Planning through Stochastic Local Search and Temporal Action Graphs in LPG

By Alfonso Gervini, Alessandro Saetti, Ivan Serina

Presented by Evan Damron

Overview

- LPG is a planning graph-based planner, capable of solving plans where actions take time (temporal), and consume resources (numerical).
- LPG locally searches the space of linear action graphs by adapting the SAT solver called Walk-SAT to the problem, calling it Walkplan.
- LPG was the best planner in the 3rd International Planning Competition (IPC-3) in 2002.

Action Graphs

- Action graphs are subgraphs of the planning graph, representing partially ordered plans.
- Action graphs contain a dummy start and end step.
- If an action is in an action graph, then so are it’s preconditions and effects.
- Action graphs can contain inconsistencies:
  - Fact nodes not supported by an action.
  - Mutexed actions.
- A solution graph is an action graph without inconsistencies.

Persistent Mutexes

- Mutexes decrease monotonically as the levels of the planning graph increase. i.e. all mutexes are guaranteed to be in all previous levels if the facts/actions involved in that mutex are in these levels.
- The *fixed-point* level of the graph is the point where nodes and mutexes will remain the same at all further levels.
- Mutexes at the fixed-point level are called *persistent mutexes*.
- In Graphplan, mutexes are computed a level at a time, which can be computationally very expensive.
- LPG speeds this up by only computing persistent mutexes.

Calculating Persistent Mutexes

ComputeMutexFacts($I, \mathcal{O}$)

Input: An initial state (I) and all ground operator instances ($\mathcal{O}$);
Output: A set of persistent mutex relations between facts (M).

```
F* ← I; F ← ∅;
M ← ∅; M* ← ∅; A ← ∅;
while F* ≠ F ∨ M* ≠ M
    F ← F*; M ← M*;
    forall a ∈ O such that Pre(a) ⊆ F* and ¬(∃p, q ∈ Pre(a) ∧ (p, q) ∈ M*)
        New(a) ← Add(a) − F*;
        forall f ∈ New(a)
            forall h ∈ Del(a)
                M* ← M* ∪ {(f, h), (h, f)};    /* Potential mutex relation */
            forall (p, q) ∈ M* such that p ∈ Pre(a) and q ∉ Del(a)
                M* ← M* ∪ {(f, q), (q, f)};    /* Potential mutex relation */
        if a ∉ A then
            forall p, q ∈ Add(a) such that (p, q) ∈ M*
                M* ← M* − {(p, q), (q, p)};    /* Invalid mutex relation */
        L ← Add(a) − New(a);
        forall (i, q) ∈ M* such that i ∈ L
            if q ∉ Del(a) ∧ ¬(∃p ∈ Pre(a) ∧ (p, q) ∈ M*) then
                M* ← M* − {(i, q), (q, i)};    /* Invalid mutex relation */
        F* ← F* ∪ New(a);
        A ← A ∪ {a};
return M.
```

Check if F1 is a positive effect for *a* and F2 is a negative effect for *a*.

Check if F1 is a positive effect for *a* and F2 is mutexed with a precondition of *a*.

F1 and F2 both belong to the add effects of an action.

F1 is an add effect of *a*, F2 is not deleted by *a*, and F2 isn't mutexed with any precondition of *a*.

Calculating Persistent Mutexes

ComputeMutexActions$(M, \mathcal{O})$

Input: A set of mutex relations between facts (M) and all ground operator instances $(\mathcal{O})$;
Output: A set of persistent mutex relations between actions (N).

$N \leftarrow \emptyset$; $\mathcal{O}^* \leftarrow \mathcal{O}$ extended with the no-op of every fact;
forall $(p, q) \in M$
    forall $a \in \mathcal{O}^*$ such that $p \in Pre(a)$
        forall $b \in \mathcal{O}^*$ such that $q \in Pre(b)$
            $N \leftarrow N \cup \{(a, b), (b, a)\}$; /* Competing needs */
forall $a \in \mathcal{O}^*$
    forall $p \in Pre(a)$
        forall $b \in \mathcal{O}$ such that $p \in Del(b)$
            $N \leftarrow N \cup \{(a, b), (b, a)\}$; /* Interference */
    forall $p \in Add(a)$
        forall $b \in \mathcal{O}$ such that $p \in Del(b)$
            $N \leftarrow N \cup \{(a, b), (b, a)\}$; /* Inconsistent effects */
return N.

Check if any preconditions of two actions are mutexed.

Check if one action deletes a precondition of another.

Check if one action deletes an add effect of another.

