Karlsruhe Institute of Technology

UNIVERSITÉ Grenoble Alpes

Efficient SAT Encodings for Hierarchical Planning

11th International Conference on Agents and Artificial Intelligence

Dominik Schreiber (Speaker), Tomáš Balyo, Damien Pellier, Humbert Fiorino | February 19, 2019

KARLSRUHE INSTITUTE OF TECHNOLOGY // UNIVERSITY GRENOBLE ALPES

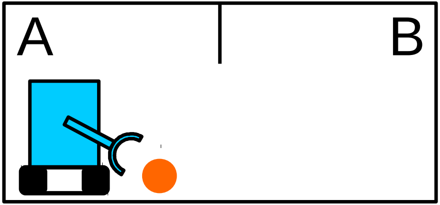
```
                (tag t3 (serve_shot ?sh ?c))
            )
        :constraints(and
                (before (and (ontable ?sh) (clean ?sh) (handempty ?h)) t1)
            )
    )

    (:method do_cocktail_in_shot2
        :parameters (?sh - shot ?i - ingredient)
        :expansion  (
                (tag t1 (do_fill_shot ?sh ?i ?h))
                (tag t2 (leave ?h ?sh))
            )
        :constraints(and
                (before (and (dispenses ?d ?i)) t1)
            )
)
```

Outline

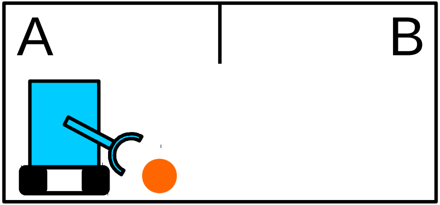
- Background: Planning, Hierarchical Planning, SAT Planning
- Related Work
- Contributions: GCT Encoding, SMS Encoding
- Evaluation
- Conclusion

Automated Planning



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

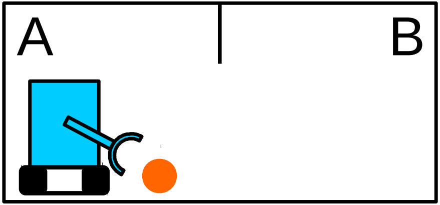
Automated Planning



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)`

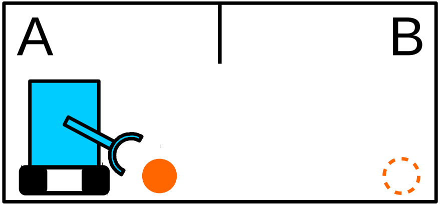
Automated Planning



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)`

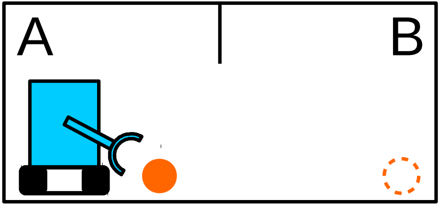
Automated Planning



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

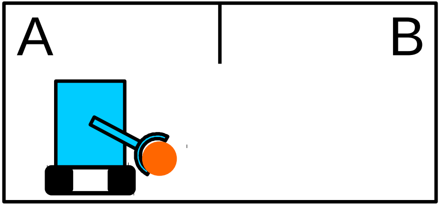
Automated Planning



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

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

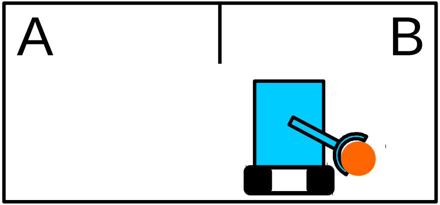
Automated Planning



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

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

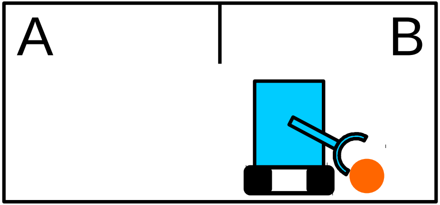
Automated Planning



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

- Plan π: 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 π: 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

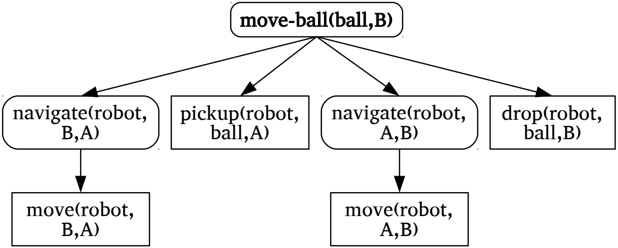
Main idea: *Share domain-specific expert knowledge with your planner.*

