On the Feasibility of Planning Graph Style Heuristics for HTN Planning

Ron Alford[1] Vikas Shivashankar[2] Ugur Kuter[3] Dana Nau[2]

[1]ASEE Postdoc
U.S. Naval Research Lab
Washington, DC, USA
ronwalf@volus.net

[2]Dept. of Computer Science
University of Maryland
College Park, MD, USA
(svikas|nau)@cs.umd.edu

[3]Smart Information Flow Technologies
Minneapolis, MN, USA
ukuter@sift.net

24th International Conference on Automated Planning and Scheduling
June 24th, 2014

Hierarchical Task Network (HTN) Planning

- HTN planning is the problem of decomposing an initial task to accomplish into a sequence of executable steps.
- Example applications of HTN planning:
  - Encoding control knowledge for planning
    - Agent planning in robotics and simulation: ARTUE (2010), Johnny (2012)
    - Planning and mission generation for games: KILLZONE 2, Elder Scrolls, Armed Assault)
  - Representing and planning with processes and hierarchies
    - Composing web services (Sirin, 2004; Tang 2013; others)
    - Program configuration (Soltani, 2012)
- More expressive than classical planning
  - Can express undecidable problems
- How does this expressivity affect what HTN heuristics can (and cannot) exist?

HTN Heuristics (Motivation)

Domain independent heuristics in search:

- Problem spaces in planning are intractably large to search
- A heuristic underestimates the distance to nearest solution, or we pretend it does
- Usually based on solving a relaxed, ground (propositional) version of the problem
  - Relax the semantics of what it means to be a solution
  - Relaxation should take low-order polynomial time to solve
  - Solution to full problem implies solution to relaxed problem (but not vice versa)
- There is currently only one (state-independent) HTN heuristic

HTN Heuristics (Motivation)

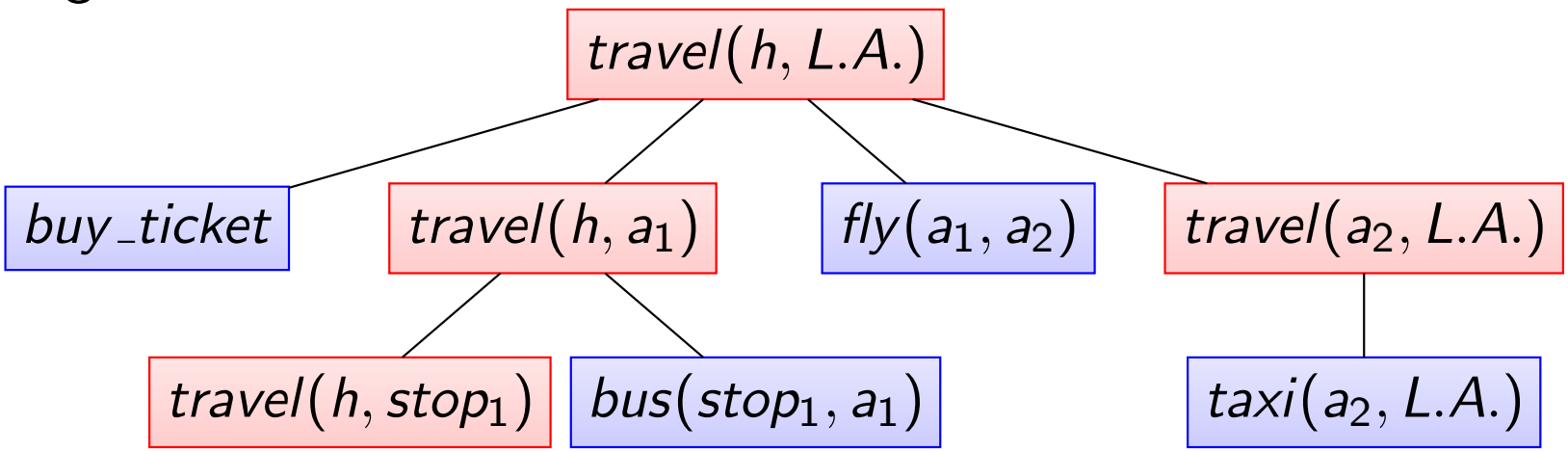
Domain independent heuristics in search:

- Problem spaces in planning are intractably large to search
- A heuristic underestimates the distance to nearest solution, or we pretend it does
- Usually based on solving a relaxed, ground (propositional) version of the problem
  - Relax the semantics of what it means to be a solution
  - Relaxation should take low-order polynomial time to solve
  - Solution to full problem implies solution to relaxed problem (but not vice versa)
- ~~There is currently only one (state-independent) HTN heuristic~~
- There are only two HTN heuristics
  - Bercher et. al. SoCS 2014

HTN Planning (Overview)

The purpose of HTN planning is to complete a *task*. Tasks are either

- *Primitive*, which corresponds to some concrete action we know how to perform
  - E.g: $walk(room, hall)$, or $drink(coffee)$
- *Non-primitive*, which is an abstract task. E.g. $travel(home, L.A.)$
- Must recursively decompose non-primitive tasks until we get primitive tasks we know how to execute directly
- We are given a set of *methods*, which are recipes on how to accomplish abstract tasks. E.g., to travel from *home* to *L.A.*, we might



Propositional HTN Planning: $D = (\mathbf{O}, M)$, $P = (D, \mathbf{s}_0, tn_0)$

Every HTN problem P contains an *initial state* s_0, which is a collection of propositions

Actions in our domain change the state. Actions have:

- A name (the task)
- A precondition (logical formula over the state)
  - Condition must hold for the action to be executed
- An effect on the state
  - A set of propositions to add
  - A set of propositions to delete

State:

$\{thirsty, tired\}$

Operator: *sleep*

pre $= tired \wedge \neg thirsty$

eff $= \{\neg tired\}$

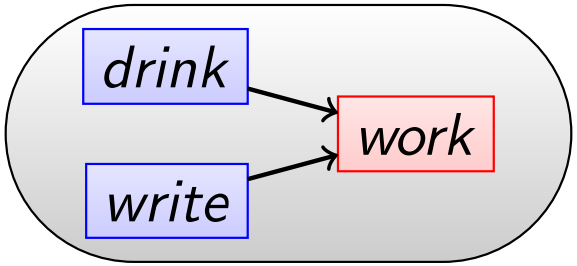
Operator: *drink*

pre $= true$

eff $= \{\neg thirsty\}$

HTN Planning: $D = (O, \mathbf{M})$, $P = (D, s_0, \mathbf{tn}_0)$

- An initial *task network*, tn_0, which is a labeled DAG of *tasks* to accomplish:
  - Edges represent ordering constraints
  - Nodes are labeled with *tasks*, which are either:
    - Primitive tasks (i.e., action names from O: $\{drink, write\}$)
    - Non-primitive tasks (say, $\{work\}$)

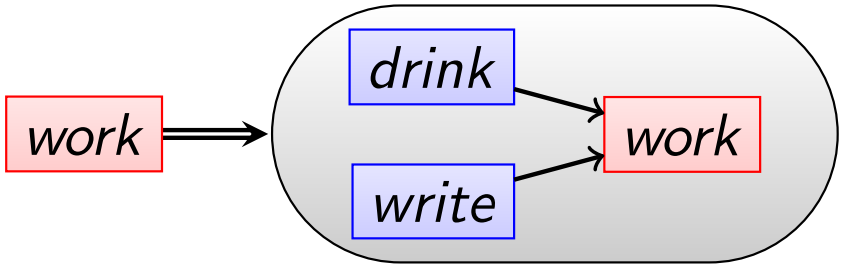


- M, a set of methods to *decompose* non-primitive tasks

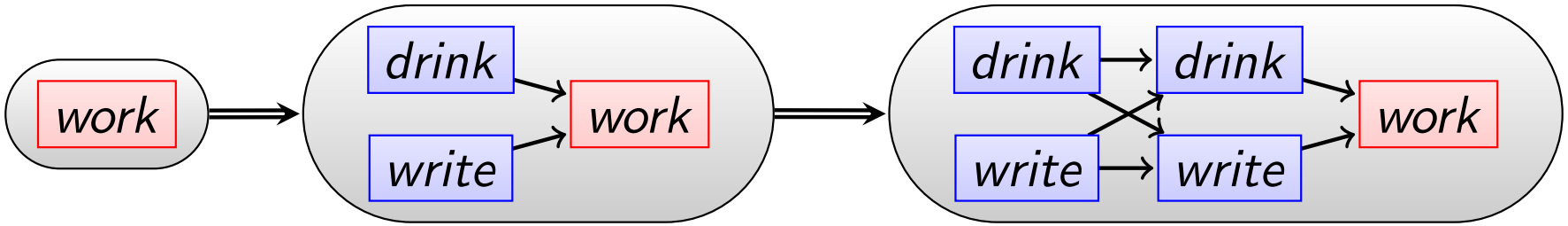
Methods and Decomposition

- A method (t, tn) is a non-primitive task t paired with a network tn

Method:

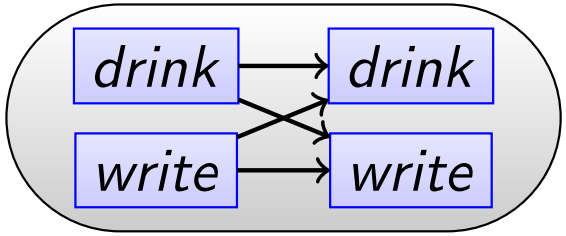


- We *decompose* a task network by replacing a node in the network with a corresponding method's network.