Walkplan

- Walkplan first computes the planning graph and persistent mutexes.
- Walkplan then creates an initial action graph. The default contains only the no-ops of the facts in the initial state until the fixed-point level (with the start and end actions).
- At each step, Walkplan will choose an inconsistency and attempt to resolve it.
  - If the inconsistency is an unsupported fact:
    - Remove an action that needs this fact, OR
    - Add an action to produce this fact.
  - If the inconsistency is a mutex, remove one of the actions in the mutex relationship.
- Walkplan has a noise parameter *p* to prevent getting stuck in local optima.
- The *neighborhood* of an inconsistency in an action plan are all the possible action plans that can result from fixing the inconsistency.

Walkplan pseudocode

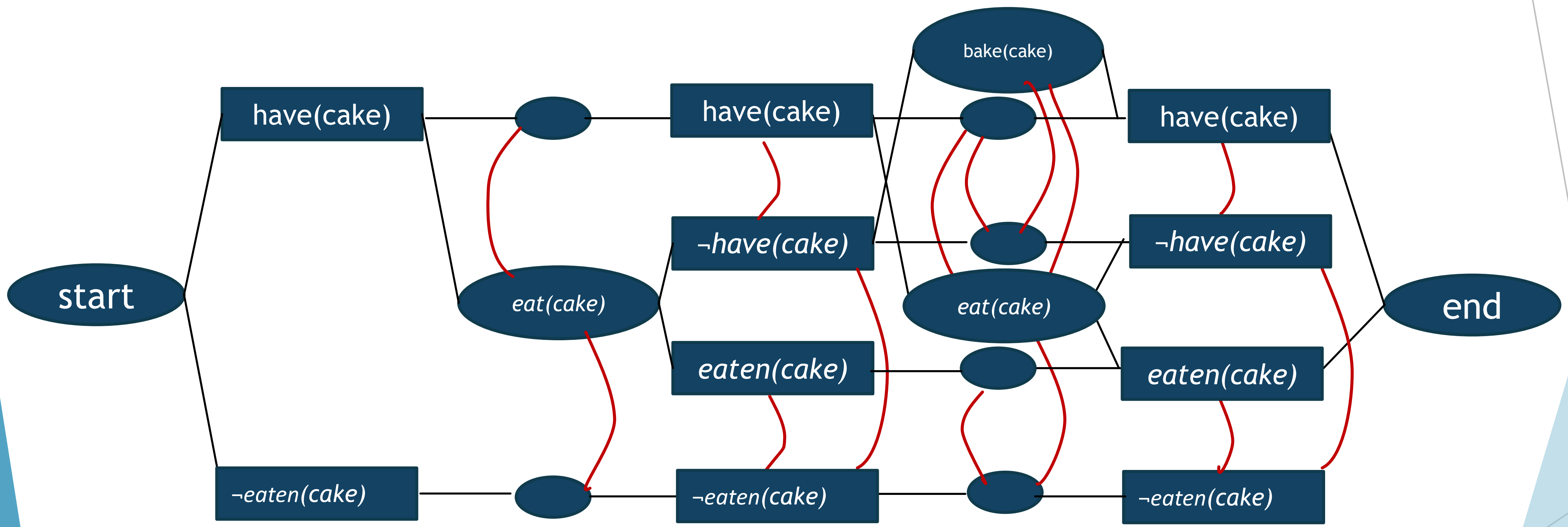
Walkplan$(\Pi, max_steps, max_restarts, p)$

Input: A planning problem Π, the maximum number of search steps max_steps,
the maximum number of search restarts $max\_restarts$, a noise factor p $(0 \leq p \leq 1)$.

