Tight Bounds for HTN Planning with Task Insertion

Ron Alford[1] Pascal Bercher[2] David W. Aha[3]

[1]ASEE/NRL Postdoctoral Fellow
ronald.alford.ctr@nrl.navy.mil

[2]Ulm University
pascal.bercher@uni-ulm.de

[3]U.S. Naval Research Laboratory
david.aha@nrl.navy.mil

July 31st, 2015

Alternative Title

Theoretical applications of "2^{2^k} Bottles of Beer on the Wall"

Hierarchical Task Network (HTN) Planning

HTN planning is the problem of decomposing an initial task to accomplish into a sequence of executable steps.

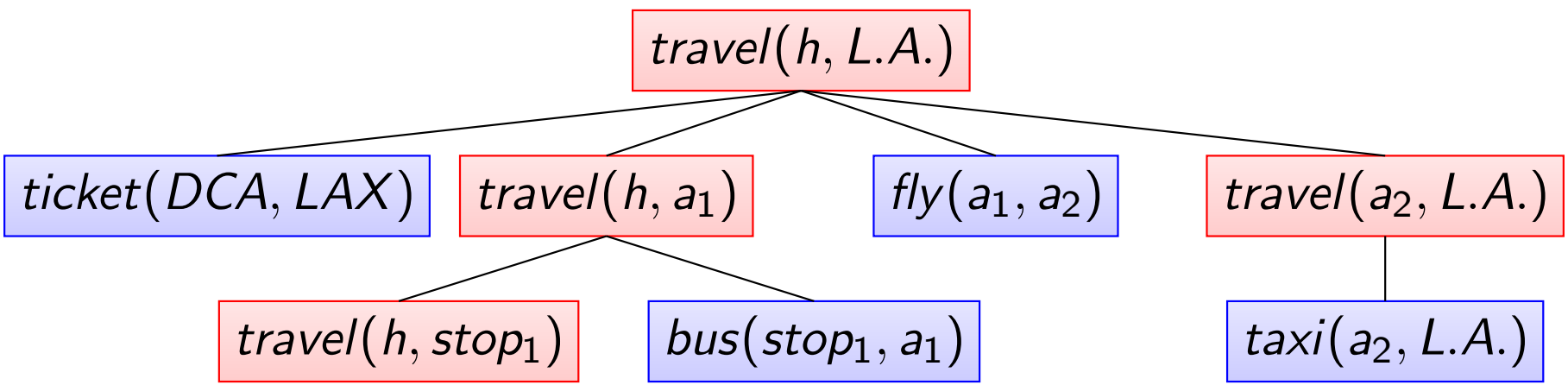
- Encoding control knowledge for planning
  - Agent planning in robotics and simulation: ARTUE (2010), Johnny (2012)
  - Planning and mission generation for games: KILLZONE 2, Elder Scrolls, Armed Assault
- Representing and planning with processes and hierarchies
  - Composing web services (Sirin, 2004; Tang 2013; others)
  - Program configuration (Soltani, 2012)

Source: Fraunhofer

HTN Planning (Overview)

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

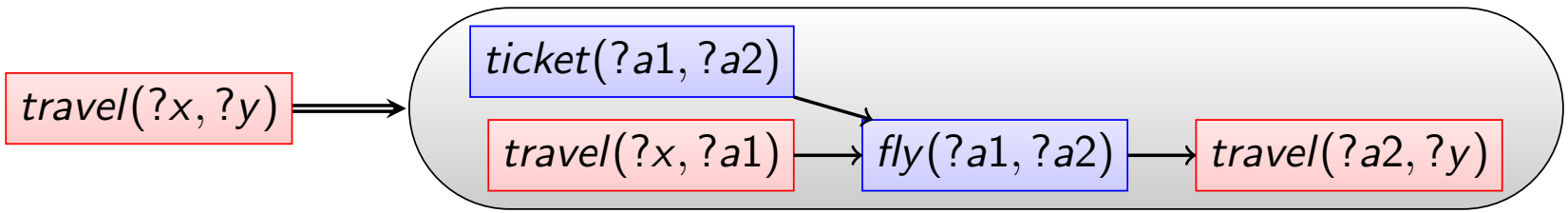
- Primitive, which corresponds to some concrete action we know how to perform, e.g., $walk(room, hall)$, or $drink(coffee)$
- Non-primitive, which is an abstract task. E.g. $travel(home, L.A.)$
- Must recursively decompose non-primitive tasks until we get primitive tasks we know how to execute directly
- 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 decompose as follows:



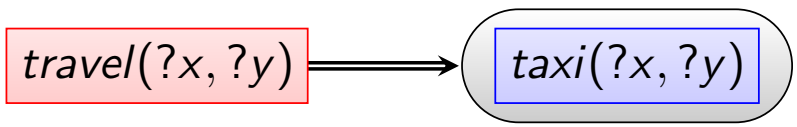
Methods and Decomposition

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

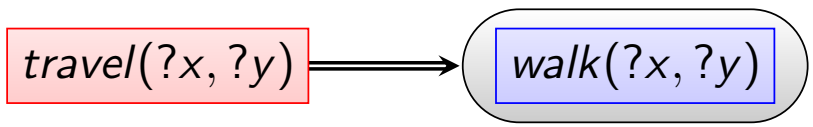
Method:



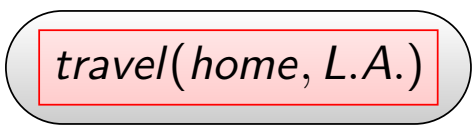
Method:



Method:



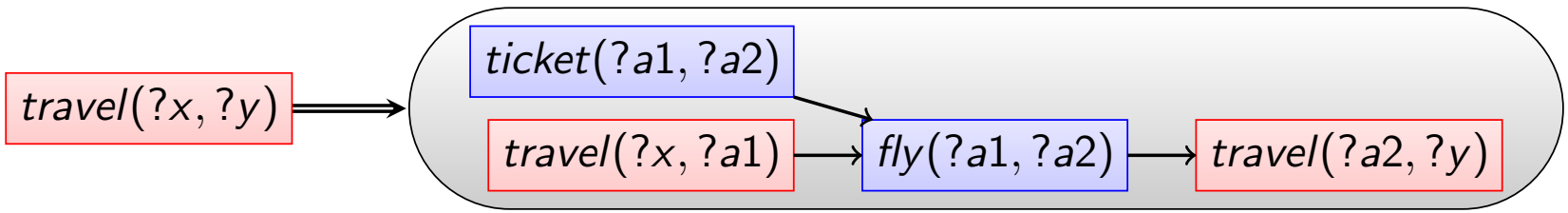
- We decompose a task network by replacing a node in the network with a corresponding method's network.



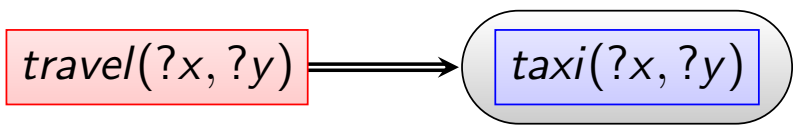
Methods and Decomposition

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

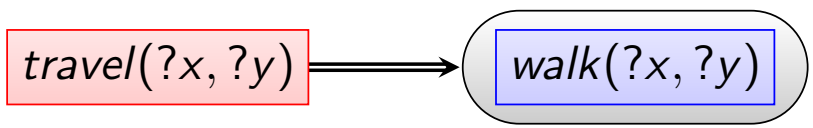
Method:



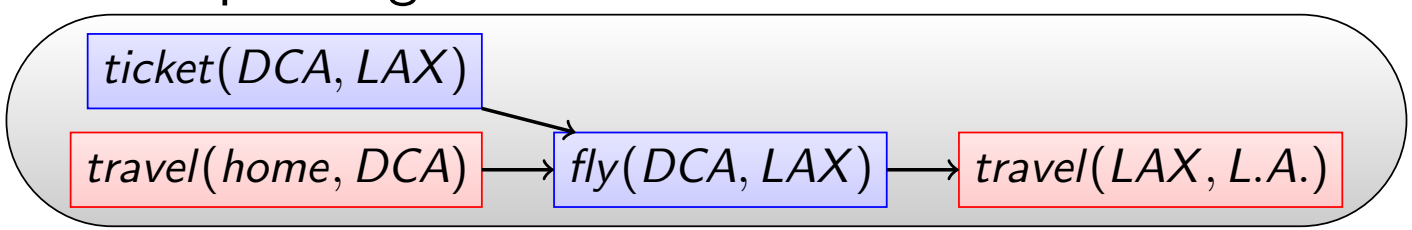
Method:



Method:



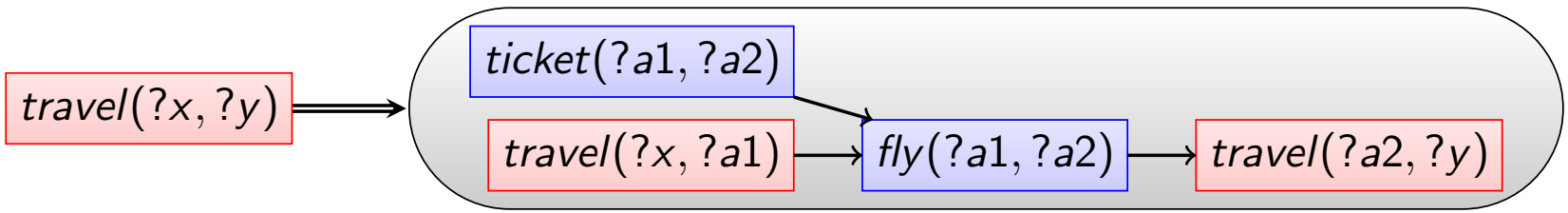
- We decompose a task network by replacing a node in the network with a corresponding method's network.