Primitive Task Networks

- Two states:
  - $thirsty, \neg thirsty$
- Two actions:
  - $\forall_s \gamma(s, write) = thirsty$
  - $\forall_s \gamma(s, drink) = \neg thirsty$



- All names are actions $\Rightarrow$ network is *primitive*
- It is *executable* in a state if there exists a consistent total order (sequence) which is executable.
- Objective: Decompose initial network to a primitive task network that is executable in s_0.

Classical (Propositional STRIPS) Planning

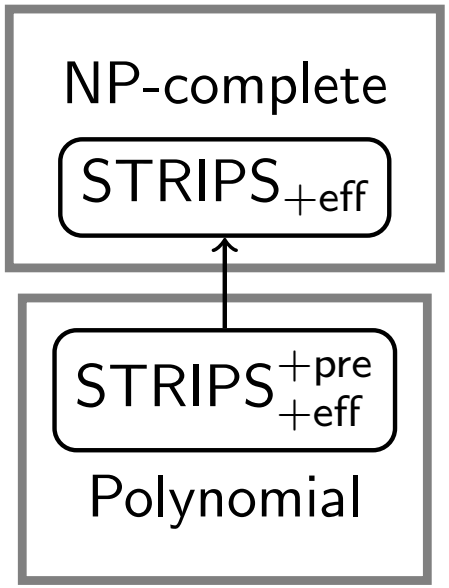
Differences between HTN and classical planning:

- $P = (O, s_0, G)$
- No methods and no initial task network.
- Objective: Find a plan (any executable sequence) that take us to a goal state.
  - No restriction (other than executability) on when planner can insert an action
- PSPACE-complete for propositional problems

Delete-Free STRIPS

Planning in propositional delete-free STRIPS (Bylander, 1994)

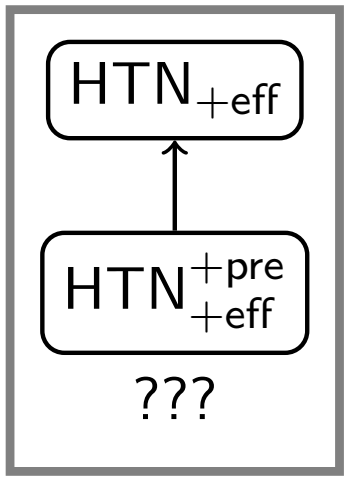
- $\text{STRIPS}^{+\text{pre}}_{+\text{eff}}$ (FF Relaxed GraphPlan)
  - All operators have positive preconditions and effects; positive goal
  - Poly-time to find a plan: just apply operators until a fixed point is reached
  - NP-hard to find optimal plan
  - Basis for many influential classical planning heuristics
- $\text{STRIPS}_{+\text{eff}}$
  - All operators have positive effects. Arbitrary goal and precondition
  - NP-complete to find a plan (same for optimality)



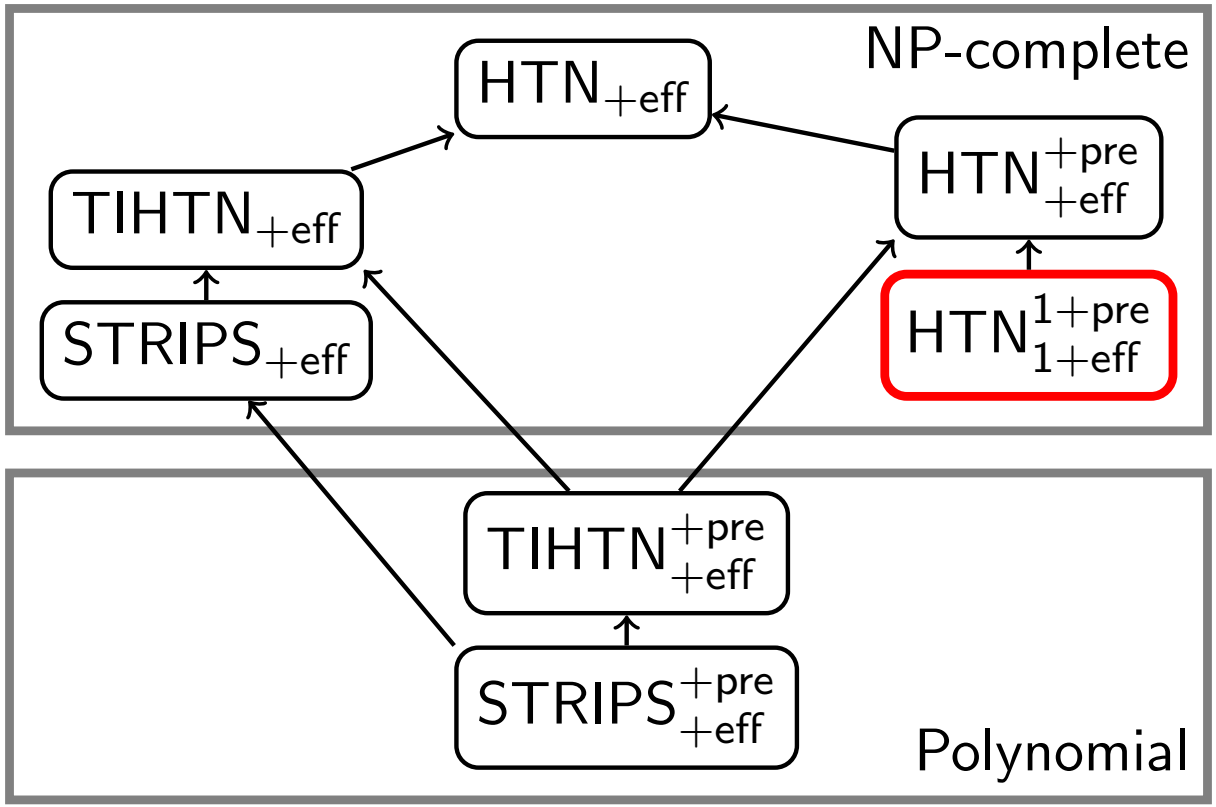
Delete-Free HTN Planning

Planning in delete-free propositional HTN domains:

- $\text{HTN}^{+\text{pre}}_{+\text{eff}}$: Positive preconditions and effects
- $\text{HTN}_{+\text{eff}}$: Positive effects
- Decidability and complexity previously unknown
- If either of these were in P, we could use them as the base of a heuristic



HTN_{1+eff}^{1+pre} is NP-hard



Almost all delete-free HTN planning is NP-hard, even the case where:

- Actions have at most 1 positive precondition and effect
- Methods are acyclic, regular (1 non-primitive task), and totally-ordered

Encoding CNF-SAT in $\text{HTN}^{1+\text{pre}}_{1+\text{eff}}$

CNF-SAT: $(V_1 \vee \neg V_3 \vee V_4) \wedge (V_2 \vee V_3 \vee \neg V_4) \wedge (\neg V_1 \vee \neg V_2 \vee \neg V_3)$
For each variable V_i

- Two propositions: V_i, nV_i
- Two methods corresponding to truth values of V_i
  - Method: 

  - Method: 

  - Method choice determines which of V_i, nV_i is set.
- For last variable, replace last task with $check\_E_1$, instead

V_1 V_2 V_3 V_4

nV_1 nV_2 nV_3 nV_4

Encoding CNF-SAT in $\text{HTN}^{1+\text{pre}}_{1+\text{eff}}$

CNF-SAT: $(V_1 \vee \neg V_3 \vee V_4) \wedge (V_2 \vee V_3 \vee \neg V_4) \wedge (\neg V_1 \vee \neg V_2 \vee \neg V_3)$

For each variable V_i

- Two propositions: V_i, nV_i
- Two methods corresponding to truth values of V_i
  - Method:

  - Method:

