HTN Problem Spaces: Structure, Algorithms, Termination

Ron Alford, Vikas Shivashankar, Ugur Kuter, and Dana Nau

July 21st, 2012

Unsolvable problems

HTN planning is semi-decidable.

- If your planner always returns if the problem is solvable, then there exist unsolvable problems for which it will never return.

Why care about unsolvable problems?

- Debugging/developing a domain
- Plan may not exist (Problem: make me a millionaire)
- Techniques that require failure:
  - NDP: uses a deterministic planner for FOND domains
  - Verification tasks

Classical Planning: $D = (S, O, \gamma)$, $P = (D, s_0, G)$

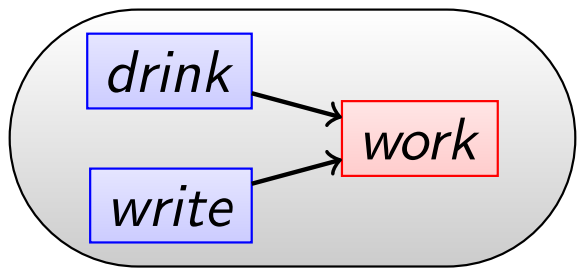
A classical planning problem consists of:

- A domain, D, with:
  - S: A finite set of states
  - O: A finite set of actions
  - $\gamma : S \times O \to S$: A partial function to transition between states
- s_0: An initial state
- G: A set of goal states
- An action a is *executable* in a state if $\gamma(s, a)$ is defined.
- A sequence of actions $\langle a_1, \ldots, a_n \rangle$ is executable in a state if the actions are executable in turn.
  - $\gamma(s_0, a_1) = s_1$
  - $\gamma(s_1, a_2) = s_2$
  - $\ldots$
  - $\gamma(s_{n-1}, a_n) = s_n$
- Objective: Find a plan (an executable sequence) that take us to a goal state.

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

All about the process, not the goal.

- Same S, O, γ, and s_0.
- An initial *task network*, tn_0, which is a DAG of *tasks* to accomplish:
  - Edges represent ordering constraints
  - Nodes either labelled with action names from O (say, $\{drink, write\}$)
  - Or non-primitive with labels from C (say, $\{work\}$)

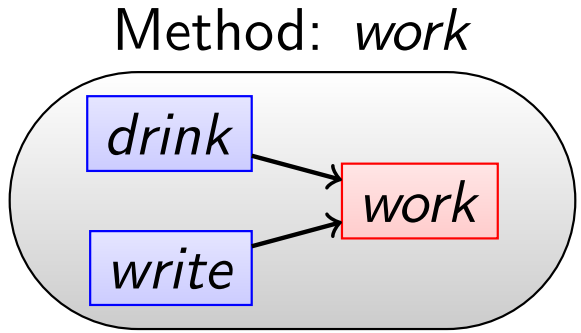


- M, a set of methods to *decompose* non-primitive tasks

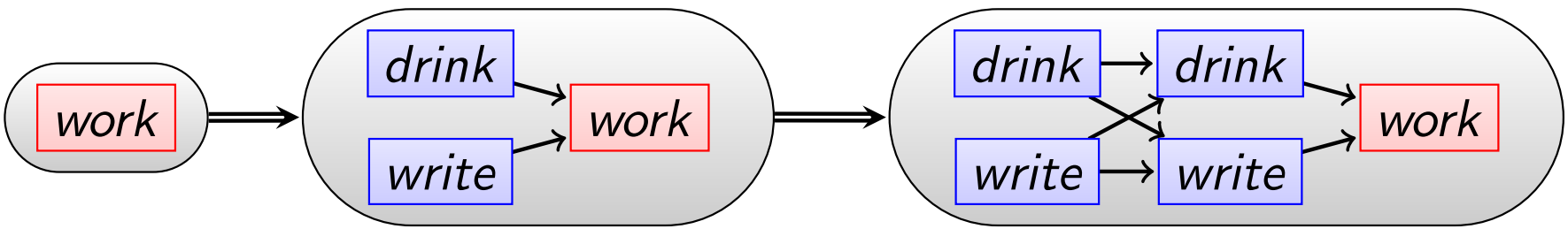
HTN Planning: $D = (S, \mathbf{C}, O, \mathbf{M}, \gamma)$, $P = (D, s_0, \mathbf{tn}_0)$

Example domain and decomposition

- A method is a non-primitive name paired with a task network.



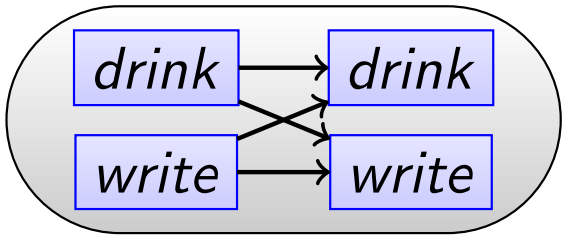
- We *decompose* a task by replacing its node in the network with its method's network.



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

Primitive task networks:

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

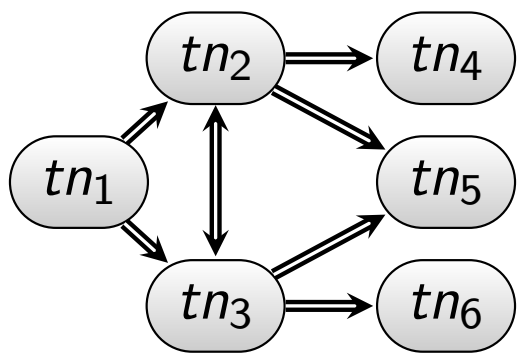


- All names are actions $\Rightarrow$ network is *primitive*
- It is *executable* in a state if there exists a consistent total order (sequence) which is executable.
  - Four possible orderings
  - All executable in both *thirsty* and $\neg thirsty$
  - Could end in either state.
- Objective: Decompose initial network to a primitive task network which is executable in s_0.

Decomposition spaces for HTN planning

Decomposition as a graph

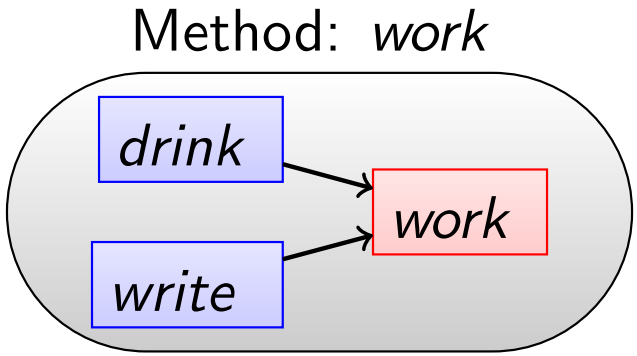
- Decomposition: Choose some non-primitive task and some method to decompose it
- Choices form a graph of task networks reachable via decomposition.
- Call this the *decomposition space* of a problem.
- Plan by searching graph (not always finite)
- Used by UMCP (Erol 94), Landmark-aware HTN planner (Elkawkagy 2011)



Decomposition spaces for HTN planning

Finiteness: $\leq_1$-stratification

- A domain is $\leq_1$-*stratifiable* if we can stratify its task names such that:
  - $(t, tn) \in M$ and $|tn| > 1 \Rightarrow$ every task in tn is on a lower stratum (acyclic decomposition).
  - $(t, tn) \in M$ and $|tn| = 1 \Rightarrow$ tn's task is on an equal or lower stratum (trivially recursive decomposition).
- Existence of a $\leq_1$-stratification implies decomposition either:
  - Replaces task with a single task (no change in size)
  - Replaces task with tasks from lower strata.