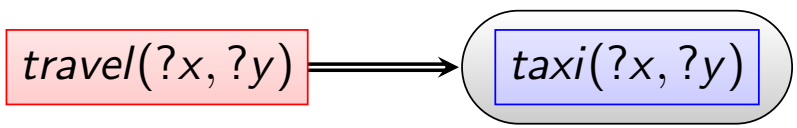
Methods and Decomposition

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

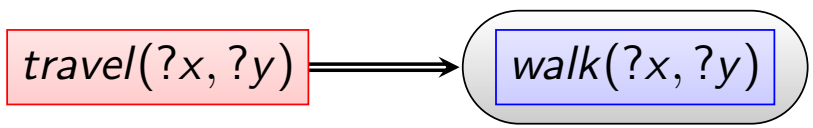
Method:



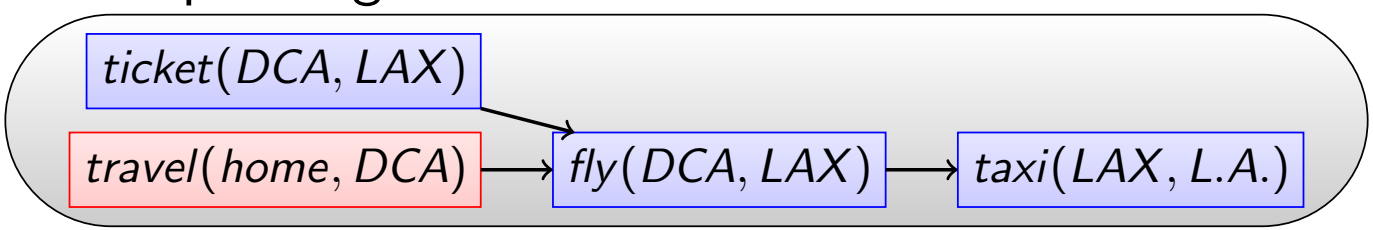
Method:



Method:



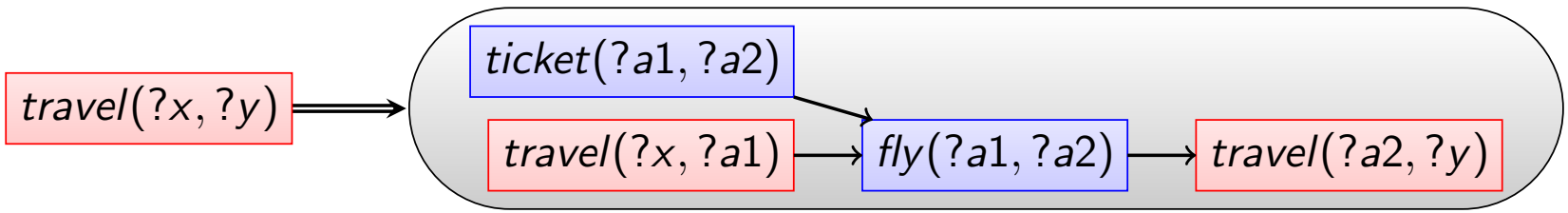
- We decompose a task network by replacing a node in the network with a corresponding method's network.



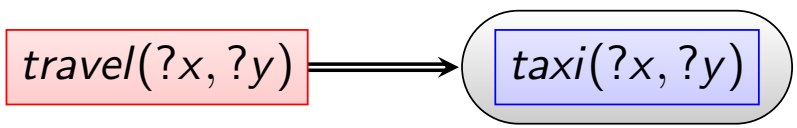
Methods and Decomposition

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

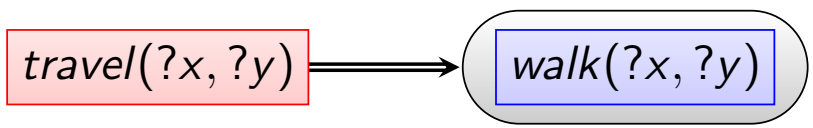
Method:



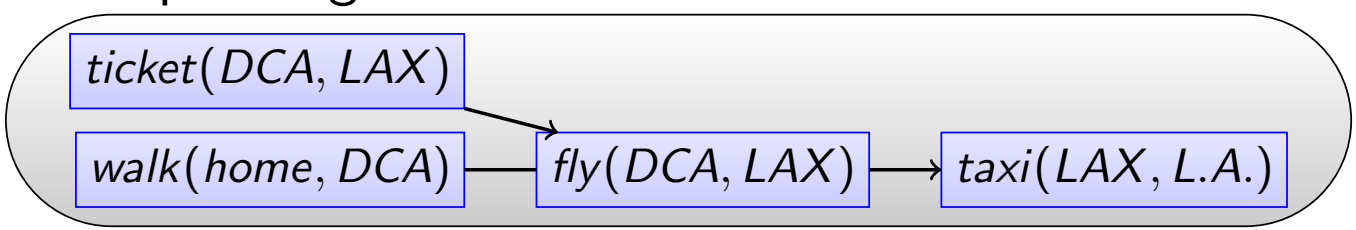
Method:



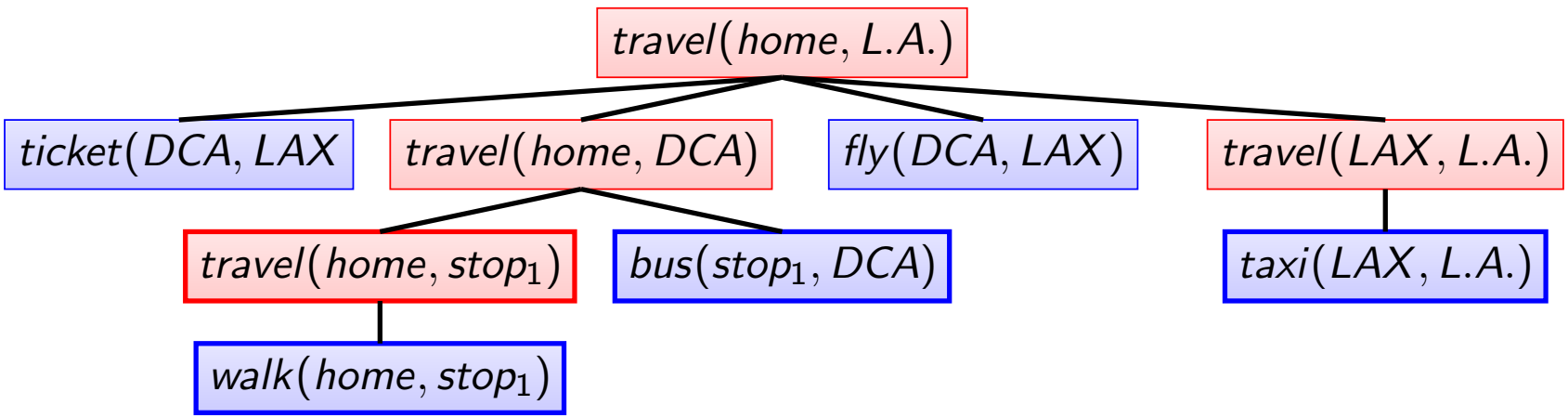
Method:



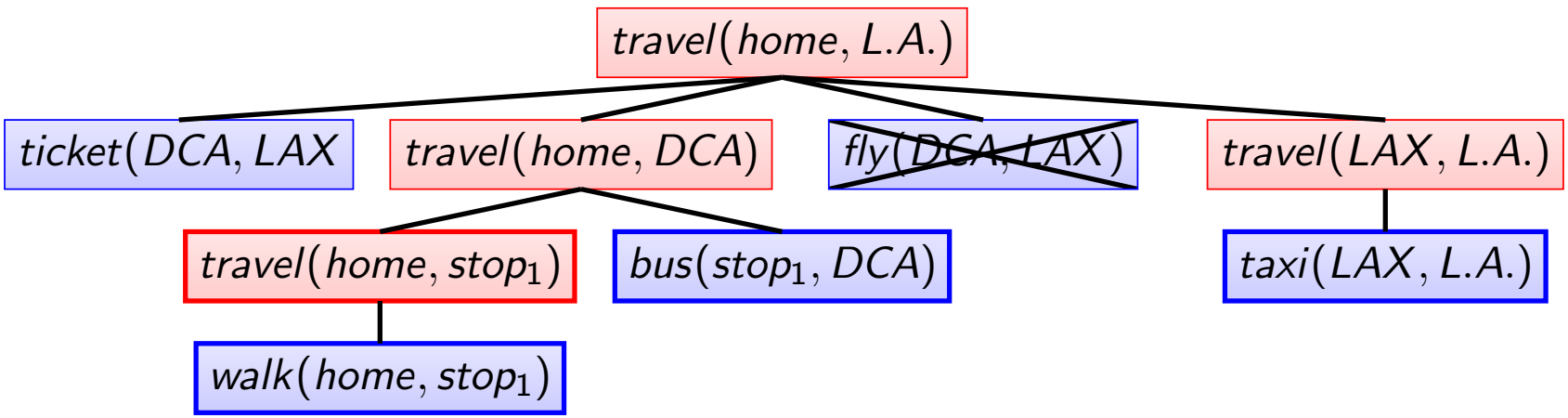
- We decompose a task network by replacing a node in the network with a corresponding method's network.



Replanning with HTNs

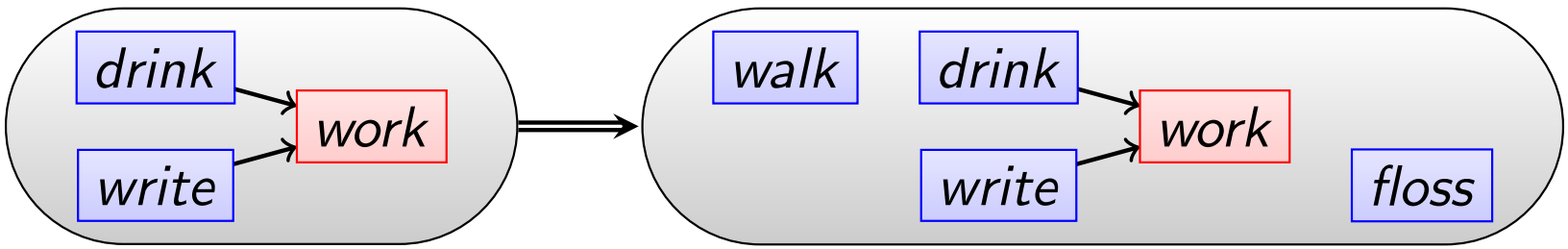


Replanning with HTNs



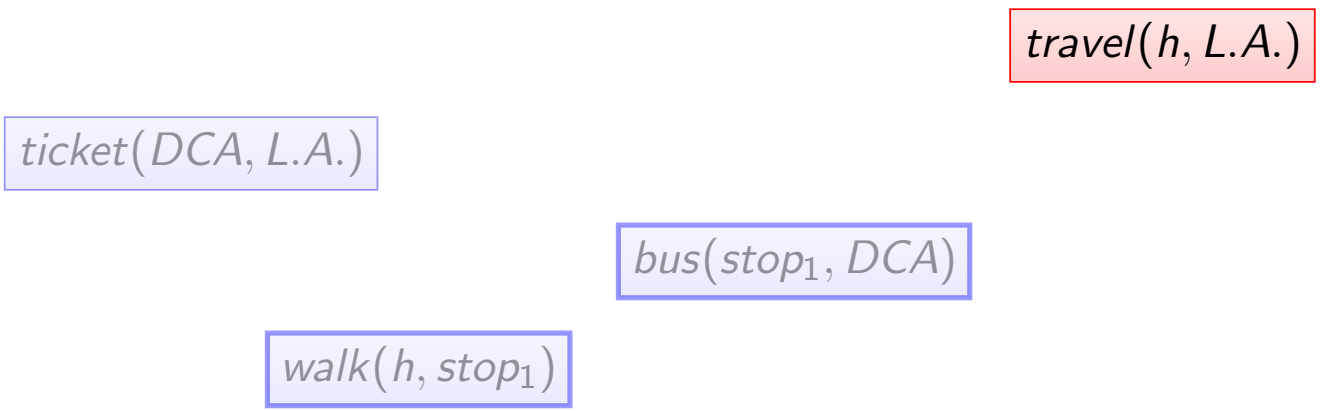
Task Insertion

- An alternate set of semantics, HTN Planning with Task Insertion (TIHTN Planning) allows the insertion of tasks without a method.

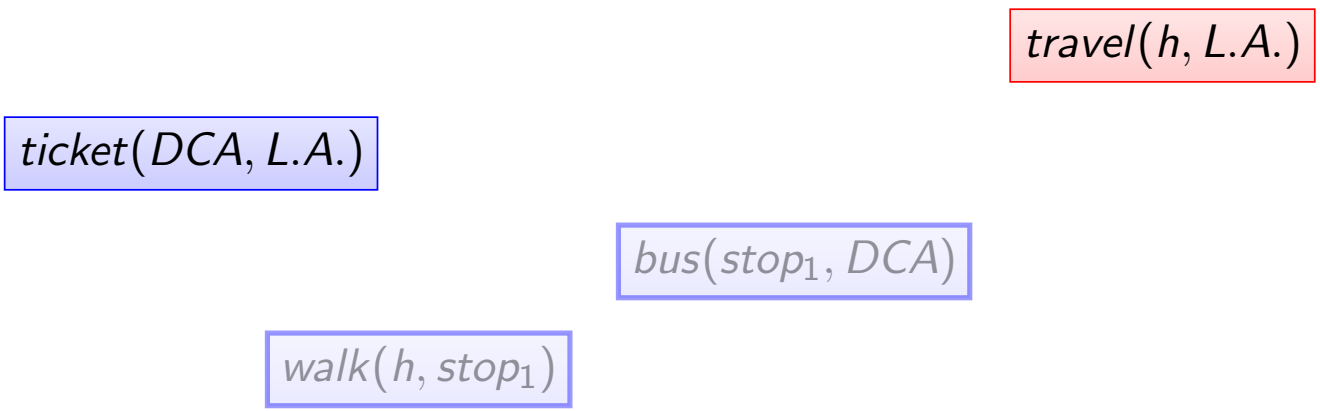


- Developed by Kambhampati (1998) and Biundo (2002)
- Used as a model for replanning and search with partial HTN knowledge