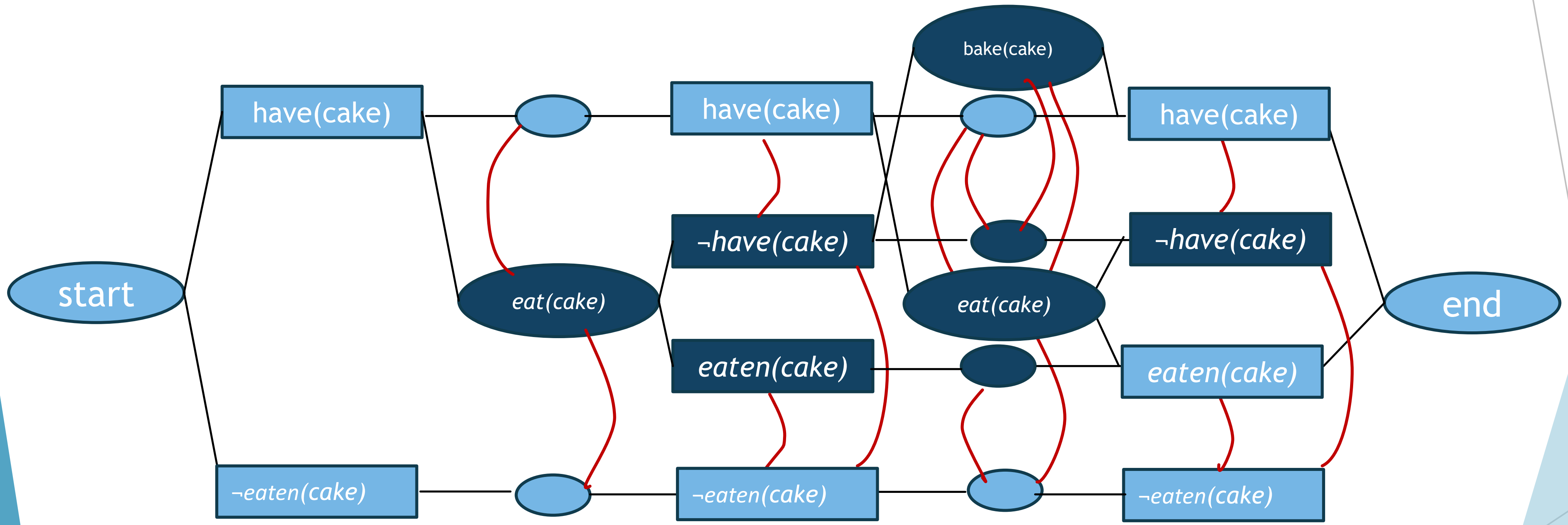
Output: A solution graph representing a plan solving Π or `fail`.

```
for i ← 1  to max_restarts do
    𝒜 ← an initial A-graph derived from the planning graph of Π;
    for j ← 1 to max_steps do
        if 𝒜 is a solution graph then
            return 𝒜
        σ ← an inconsistency in 𝒜;
        N(σ, 𝒜) ← neighborhood of 𝒜 for σ;
        if ∃ 𝒜′ ∈ N(σ, 𝒜) such that the quality of 𝒜′ is not worse than the quality of 𝒜
            then 𝒜 ← 𝒜′ (if there is more than one 𝒜′-graph, choose randomly one)
        else if random < p then
                𝒜 ← an element of N(σ, 𝒜) randomly chosen
                else 𝒜 ← best element in N(σ, 𝒜);
return fail.
```

