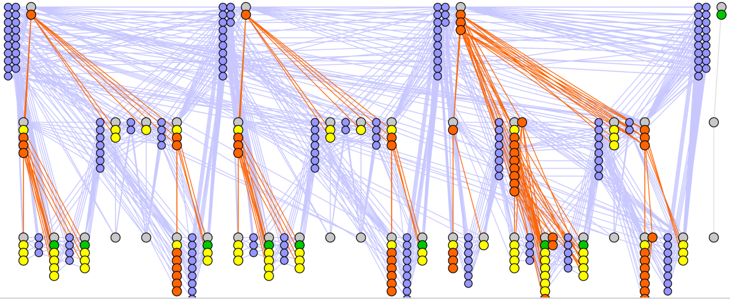
# Tree-REX: SAT-based Tree Exploration For Efficient and High-Quality HTN Planning

29th International Conference on Automated Planning and Scheduling

Dominik Schreiber, Damien Pellier, Humbert Fiorino, Tomáš Balyo | July 13, 2019

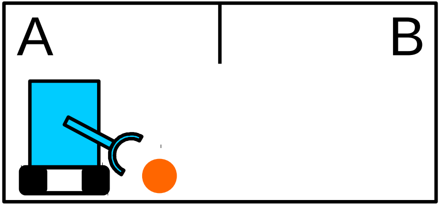
KARLSRUHE INSTITUTE OF TECHNOLOGY // UNIVERSITY GRENOBLE ALPES



# Outline

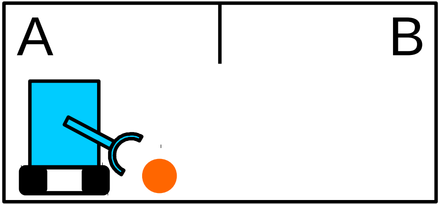
- Introduction
  - $\{$Automated, Hierarchical, SAT-based$\}$ Planning
- Related Work
- Tree-REX Planning approach
  - Algorithm; Encoding; Plan length optimization
- Evaluation
- Conclusion

# Automated Planning




*Find a valid sequence of actions from some initial world state to a desired goal state.*

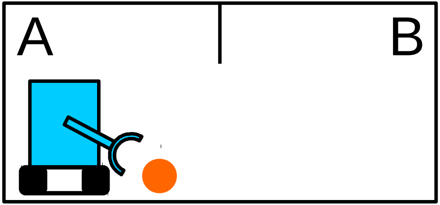
# Automated Planning




*Find a valid sequence of actions from some initial world state to a desired goal state.*

- (World) State: Consistent set of boolean atoms;
  e.g. `at(ball,A)`, `at(robot,B)`

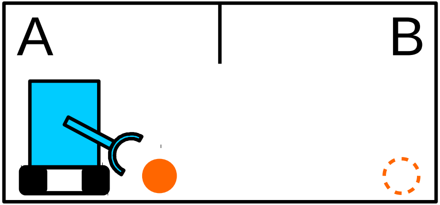
# Automated Planning




*Find a valid sequence of actions from some initial world state to a desired goal state.*

- Action *a*: Has boolean preconditions and effects;
  e.g. action `move(robot,A,B)` requires `at(robot,A)`, deletes `at(robot,A)`, adds `at(robot,B)`

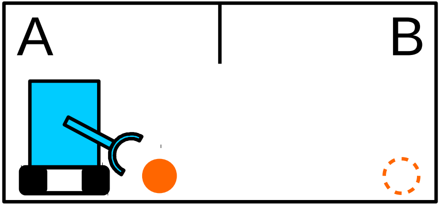
# Automated Planning




*Find a valid sequence of actions from some initial world state to a desired goal state.*

- Goal *g*: Subset of possible states, e.g. `at(ball,B)` must hold

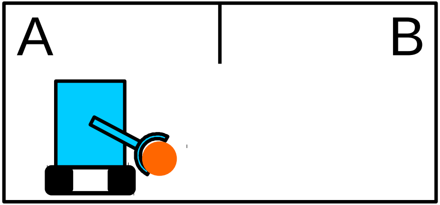
# Automated Planning




*Find a valid sequence of actions from some initial world state to a desired goal state.*

- Plan $\pi$: Action sequence transforming an initial state to a goal state
  e.g. $\pi =$

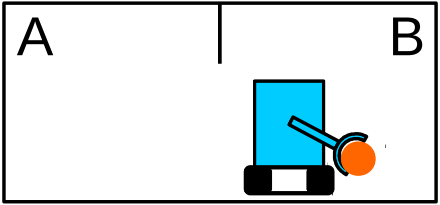
# Automated Planning




*Find a valid sequence of actions from some initial world state to a desired goal state.*