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

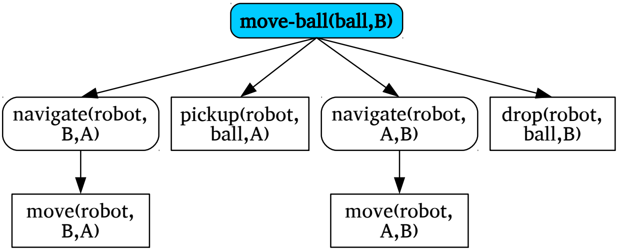
Most popular: Hierarchical Task Network (HTN) Planning [Erol et al., 1994]

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

Hierarchical Task Networks



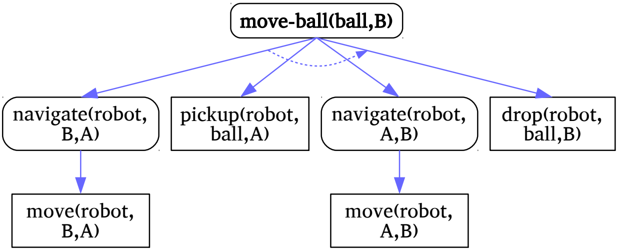
Hierarchical Task Networks



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

- Abstract notion of what needs to be achieved

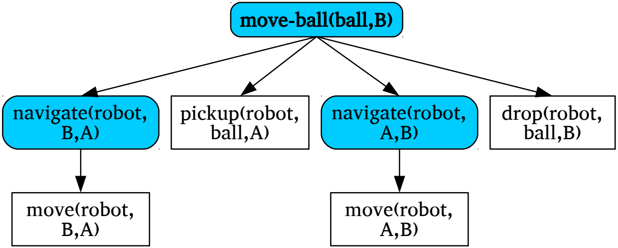
Hierarchical Task Networks



Directed edges: Subtask relationships

- Here: totally ordered
- Span a tree of tasks

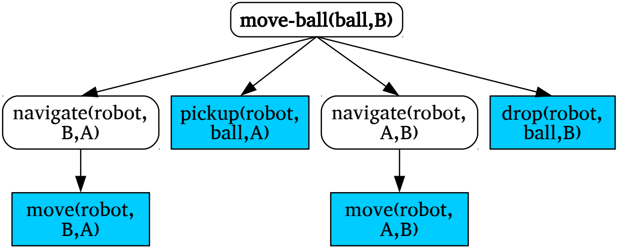
Hierarchical Task Networks



Inner nodes: Compound tasks

- Can be achieved by choosing 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

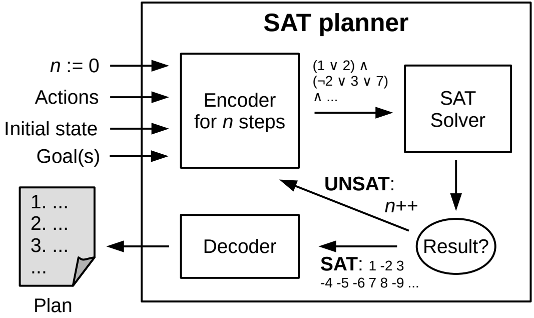


Leaf nodes: Primitive tasks

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

SAT-based Planning

- Encode planning problem into propositional logic up to a certain number of steps [Kautz and Selman, 1992]
- Use Satisfiability (SAT) Solver to find satisfying assignment
- Decode assignment back into a plan



Related Work

Introduction of incremental SAT solving to planning [Gocht and Balyo, 2017]

- Maintain and iteratively extend a single logical formula
- Remember logical conflicts from previous iterations

Related Work

Introduction of incremental SAT solving to planning
[Gocht and Balyo, 2017]

- Maintain and iteratively extend a single logical formula
- Remember logical conflicts from previous iterations

SAT-based HTN planning: Few research before 2018
[Mali and Kambhampati, 1998]

- Previous 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
- In practice, infeasible for today's problem instances

Encoding: Grammar-Constrained Tasks

Enhancement of previous *bottom-up linear forward* encoding [Mali and Kambhampati, 1998]

- Focused on totally ordered HTN planning
- Fully supporting recursive subtask relationships
- Resulting in smaller encoding size (quadratic in #steps, #tasks)

