Plan Graphs and Graphplan

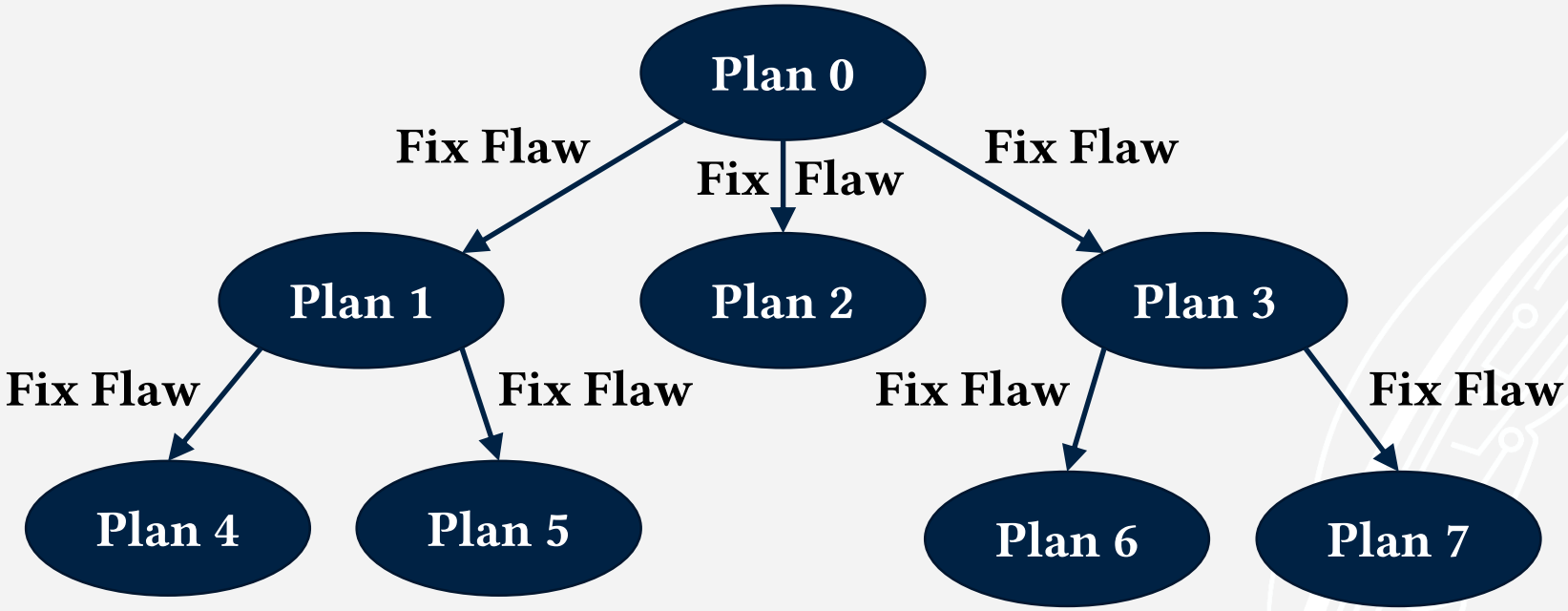
Stephen G. Ware

CSCI 4525 / 5525

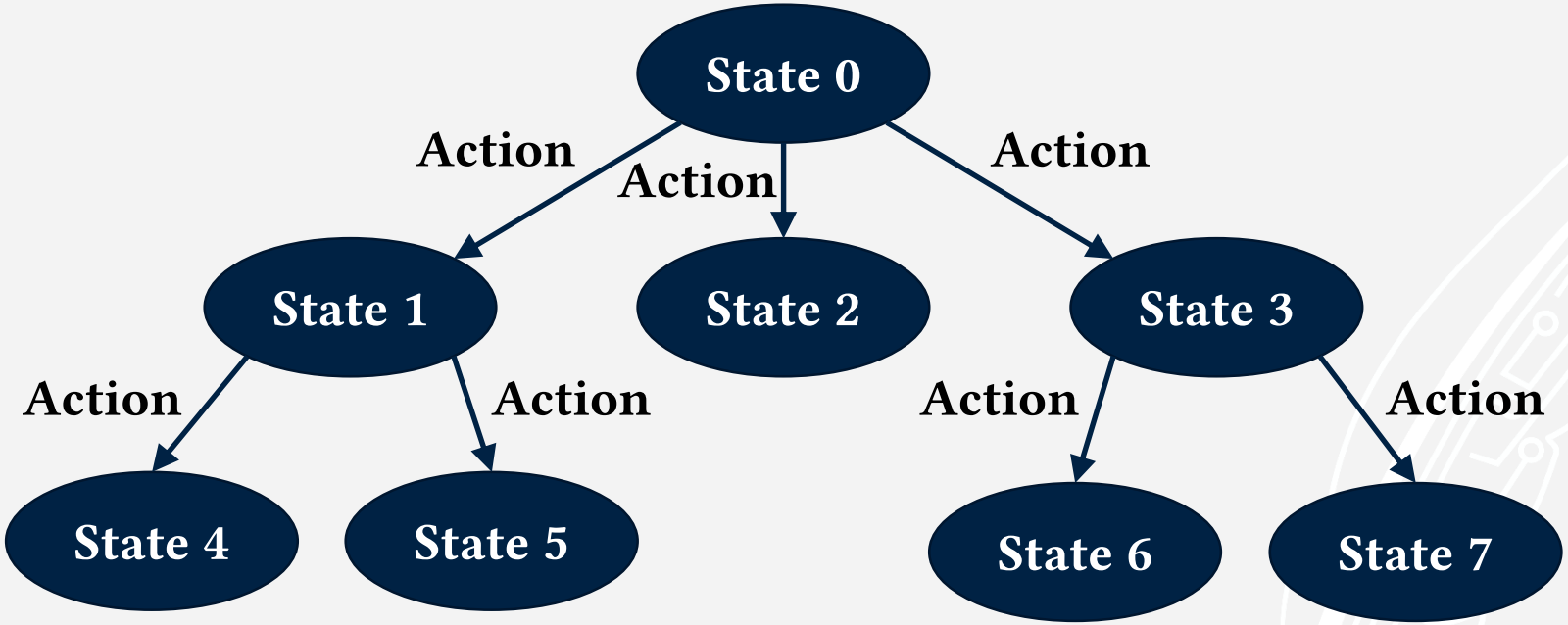
Planning So Far

- The search space of planning problems explode quickly.
- Abstraction helps by grouping many choices into a single choice.
- The abstractions discussed so far require complicated plan-space search techniques which may never terminate.
- Highly accurate heuristics may allow for traditional state-space search.