  - Method choice determines which of V_i, nV_i is set.
- For last variable, replace last task with $check\_E_1$, instead

$$V_1 \quad V_2 \quad V_3 \quad V_4$$

$$nV_1 \quad nV_2 \quad nV_3 \quad nV_4$$

Encoding CNF-SAT in $\text{HTN}^{1+\text{pre}}_{1+\text{eff}}$

CNF-SAT: $(V_1 \vee \neg V_3 \vee V_4) \wedge (V_2 \vee V_3 \vee \neg V_4) \wedge (\neg V_1 \vee \neg V_2 \vee \neg V_3)$

For each variable V_i

- Two propositions: V_i, nV_i
- Two methods corresponding to truth values of V_i
  - Method: $set\_V_i \Longrightarrow assert\_V_i \rightarrow set\_V_{i+1}$
  - Method: $set\_V_i \Longrightarrow assert\_nV_i \rightarrow set\_V_{i+1}$
  - Method choice determines which of V_i, nV_i is set.
- For last variable, replace last task with $check\_E_1$, instead

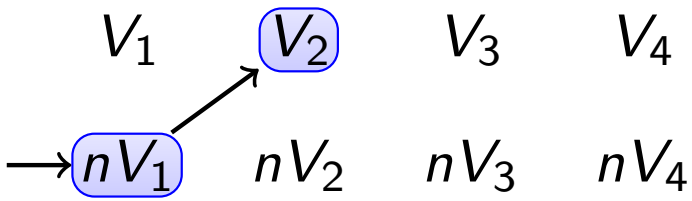
V_1	V_2	V_3	V_4
$\rightarrow nV_1$	nV_2	nV_3	nV_4

Encoding CNF-SAT in $\text{HTN}^{1+\text{pre}}_{1+\text{eff}}$

CNF-SAT: $(V_1 \vee \neg V_3 \vee V_4) \wedge (V_2 \vee V_3 \vee \neg V_4) \wedge (\neg V_1 \vee \neg V_2 \vee \neg V_3)$

For each variable V_i

- Two propositions: V_i, nV_i
- Two methods corresponding to truth values of V_i
  - Method: $set\_V_i \Longrightarrow assert\_V_i \rightarrow set\_V_{i+1}$
  - Method: $set\_V_i \Longrightarrow assert\_nV_i \rightarrow set\_V_{i+1}$
  - Method choice determines which of V_i, nV_i is set.
- For last variable, replace last task with $check\_E_1$, instead

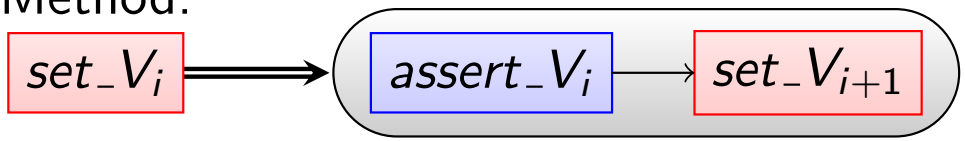


Encoding CNF-SAT in $\text{HTN}^{1+\text{pre}}_{1+\text{eff}}$

CNF-SAT: $(V_1 \vee \neg V_3 \vee V_4) \wedge (V_2 \vee V_3 \vee \neg V_4) \wedge (\neg V_1 \vee \neg V_2 \vee \neg V_3)$

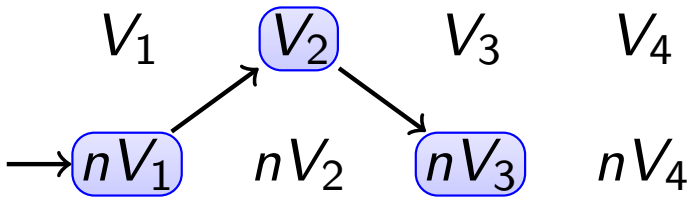
For each variable V_i

- Two propositions: V_i, nV_i
- Two methods corresponding to truth values of V_i
  - Method:



  - Method:

  - Method choice determines which of V_i, nV_i is set.
- For last variable, replace last task with $check\_E_1$, instead

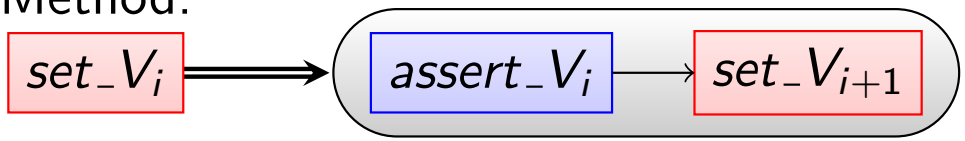


Encoding CNF-SAT in $\text{HTN}^{1+\text{pre}}_{1+\text{eff}}$

CNF-SAT: $(V_1 \vee \neg V_3 \vee V_4) \wedge (V_2 \vee V_3 \vee \neg V_4) \wedge (\neg V_1 \vee \neg V_2 \vee \neg V_3)$

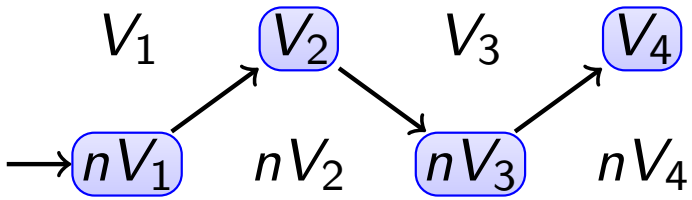
For each variable V_i

- Two propositions: V_i, nV_i
- Two methods corresponding to truth values of V_i
  - Method:



  - Method:

  - Method choice determines which of V_i, nV_i is set.
- For last variable, replace last task with $check\_E_1$, instead

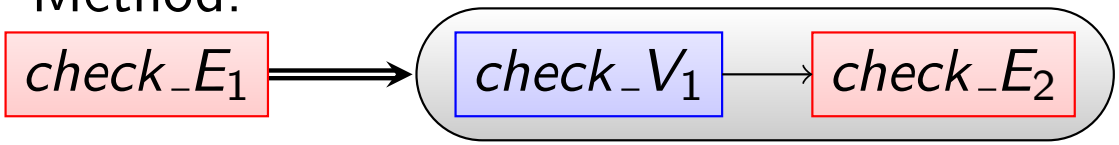


Delete-Free HTN planning is NP-hard

CNF-SAT: $(V_1 \vee \neg V_3 \vee V_4) \wedge (V_2 \vee V_3 \vee \neg V_4) \wedge (\neg V_1 \vee \neg V_2 \vee \neg V_3)$

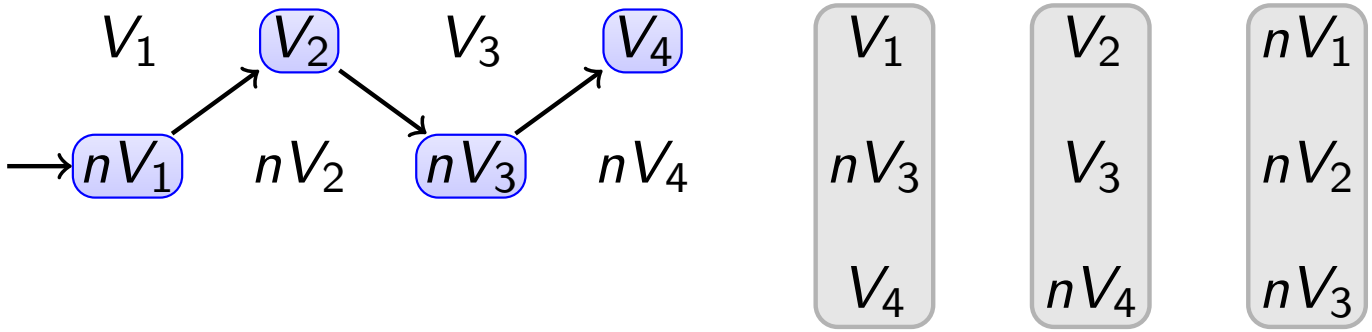
For disjunctive clause E_i, methods test whether at least one term is true:

- Method:



- Method:

- Method:

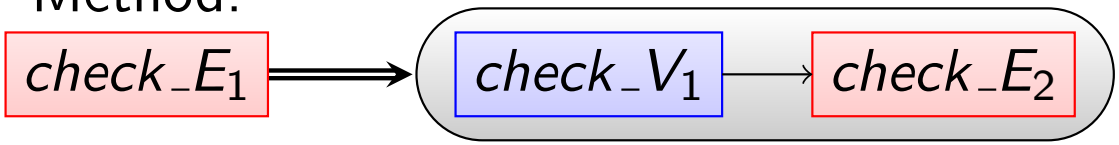


Delete-Free HTN planning is NP-hard

CNF-SAT: $(V_1 \vee \neg V_3 \vee V_4) \wedge (V_2 \vee V_3 \vee \neg V_4) \wedge (\neg V_1 \vee \neg V_2 \vee \neg V_3)$