- Plan $\pi$: Action sequence transforming an initial state to a goal state
  e.g. $\pi = \langle$ `pickup(robot,ball,A)`

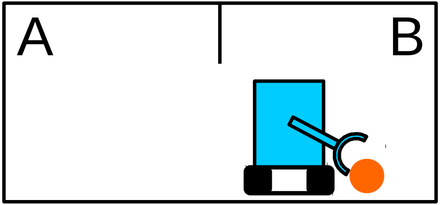
# Automated Planning




*Find a valid sequence of actions from some initial world state to a desired goal state.*

- Plan $\pi$: Action sequence transforming an initial state to a goal state
  e.g. $\pi = \langle$ `pickup(robot,ball,A)`, `move(robot,A,B)`

# Automated Planning




*Find a valid sequence of actions from some initial world state to a desired goal state.*

- Plan $\pi$: Action sequence transforming an initial state to a goal state
  e.g. $\pi = \langle$ `pickup(robot,ball,A)`, `move(robot,A,B)`,
  `drop(robot,ball,B)` $\rangle$

# Hierarchical Planning

Main idea: *Share domain-specific expert knowledge with your planner.*

- Which tasks need to be achieved
- How to directly achieve simple tasks
- How to break down complex tasks into simpler ones

# Hierarchical Planning

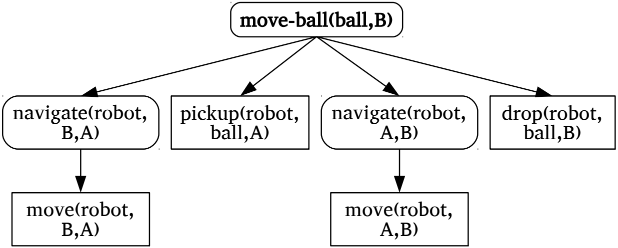
Main idea: *Share domain-specific expert knowledge with your planner.*

- Which tasks need to be achieved
- How to directly achieve simple tasks
- How to break down complex tasks into simpler ones

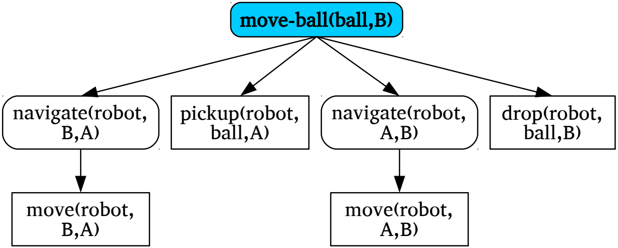
Most popular: Hierarchical Task Network (HTN) Planning [Erol et al., 1994]

- Extension of classical planning (states, actions, plans)
- More expressive than classical planning
- More focused search, enables more efficient planning

# Hierarchical Task Networks



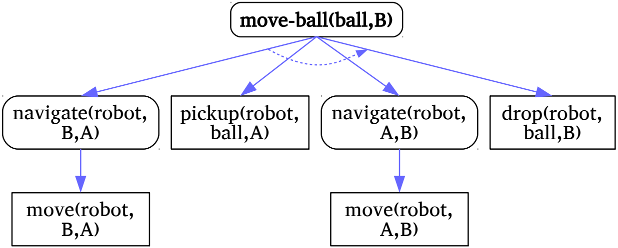
# Hierarchical Task Networks




Root(s): Initial task(s), part of problem input

- Abstract notion of what needs to be achieved

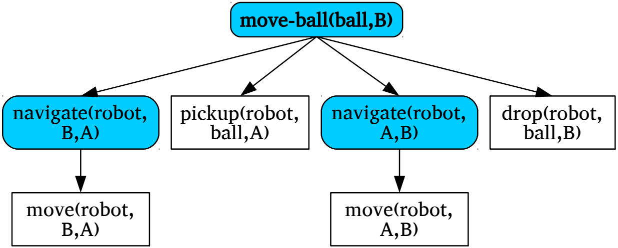
# Hierarchical Task Networks




Directed edges: Subtask relationships

- Totally ordered HTN planning ⇒ Total order on subtasks
- Span a tree of tasks

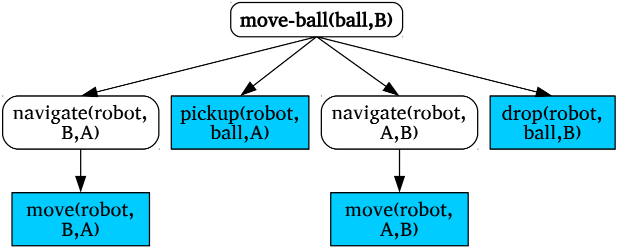
# Hierarchical Task Networks




Inner nodes: Compound tasks

- Can be achieved by picking a method and achieving each subtask
- Example: Method `move-ball(ball, to, r, x, y)`
  Preconditions { `at(ball,x)`, `at(r,y)` },
  Subtasks ⟨ (1) `navigate(r,y,x)`, (2) `pickup(r,ball,x)`,
  (3) `navigate(r,x,to)`, (4) `drop(r,ball,to)` ⟩