Plan-Space Search



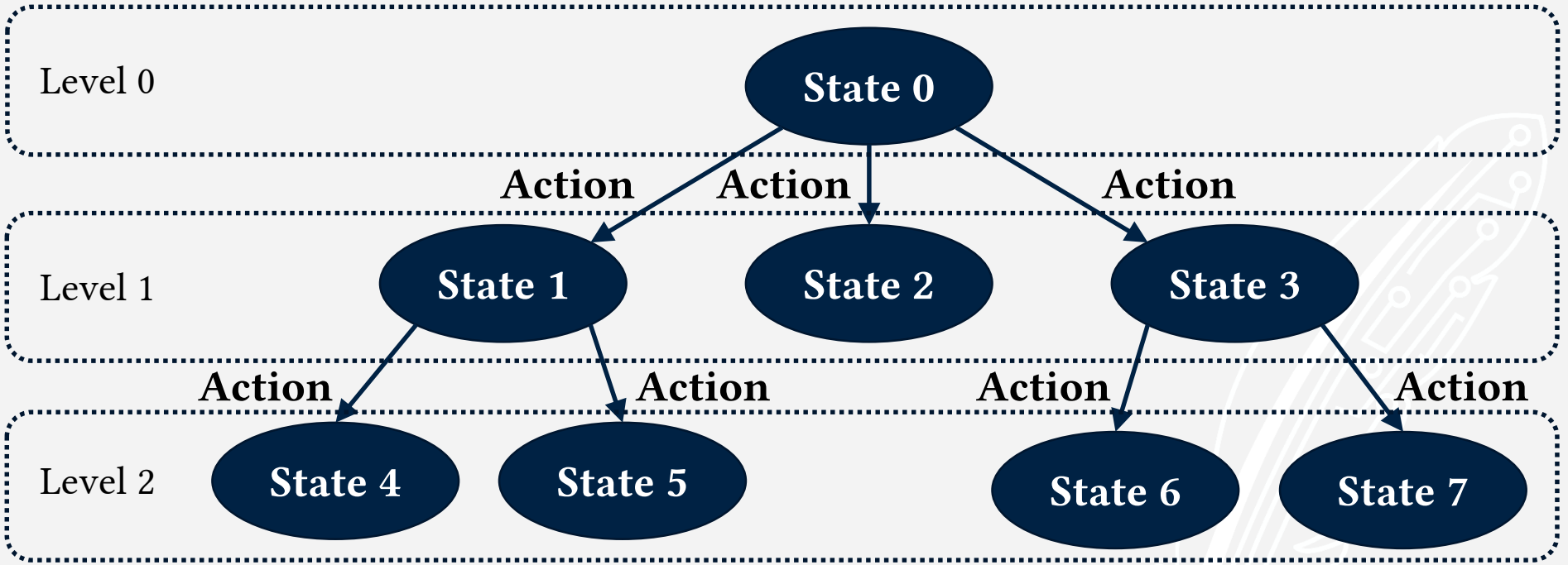
State-Space Search



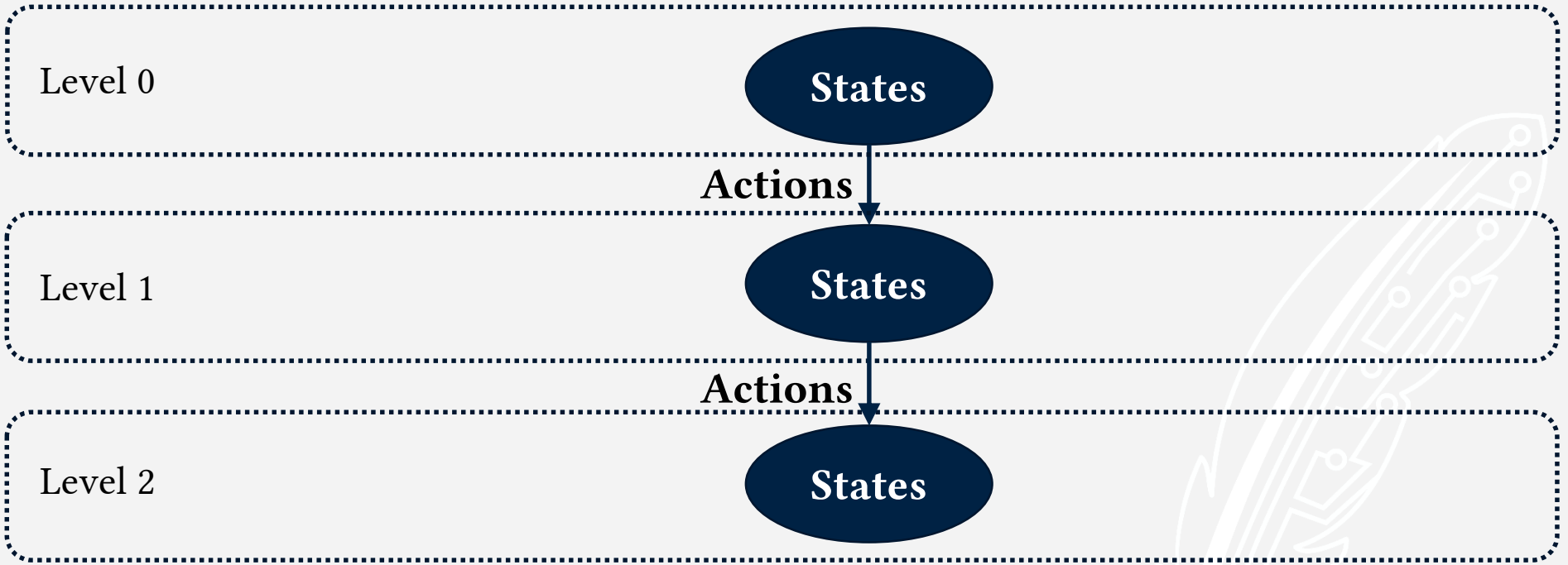
Plan Graphs: Concept

By abstracting the state space, we can get accurate estimates from a data structure that is much smaller and easier to build.

State Space


Level 0
State 0
Action
Action
Action
Level 1
State 1
State 2
State 3
Action
Action
Action
Action
Level 2
State 4
State 5
State 6
State 7

State Space Abstracted



Plan Graph

A **plan graph** is a directed, leveled graph with two kinds of nodes:

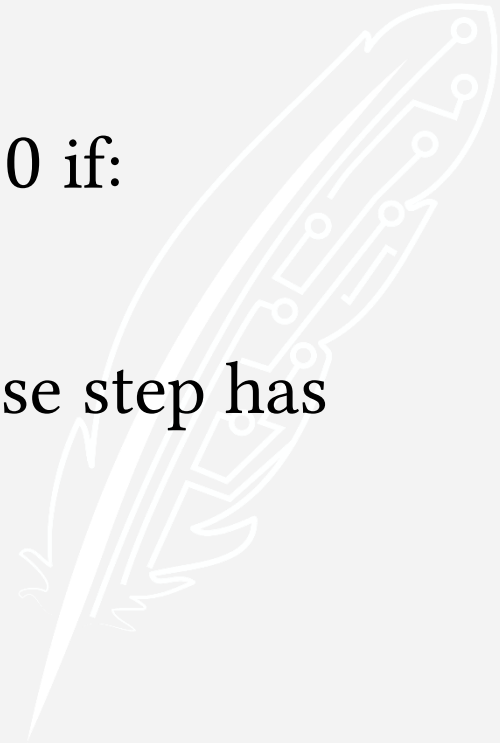
- A **literal nodes** represents a single ground predicate literal.
- A **step node** represents a ground action.

Literal Nodes

A node for literal L appears at level 0 if L is true in the initial state.