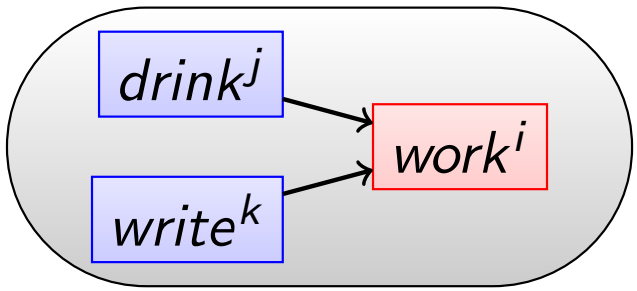


Decomposition spaces for HTN planning

Finiteness: $\leq_1$-stratification

- A domain is $\leq_1$-*stratifiable* if we can stratify its task names such that:
  - $(t, tn) \in M$ and $|tn| > 1 \Rightarrow$ every task in tn is on a lower stratum (acyclic decomposition).
  - $(t, tn) \in M$ and $|tn| = 1 \Rightarrow$ tn's task is on an equal or lower stratum (trivially recursive decomposition).
- Existence of a $\leq_1$-stratification implies decomposition either:
  - Replaces task with a single task (no change in size)
  - Replaces task with tasks from lower strata.

Method: $work^i$



Constraints:

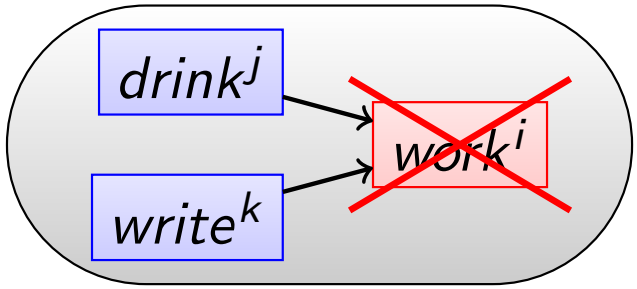
$j < i,\ k < i,\ \ i < i$

Decomposition spaces for HTN planning

Finiteness: $\leq_1$-stratification

- A domain is $\leq_1$-*stratifiable* if we can stratify its task names such that:
  - $(t, tn) \in M$ and $|tn| > 1 \Rightarrow$ every task in tn is on a lower stratum (acyclic decomposition).
  - $(t, tn) \in M$ and $|tn| = 1 \Rightarrow$ tn's task is on an equal or lower stratum (trivially recursive decomposition).
- Existence of a $\leq_1$-stratification implies decomposition either:
  - Replaces task with a single task (no change in size)
  - Replaces task with tasks from lower strata.

Method: $work^i$



Constraints:

$j < i$, $k < i$, ~~$i < i$~~

2	work
1	drink, write

Decomposition spaces for HTN planning

$\leq_1$-stratification implications

- Exists $\leq_1$-stratification $\Rightarrow$ finite decomposition space (task networks reachable by decomposition)
- No $\leq_1$-stratification $\Rightarrow$ infinite decomposition space
- Dealing with trivial recursion
  - Check for loops during decomposition
  - Compile it away

Progression spaces for HTN planning

For an HTN problem $P = (D, s_0, tn_0)$, if t is a task with no predecessors in tn_0, there are two ways to *progress* it:

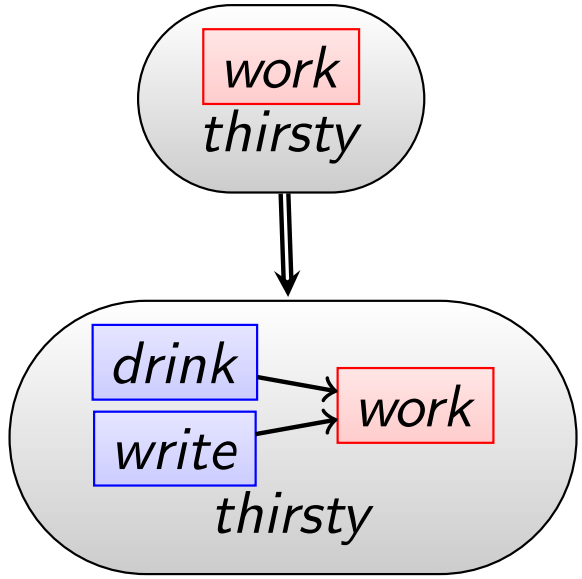
- If t is non-primitive:
  - Let tn' be a decomposition of t in tn_0
  - Then (D, s_0, tn') is a progression of P.
- If t is primitive ($t \in O$) and executable, then:
  - Execute t: $\gamma(s_0, t) = s_1$
  - Remove it from tn_0: $tn' = tn_0 \setminus \{t\}$
  - Then (D, s_1, tn') is a progression of P.
- Progression interleaves decomposition and finding a total order.
- Choices form a graph of problems reachable by progression.