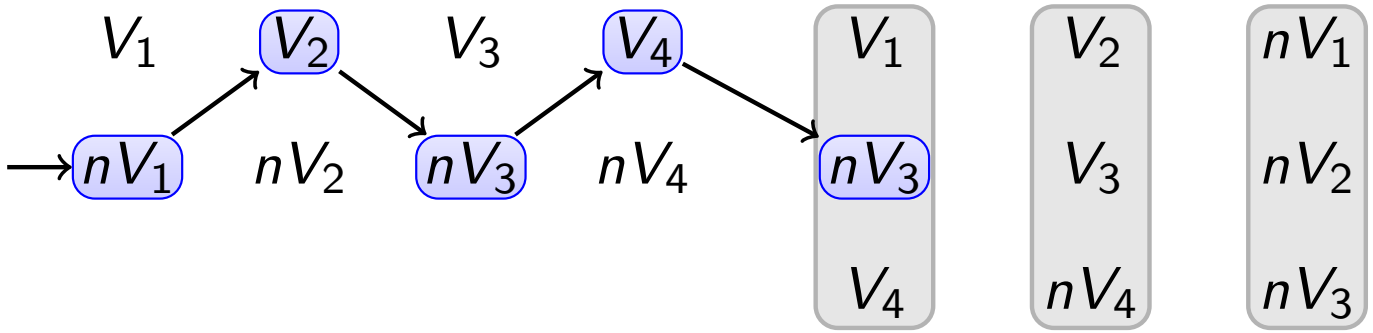
For disjunctive clause E_i, methods test whether at least one term is true:

- Method:



- Method:

- Method:

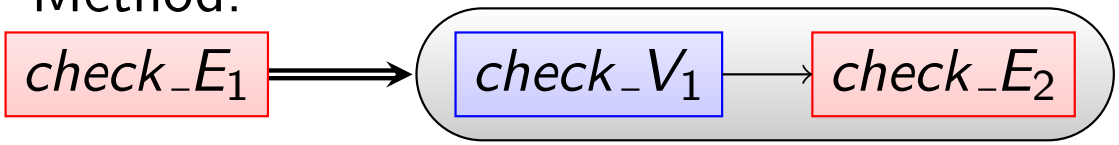


Delete-Free HTN planning is NP-hard

CNF-SAT: $(V_1 \vee \neg V_3 \vee V_4) \wedge (V_2 \vee V_3 \vee \neg V_4) \wedge (\neg V_1 \vee \neg V_2 \vee \neg V_3)$

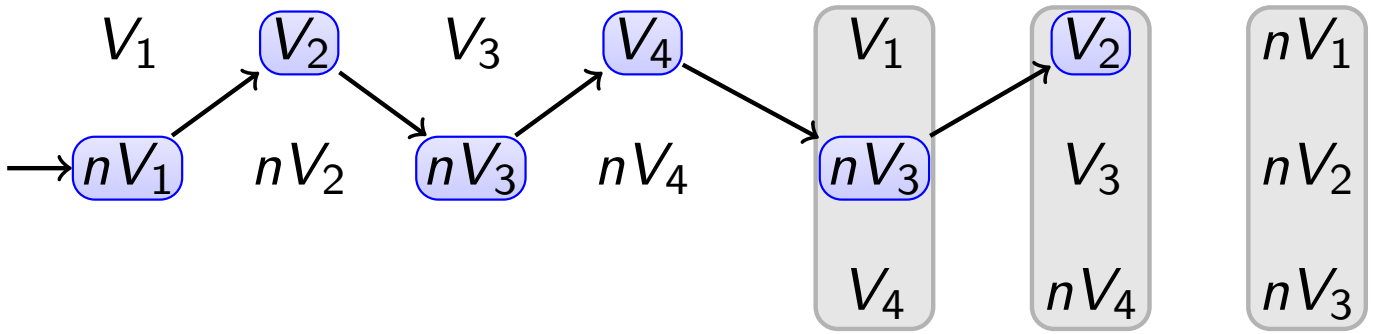
For disjunctive clause E_i, methods test whether at least one term is true:

- Method:



- Method:

- Method:



Delete-Free HTN planning is NP-hard

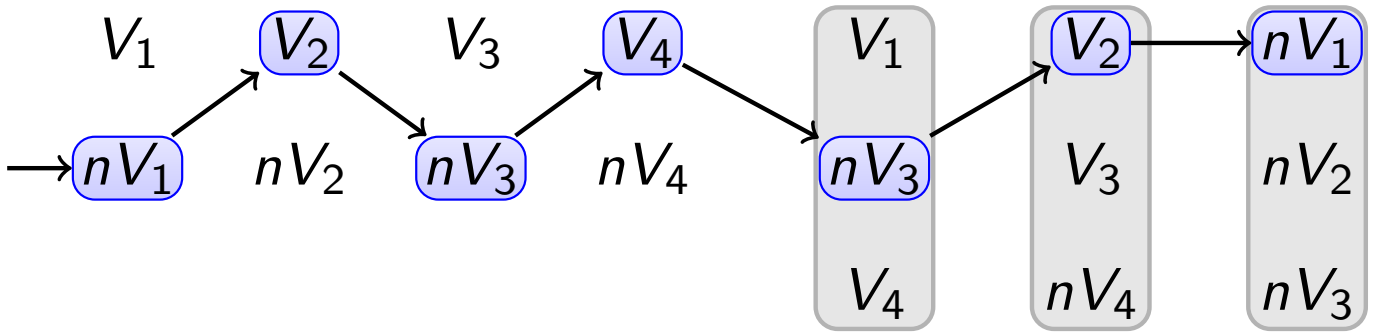
CNF-SAT: $(V_1 \vee \neg V_3 \vee V_4) \wedge (V_2 \vee V_3 \vee \neg V_4) \wedge (\neg V_1 \vee \neg V_2 \vee \neg V_3)$

For disjunctive clause E_i, methods test whether at least one term is true:

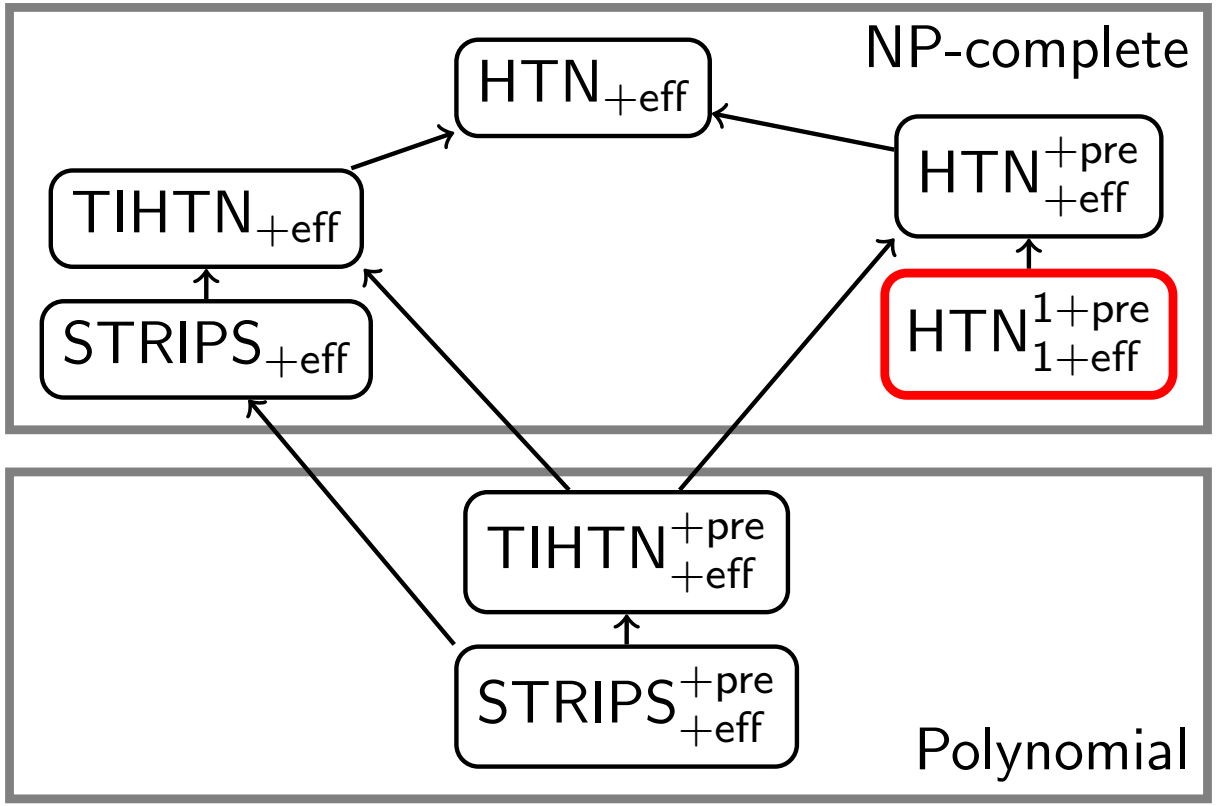
- Method:

- Method:

- Method:

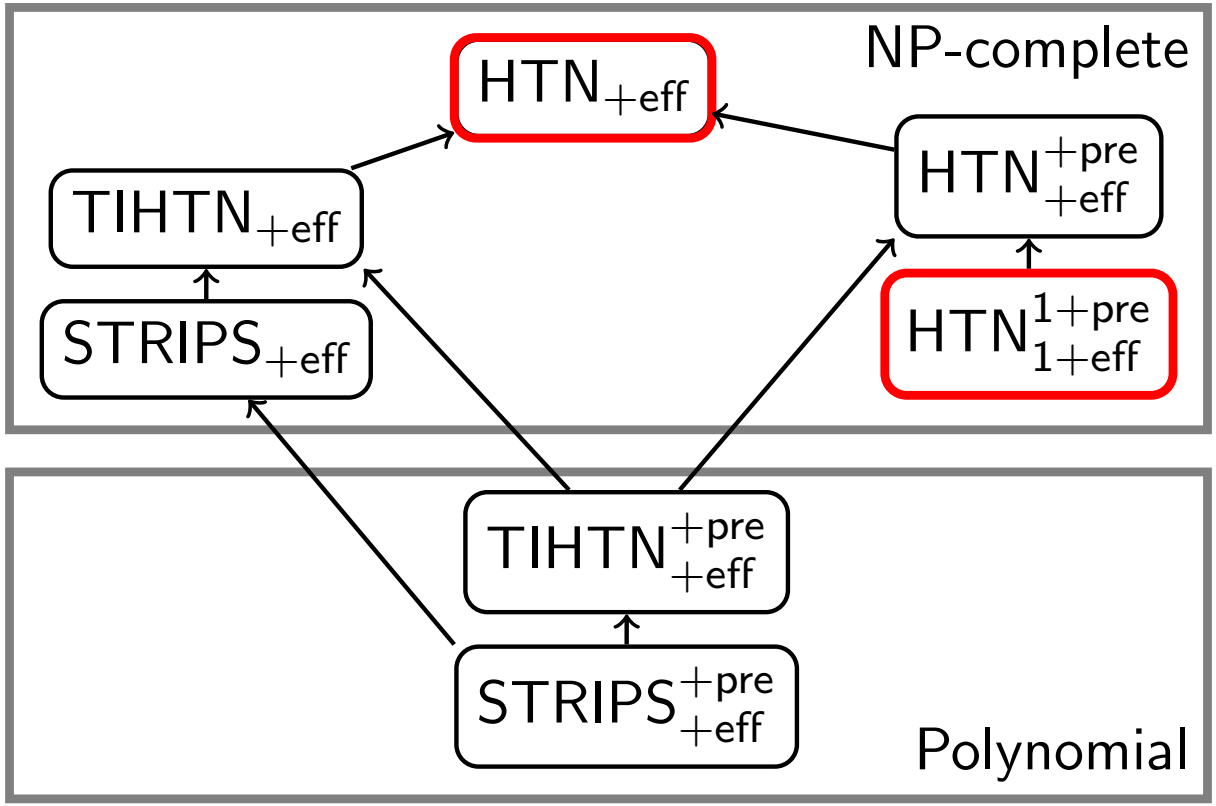


NP-Hardness and Completeness



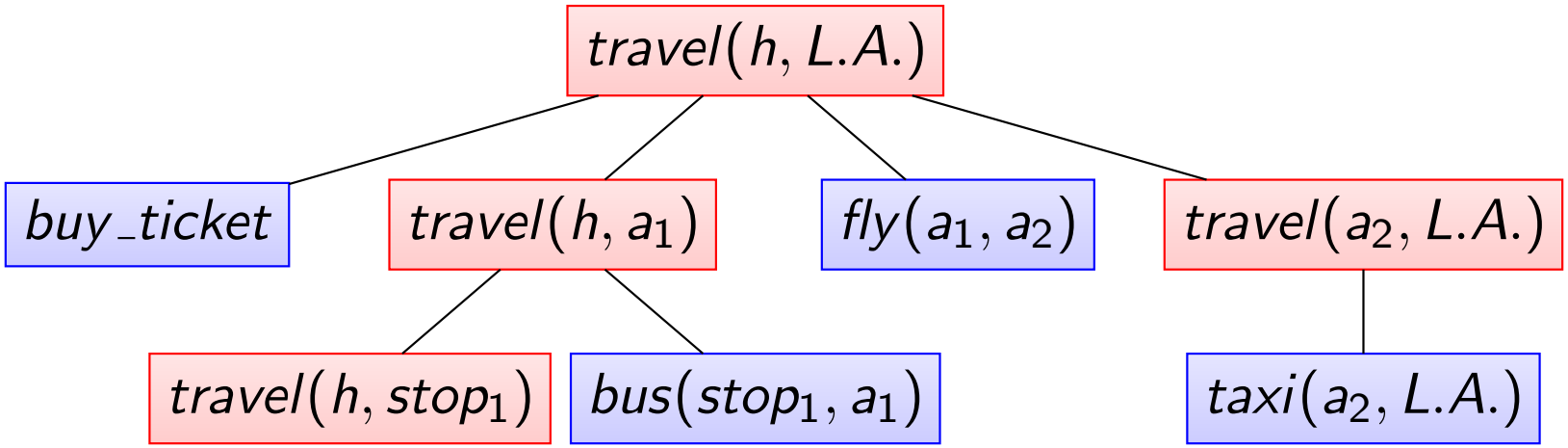
- Even highly restricted delete-free HTN planning is NP-hard
- Now we show that every solvable delete-free HTN problem has a polynomial-size witness

NP-Hardness and Completeness



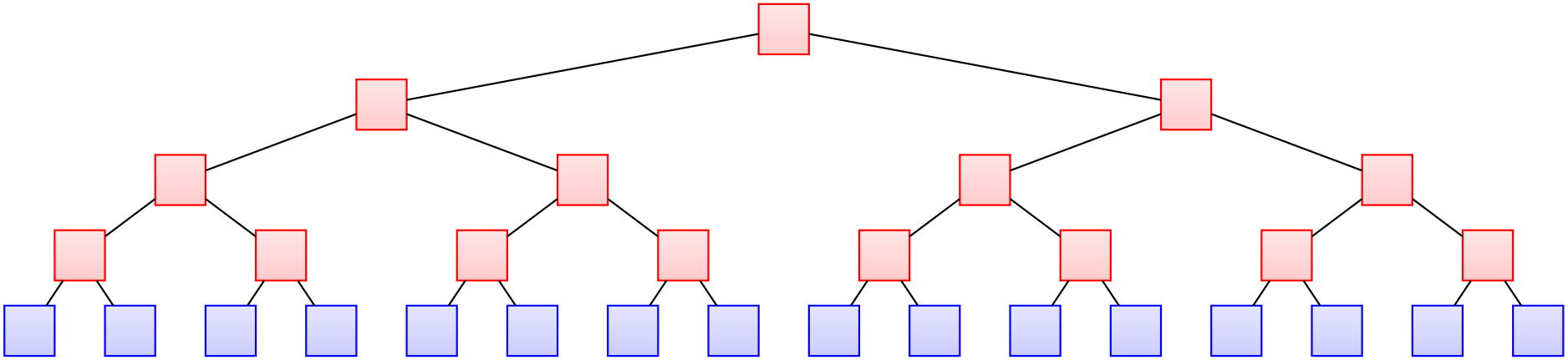
- Even highly restricted delete-free HTN planning is NP-hard
- Now we show that every solvable delete-free HTN problem has a polynomial-size witness

HTN Decomposition Trees



- History of decompositions forms a tree where the leaves are the current task network (Geier and Bercher, 2011)
  - If that task network is executable, this tree is a proof that the problem is solvable
- Even if no actions change the state, min tree size may still be exponential
  - To show $\text{HTN}_{+\text{eff}} \in$ NP, we need a polynomial size witness