A node for literal L appears at level $n > 0$ if:

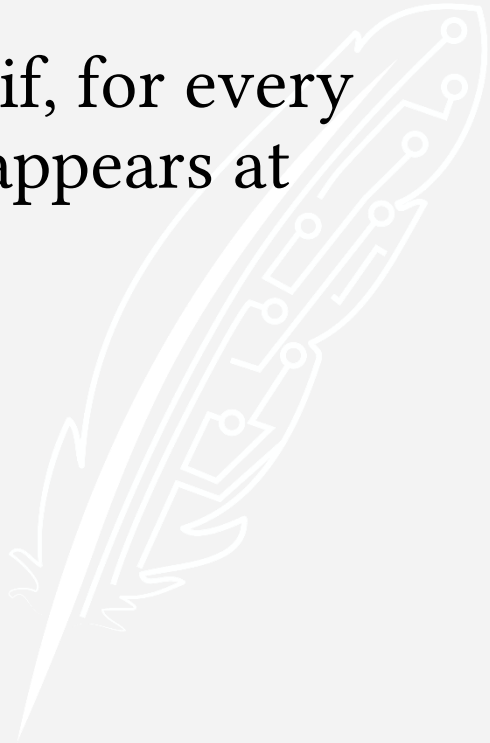
- A node for L appears at level $n - 1$.
- There exists a step node at level n whose step has L as an effect.



Step Nodes

No step nodes exist at layer 0.

A node for step S appears at level $n > 0$ if, for every precondition literal P of S, a node for P appears at level $n - 1$.



Cake Domain

Initial State: $have(Cake)$

Goal: $have(Cake) \wedge eaten(Cake)$

Action: $eat(Cake)$

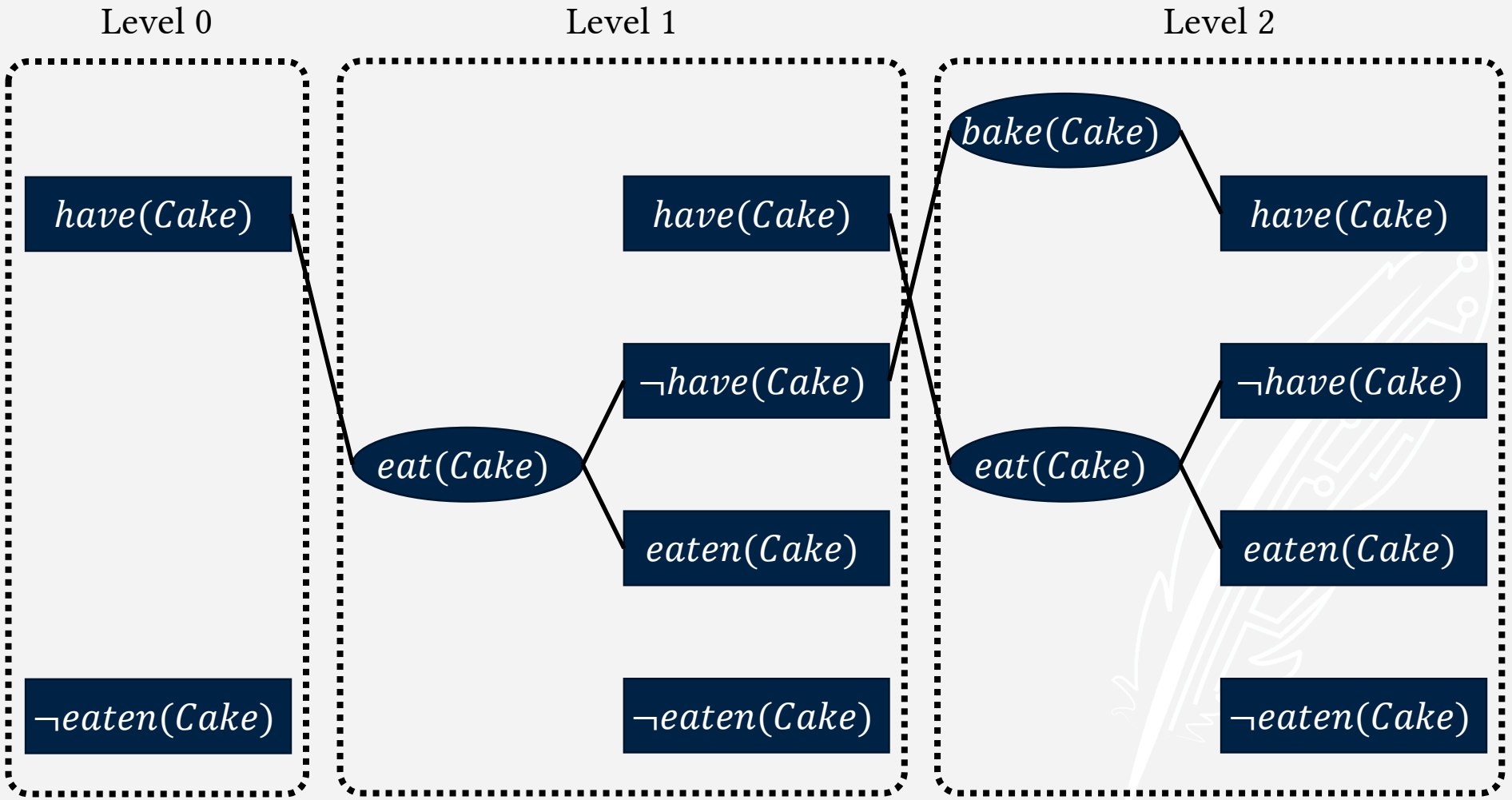
Precondition: $have(Cake)$

Effect: $eaten(Cake) \wedge \neg have(Cake)$

Action: $bake(Cake)$

Precondition: $\neg have(Cake)$

Effect: $have(Cake)$

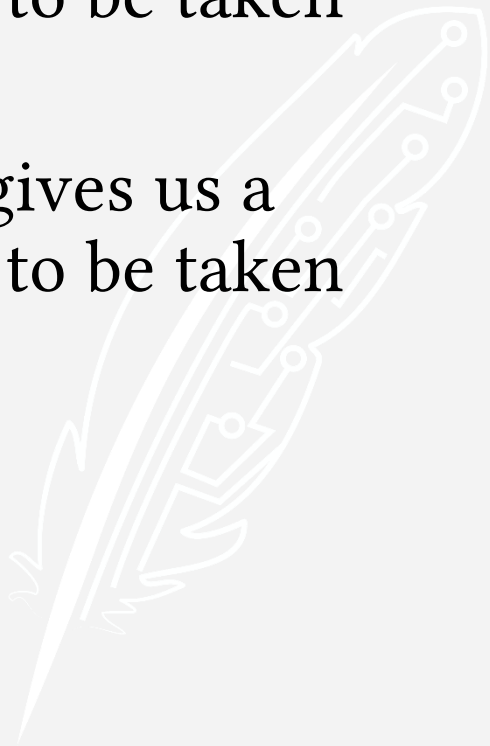

Level 0
Level 1
Level 2
have(Cake)
¬eaten(Cake)
have(Cake)
¬have(Cake)
eat(Cake)
eaten(Cake)
¬eaten(Cake)
bake(Cake)
have(Cake)
¬have(Cake)
eat(Cake)
eaten(Cake)
¬eaten(Cake)