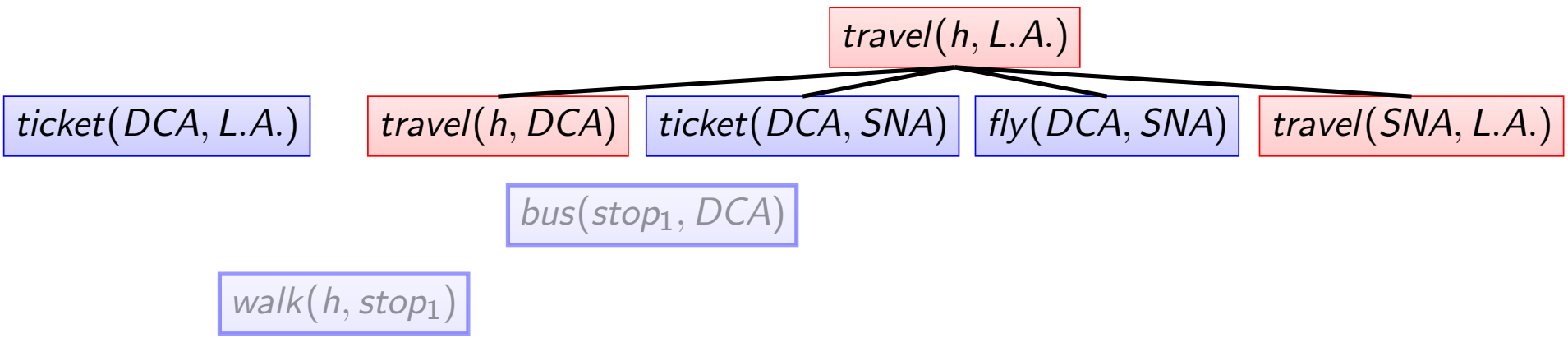
Replanning with TIHTNs



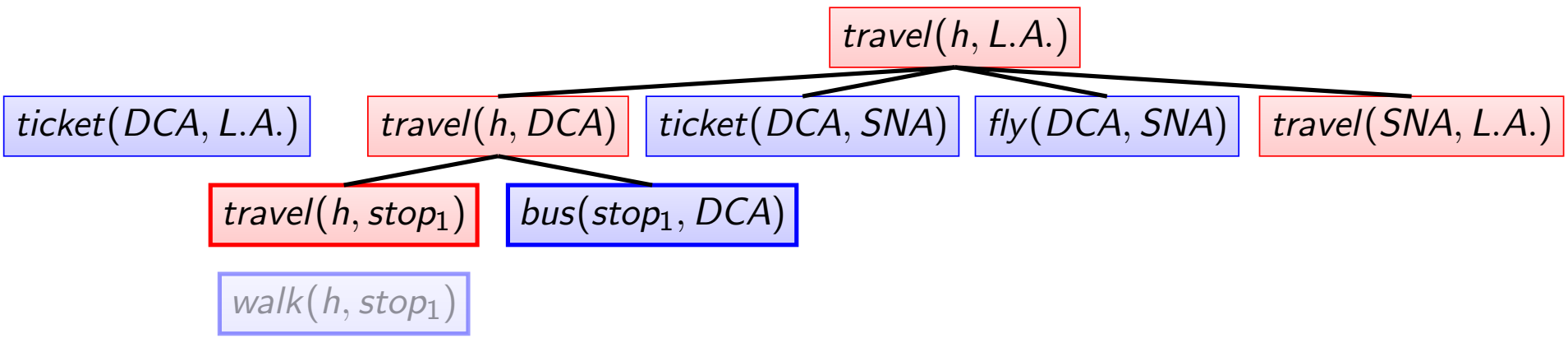
Replanning with TIHTNs



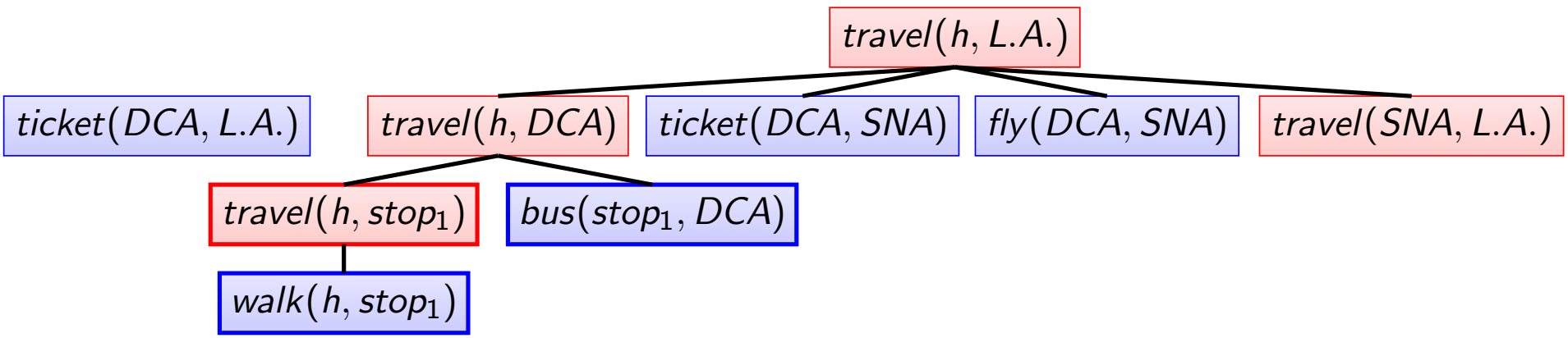
Replanning with TIHTNs



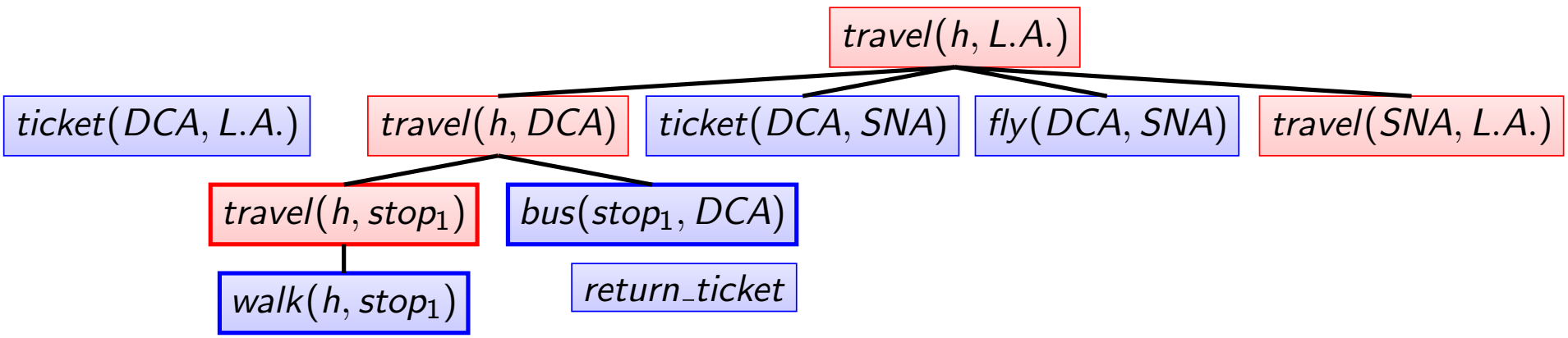
Replanning with TIHTNs



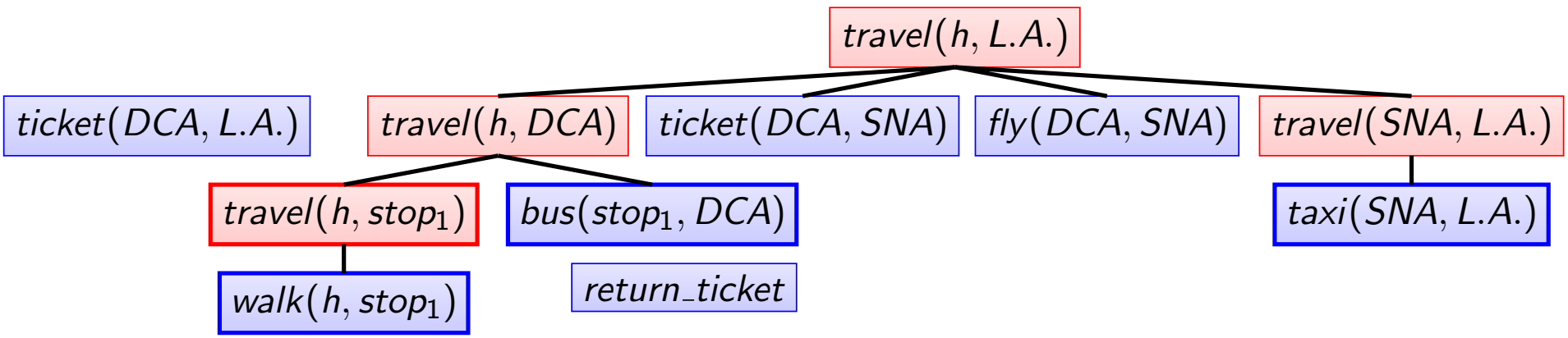
Replanning with TIHTNs



Replanning with TIHTNs



Replanning with TIHTNs



Why do we study complexity?