# Hierarchical Task Networks

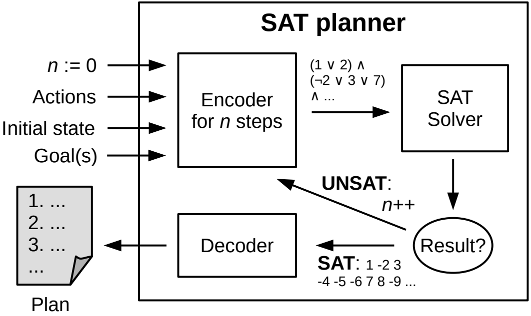



Leaf nodes: Primitive tasks

- Directly correspond to applying a certain action
- In-order traversal of all leaves $\Rightarrow$ Plan!

# SAT-based (Classical) Planning

Iteratively encode planning problem into propositional logic up to a certain number $n$ of steps [Kautz and Selman, 1992]



# SAT-based Totally Ordered HTN Planning

1998: Inception of SAT-based HTN planning [Mali and Kambhampati, 1998]

- Encodings do not address recursive task relationships (fixed maximum amount of actions for each task)
- Complexity of clauses and variables cubic in amount of steps

# SAT-based Totally Ordered HTN Planning

1998: Inception of SAT-based HTN planning [Mali and Kambhampati, 1998]

- Encodings do not address recursive task relationships (fixed maximum amount of actions for each task)
- Complexity of clauses and variables cubic in amount of steps

2018: totSAT [Behnke et al., 2018]

- Encode problem's hierarchy up to depth *k* until satisfiable for some *k*
- Method preconditions compiled into "virtual actions"

# SAT-based Totally Ordered HTN Planning

1998: Inception of SAT-based HTN planning [Mali and Kambhampati, 1998]

- Encodings do not address recursive task relationships (fixed maximum amount of actions for each task)
- Complexity of clauses and variables cubic in amount of steps

2018: totSAT [Behnke et al., 2018]

- Encode problem's hierarchy up to depth *k* until satisfiable for some *k*
- Method preconditions compiled into "virtual actions"

2019: SAT-based Stack Machine Simulation [Schreiber et al., 2019]

- Uses incremental SAT solving: Maintains a single logical formula
- Requires hyperparameter (maximum stack size to encode)

# Tree-like Reduction Exploration

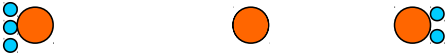
Explore hierarchy breadth-first (layer by layer)

- Each layer contains potential abstract plans
- Expand hierarchy until actual valid plan is found
  - Primitive subtasks only
  - All preconditions, effects, goals hold

# Tree-like Reduction Exploration

Explore hierarchy breadth-first (layer by layer)

- Each layer contains potential abstract plans
- Expand hierarchy until actual valid plan is found
  - Primitive subtasks only
  - All preconditions, effects, goals hold

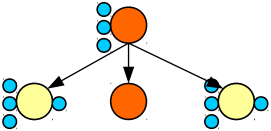


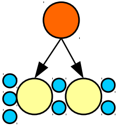
Layer 0:
3 initial tasks

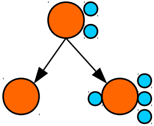
# Tree-like Reduction Exploration

Explore hierarchy breadth-first (layer by layer)

- Each layer contains potential abstract plans
- Expand hierarchy until actual valid plan is found
  - Primitive subtasks only
  - All preconditions, effects, goals hold







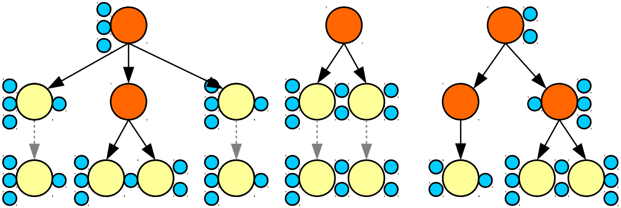
Layer 0:
3 initial tasks

Layer 1

# Tree-like Reduction Exploration

Explore hierarchy breadth-first (layer by layer)

- Each layer contains potential abstract plans
- Expand hierarchy until actual valid plan is found
  - Primitive subtasks only
  - All preconditions, effects, goals hold