Progression spaces

Example

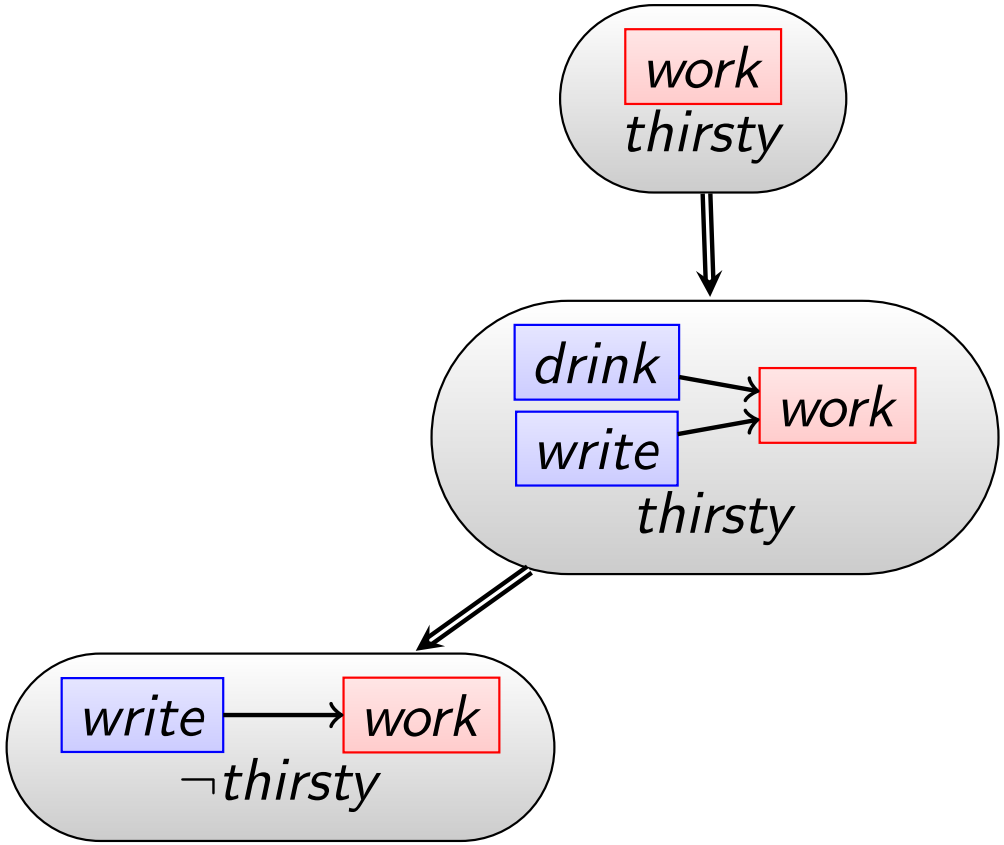
Progression spaces

Example



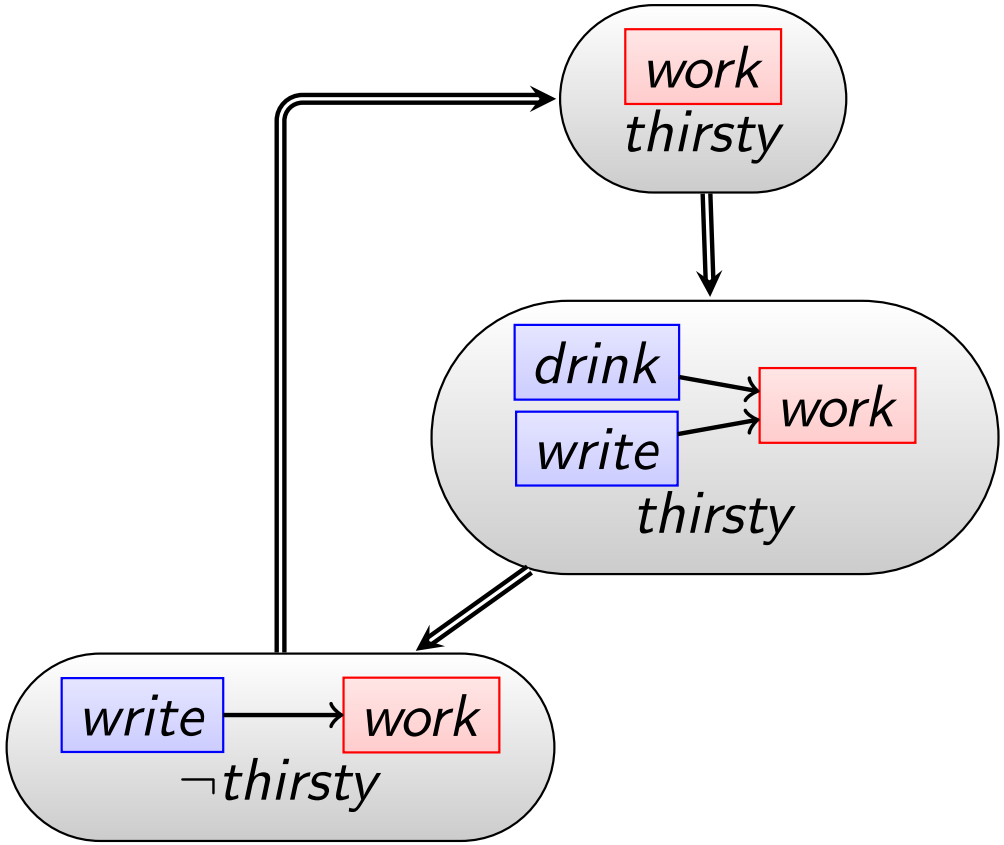
Progression spaces

Example



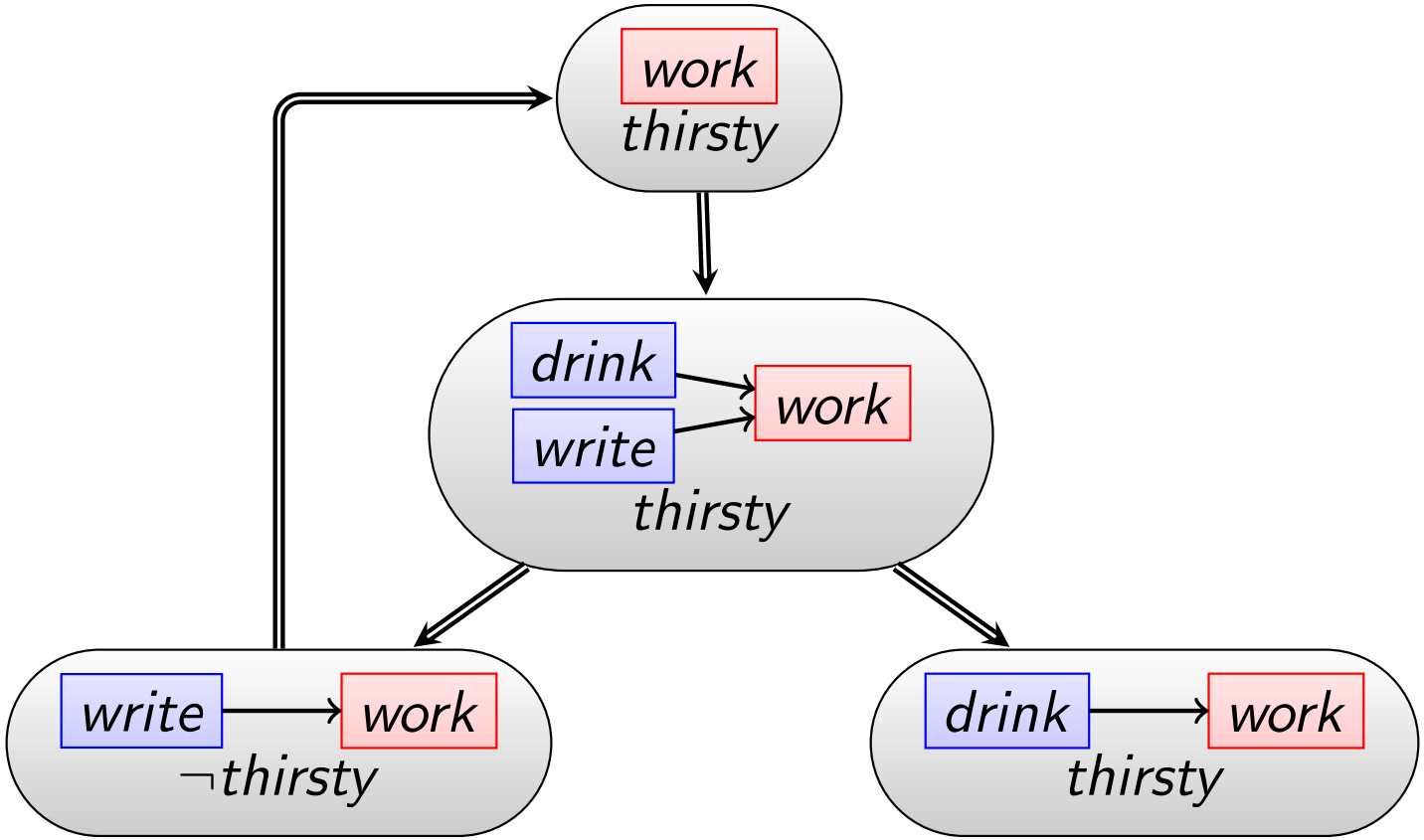
Progression spaces

Example



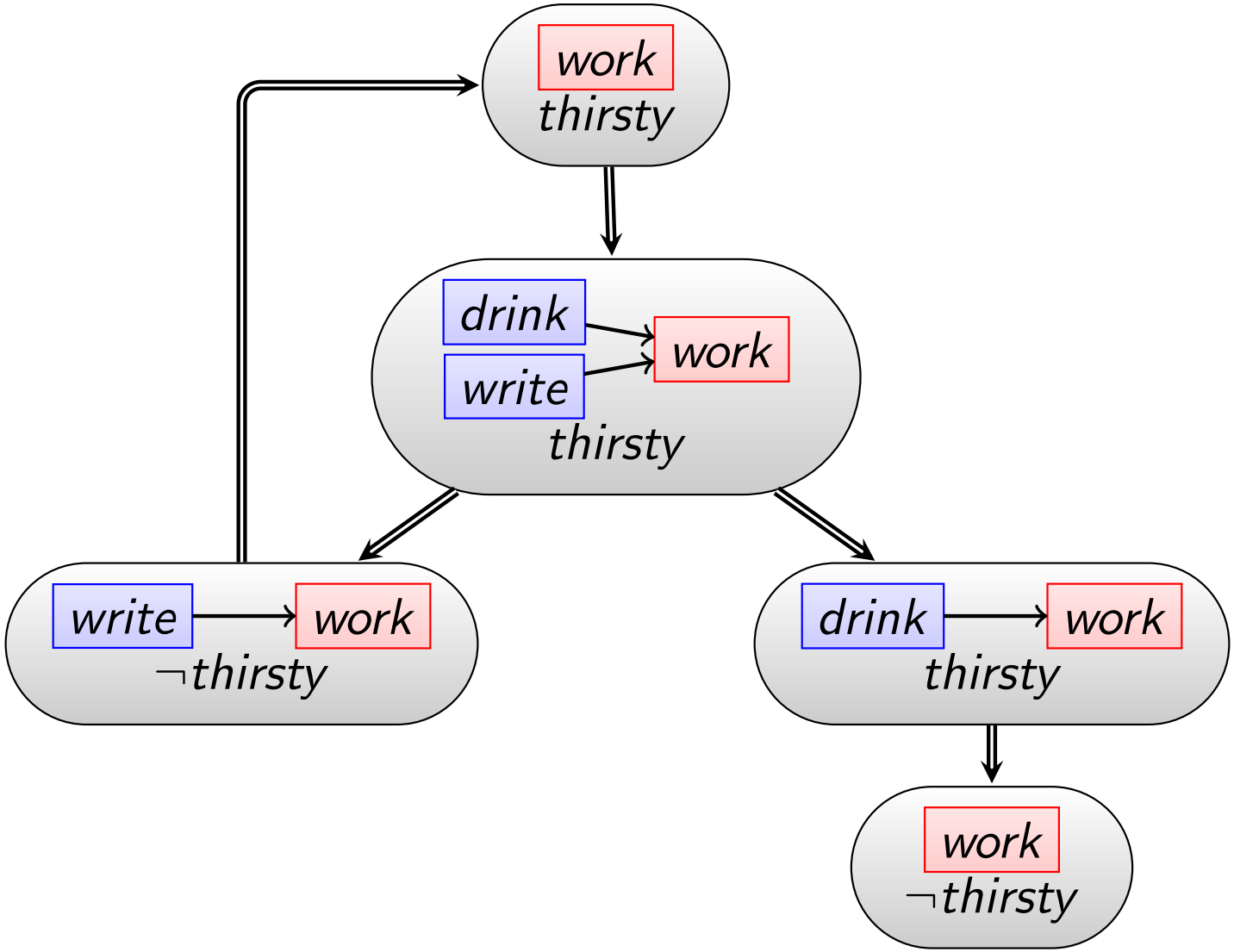
Progression spaces

Example



Progression spaces

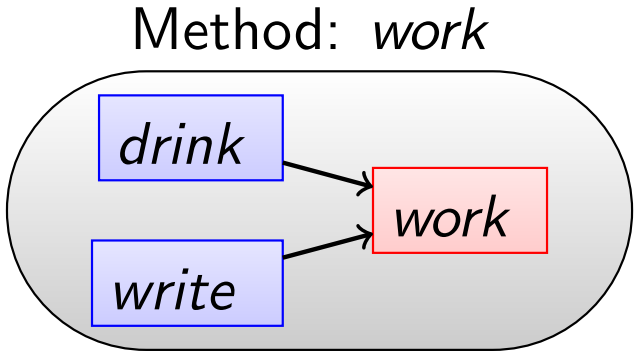
Example



Progression spaces for HTN planning

$\leq_r$-stratification

- A $\leq_r$-stratification of task names: For every method (t, tn),
  - All but the last task of tn must be on lower strata.
  - **Last task** on equal or lower stratum (tail recursion).

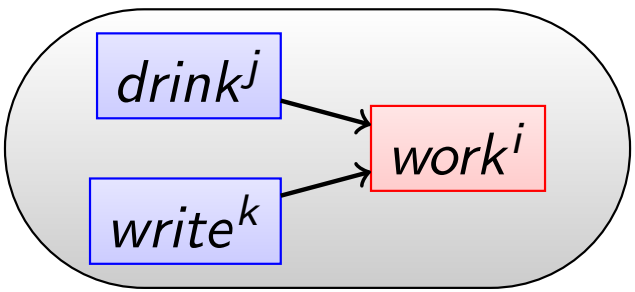


Progression spaces for HTN planning

$\leq_r$-stratification

- A $\leq_r$-stratification of task names: For every method (t, tn),
  - All but the last task of tn must be on lower strata.
  - **Last task** on equal or lower stratum (tail recursion).

Method: $work^i$



Constraints: $j < i, k < i, i \leq i$

2	work
1	drink, write

Progression spaces for HTN planning

$\leq_r$-stratification implications

- Includes all $\leq_1$-stratifiable problems
- Exists $\leq_r$-stratification $\Rightarrow$
  - Can only decompose to a fixed depth before executing actions.
  - Finite progression space (problems reachable by progression)
- No $\leq_r$-stratification: Progression space might be infinite:
  - γ structure can prune problems from progression space
- Most (all?) SHOP and SHOP2 domains are $\leq_r$-stratifiable
  - Can guarantee termination by adding a loop check