Encoding: Grammar-Constrained Tasks

Enhancement of previous *bottom-up linear forward* encoding [Mali and Kambhampati, 1998]

- Focused on totally ordered HTN planning
- Fully supporting recursive subtask relationships
- Resulting in smaller encoding size (quadratic in #steps, #tasks)

Limitations of new encoding:

- Encoding still too large for realistic problem sizes
- Allows for interleaving of tasks in some special cases

Encoding: Grammar-Constrained Tasks

Enhancement of previous *bottom-up linear forward* encoding [Mali and Kambhampati, 1998]

- Focused on totally ordered HTN planning
- Fully supporting recursive subtask relationships
- Resulting in smaller encoding size (quadratic in #steps, #tasks)

Limitations of new encoding:

- Encoding still too large for realistic problem sizes
- Allows for interleaving of tasks in some special cases

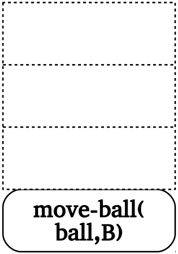
Observation: HTN is like enforcing a grammar on valid plans

- Totally ordered HTN corresponds to context-free grammar
- Finding a plan equivalent to deriving a word from the grammar

Encoding: Stack Machine Simulation (1)

Based on idea of context-free grammar:

- Encode stack of tasks at each step of future plan
- Add transition rules (**pop**, **push**) to process tasks until stack is empty

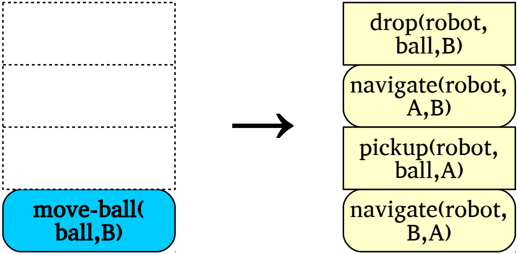


Plan: $\langle$ $\rangle$

Encoding: Stack Machine Simulation (1)

Based on idea of context-free grammar:

- Encode stack of tasks at each step of future plan
- Add transition rules (**pop**, **push**) to process tasks until stack is empty

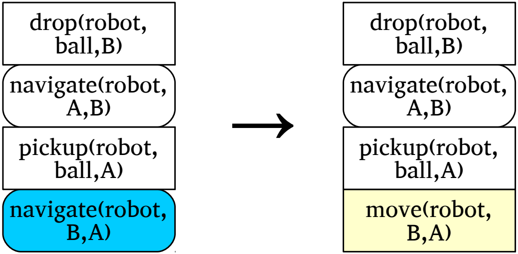


Plan: $\langle$ $\rangle$

Encoding: Stack Machine Simulation (1)

Based on idea of context-free grammar:

- Encode stack of tasks at each step of future plan
- Add transition rules (**pop**, **push**) to process tasks until stack is empty

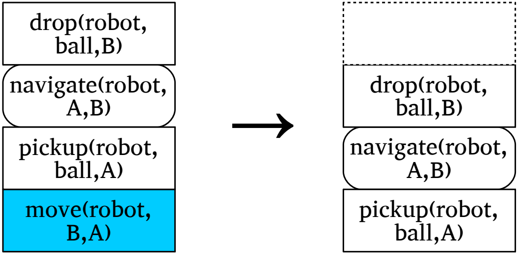


Plan: $\langle \qquad \rangle$

Encoding: Stack Machine Simulation (1)

Based on idea of context-free grammar:

- Encode stack of tasks at each step of future plan
- Add transition rules (**pop**, **push**) to process tasks until stack is empty

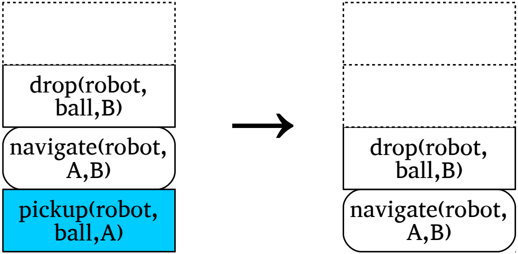


Plan: $\langle$ `move(robot,room2,room1)` $\rangle$

Encoding: Stack Machine Simulation (1)

Based on idea of context-free grammar:

- Encode stack of tasks at each step of future plan
- Add transition rules (**pop**, **push**) to process tasks until stack is empty