Layer 0:
3 initial tasks

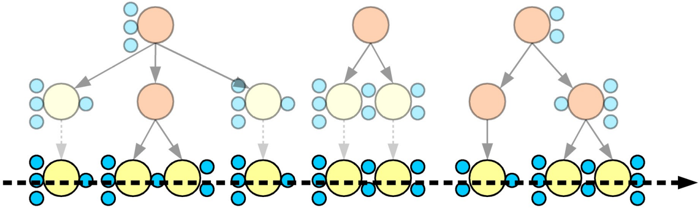
Layer 1

Layer 2

# Tree-like Reduction Exploration

Explore hierarchy breadth-first (layer by layer)

- Each layer contains potential abstract plans
- Expand hierarchy until actual valid plan is found
    - Primitive subtasks only
    - All preconditions, effects, goals hold



Layer 0:
3 initial tasks

Layer 1

Layer 2: Plan!

# Tree-REX: Algorithm

**Algorithm:** Tree-REX Planning Procedure

1: $\Pi :=$ Preprocess and ground HTN planning problem;
2: $C := \emptyset$; *// Logical clauses*
3: $\mathit{nextLayer} :=$ GenerateFirstLayer($\Pi$);
4: $\mathit{prevLayer} := \mathit{nextLayer}$;
5: $\mathit{result} :=$ NONE;

# Tree-REX: Algorithm

**Algorithm:** Tree-REX Planning Procedure

```
Π := Preprocess and ground HTN planning problem;
C := ∅;                                               // Logical clauses
nextLayer := GenerateFirstLayer(Π);
prevLayer := nextLayer;
result := NONE;
while result = NONE do
    prevLayer := nextLayer;
    nextLayer := Expand(nextLayer);                   // Create new hierarchical layer
```

# Tree-REX: Algorithm

Karlsruhe Institute of Technology

**Algorithm:** Tree-REX Planning Procedure

$\Pi :=$ Preprocess and ground HTN planning problem;
$C := \emptyset$; *// Logical clauses*
$\mathit{nextLayer} :=$ GenerateFirstLayer($\Pi$);
$\mathit{prevLayer} := \mathit{nextLayer}$;
$\mathit{result} :=$ NONE;
**while** $\mathit{result} =$ NONE **do**
    $\mathit{prevLayer} := \mathit{nextLayer}$;
    $\mathit{nextLayer} :=$ Expand($\mathit{nextLayer}$); *// Create new hierarchical layer*
    $C := C \cup$ Encode($\mathit{prevLayer}$, $\mathit{nextLayer}$); *// Add clauses*
    $\mathit{result} :=$ Solve($C$, AssumeAllPrimitive($\mathit{nextLayer}$)); *// SAT Solving*

# Tree-REX: Algorithm

**Algorithm:** Tree-REX Planning Procedure

```
Π := Preprocess and ground HTN planning problem;
C := ∅;                                                   // Logical clauses
nextLayer := GenerateFirstLayer(Π);
prevLayer := nextLayer;
result := NONE;
while result = NONE do
    prevLayer := nextLayer;
    nextLayer := Expand(nextLayer);      // Create new hierarchical layer
    C := C ∪ Encode(prevLayer, nextLayer);                // Add clauses
    result := Solve(C, AssumeAllPrimitive(nextLayer));    // SAT Solving
return Decode(Π, result).          // Extract plan from sat. assignment
```

# Tree-REX: Encoding into SAT

Karlsruhe Institute of Technology

Boolean variables:

- Variable for each possible fact/action/method at each position $(l, i)$
- Variable indicating primitiveness of each position $(l, i)$

# Tree-REX: Encoding into SAT

Boolean variables:

- Variable for each possible fact/action/method at each position $(l, i)$
- Variable indicating primitiveness of each position $(l, i)$

Clauses:

- Initial layer: Enforce initial task network
- Every layer: Classical planning clauses for occurring facts, actions [Kautz and Selman, 1992]
- In between layers: Propagation of facts, actions, Reduction of methods into any of its possible children
- Final layer: Enforce actions only (no methods!)

# Tree-REX: Plan Length Optimization

Solution is found at layer $m$: Plan has some length $n \leq \text{size}(\mathit{lastLayer})$

$\Rightarrow$ Length of plan $\equiv$ Amount of actual actions at final layer

$\Rightarrow$ No incentive for SAT solver to prefer solutions with fewer actions

# Tree-REX: Plan Length Optimization

Solution is found at layer $m$: Plan has some length $n \leq \text{size}(\mathit{lastLayer})$

$\Rightarrow$ Length of plan $\equiv$ Amount of actual actions at final layer

$\Rightarrow$ No incentive for SAT solver to prefer solutions with fewer actions

**Quality awareness**: Use incremental SAT solving to find better plans

- Encode simple plan length counter at final layer:
  Boolean variable for "*plan length is greater than or equal to x*"