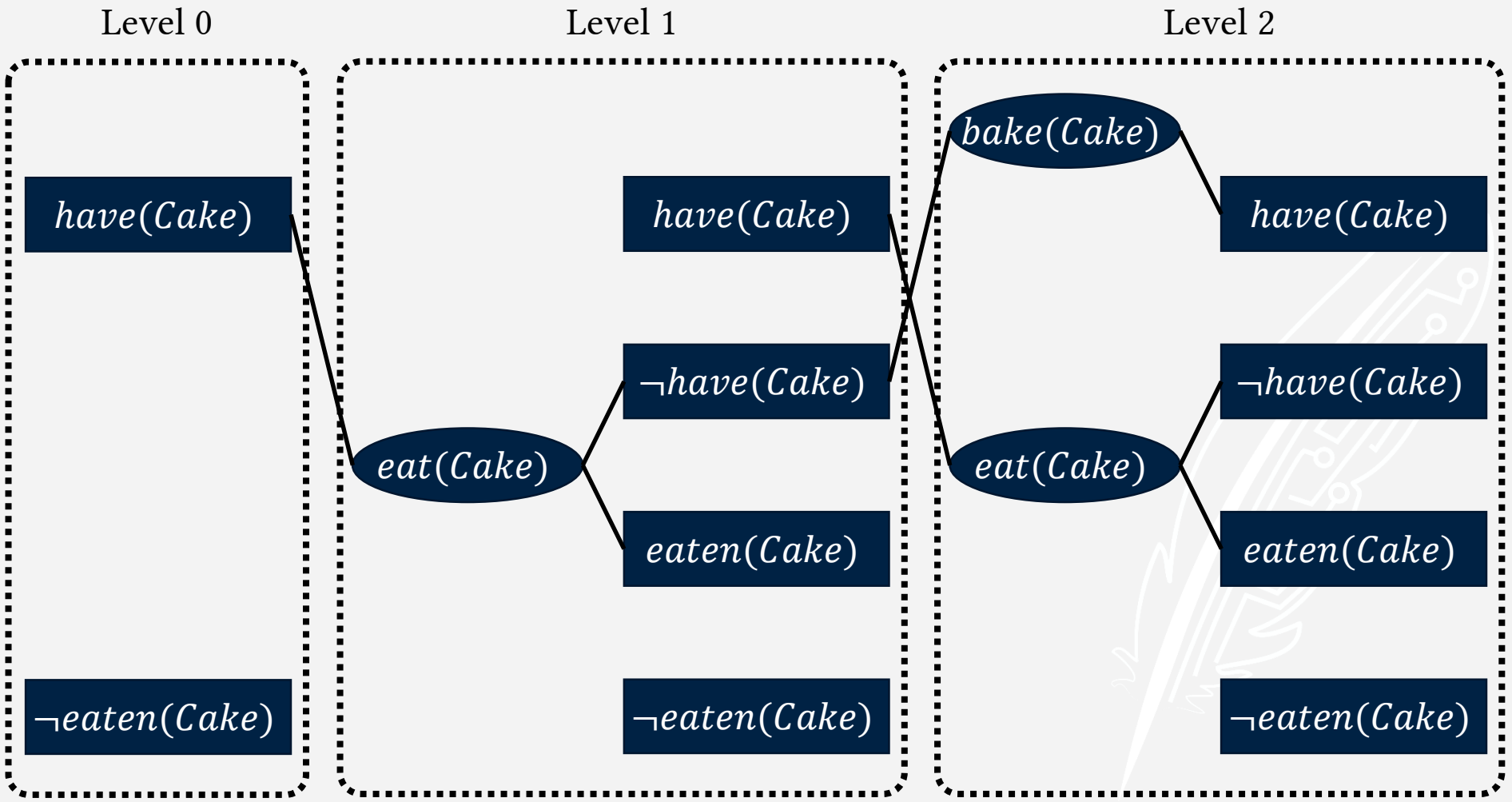
Features of Plan Graphs

- The literals and steps at each level are monotonically increasing.
- It is possible that the literals X and $\neg X$ appear at the same level. This impossible state is the price we pay for abstraction.
- A plan graph can be built in polynomial time!

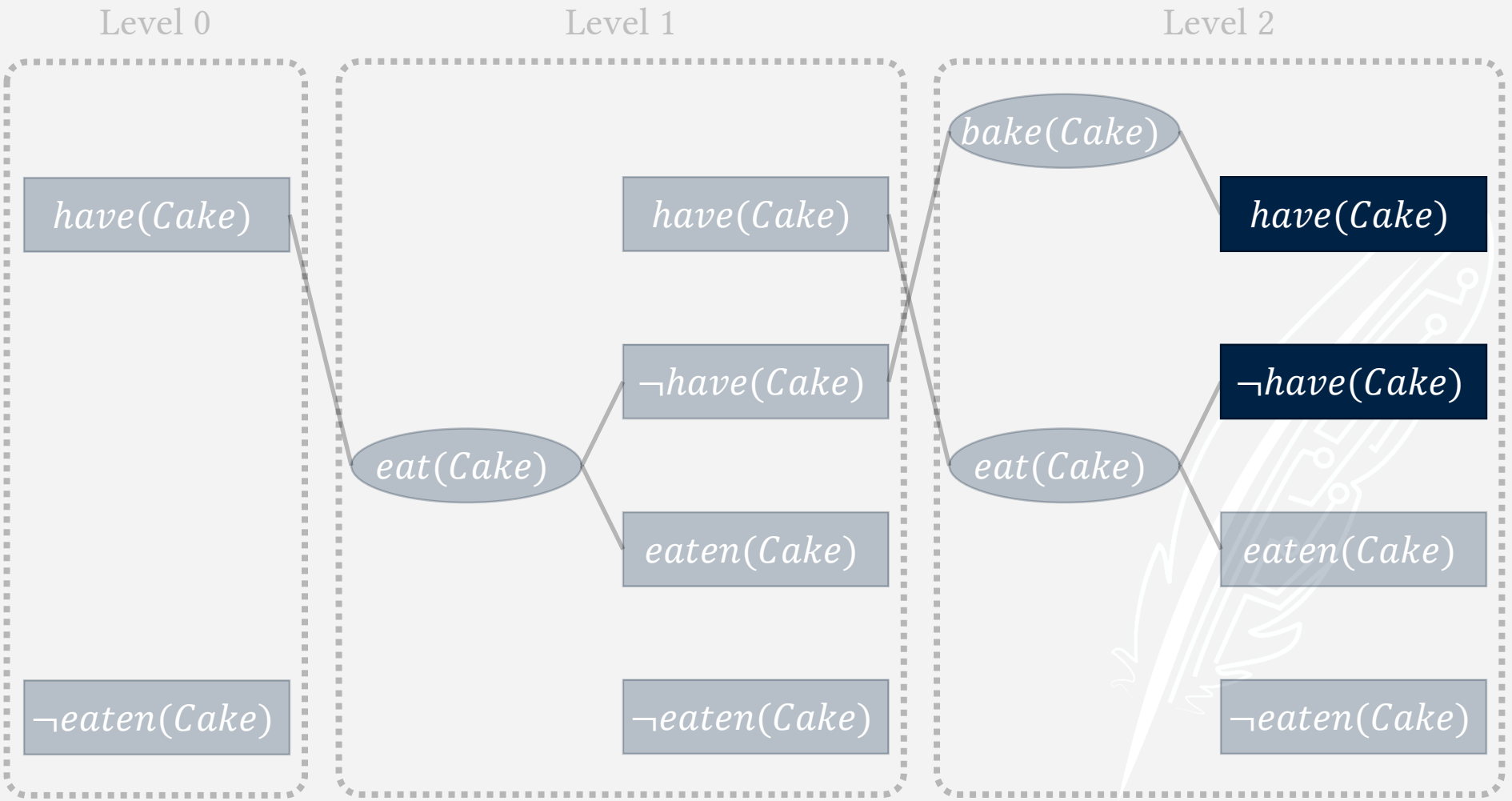
Plan Graphs Heuristics

- The level at which a literal first appears gives us a good estimate of how many steps need to be taken before the literal can be achieved.
- The level at which a step first appears gives us a good estimate of how many steps need to be taken before that step can be taken.
- Are these estimates admissible? Yes
- Will they every underestimate? Yes