Walkplan Example

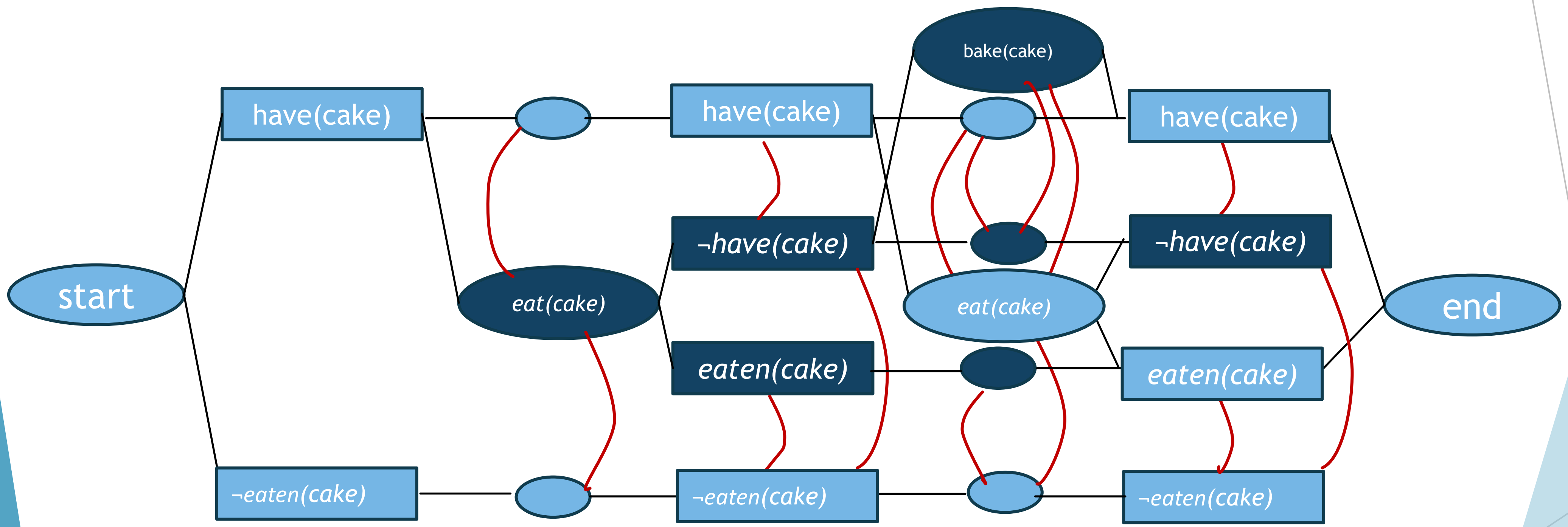


Walkplan Example



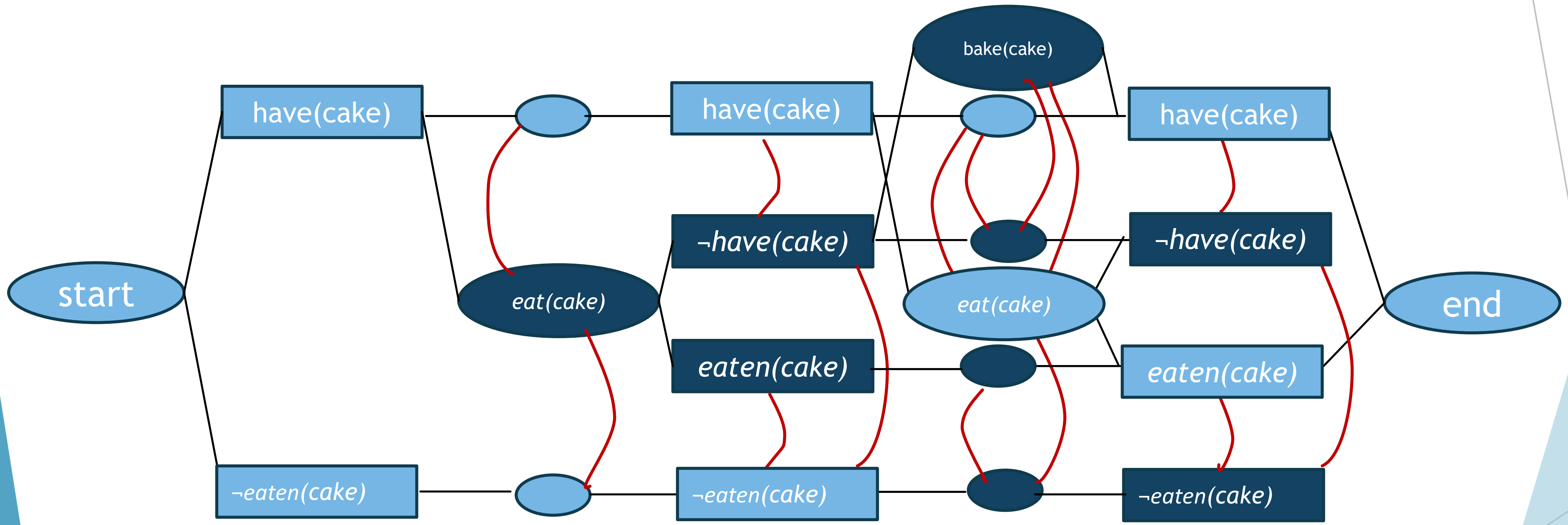
Inconsistency count: 1
Selected Inconsistency: *eaten(cake)* is unsupported
Neighborhood: Add supporting action *eat(cake)* or no-op

Walkplan Example



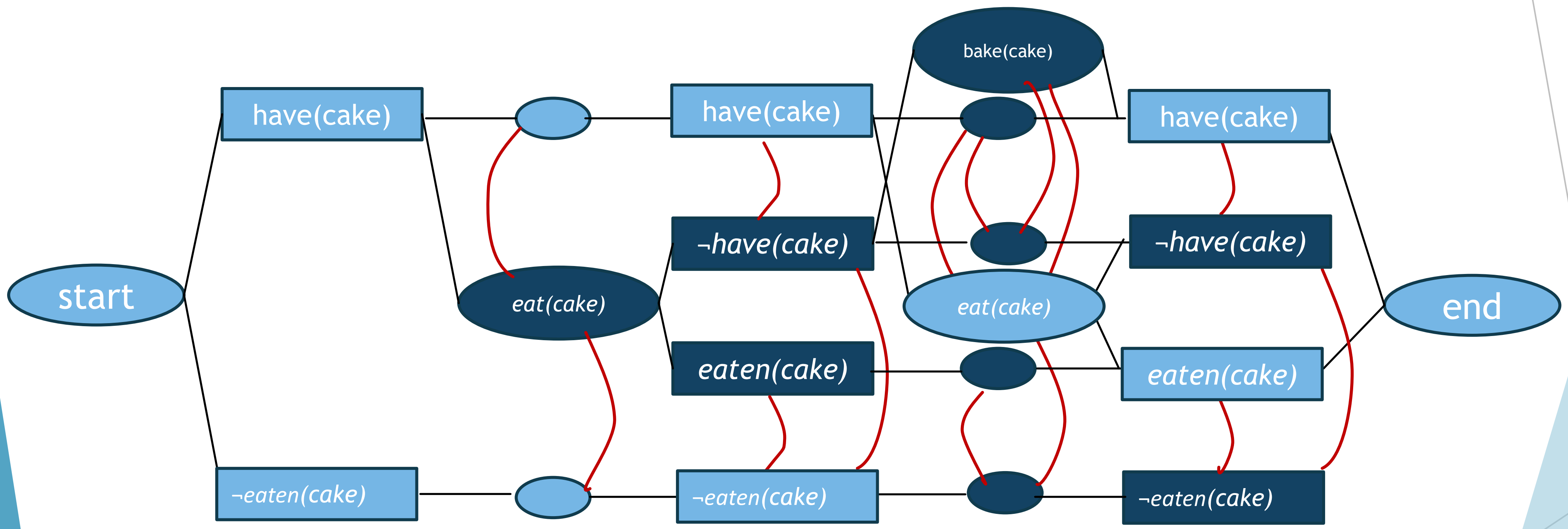
Inconsistency count: 2
Selected Inconsistency: *eat(cake)* mutex with *¬eaten(cake)* no-op
Neighborhood: Remove *eat(cake)* or *¬eaten(cake)* no-op