# Tree-REX: Plan Length Optimization

Solution is found at layer $m$: Plan has some length $n \leq \text{size}(\textit{lastLayer})$

$\Rightarrow$ Length of plan $\equiv$ Amount of actual actions at final layer

$\Rightarrow$ No incentive for SAT solver to prefer solutions with fewer actions

**Quality awareness**: Use incremental SAT solving to find better plans

- Encode simple plan length counter at final layer:
  Boolean variable for "*plan length is greater than or equal to x*"
- Enforce plan length smaller than $n$ as an additional assumption

  $\Rightarrow$ SAT: Better plan has been found (length $< n$)

  $\Rightarrow$ UNSAT: Plan length $n$ is already optimal at this layer
- Anytime procedure: Better results the more resources are invested

# Evaluations

- PDDL benchmarks from previous HTN planning evaluations [Behnke et al., 2018, Ramoul et al., 2017]
- Competitors:
  - Tree-REX (no plan length optimization)
  - Tree-REX-o (with plan length optimization)
  - totSAT [Behnke et al., 2018] (AAAI-18 configuration)
- SAT solver: MiniSat [Eén and Sörensson, 2003] for all competitors
- Up to 5 minutes of total run time

# Evaluations: Run Times

| Domain | #Inst. | totSAT | Tree-REX | Tree-REX-o |
|---|---|---|---|---|
| Barman | 20 | 5.00 | **20.00** | 9.19 |
| Blocksworld | 20 | 4.02 | **20.00** | 19.22 |
| Childsnack | 20 | 0.91 | **20.00** | 19.99 |
| Depots | 20 | 7.81 | **19.36** | 18.95 |
| Entertainment | 12 | 6.73 | **9.25** | 8.94 |
| Gripper | 20 | 2.92 | **20.00** | 19.85 |
| Hiking | 20 | 0.93 | **20.00** | 17.93 |
| Rover | 20 | 0.87 | **20.00** | 10.12 |
| Satellite | 20 | 1.23 | **16.00** | 8.62 |
| Transport | 30 | 3.77 | **30.00** | 21.51 |

Table: Run time IPC scores

# Evaluations: Plan Quality

| Domain | #Inst. | totSAT | Tree-REX | Tree-REX-o |
|---|---|---|---|---|
| Barman | 20 | 19.67 | 18.67 | **20.00** |
| Blocksworld | 20 | 18.36 | **19.68** | **19.68** |
| Childsnack | 20 | 12.00 | **20.00** | **20.00** |
| Depots | 20 | 18.59 | 19.74 | **19.93** |
| Entertainment | 12 | 11.92 | 11.79 | **12.00** |
| Gripper | 20 | **20.00** | **20.00** | **20.00** |
| Hiking | 20 | 11.00 | **20.00** | **20.00** |
| Rover | 20 | 4.83 | 13.08 | **20.00** |
| Satellite | 20 | 9.80 | 9.66 | **16.00** |
| Transport | 30 | 23.71 | 26.04 | **30.00** |

Table: Plan length IPC scores

# Conclusion

- Tree-REX: SAT-based totally ordered HTN planning
- Rapid encoding and solving through incremental SAT solving
- Anytime plan length optimization
  $\Rightarrow$ First quality-aware SAT-based HTN planner
- Outperforms state-of-the-art SAT-based planning regarding run times and/or plan quality

# Conclusion

- Tree-REX: SAT-based totally ordered HTN planning
- Rapid encoding and solving through incremental SAT solving
- Anytime plan length optimization
  $\Rightarrow$ First quality-aware SAT-based HTN planner
- Outperforms state-of-the-art SAT-based planning regarding run times and/or plan quality

## Future work

- Integrate into framework with additional features (disjunctive / ADL conditions, conditional effects, . . . )
- Interleave expansion and resolution of hierarchical layers
- Investigate (partially) lifted SAT encodings for HTN

# References I


Behnke, G., Höller, D., and Biundo, S. (2018).
totSAT–totally-ordered hierarchical planning through SAT.
In *Proceedings of the 32th AAAI conference on AI (AAAI 2018). AAAI Press*.

Eén, N. and Sörensson, N. (2003).
Temporal induction by incremental SAT solving.
*Electronic Notes in Theoretical Computer Science*, 89(4):543–560.

Erol, K., Hendler, J., and Nau, D. (1994).
UMCP: A sound and complete procedure for hierarchical task-network planning.
In *Proceedings of the Artificial Intelligence Planning Systems*, volume 94, pages 249–254.

Kautz, H. and Selman, B. (1992).
Planning as Satisfiability.
In *Proceedings of the European Conference on Artificial Intelligence*, pages 359–363.