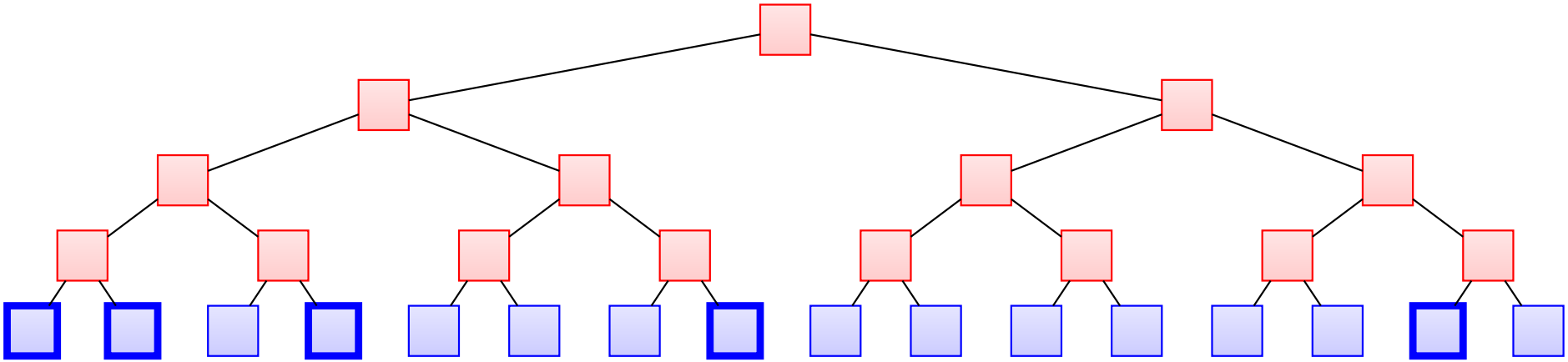
A Polynomial-Size Witness for $\text{HTN}_{+\text{eff}}$



Given a solution:

- In delete-free planning, only as many actions as there are propositions can change the state
- Prune tree to the minimal tree that contains those actions
- Construct a poly-size table to show the pruned tree has an executable expansion
- Use a pumping lemma to shorten the tree to polynomial height
- Poly height, poly width tree and a poly-size table $\implies$ $\text{HTN}_{+\text{eff}} \in \text{NP}$

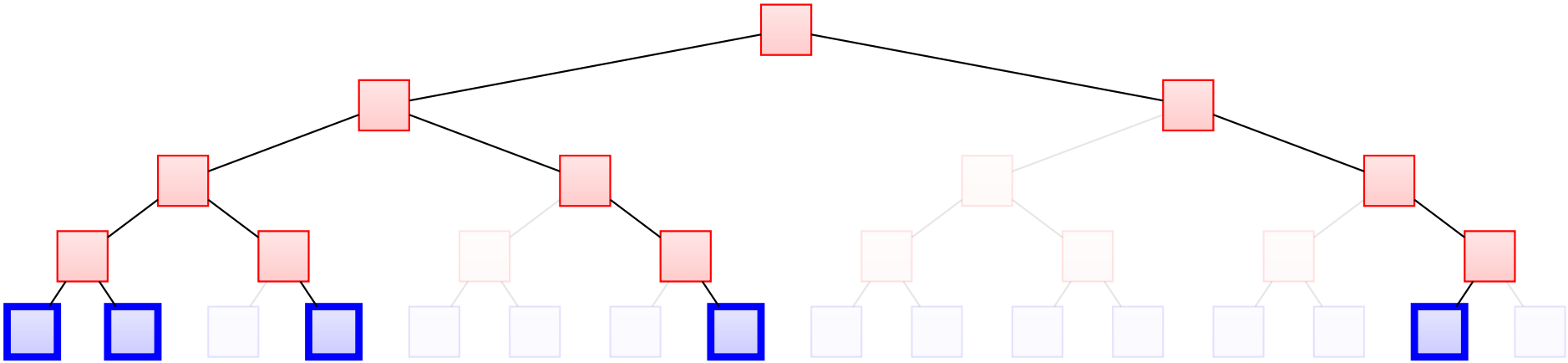
A Polynomial-Size Witness for $\text{HTN}_{+\text{eff}}$



Given a solution:

- In delete-free planning, only as many actions as there are propositions can change the state
- Prune tree to the minimal tree that contains those actions
- Construct a poly-size table to show the pruned tree has an executable expansion
- Use a pumping lemma to shorten the tree to polynomial height
- Poly height, poly width tree and a poly-size table $\implies$ $\text{HTN}_{+\text{eff}} \in \text{NP}$

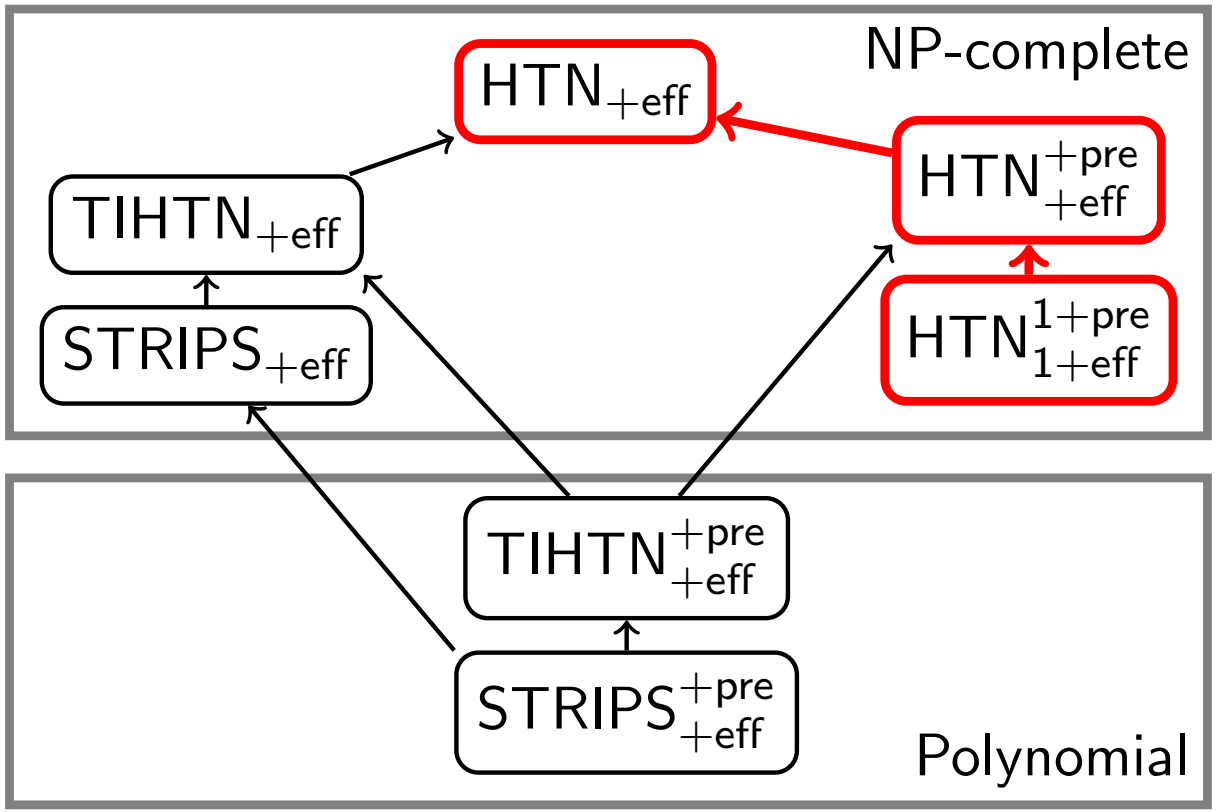
A Polynomial-Size Witness for $\text{HTN}_{+\text{eff}}$



Given a solution:

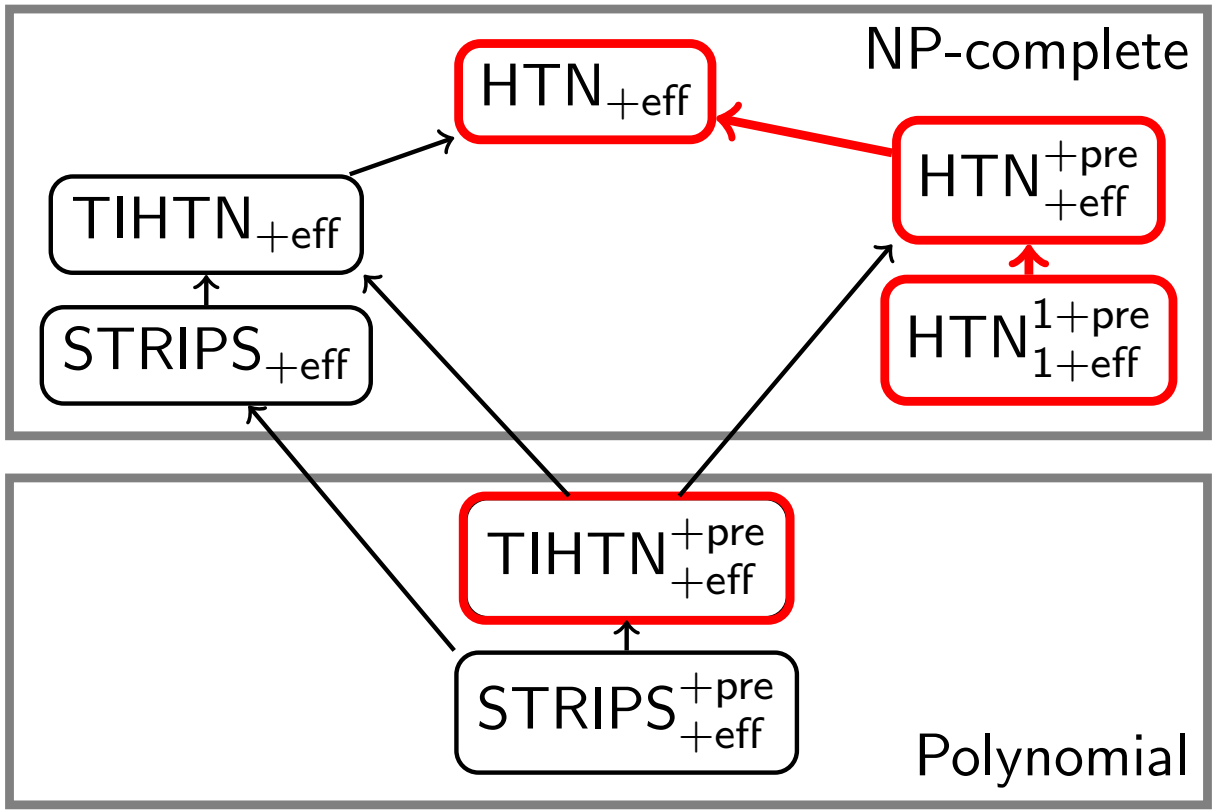
- In delete-free planning, only as many actions as there are propositions can change the state
- Prune tree to the minimal tree that contains those actions
- Construct a poly-size table to show the pruned tree has an executable expansion
- Use a pumping lemma to shorten the tree to polynomial height
- Poly height, poly width tree and a poly-size table $\implies$ $\text{HTN}_{+\text{eff}} \in \text{NP}$

Delete-Free HTN planning is NP-complete



- HTN^{+pre}_{+eff} has a poly-size witness and can encode SAT, so it is NP-complete
- Just ignoring the deletes and negative preconditions is not enough to create an HTN heuristic - we must further relax the semantics

Delete-Free HTN planning is NP-complete



- HTN^{+pre}_{+eff} has a poly-size witness and can encode SAT, so it is NP-complete
- Just ignoring the deletes and negative preconditions is not enough to create an HTN heuristic - we must further relax the semantics

Delete-Free Task-Insertion HTN Planning

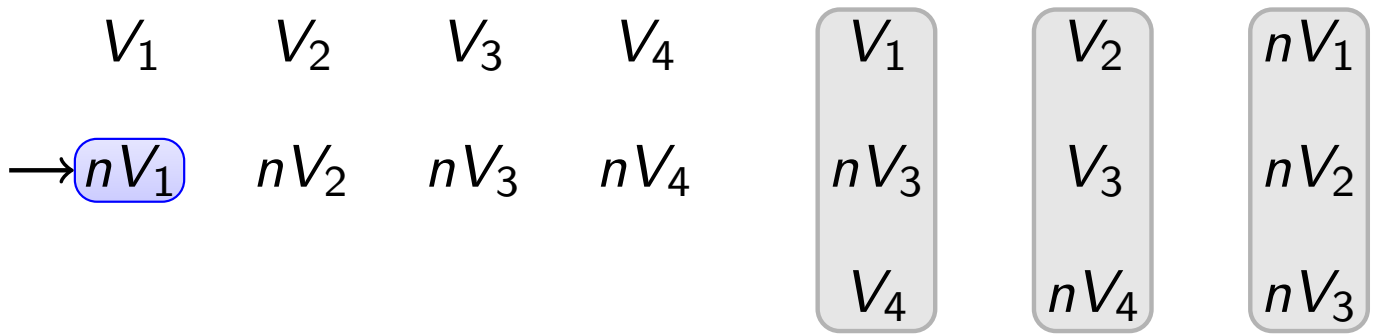
- Task-Insertion HTN (TIHTN) Planning (Geier & Bercher, 2011):
  - Allows insertion of arbitrary operators into task network
- $\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$ planning is poly-time:
  - Insert operators until you reach a fixpoint
  - Test to see if tasks are executable in the fixpoint state (similar to testing if a CFG is empty).
- Poly-time $\implies$ $\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$ is a candidate for an HTN heuristic.
- We can't reduce SAT to $\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$:

$$V_1 \quad V_2 \quad V_3 \quad V_4$$
$$nV_1 \quad nV_2 \quad nV_3 \quad nV_4$$

$$\begin{matrix} V_1 \\ nV_3 \\ V_4 \end{matrix} \qquad \begin{matrix} V_2 \\ V_3 \\ nV_4 \end{matrix} \qquad \begin{matrix} nV_1 \\ nV_2 \\ nV_3 \end{matrix}$$

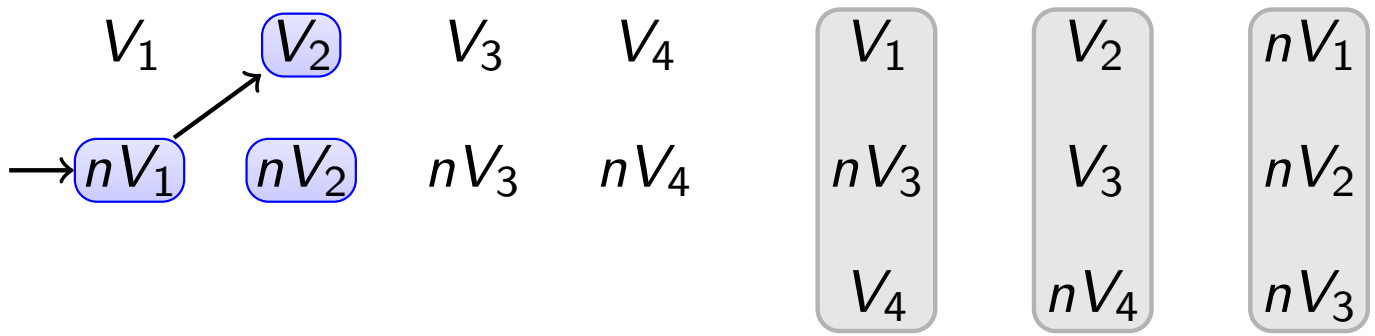
Delete-Free Task-Insertion HTN Planning

- Task-Insertion HTN (TIHTN) Planning (Geier & Bercher, 2011):
  - Allows insertion of arbitrary operators into task network
- $\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$ planning is poly-time:
  - Insert operators until you reach a fixpoint
  - Test to see if tasks are executable in the fixpoint state (similar to testing if a CFG is empty).
- Poly-time $\implies$ $\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$ is a candidate for an HTN heuristic.
- We can't reduce SAT to $\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$:



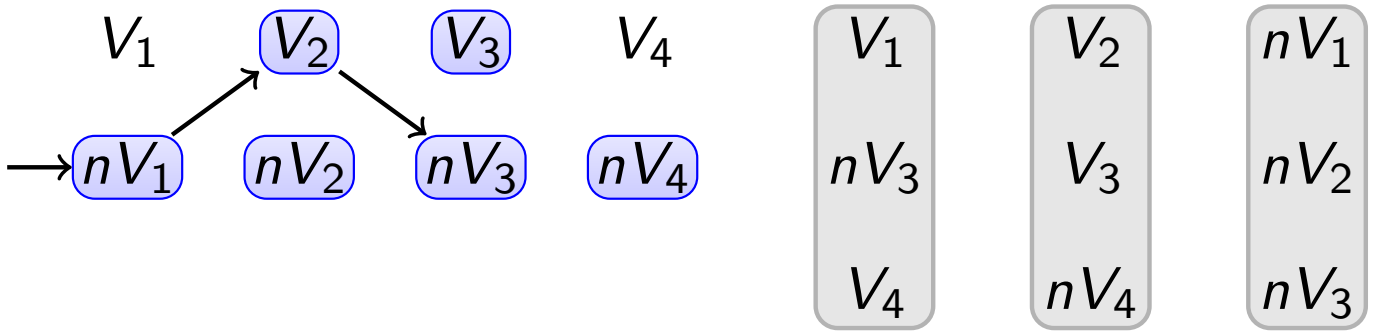
Delete-Free Task-Insertion HTN Planning

- Task-Insertion HTN (TIHTN) Planning (Geier & Bercher, 2011):
  - Allows insertion of arbitrary operators into task network
- $\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$ planning is poly-time:
  - Insert operators until you reach a fixpoint
  - Test to see if tasks are executable in the fixpoint state (similar to testing if a CFG is empty).
- Poly-time $\implies$ $\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$ is a candidate for an HTN heuristic.
- We can't reduce SAT to $\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$:



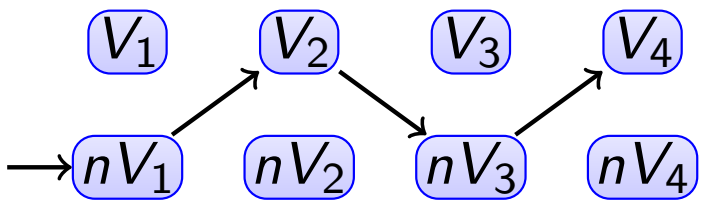
Delete-Free Task-Insertion HTN Planning

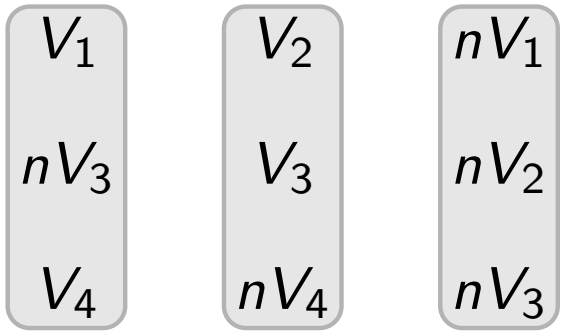
- Task-Insertion HTN (TIHTN) Planning (Geier & Bercher, 2011):
  - Allows insertion of arbitrary operators into task network
- $\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$ planning is poly-time:
  - Insert operators until you reach a fixpoint
  - Test to see if tasks are executable in the fixpoint state (similar to testing if a CFG is empty).
- Poly-time $\implies$ $\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$ is a candidate for an HTN heuristic.
- We can't reduce SAT to $\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$:



Delete-Free Task-Insertion HTN Planning

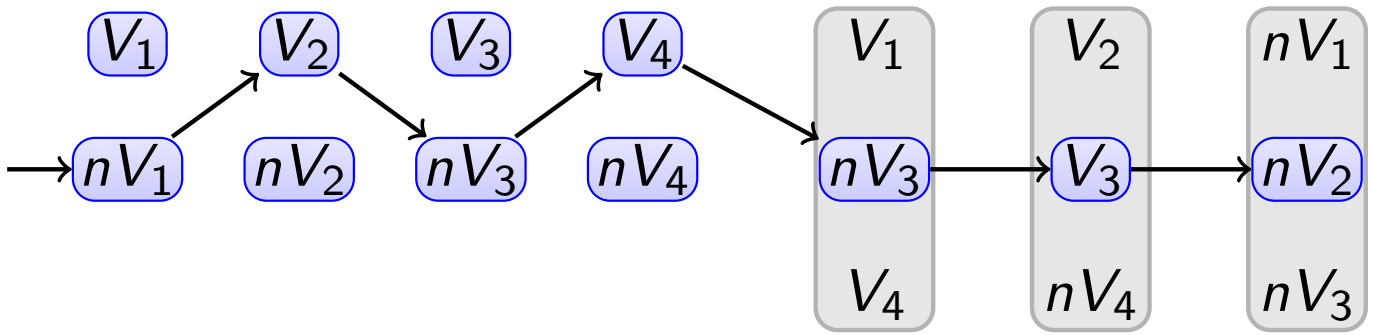
- Task-Insertion HTN (TIHTN) Planning (Geier & Bercher, 2011):
  - Allows insertion of arbitrary operators into task network
- $\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$ planning is poly-time:
  - Insert operators until you reach a fixpoint
  - Test to see if tasks are executable in the fixpoint state (similar to testing if a CFG is empty).
- Poly-time $\implies$ $\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$ is a candidate for an HTN heuristic.
- We can't reduce SAT to $\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$:



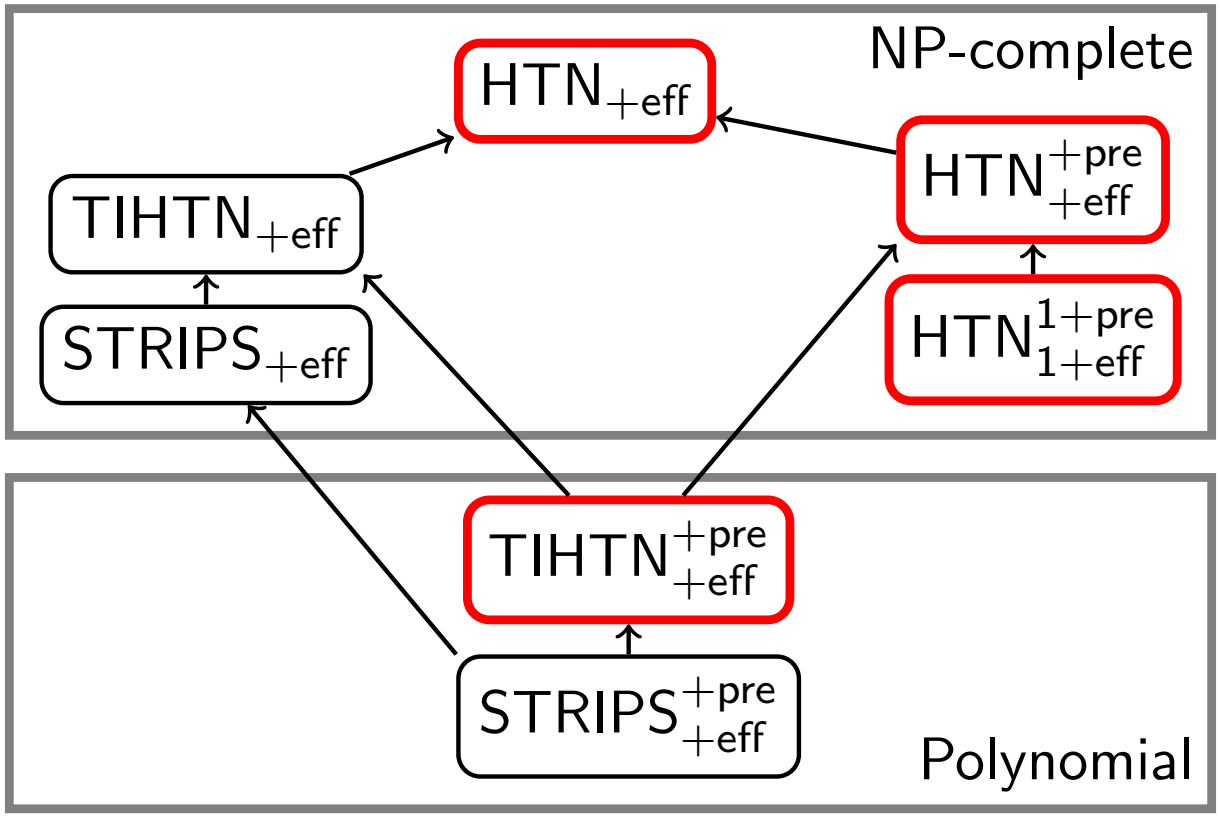


Delete-Free Task-Insertion HTN Planning

- Task-Insertion HTN (TIHTN) Planning (Geier & Bercher, 2011):
  - Allows insertion of arbitrary operators into task network
- $\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$ planning is poly-time:
  - Insert operators until you reach a fixpoint
  - Test to see if tasks are executable in the fixpoint state (similar to testing if a CFG is empty).
- Poly-time $\implies$ $\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$ is a candidate for an HTN heuristic.
- We can't reduce SAT to $\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$:



Conclusions



Delete-free HTN planning is NP-complete!

- Heuristics based on delete-free HTNs must further relax semantics
- $\text{TIHTN}^{+\text{pre}}_{+\text{eff}}$ (HTN planning with task insertion) is one candidate
- Need to explore the trade-offs between complexity & fidelity