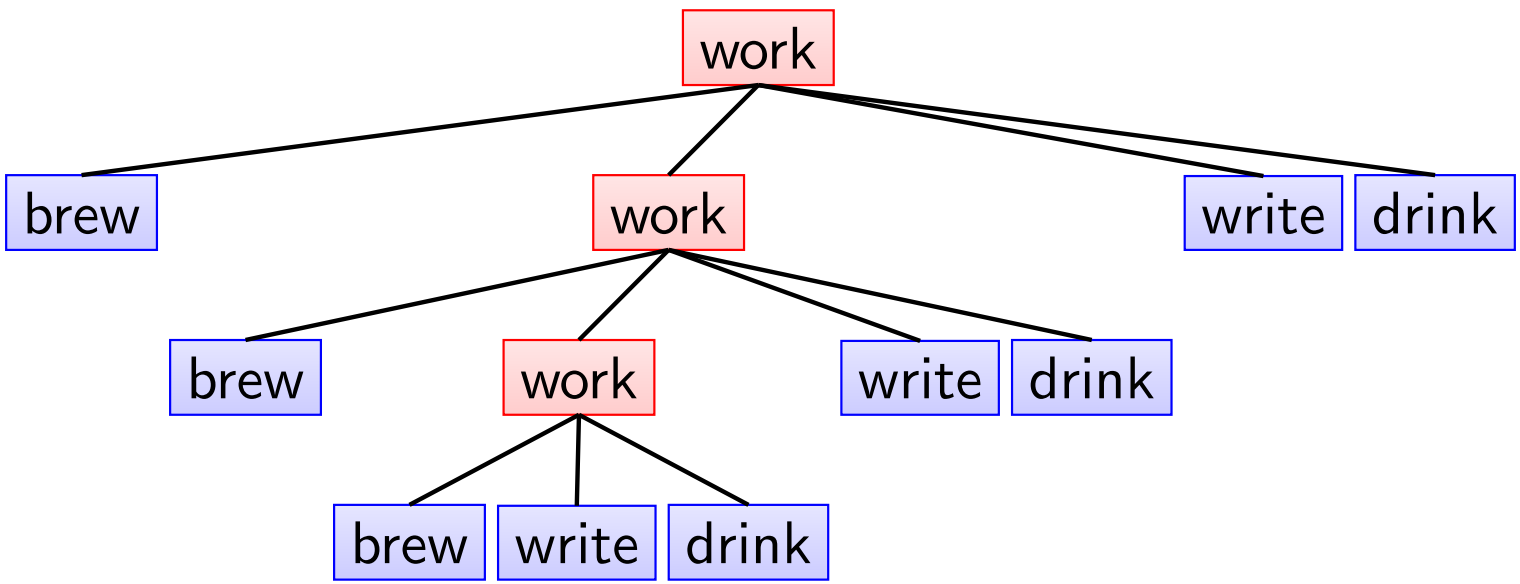
- TIHTN is decidable (Geier & Bercher, 2011)
  - Very little research on algorithms for TIHTN planning
  - No theoretical comparison, none guaranteed to terminate
- Decision procedures are important for model checking, non-deterministic planning, translation-based planning
- Method structures impact decidability and complexity:
  - Whether the method's tasks are totally-ordered
  - Where in the method recursion may occur
  - Whether the domain is relational (non-propositional)

Why do we study complexity?

- TIHTN is decidable (Geier & Bercher, 2011)
  - Very little research on algorithms for TIHTN planning
  - No theoretical comparison, none guaranteed to terminate
- Decision procedures are important for model checking, non-deterministic planning, translation-based planning
- Method structures impact decidability and complexity:
  - **Whether the method's tasks are totally-ordered**
  - **Where in the method recursion may occur**
  - Whether the domain is relational (non-propositional)

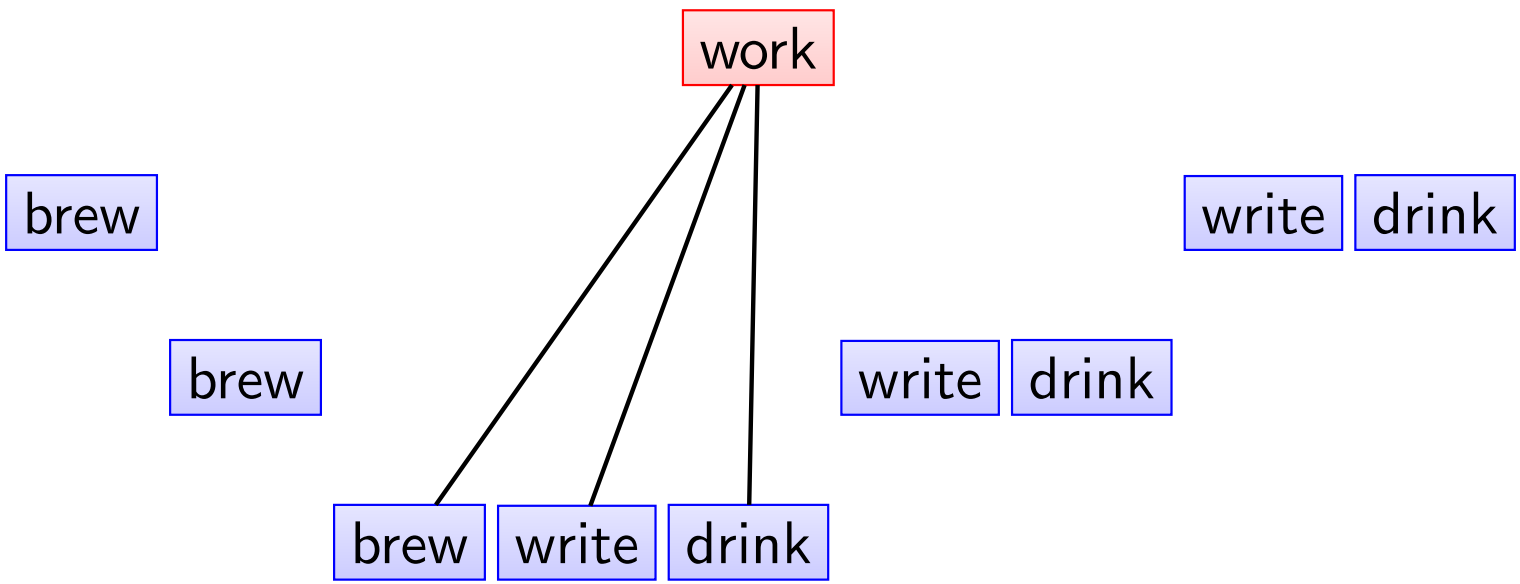
Acyclic Decomposition (Geier & Bercher, 2011)

- TIHTN planning is decidable (Geier & Bercher, 2011)
- If a TIHTN problem is solvable, there is a solution with an acyclic decomposition

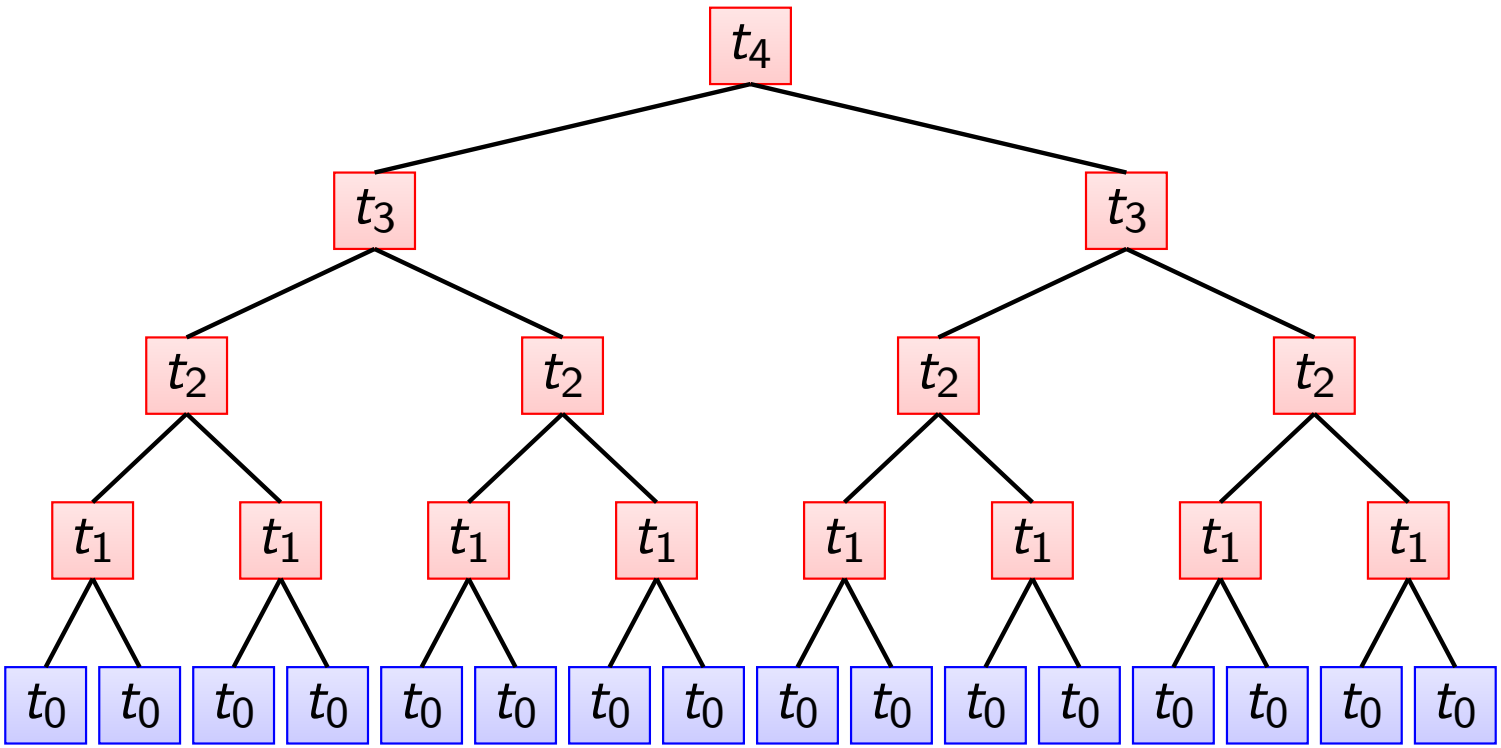


Acyclic Decomposition (Geier & Bercher, 2011)