# References II


Mali, A. and Kambhampati, S. (1998).
Encoding HTN planning in propositional logic.
In *Proceedings International Conference on Artificial Intelligence Planning and Scheduling*, pages 190–198.

Ramoul, A., Pellier, D., Fiorino, H., and Pesty, S. (2017).
Grounding of HTN planning domain.
*International Journal on Artificial Intelligence Tools*, 26(5):1–24.

Schreiber, D., Balyo, T., Pellier, D., and Fiorino, H. (2019).
Efficient SAT encodings for hierarchical planning.
In *Proceedings of the 11th International Conference on Agents and Artificial Intelligence, ICAART 2019*, volume 2, pages 531–538.

# Hierarchical Planning

- Generally, can allow partially ordered and interleaving subtasks (here: total order on subtasks)
- Difficulty of HTN planning: Choosing the “right” method to fulfill a task (non-deterministic / randomized choice, heuristics, . . . )

Compared to classical planning:

- Smaller, more directed search space to find a plan
- More expressive [Erol et al., 1994]

# Tree-like Reduction Exploration (2/2)

Layer $l$: Array $[0, \ldots, s_l - 1]$ of sets of facts, actions, and methods

- Lookup for which objects need to be encoded where at layer $l$
- $(l, i) :=$ $i$-th position of layer $l$
- $x \in (l, i) \; :\Leftrightarrow$ encode method / action / fact $x$ at position $(l, i)$

# Tree-like Reduction Exploration (2/2)

Layer $l$: Array $[0, \ldots, s_l - 1]$ of sets of facts, actions, and methods

- Lookup for which objects need to be encoded where at layer $l$
- $(l, i) :=$ $i$-th position of layer $l$
- $x \in (l, i)$ $:\Leftrightarrow$ encode method / action / fact $x$ at position $(l, i)$

Inductive construction of layers:

- Layer 0: Fully defined by initial state and initial task network
- Layer $n - 1 \rightsquigarrow n$: Propagate facts and actions, reduce methods, add facts implied by new actions and methods

# Tree-like Reduction Exploration (2/2)

Layer $l$: Array $[0, \ldots, s_l - 1]$ of sets of facts, actions, and methods

- Lookup for which objects need to be encoded where at layer $l$
- $(l, i) :=$ $i$-th position of layer $l$
- $x \in (l, i) \;:\Leftrightarrow$ encode method / action / fact $x$ at position $(l, i)$

Inductive construction of layers:

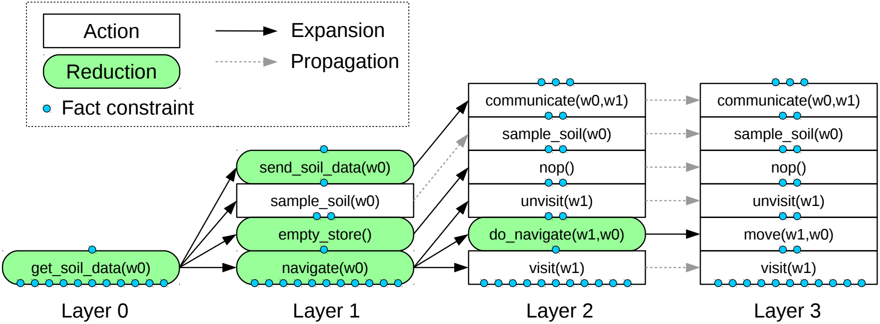
- Layer 0: Fully defined by initial state and initial task network
- Layer $n - 1 \rightsquigarrow n$: Propagate facts and actions, reduce methods, add facts implied by new actions and methods

Each subtask may have many different realizations to choose from!

- Example: `walk(a,b)` may be achieved by walking over `c`, over `d`, ...
- Encode all options – SAT solver will decide on which option to take

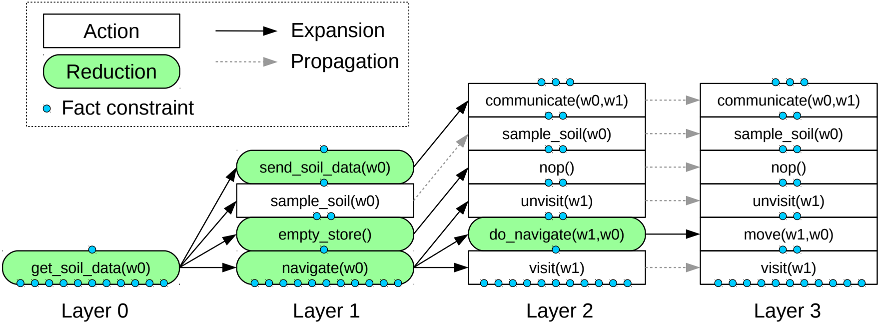
# Tree-like Reduction Exploration (3/2)

**Example**: Rover planning problem with one initial task,
showing only one of the possible actions/methods at each position



# Tree-like Reduction Exploration (3/2)

**Example**: Rover planning problem with one initial task,
showing only one of the possible actions/methods at each position




Layer 3: Valid sequence of primitive actions ⇒ Finished plan!