Walkplan Example



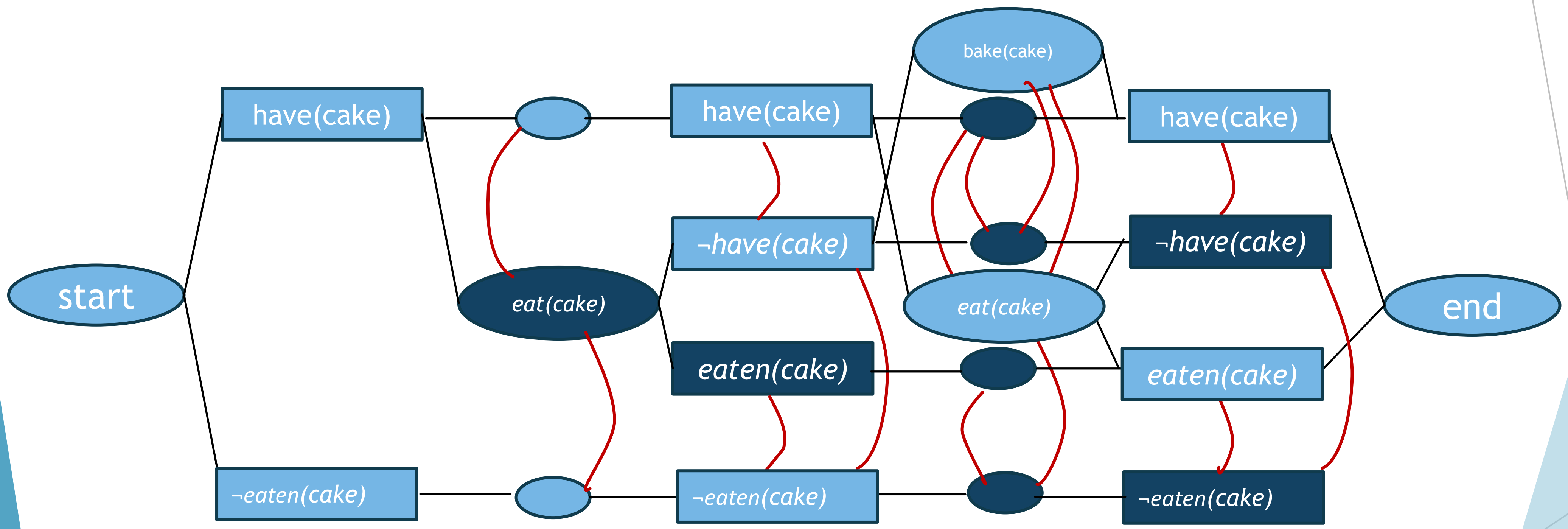
Inconsistency count: 1
Selected Inconsistency: *eat(cake)* mutex with *have(cake)* no-op
Neighborhood: Remove *eat(cake)* or *¬have(cake)* no-op