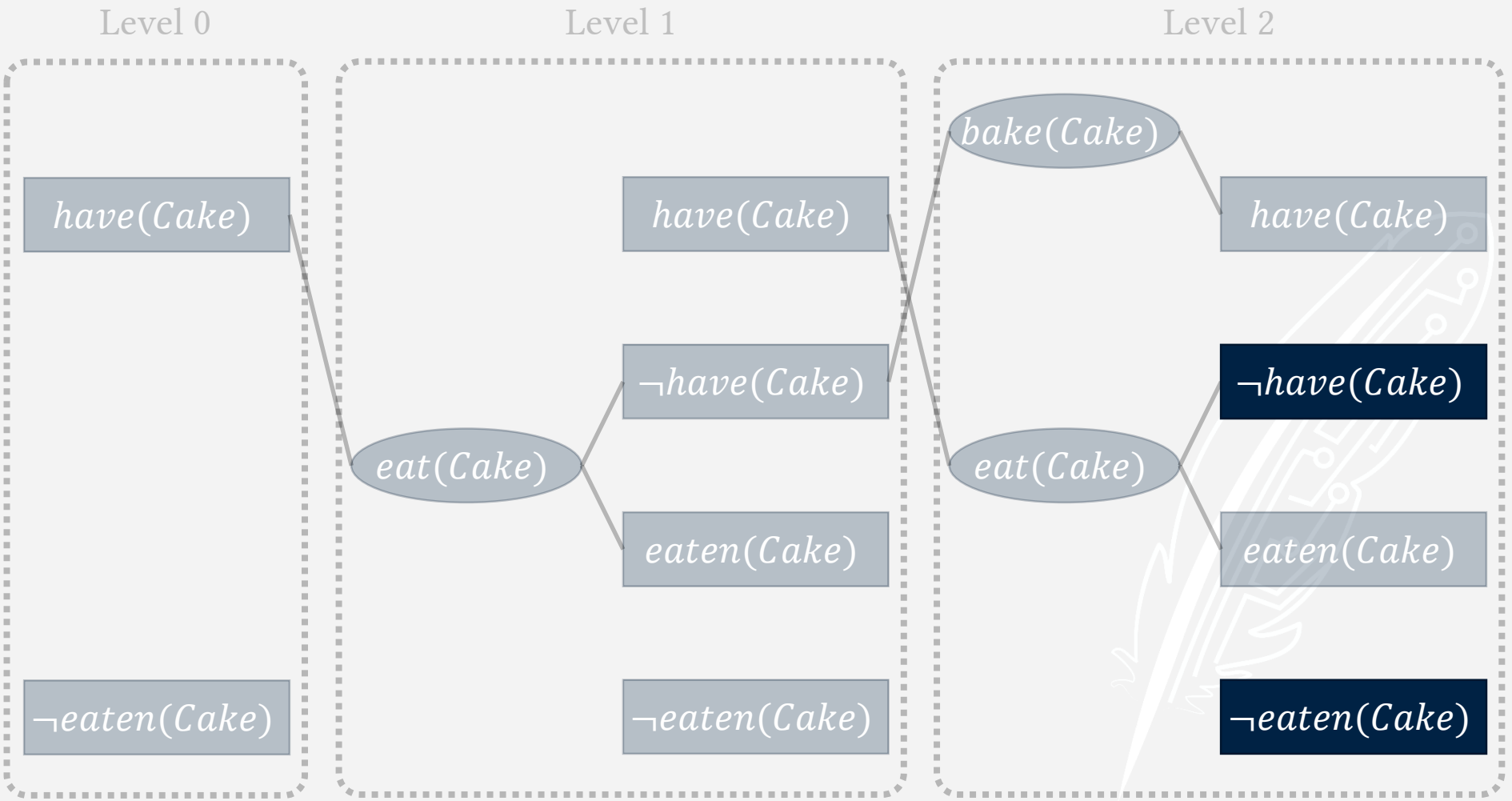



Level 0
Level 1
Level 2
have(Cake)
¬eaten(Cake)
have(Cake)
¬have(Cake)
eat(Cake)
eaten(Cake)
¬eaten(Cake)
bake(Cake)
have(Cake)
¬have(Cake)
eat(Cake)
eaten(Cake)
¬eaten(Cake)

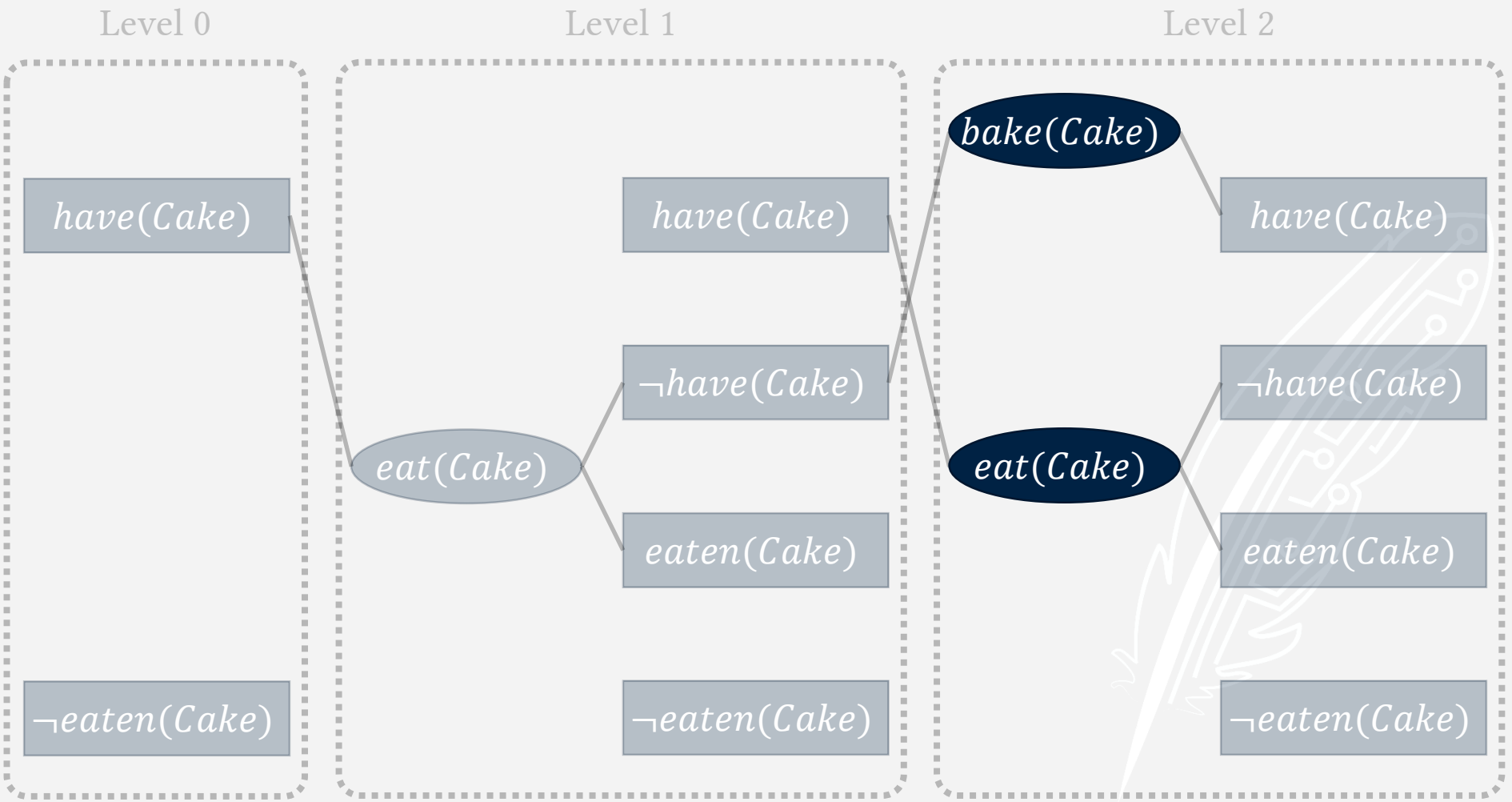
Can these two literals ever be true in the same state?