# Tree-REX: Encoding into SAT (1/2)

Boolean variables:

- Variable for each possible fact/action/method at each position $(l, i)$
- Variable indicating primitiveness of each position $(l, i)$

# Tree-REX: Encoding into SAT (1/2)

Boolean variables:

- Variable for each possible fact/action/method at each position $(l, i)$
- Variable indicating primitiveness of each position $(l, i)$

Classical planning clauses: mostly from [Kautz and Selman, 1992]

1. The initial state holds at $(l, 0)$ and the goals hold at $(l, s_l)$.
2. An action at $(l, i)$ implies its preconditions at $(l, i)$ and its effects at $(l, i + 1)$.
3. At most one action occurs at each position $(l, i)$.
4. A fact changes its value from $(l, i)$ to $(l, i + 1)$ only if some action supporting the fact change occurs at $(l, i)$ or $(l, i)$ is not primitive.

# Tree-REX: Encoding into SAT (2/2)

Primitiveness and method preconditions:

5. An action (*method*) at position $p$ implies that $p$ is (*not*) primitive.
6. A method at $(l, i)$ implies its preconditions at $(l, i)$.

# Tree-REX: Encoding into SAT (2/2)

Primitiveness and method preconditions:

5. An action (*method*) at position $p$ implies that $p$ is (*not*) primitive.
6. A method at $(l, i)$ implies its preconditions at $(l, i)$.

Transitions from layer $l$ to layer $l+1$, former position $i$ shifted to $i' \geq i$:

7. A fact holds at $(l, i)$ iff it holds at $(l+1, i')$.
8. An action at $(l, i)$ implies the same action at $(l+1, i')$.
9. A method at $(l, i)$ implies any of its valid children at $(l+1, i'), \ldots, (l+1, i'+k)$.

# Tree-REX: Encoding into SAT (2/2)

Primitiveness and method preconditions:

5. An action (*method*) at position $p$ implies that $p$ is (*not*) primitive.
6. A method at $(l, i)$ implies its preconditions at $(l, i)$.

Transitions from layer $l$ to layer $l + 1$, former position $i$ shifted to $i' \geq i$:

7. A fact holds at $(l, i)$ iff it holds at $(l + 1, i')$.
8. An action at $(l, i)$ implies the same action at $(l + 1, i')$.
9. A method at $(l, i)$ implies any of its valid children at $(l + 1, i'), \ldots, (l + 1, i' + k)$.

Initial and final layer:

10. Each initial task $t_i$ implies some according method/action at $(0, i)$.
11. Each position of the final layer is primitive. (Goal assumptions)

# Tree-REX: Realization

## Encoding

- Enforce virtual "blank" actions where positions should remain empty
- Additional (redundant) clauses: *Backwards propagation* –
  Necessary conditions for an action / method to occur somewhere
- Various optimizations reducing clauses and/or variables

# Tree-REX: Realization

## Encoding

- Enforce virtual "blank" actions where positions should remain empty
- Additional (redundant) clauses: *Backwards propagation* –
  Necessary conditions for an action / method to occur somewhere
- Various optimizations reducing clauses and/or variables

## Implementation

- Preprocessing and grounding from [Ramoul et al., 2017]
- Separate interpreter application
  – Receives abstract encoding "blueprint"
  – Instantiates clauses as necessary
- IPASIR as generic incremental SAT interface
  $\Rightarrow$ Plug in any modern incremental SAT solver

# Tree-REX

Karlsruhe Institute of Technology

## Expansion of Methods.

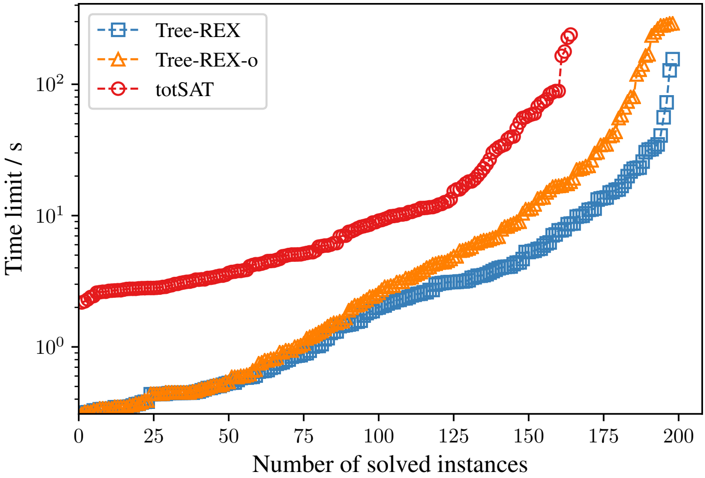
Consider the $l$-th and $(l+1)$-th hierarchical layers of an HTN planning problem. For some $i \geq 0$, all facts and actions at $(l, i)$ are propagated to $(l+1, i')$, where $i' \geq i$. Let method $r$ occur at $(l, i)$.