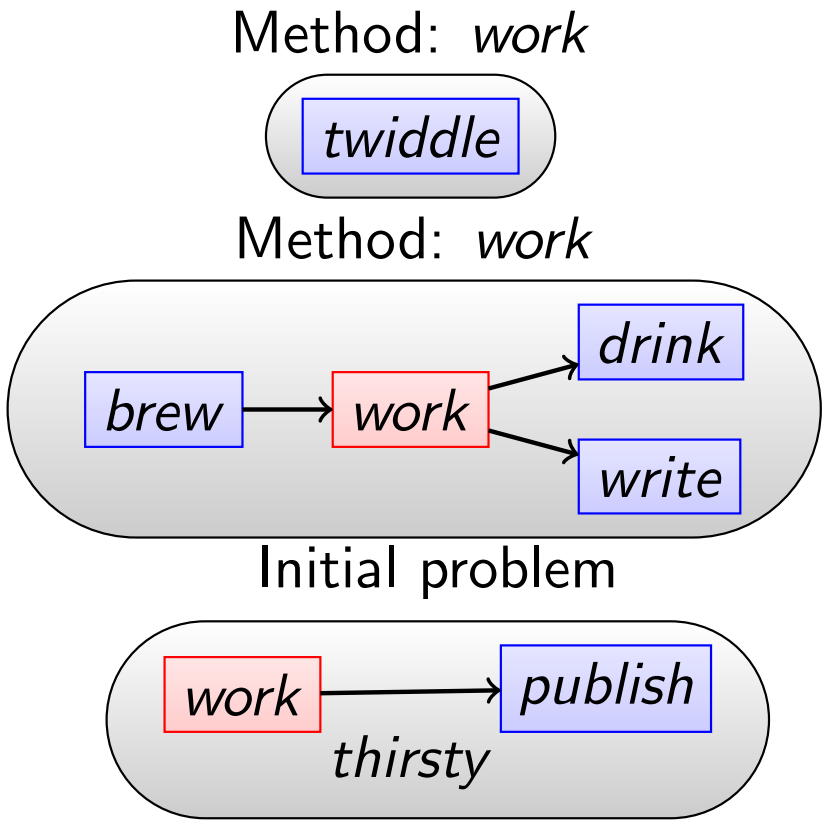
Total Order Partition spaces

Motivating Example: Morning coffee

- Brew as much coffee in the morning as you need throughout the day.
- Easy to express as HTN
  - Difficult in classical planning (artificially limit coffee?)
  - Not $\leq_r$-stratifiable ('work' now in the middle)

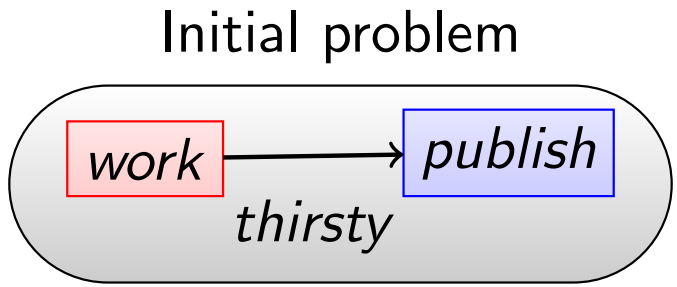
New actions:

- $\gamma(s, \mathit{brew}) = s$
- $\gamma(s, \mathit{twiddle}) = s$
- $\gamma(s, \mathit{publish}) = \emptyset$



Total Order Partition spaces

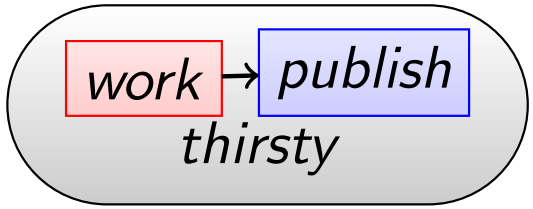
Serialization



- Plan by splitting into smaller parts
- If there is a total order over chunks of the task network:
  - Serialize network into smaller chunks
  - End state of one is the start state of the next.
  - If last chunk solvable, then the whole problem is solvable
- When no total order, use progression
- Call this the Total Order Progression (TOP) space