Can these two literals ever be true in the same state?



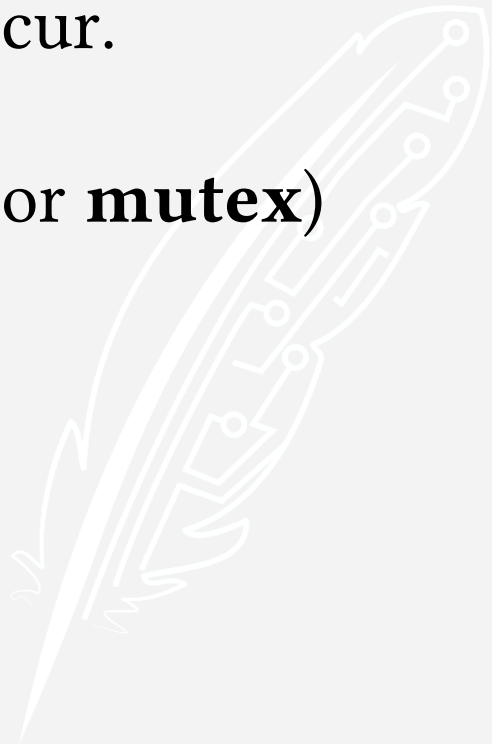
Can these two actions be taken in parallel without interfering with one another?



Increasing Accuracy

- We know that some literals cannot co-occur.
- We know that some steps cannot co-occur.

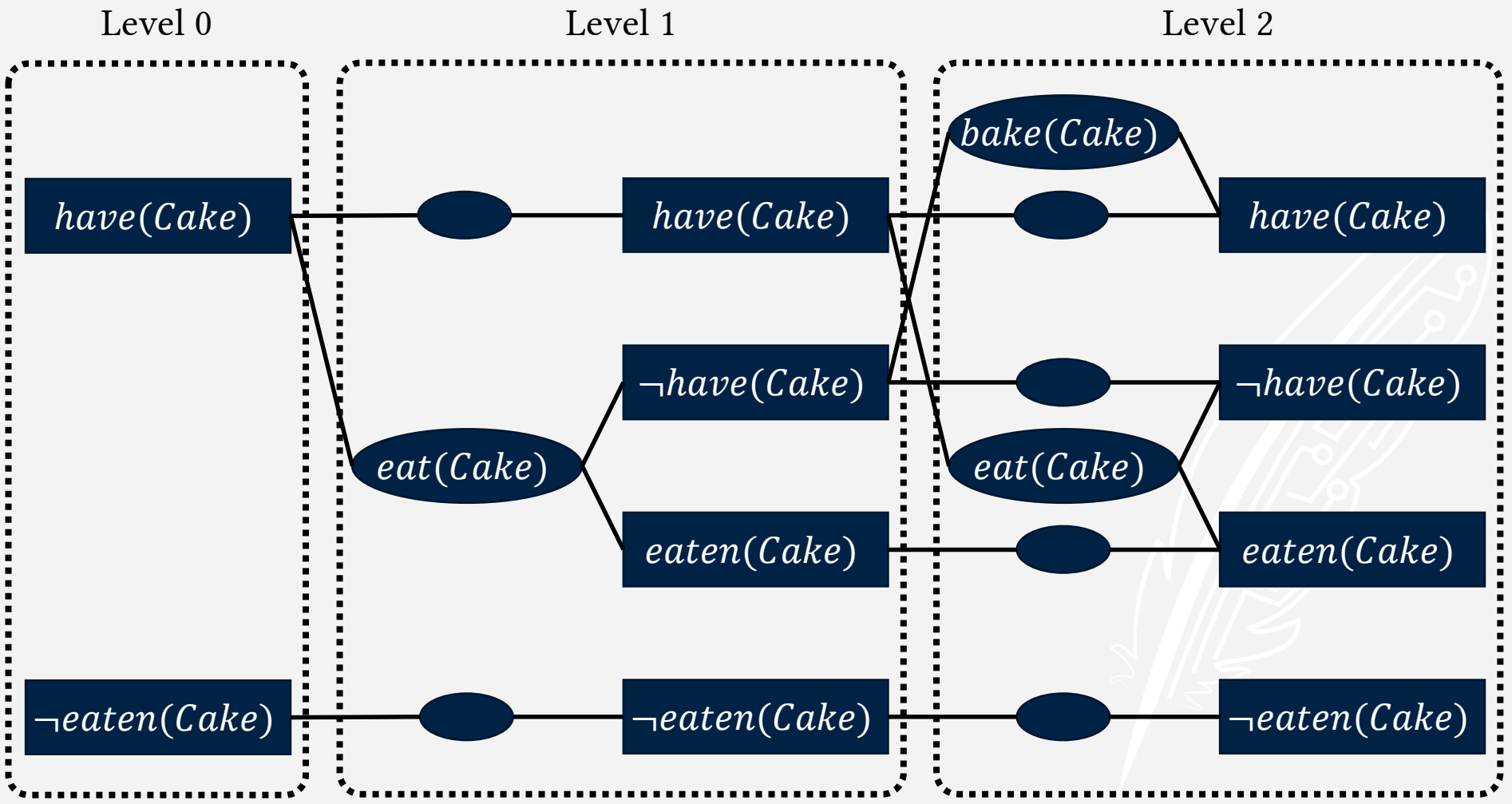
Improvement: Record mutual exclusion (or **mutex**) relations that express these ideas.



Persistence Steps

A **persistence step** for some literal L is a dummy step which has L as its precondition and L as its effect.