- If the $k$-th subtask $t_k$ of $r$ is primitive, the corresponding action $a_{t_k}$ is added to $(l+1, i')$.
- If $t_k$ is compound, each possible corresponding method $r' \in R(t_k)$ is added to $(l+1, i'+k)$.

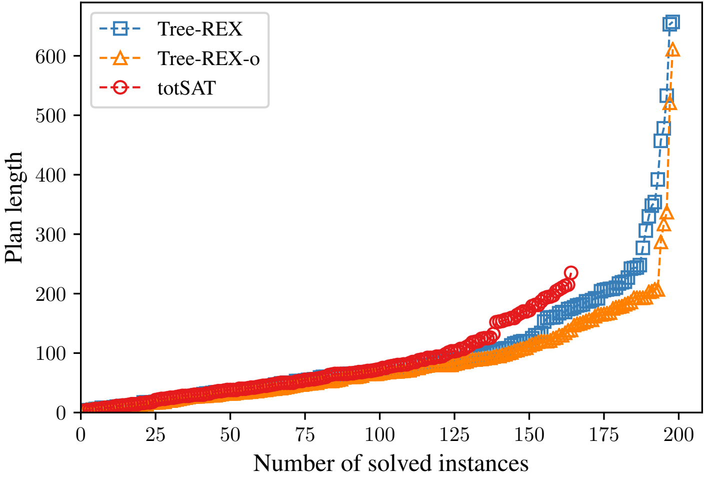
How to incrementally build formula layer by layer?

- Add clauses (9) once, clauses (1-8) for each new layer $l$
- Assume clauses (10) before each solving attempt (drop afterwards)

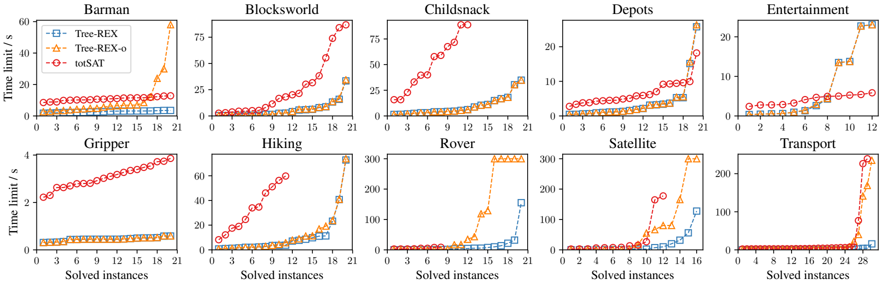
# Evaluation Plots I



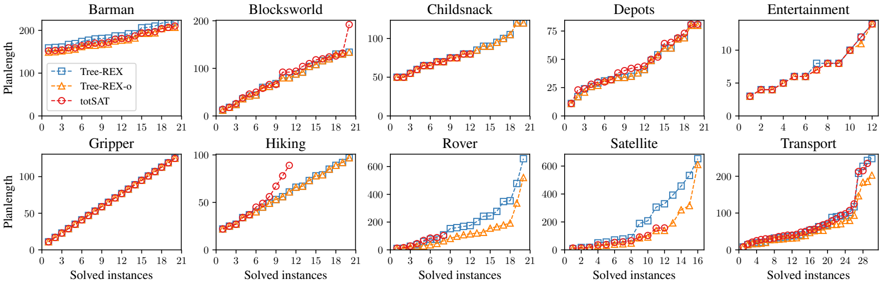
# Evaluation Plots II



# Evaluation Plots III



# Evaluation Plots IV



# Evaluations: Insights

- Tree-REX dominates run times, Tree-REX-o dominates plan quality
- Plan length optimization heavily domain-dependent
  - Not improvable due to rigid hierarchy (e.g. Childsnack, Gripper): instant termination after initial plan is found
  - Slightly improvable (e.g. Barman): quick optimization process
  - Highly improvable (e.g. Rover, Satellite): long optimization process with many little improvements
- Bottleneck for large instances: Grounding, SAT solving