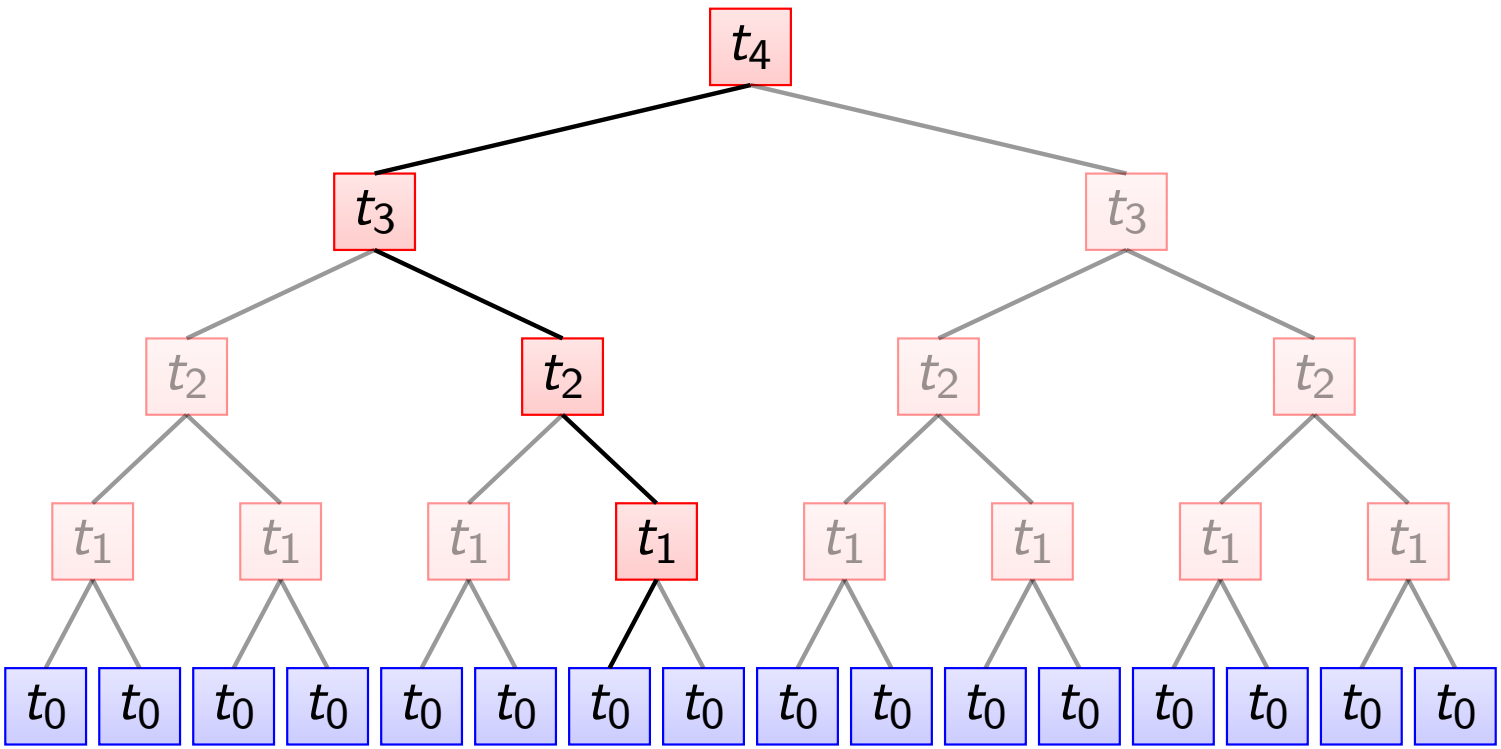
- TIHTN planning is decidable (Geier & Bercher, 2011)
- If a TIHTN problem is solvable, there is a solution with an acyclic decomposition



Planning with finite trees



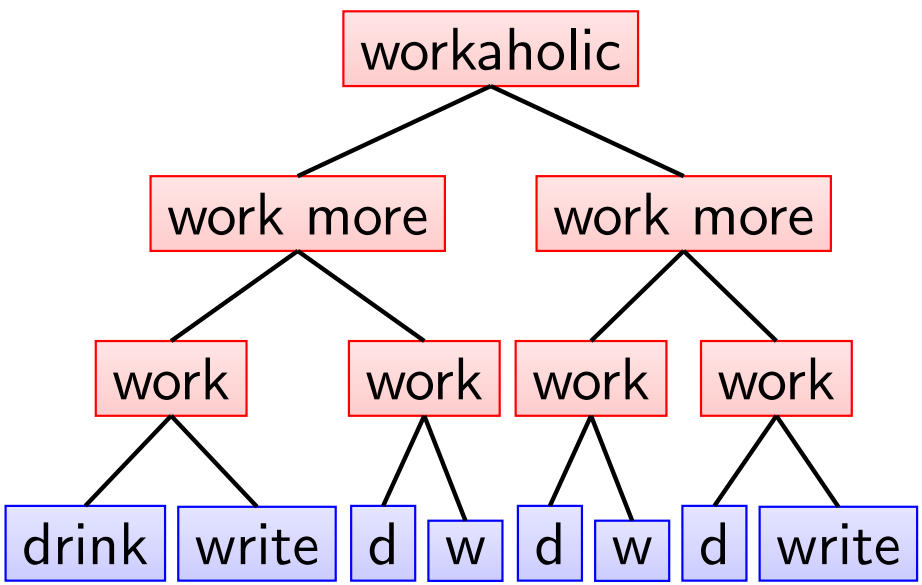
Planning with finite trees



Acyclic HTN planning

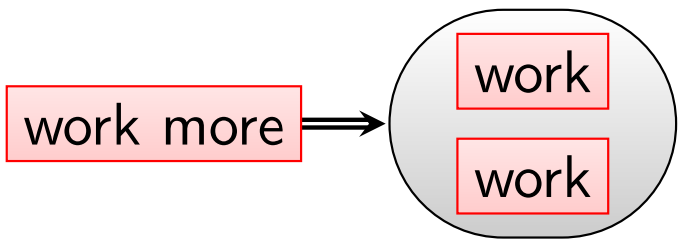
Acyclic HTNs:

- No recursion allowed
- Useful for program configuration (Soltani, 2012)

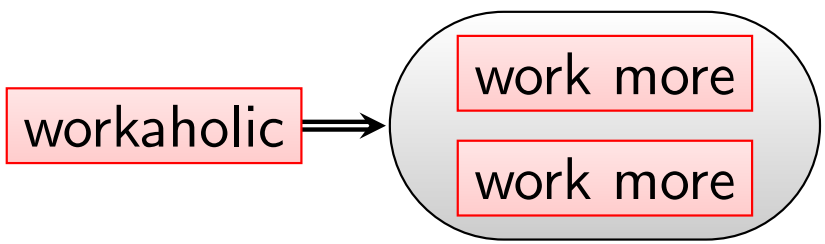


Method (totally-ordered):

Method (partially-ordered):



Method (partially-ordered):



Acyclic HTN planning, with and without task insertion

Acyclic HTNs:

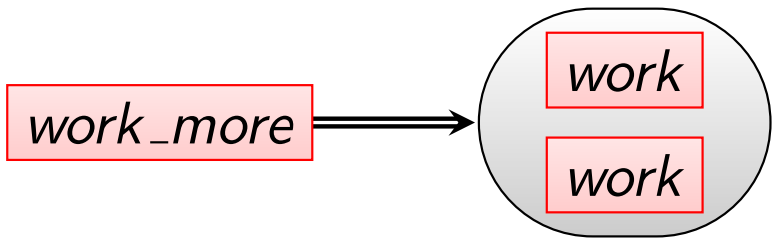
- No recursion allowed
- NEXPTIME-complete
- Every decomposition in NEXPTIME proof has same number of tasks

With insertion:

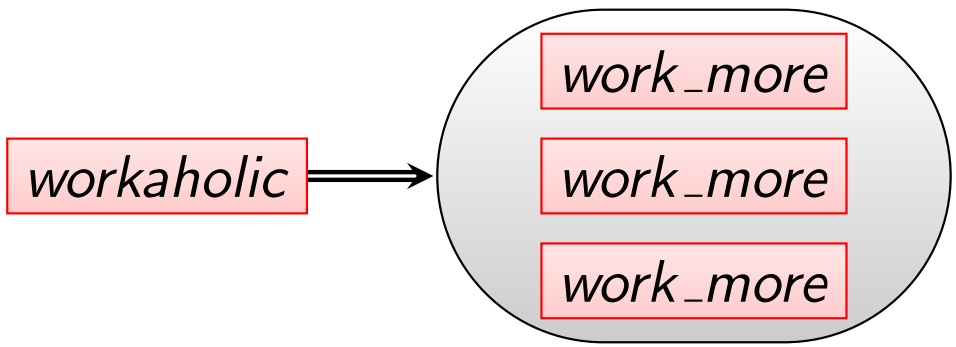
- Use counting technique to limit insertions
- TIHTN planning is NEXPTIME-complete