Walkplan Example



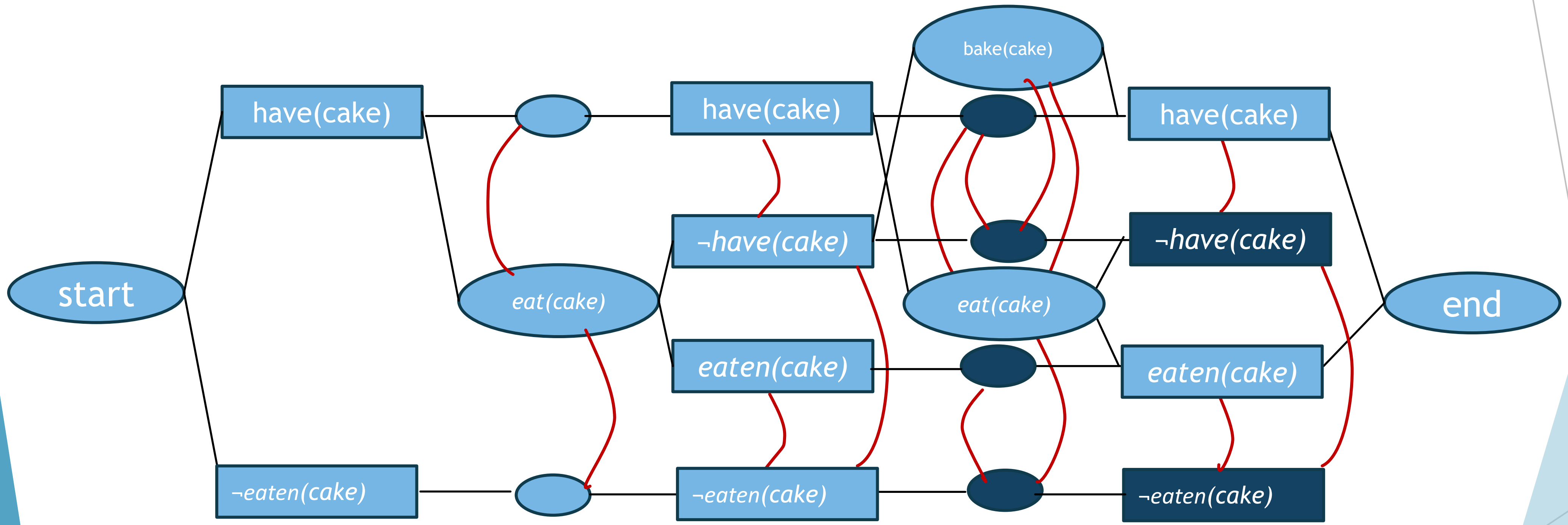
Inconsistency count: 1
Selected Inconsistency: unsupported fact *have(cake)*
Neighborhood: add *bake(cake)* action or no-op

Walkplan Example



Inconsistency count: 2
Selected Inconsistency: unsupported fact ¬ *have(cake)*
Neighborhood: add *bake(cake)* action

