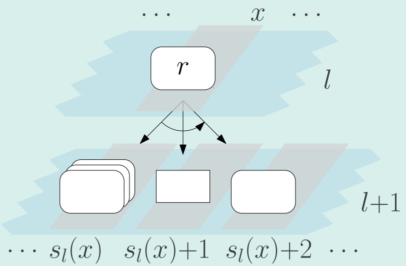
Clauses (1/2)

- Initial state s_0: $\forall f \in s_0 : f_0^0$, $\forall f \notin s_0 : \neg f_0^0$
- At-most-one constraints over operations at each position
- Preconditions and effects: $o_x^l \Rightarrow \bigwedge_{f \in pre(o)} f_x^l \wedge \bigwedge_{f \in eff(o)} f_{x+1}^l$

Our Encoding

Clauses (1/2)

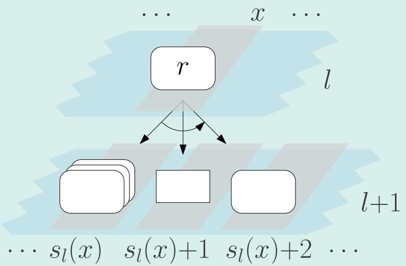
- Initial state s_0: $\forall f \in s_0 : f_0^0$, $\forall f \notin s_0 : \neg f_0^0$
- At-most-one constraints over operations at each position
- Preconditions and effects: $o_x^l \Rightarrow \bigwedge_{f \in pre(o)} f_x^l \wedge \bigwedge_{f \in eff(o)} f_{x+1}^l$
- Propagation of facts, actions: $f_x^l \Leftrightarrow f_{s_l(x)}^{l+1}$, $o_x^l \Rightarrow o_{s_l(x)}^{l+1}$



Our Encoding

Clauses (1/2)

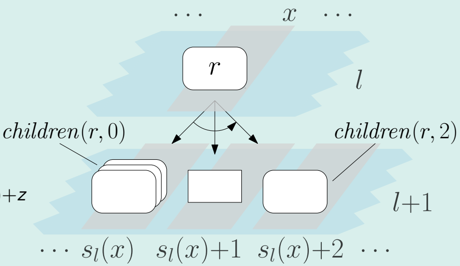
- Initial state s_0: $\forall f \in s_0 : f_0^0$, $\forall f \notin s_0 : \neg f_0^0$
- At-most-one constraints over operations at each position
- Preconditions and effects: $o_x^l \Rightarrow \bigwedge_{f \in pre(o)} f_x^l \wedge \bigwedge_{f \in eff(o)} f_{x+1}^l$
- Propagation of facts, actions: $f_x^l \Leftrightarrow f_{s_l(x)}^{l+1}$, $o_x^l \Rightarrow o_{s_l(x)}^{l+1}$
- Expansion of a reduction: $\forall z : o_x^l \Rightarrow \bigvee_{o' \in children(o,z)} (o')_{s_l(x)+z}^{l+1}$



Our Encoding

Clauses (1/2)

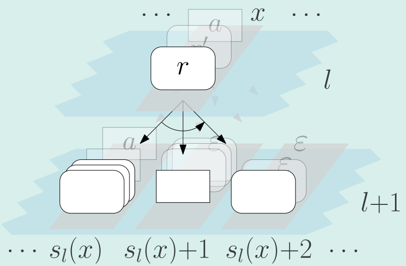
- Initial state s_0: $\forall f \in s_0 : f_0^0$, $\forall f \notin s_0 : \neg f_0^0$
- At-most-one constraints over operations at each position
- Preconditions and effects: $o_x^l \Rightarrow \bigwedge_{f \in pre(o)} f_x^l \wedge \bigwedge_{f \in eff(o)} f_{x+1}^l$
- Propagation of facts, actions: $f_x^l \Leftrightarrow f_{s_l(x)}^{l+1}$, $o_x^l \Rightarrow o_{s_l(x)}^{l+1}$
- Expansion of a reduction: $\forall z : o_x^l \Rightarrow \bigvee_{o' \in children(o,z)} (o')_{s_l(x)+z}^{l+1}$



Our Encoding

Clauses (1/2)

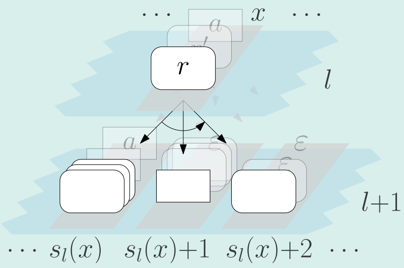
- Initial state s_0: $\forall f \in s_0 : f_0^0$, $\forall f \notin s_0 : \neg f_0^0$
- At-most-one constraints over operations at each position
- Preconditions and effects: $o_x^l \Rightarrow \bigwedge_{f \in pre(o)} f_x^l \wedge \bigwedge_{f \in eff(o)} f_{x+1}^l$
- Propagation of facts, actions: $f_x^l \Leftrightarrow f_{s_l(x)}^{l+1}$, $o_x^l \Rightarrow o_{s_l(x)}^{l+1}$
- Expansion of a reduction: $\forall z : o_x^l \Rightarrow \bigvee_{o' \in children(o,z)} (o')_{s_l(x)+z}^{l+1}$



Our Encoding

Clauses (1/2)

- Initial state s_0: $\forall f \in s_0 : f^0_0$, $\forall f \notin s_0 : \neg f^0_0$
- At-most-one constraints over operations at each position
- Preconditions and effects: $o^l_x \Rightarrow \bigwedge_{f \in pre(o)} f^l_x \wedge \bigwedge_{f \in eff(o)} f^l_{x+1}$
- Propagation of facts, actions: $f^l_x \Leftrightarrow f^{l+1}_{s_l(x)}$, $o^l_x \Rightarrow o^{l+1}_{s_l(x)}$
- Expansion of a reduction: $\forall z : o^l_x \Rightarrow \bigvee_{o' \in children(o,z)} (o')^{l+1}_{s_l(x)+z}$
- Assume fully expanded network at deepest layer l': $prim^{l'}_0, prim^{l'}_1, prim^{l'}_2, \ldots$



Our Encoding

Karlsruher Institut für Technologie

Clauses (2/2)

- Domain of pseudo-constant α in o: $o^l_x \Rightarrow \bigvee_{c \in dom(\alpha)} [\alpha/c]$
  + at-most-one constraints over $\{[\alpha/c] \mid c \in dom(\alpha)\}$
- Link between pseudo-atom f and actual atom $f' := f[\alpha_1/c_1][\alpha_2/c_2]$:
  $([\alpha_1/c_1] \wedge [\alpha_2/c_2]) \Rightarrow (f^l_x \Leftrightarrow (f')^l_x)$
- Frame axioms: $(f^l_x \wedge \neg f^l_{x+1}) \Rightarrow (\neg prim^l_x \vee \bigvee_{o \in supp(\neg f)} o^l_x \vee \bigvee_{o \in isupp(\neg f)} o^l_x)$
  + If $o \in isupp(\neg f)$ is active, then active substitutions must unify an effect of o with f.

Further Challenges

- Special handling of actions whose effects may become contradictory
- Enforce restrictions on a pseudo-constant's domain imposed by argument types
- Make "more general" operations subsume "less general" operations