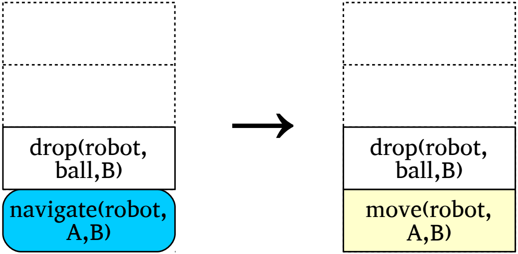


Plan: $\langle$ `move(robot,room2,room1)`, `pickup(robot,ball,room1)` $\rangle$

Encoding: Stack Machine Simulation (1)

Based on idea of context-free grammar:

- Encode stack of tasks at each step of future plan
- Add transition rules (**pop**, **push**) to process tasks until stack is empty

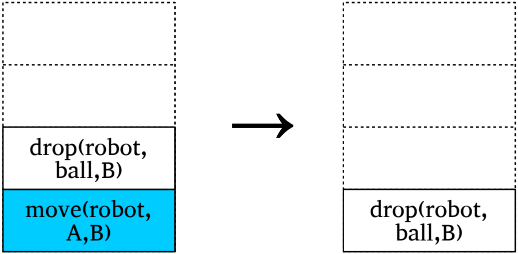


Plan: $\langle$ `move(robot,room2,room1)`, `pickup(robot,ball,room1)` $\rangle$

Encoding: Stack Machine Simulation (1)

Based on idea of context-free grammar:

- Encode stack of tasks at each step of future plan
- Add transition rules (**pop**, **push**) to process tasks until stack is empty

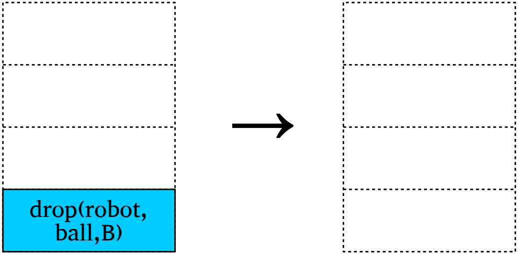


Plan: $\langle$ `move(robot,room2,room1)`, `pickup(robot,ball,room1)`, `move(robot,room1,room2)` $\rangle$

Encoding: Stack Machine Simulation (1)

Based on idea of context-free grammar:

- Encode stack of tasks at each step of future plan
- Add transition rules (**pop**, **push**) to process tasks until stack is empty



Plan: $\langle$ `move(robot,room2,room1)`, `pickup(robot,ball,room1)`, `move(robot,room1,room2)`, `drop(robot,ball,room2)` $\rangle$

Encoding: Stack Machine Simulation (2)

Realization in propositional logic:

- Boolean variables for each task at each stack position at each step, for each action at each step, for each atom at each step

Encoding: Stack Machine Simulation (2)

Realization in propositional logic:

- Boolean variables for each task at each stack position at each step, for each action at each step, for each atom at each step
- All clauses only contain variables from adjacent steps
  $\Rightarrow$ Formula can be expanded incrementally

Encoding: Stack Machine Simulation (2)

Realization in propositional logic:

- Boolean variables for each task at each stack position at each step, for each action at each step, for each atom at each step
- All clauses only contain variables from adjacent steps
  $\Rightarrow$ Formula can be expanded incrementally
- Assertion to SAT solver: stack must be empty at final step n
  $\Rightarrow$ Assignment found: Extract plan from `true` action variables
  $\Rightarrow$ Unsatisfiable: Increase n, add new clauses, repeat

Encoding: Stack Machine Simulation (2)

Realization in propositional logic:

- Boolean variables for each task at each stack position at each step, for each action at each step, for each atom at each step
- All clauses only contain variables from adjacent steps
  $\Rightarrow$ Formula can be expanded incrementally
- Assertion to SAT solver: stack must be empty at final step n
  $\Rightarrow$ Assignment found: Extract plan from `true` action variables
  $\Rightarrow$ Unsatisfiable: Increase n, add new clauses, repeat

Properties

- Handles all special cases (recursive subtasks, no interleaving, etc.)
- Requires parameter σ: Maximum stack size to encode
- $\mathcal{O}(\#\mathit{steps} \cdot (\sigma \cdot \#\mathit{tasks} + \#\mathit{methods} + \#\mathit{actions}))$ clauses