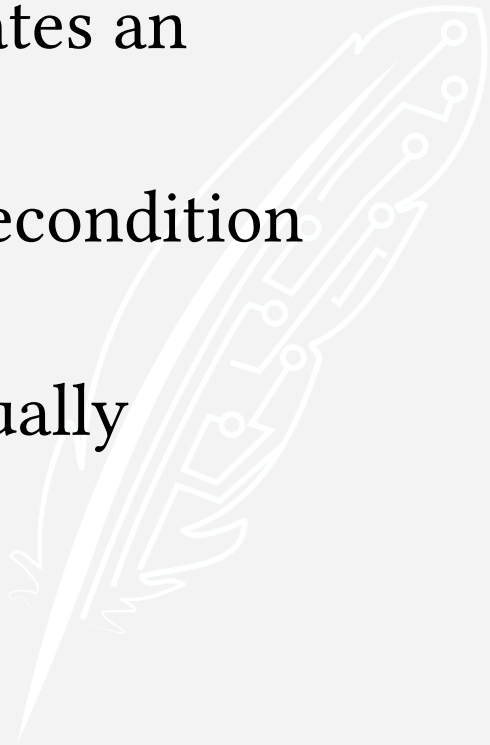
Persistence steps represent frame axioms (i.e. things that do not change from one moment to the next).


Level 0
Level 1
Level 2
have(Cake)
¬eaten(Cake)
have(Cake)
¬have(Cake)
eat(Cake)
eaten(Cake)
¬eaten(Cake)
bake(Cake)
have(Cake)
¬have(Cake)
eat(Cake)
eaten(Cake)
¬eaten(Cake)

Step Mutex Relations

Two steps cannot be taken in parallel when:

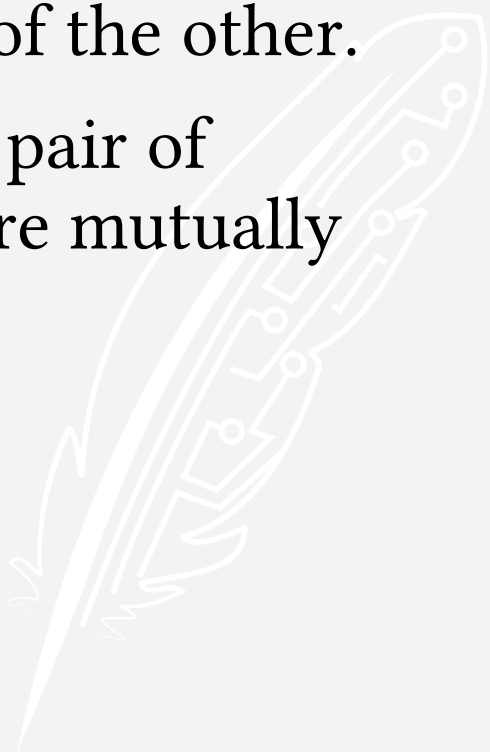
- **Inconsistent Effects:** One action negates an effect of the other.
- **Interference:** One action negates a precondition of the other.
- **Competing Needs:** Actions have mutually exclusive preconditions.