Walkplan Example

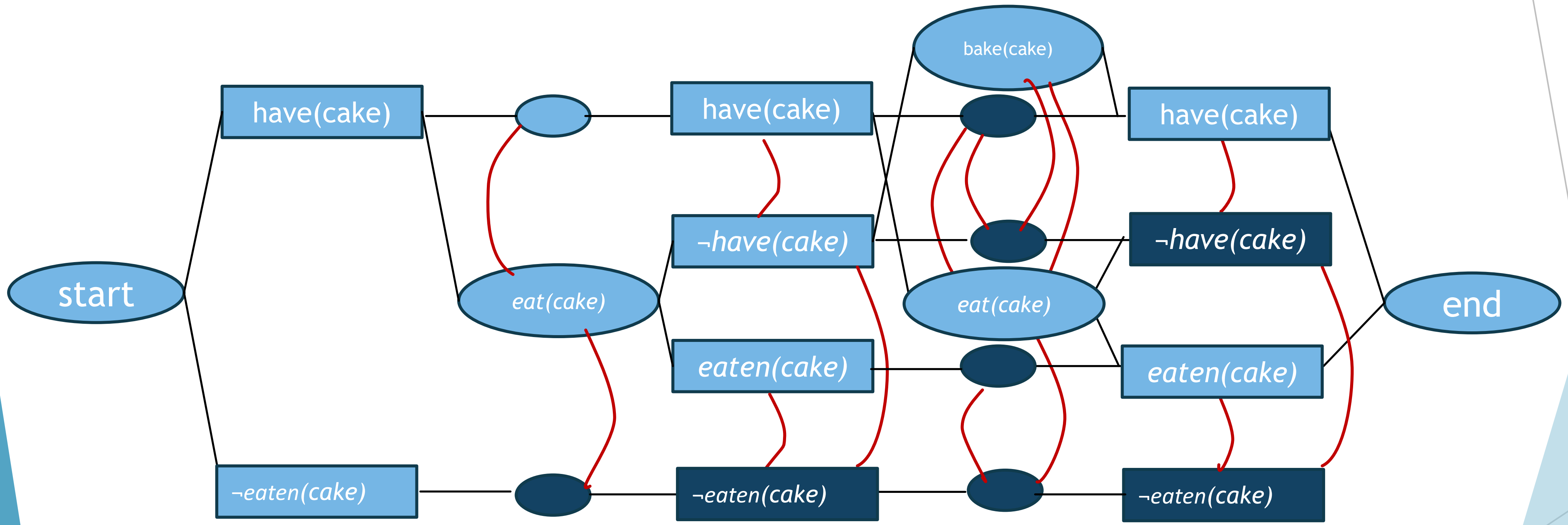


Inconsistency count: 3

Selected Inconsistency: mutex between *eat(cake)* and *¬ eaten(cake) no-op*

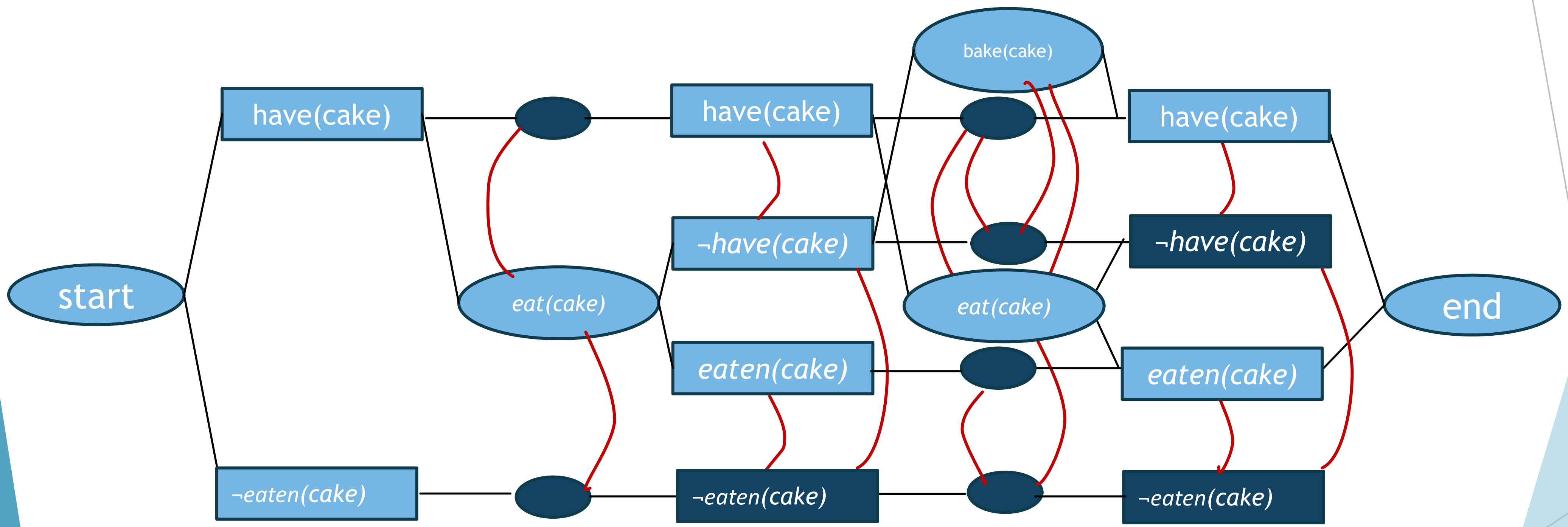
Neighborhood: remove *eat(cake) or ¬ eaten(cake) no-op*

Walkplan Example



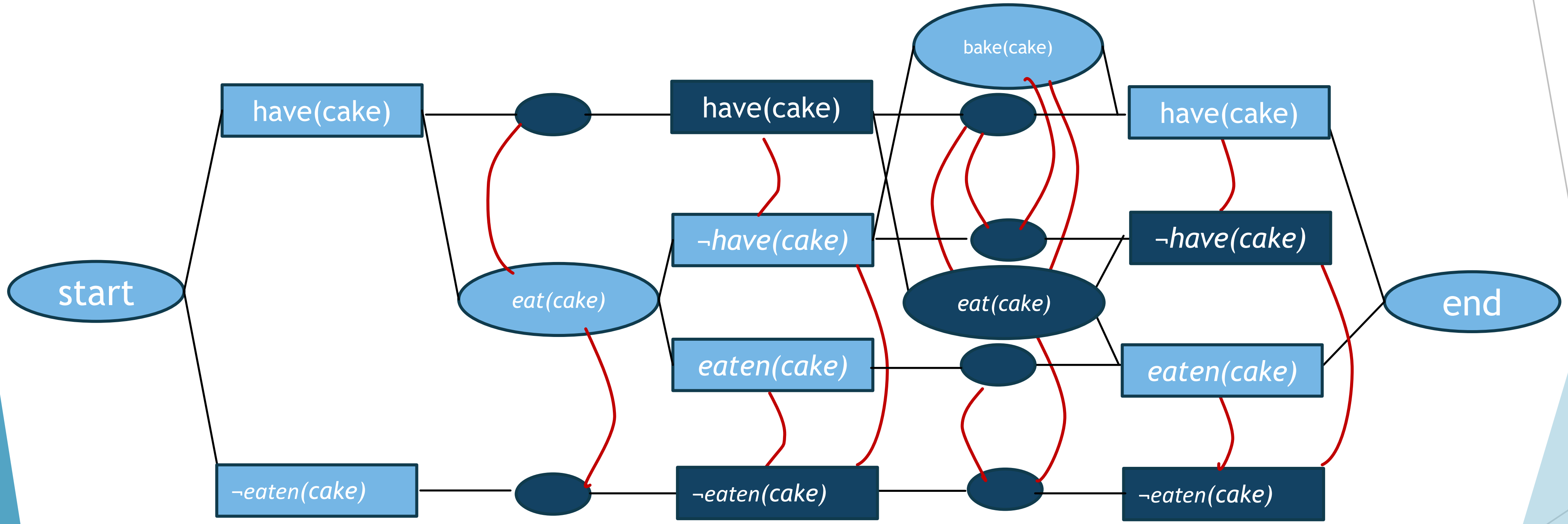
Inconsistency count: 2
Selected Inconsistency: mutex between *eat(cake)* and *have(cake) no-op*
Neighborhood: remove *eat(cake) or have(cake) no-op*