Evaluation

Internal evaluation of approaches

- GCT Encoding
- SMS Encoding (3 variants)

Evaluation

Karlsruhe Institute of Technology

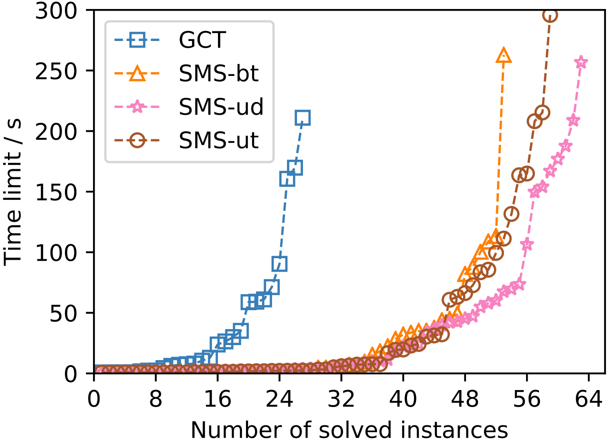
Internal evaluation of approaches

- GCT Encoding
- SMS Encoding (3 variants)

Evaluation environment:

- 120 benchmark instances from six IPC domains
  Barman, Blocksworld, Childsnack, Elevator, Rover, Satellite
- 24 core Intel Xeon CPU E5-2630 @ 2.30 GHz, 264 GB of RAM
- Limits per run: five minutes; 12 GB of RAM

Comparison of Run Times



Run Time Scores per Domain

Domain	GCT	SMS-bt	SMS-ut	SMS-ur
Barman	0.09	1.90	1.96	**4.68**
Blocksworld	0.08	9.22	**10.94**	6.74
Childsnack	0.98	3.90	**9.95**	4.50
Elevator	4.21	**14.86**	13.32	10.29
Rover	0.44	**6.17**	5.40	5.58
Satellite	0.96	7.08	7.17	**16.08**
Total	6.75	43.13	**48.74**	47.88

Score for each instance and competitor: $\frac{T^*}{T} = \frac{\text{best competitor's run time}}{\text{run time}}$

Plan Length Scores per Domain

Karlsruhe Institute of Technology

Domain	GCT	SMS-bt	SMS-ut	SMS-ur
Barman	0.85	2.72	2.00	**5.00**
Blocksworld	2.00	10.00	**13.00**	11.00
Childsnack	3.00	6.00	**10.00**	8.00
Elevator	13.00	**16.00**	15.00	15.00
Rover	3.86	**6.62**	6.55	**6.62**
Satellite	4.00	9.61	11.79	**16.77**
Total	26.70	50.96	58.33	**62.40**

Score for each instance and competitor: $\frac{T^*}{T} = \frac{\text{best competitor's plan length}}{\text{plan length}}$

Conclusion

- Two new SAT encodings for totally ordered HTN planning
  GCT: Handles recursive subtask relationships
  SMS: Introduces incremental SAT solving to HTN planning

Conclusion

- Two new SAT encodings for totally ordered HTN planning
  GCT: Handles recursive subtask relationships
  SMS: Introduces incremental SAT solving to HTN planning
- Evaluation: Incremental SMS encoding significantly outperforms more conventional GCT encoding

Conclusion

- Two new SAT encodings for totally ordered HTN planning
  GCT: Handles recursive subtask relationships
  SMS: Introduces incremental SAT solving to HTN planning
- Evaluation: Incremental SMS encoding significantly outperforms more conventional GCT encoding

Future work

- Enhance SMS to expand tasks more rapidly
- Eliminate hyper-parameter σ by changing structure of encoding
- Compare to recent related work [Behnke et al., 2018]

Conclusion

- Two new SAT encodings for totally ordered HTN planning
  GCT: Handles recursive subtask relationships
  SMS: Introduces incremental SAT solving to HTN planning
- Evaluation: Incremental SMS encoding significantly outperforms more conventional GCT encoding

Future work

- Enhance SMS to expand tasks more rapidly
- Eliminate hyper-parameter σ by changing structure of encoding
- Compare to recent related work [Behnke et al., 2018]

Thank you for your attention!

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*.

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.

Gocht, S. and Balyo, T. (2017).
Accelerating SAT based planning with incremental SAT solving.
Proceedings of the International Conference on Automated Planning and Scheduling, pages 135–139.

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.