Method (totally-ordered):

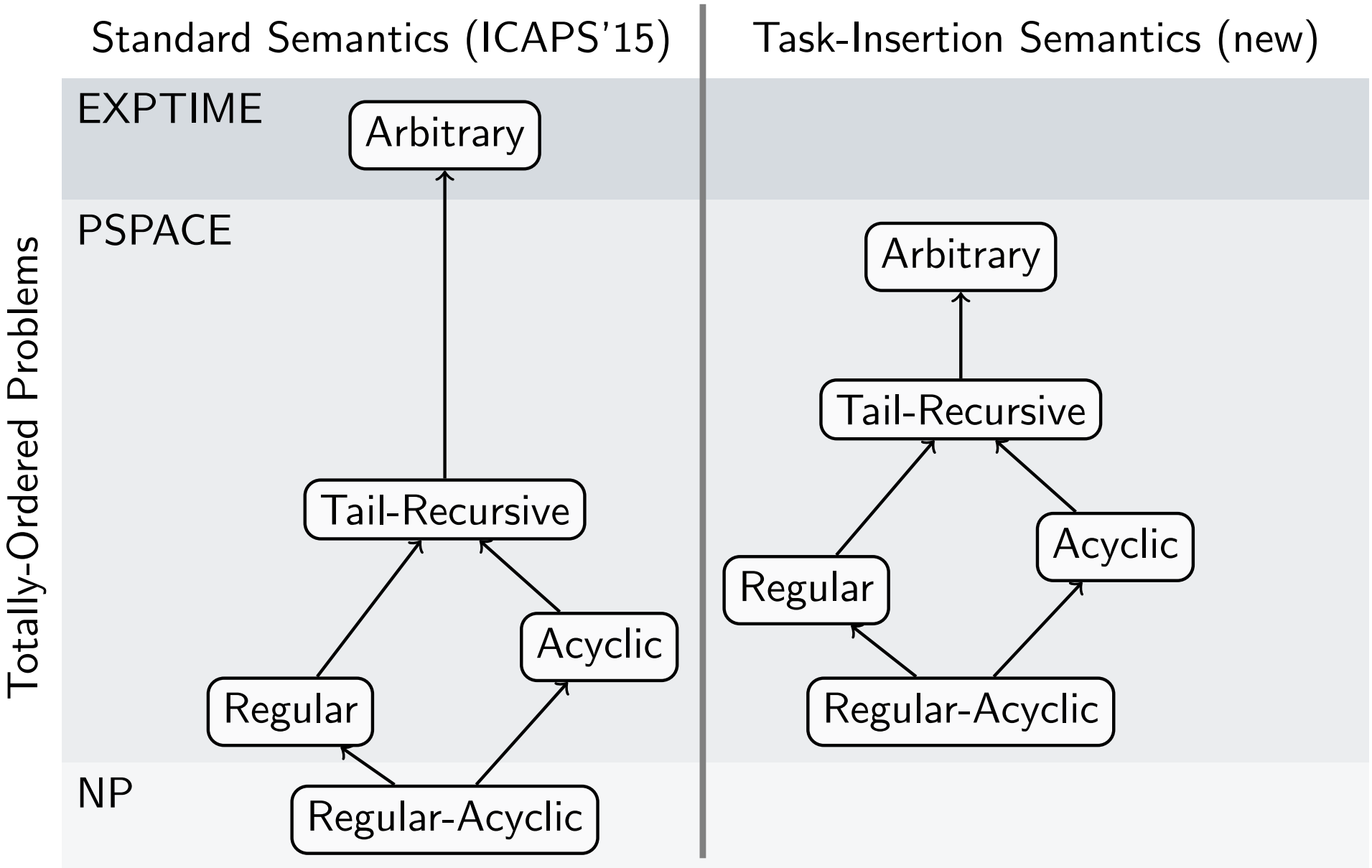
Method (partially-ordered):



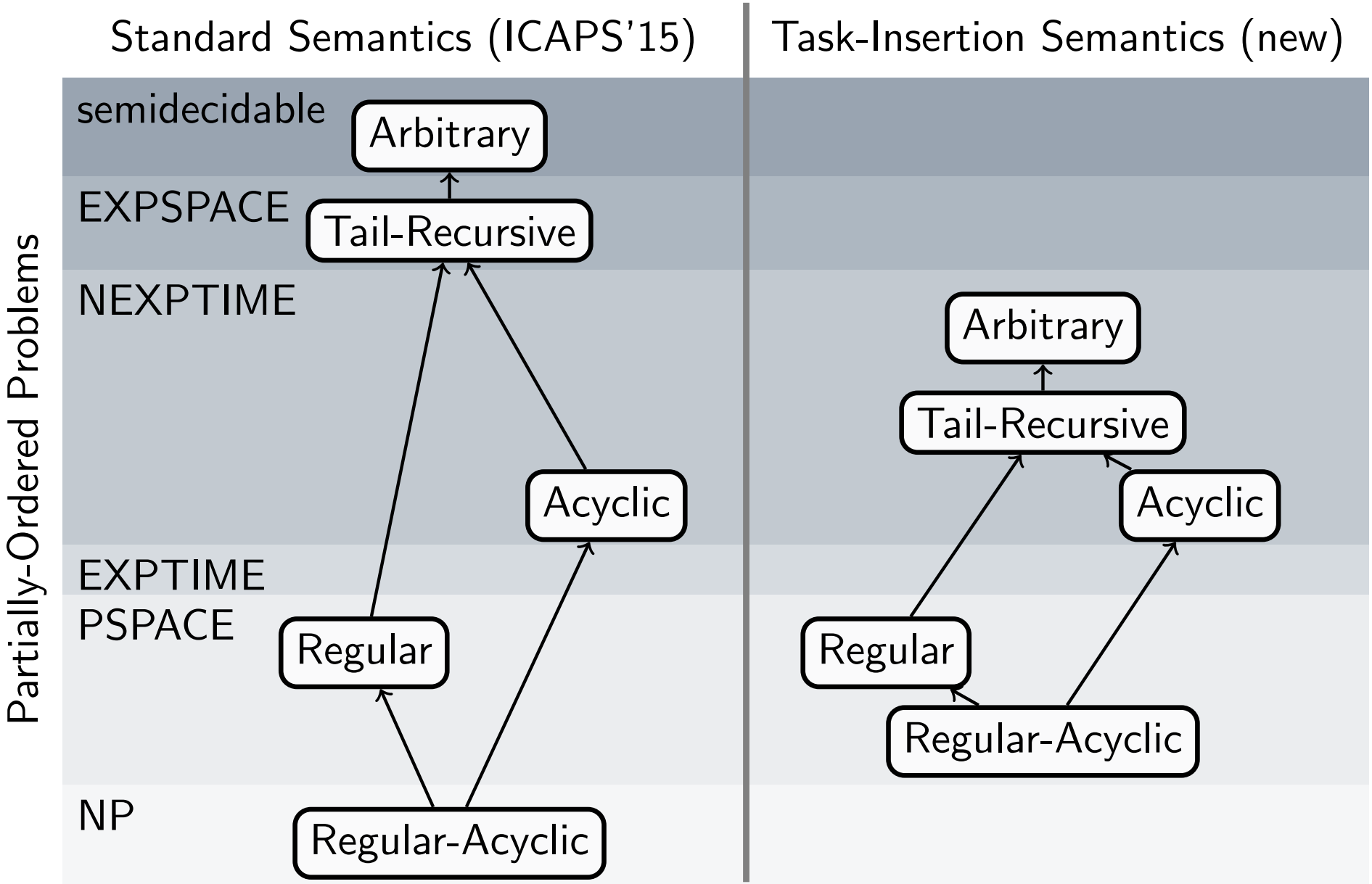
Method (partially-ordered):