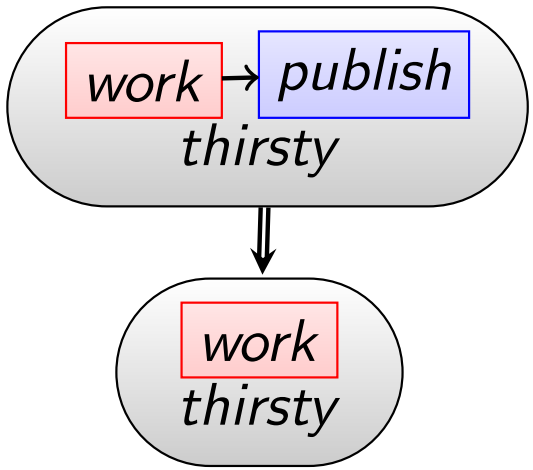
Total Order Partition spaces

Example



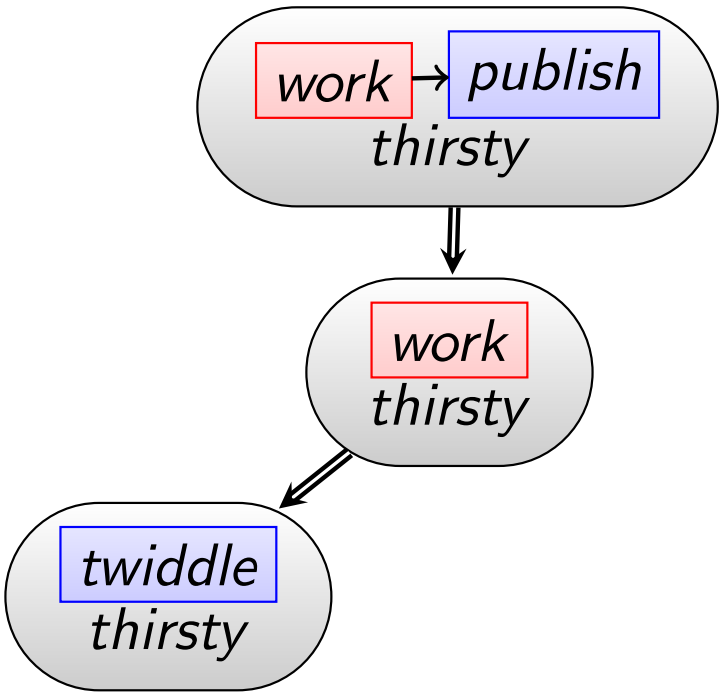
Total Order Partition spaces

Example



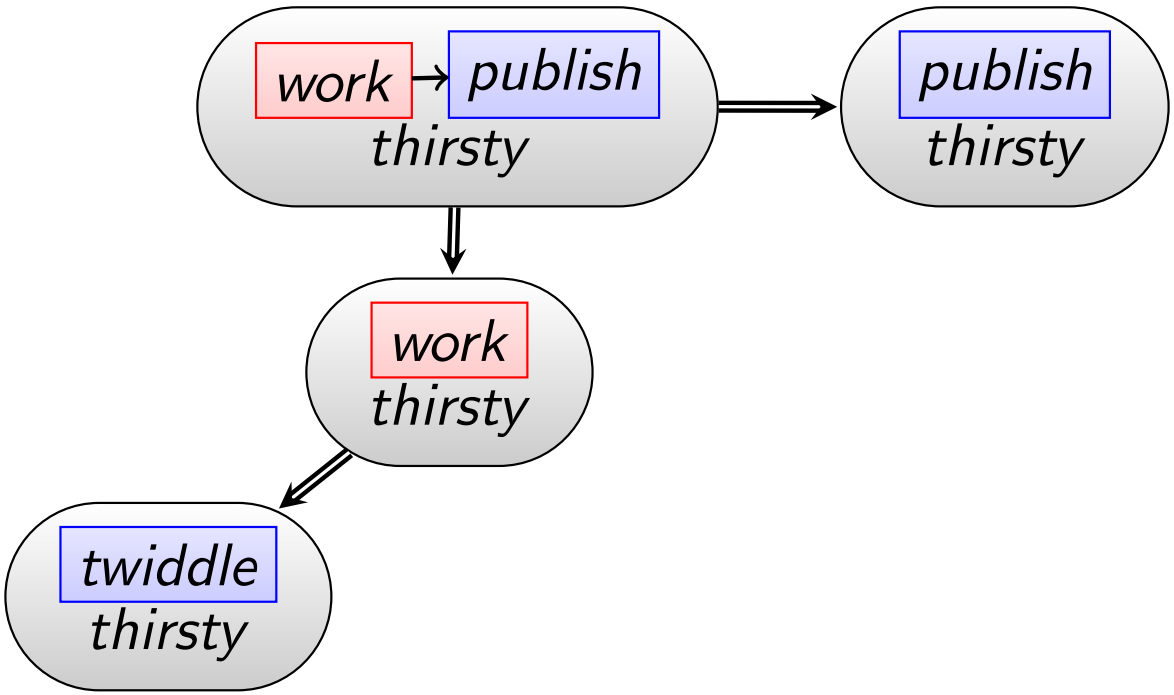
Total Order Partition spaces

Example



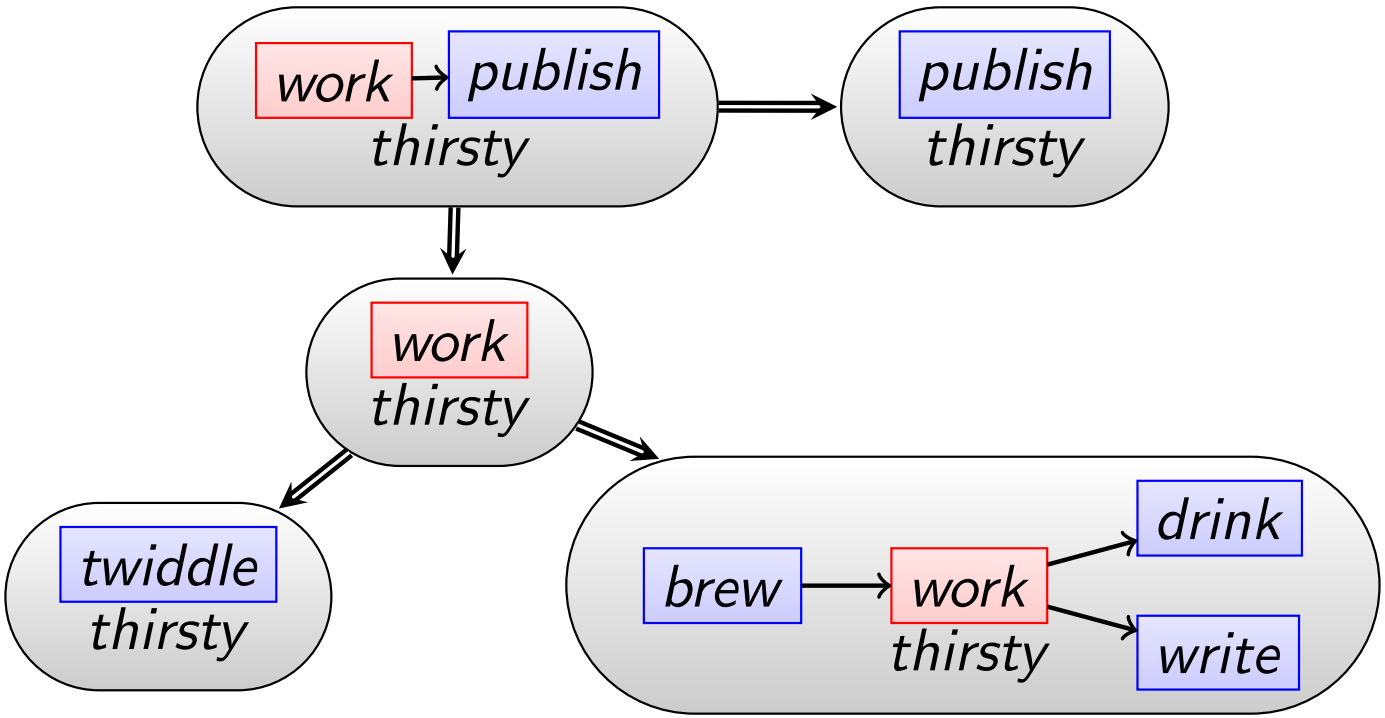
Total Order Partition spaces

Example



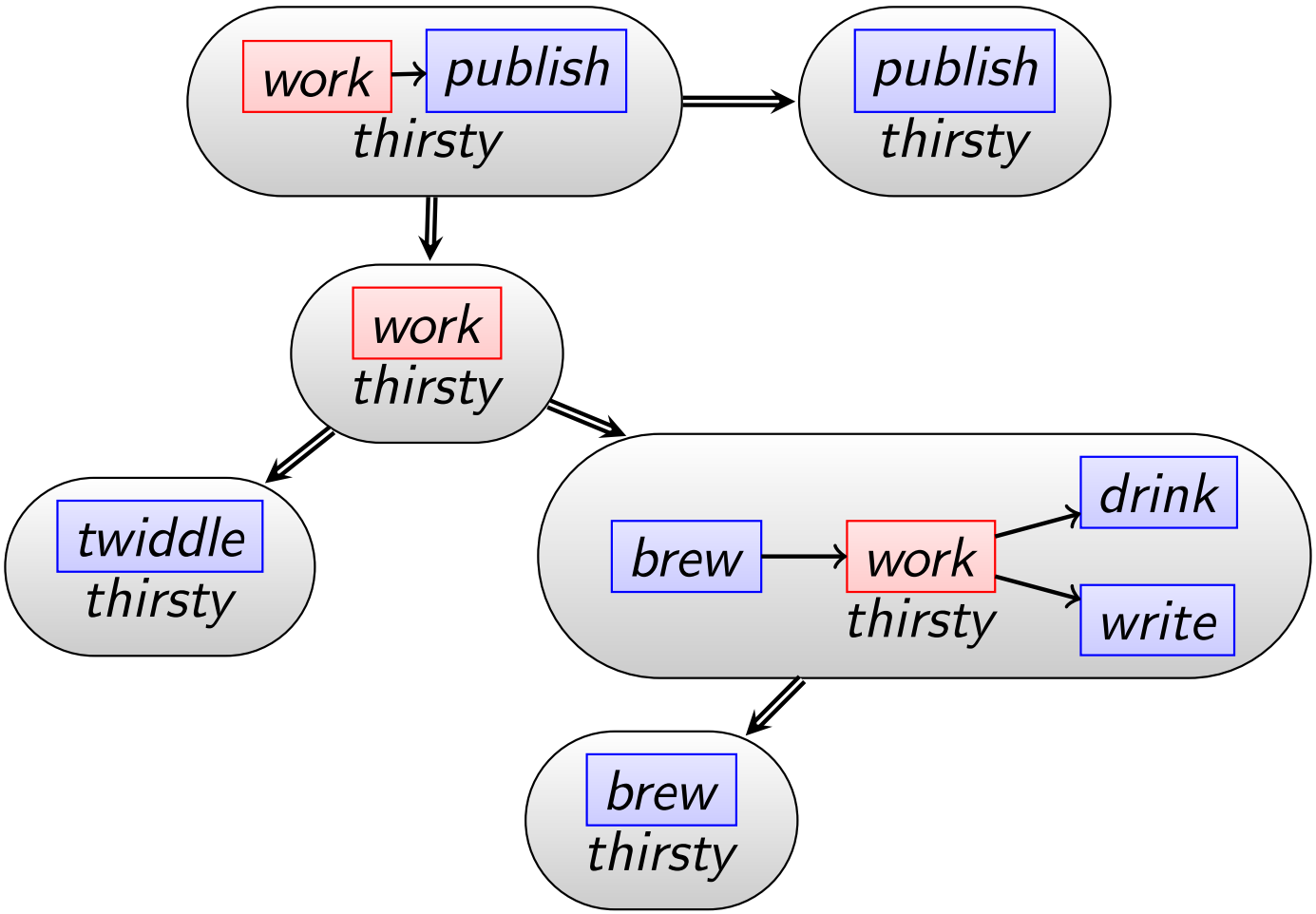
Total Order Partition spaces

Example



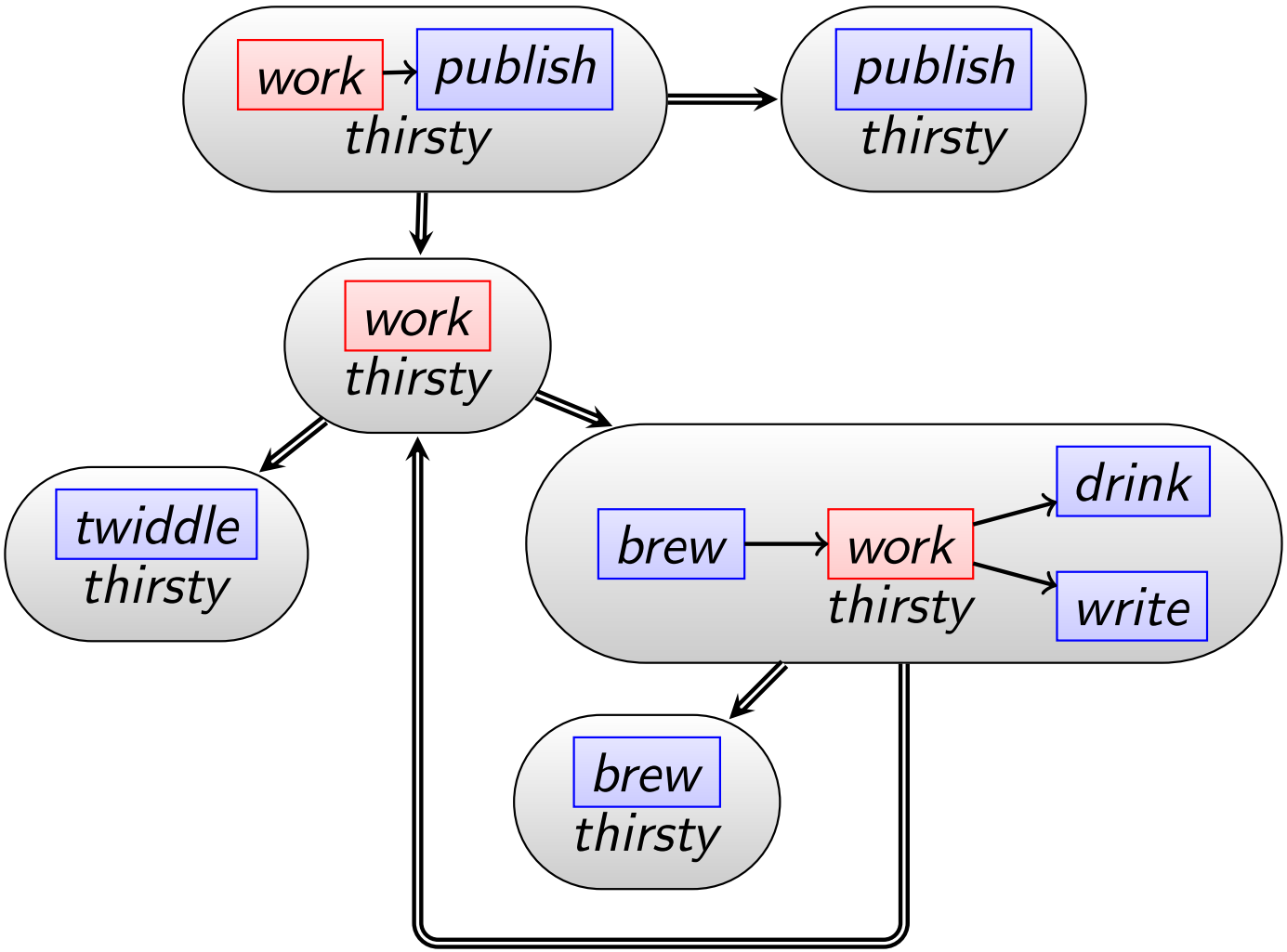
Total Order Partition spaces

Example



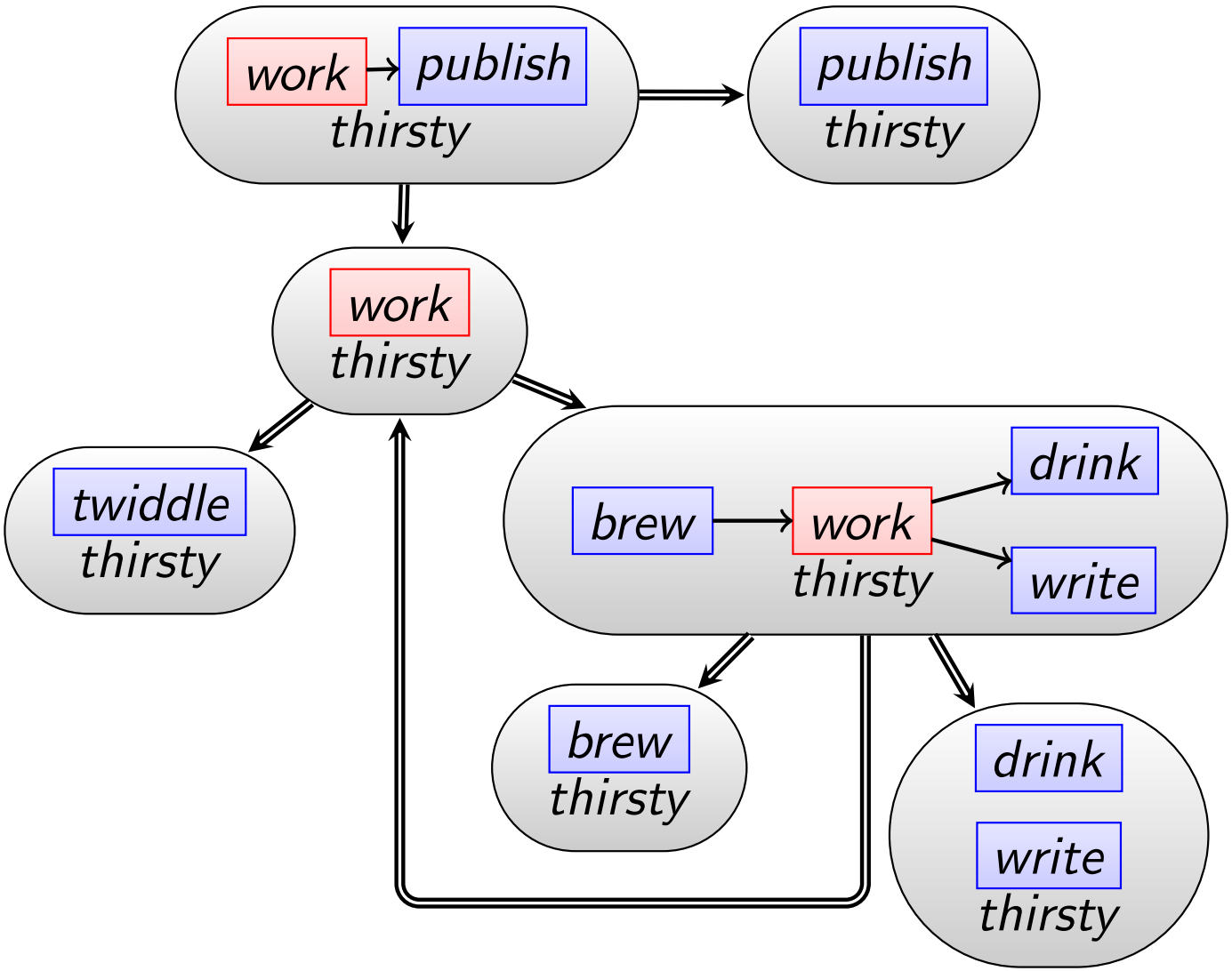
Total Order Partition spaces

Example



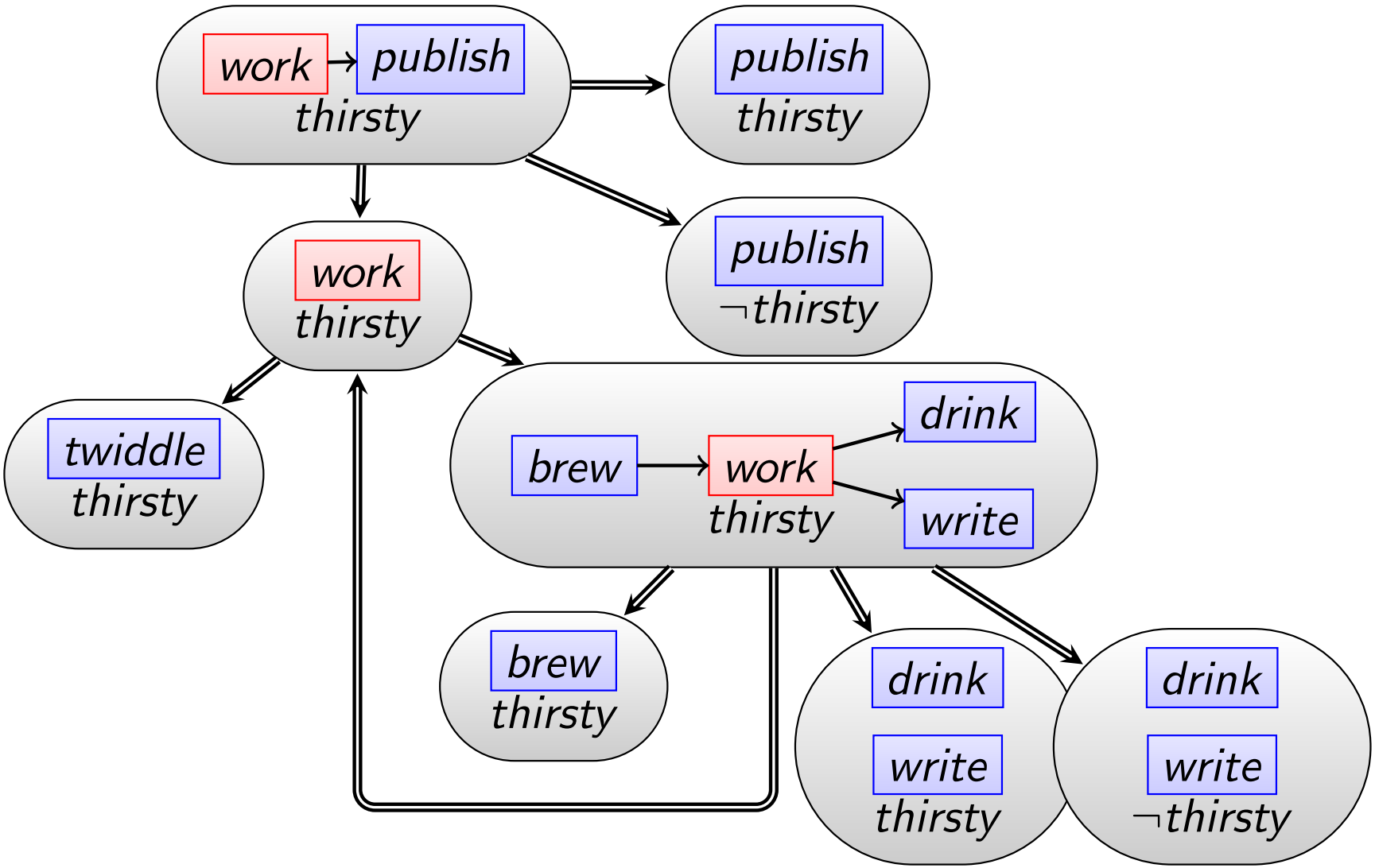
Total Order Partition spaces

Example



Total Order Partition spaces

Example



Total Order Partition spaces

$\leq_r$-ordered

- For finiteness, need to consider the structure of networks in the problem.
- Let $\langle tn_1, \ldots, tn_n \rangle$ be the totally ordered chunks of a network tn. tn is $\leq_r$-ordered if every tn_i is either:
  - singular (only one task)
  - $\leq_r$-stratifiable
- Initial task network and all methods $\leq_r$-ordered $\Rightarrow$ problem is $\leq_r$-ordered
- Checks that we never introduce a partial order that can't be solved via progression.

Total Order Partition spaces

$\leq_r$-ordered implications

- Includes $\leq_r$-stratifiable problems
- Includes totally ordered HTN planning.
  - Every finite-state finite-task domain that SHOP can handle.
  - SHOP may not terminate on these
  - Possible to develop a planner that does.
- Heuristic search may be fun:
  - Loops
  - Sequential dependencies between children

Wrap-up

Contributions:

- Characterized three different ways to search for HTN problems (fourth in paper)
- Two ways well implemented, but now we know when we can get them to terminate.
  - Stratification checks may be helpful for domain authors
- A new way to search, which terminates on more problems.

Future work:

- Heuristic search for TOP spaces
- Use $\leq_r$-stratification to find automatic bounds for translation based planning