Walkplan Example



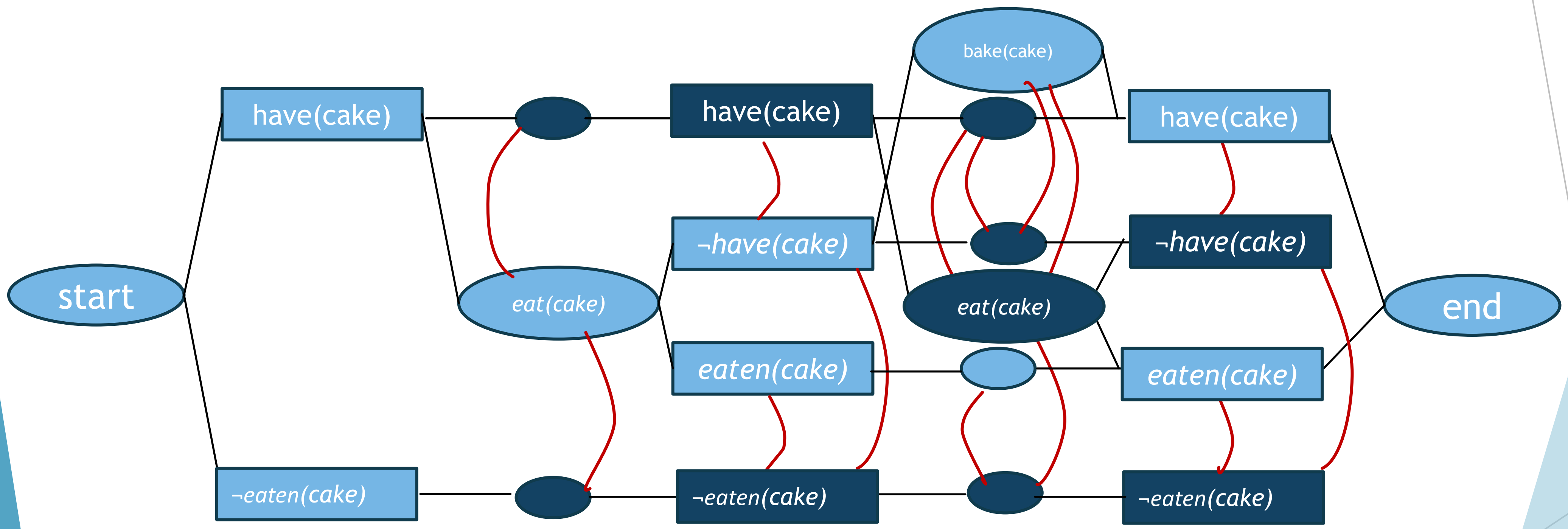
Inconsistency count: 2
Selected Inconsistency: unsupported fact *have(cake)*
Neighborhood: remove *eat(cake) or* add *have(cake)* no-op

Walkplan Example



Inconsistency count: 1
Selected Inconsistency: unsupported fact *eaten(cake)*
Neighborhood: add *eat(cake) or* add *eaten(cake)* no-op

Walkplan Example



Inconsistency count: 0

Return finished plan

Takeaways

- Local search through the space of planning graphs can be more effective and scalable than depth-first search.
- Calculating persisting mutexes is more efficient.
- Local search allows for plan refinement if you have an initial plan to start with.