Totally-Ordered Propositional HTN Complexity



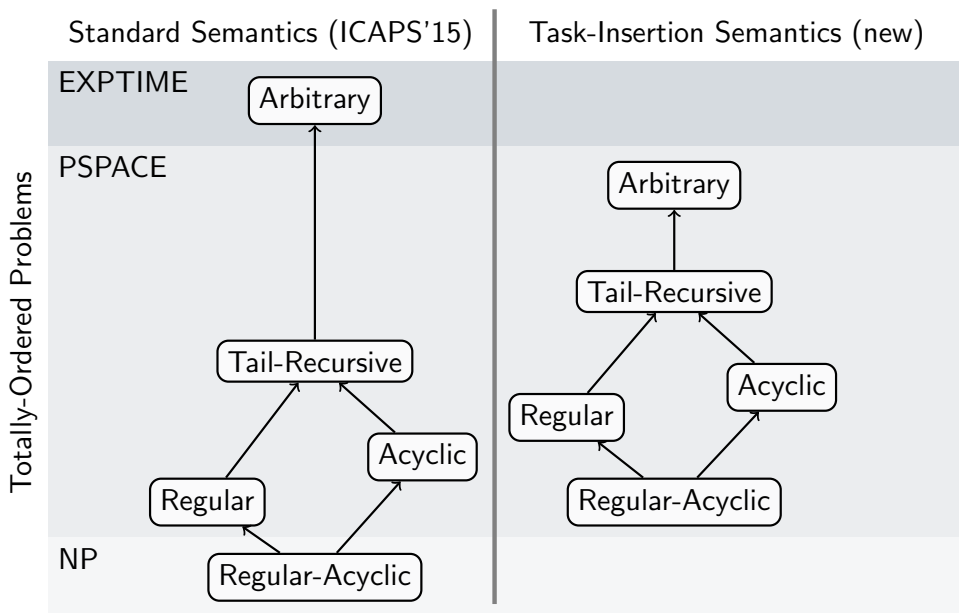
Partially-Ordered Propositional HTN Complexity

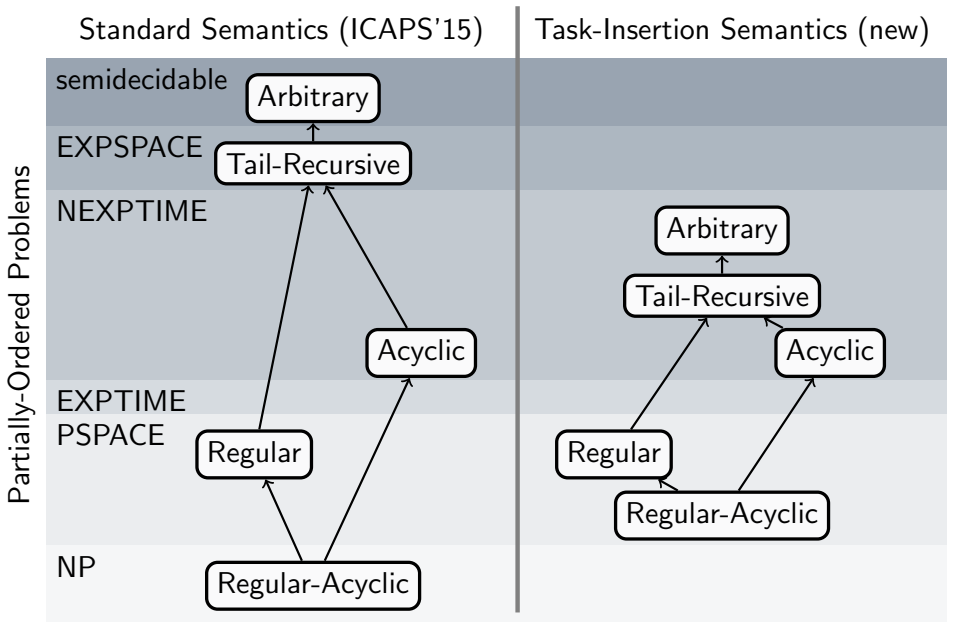


Contributions

Provided:

- New complexity bounds for all combinations of:
  - Propositional or non-propositional methods and actions
  - Partially or totally-ordered methods
  - General or regular method structures
- Theoretically-efficient algorithm for TIHTN planning





Implications

For end users:

Implications

For end users: Wait five years?

Implications

For end users: Wait five years?

- Totally-ordered propositional TIHTNs are PSPACE-complete
  - Poly-time (quadratic) translatable to propositional STRIPS
  - Probably only useful when it is linear

Implications

For end users: Wait five years?

- Totally-ordered propositional TIHTNs are PSPACE-complete
  - Poly-time (quadratic) translatable to propositional STRIPS
  - Probably only useful when it is linear

For search people: We state-space search algorithm for TIHTN planning

- A^* needs duplicate detection
  - DAG isomorphism is GI-complete
  - Totally-ordered problems can use lists
- Acyclic progression is just one blocking technique (may be others)

Implications

For end users: Wait five years?

- Totally-ordered propositional TIHTNs are PSPACE-complete
  - Poly-time (quadratic) translatable to propositional STRIPS
  - Probably only useful when it is linear

For search people: We state-space search algorithm for TIHTN planning

- A^* needs duplicate detection
  - DAG isomorphism is GI-complete
  - Totally-ordered problems can use lists
- Acyclic progression is just one blocking technique (may be others)

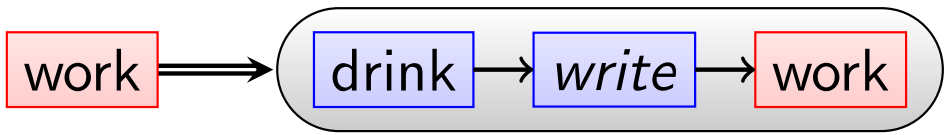
For domain-independent heuristicists:

- Very few domain-independent TIHTN heuristics
- Delete-free TIHTN planning is poly-time decidable, but?

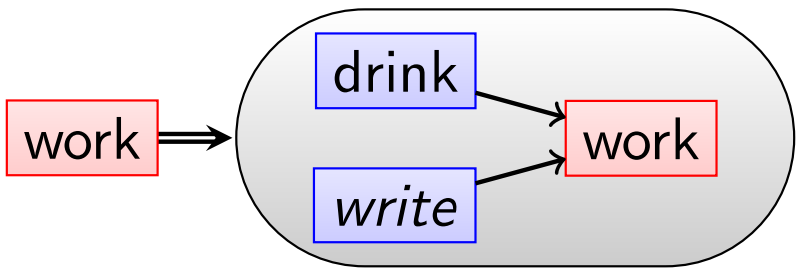
Method structures: Regular & Tail-Recursive

- Only allowed to recurse through last task
- Only last task in regular HTNs can be non-primitive
- Tail-recursion generalizes acyclic HTNs

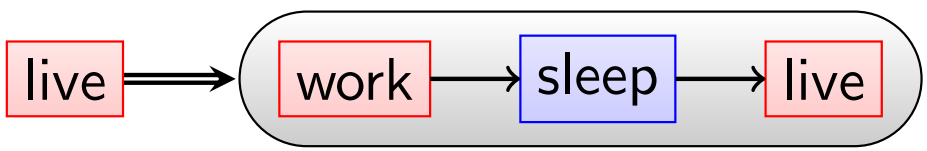
Method (totally-ordered regular):



Method (partially-ordered regular):



Method (totally-ordered tail-recursive):



Method structures: Arbitrary

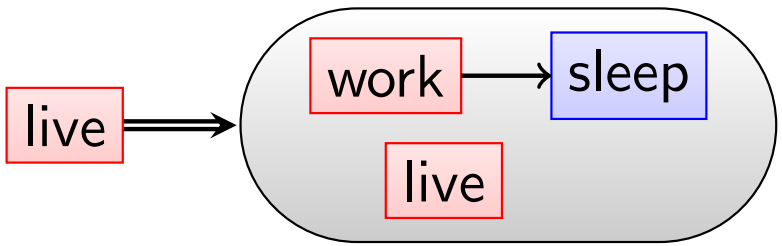
Partially-ordered arbitrary problems:

- Can express intersection of CFGs
- Semi-decidable

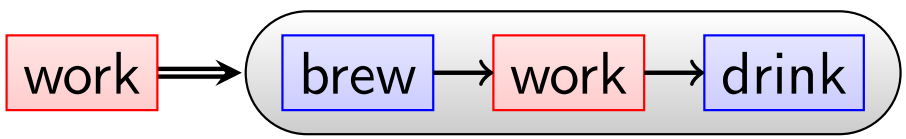
Totally-ordered arbitrary problems:

- Sometimes used for program representation
  - Web services (Sirin, 2004)
  - Java security (Kuter, 2015)
- Relatively easy to encode 2-player game tree search (chess, checkers)

Method (partially-ordered):



Method (totally-ordered):



Results on Acyclic Progression

- Task networks are size-bounded under acyclic progression
- Size bounds if n is the number of task names and m is the size of the largest method
  - $m \cdot n$ for totally-ordered problems
  - m^n for partially-ordered problems