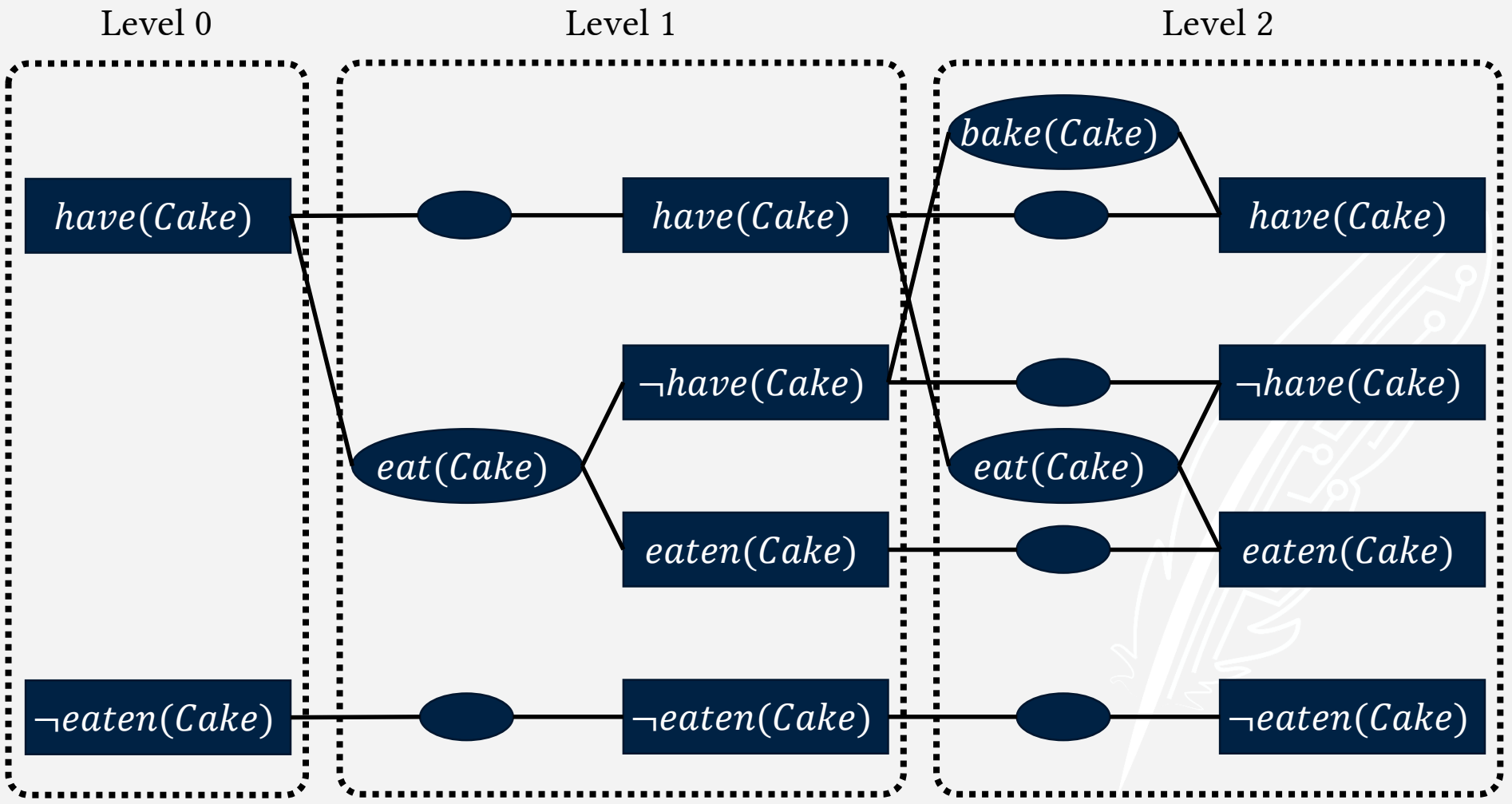


Literal Mutex Relations

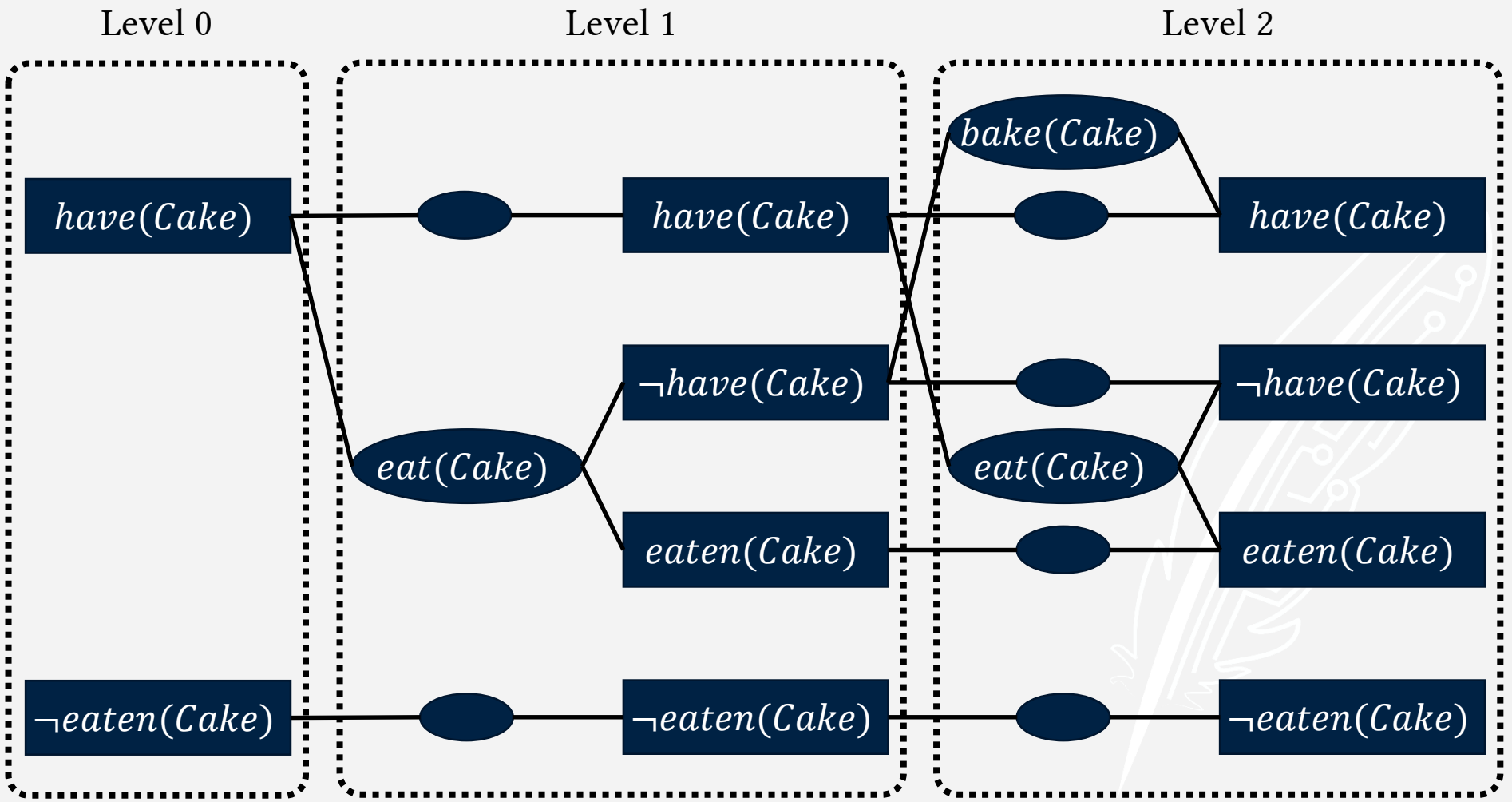
Two literals cannot occur at the same time when:

- **Opposites:** One literal is the negation of the other.
- **Inconsistent Support:** Every possible pair of actions that could achieve the literals are mutually exclusive.

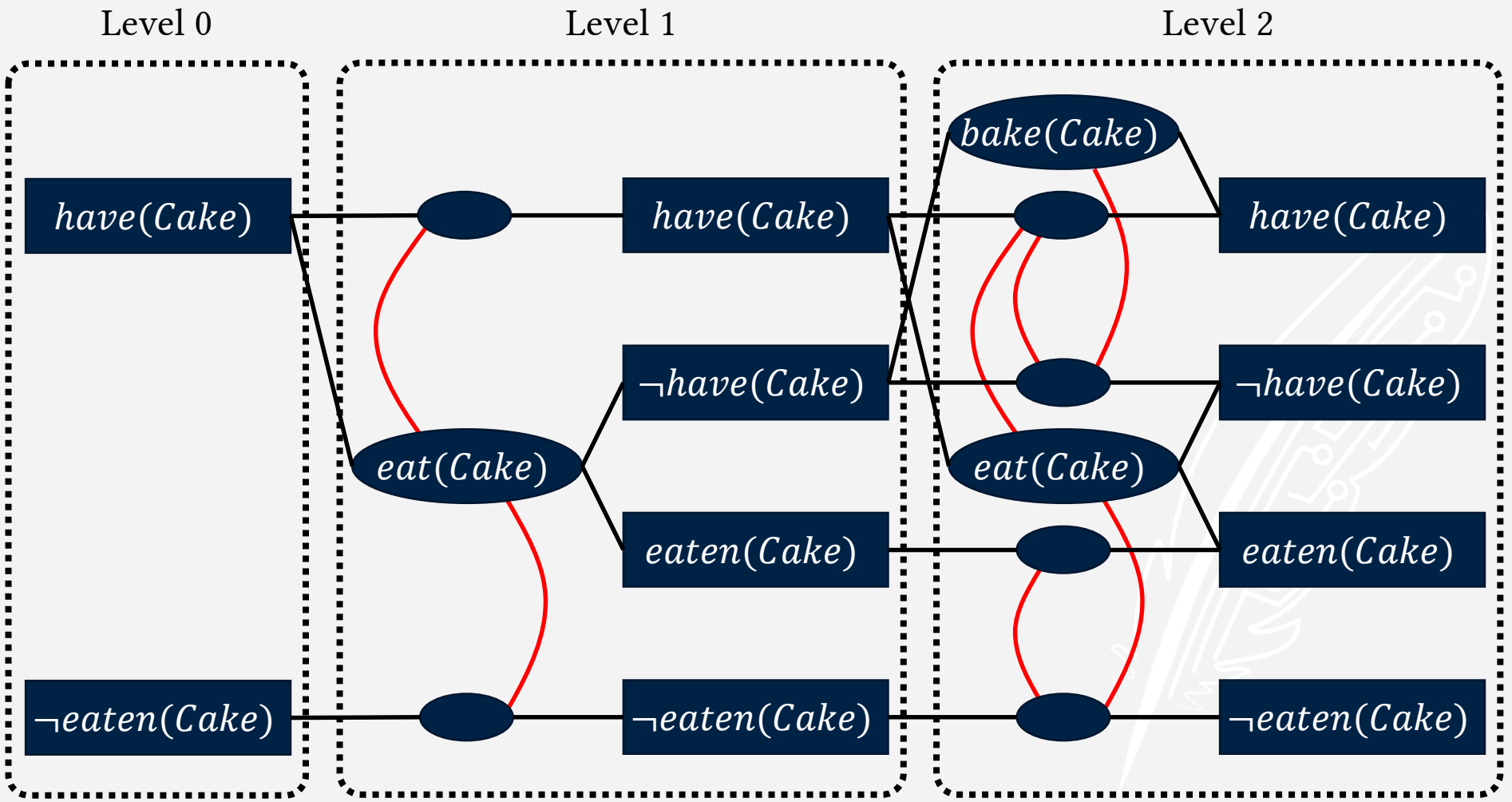



Level 0
Level 1
Level 2
have(Cake)
¬eaten(Cake)
have(Cake)
¬have(Cake)
eat(Cake)
eaten(Cake)
¬eaten(Cake)
bake(Cake)
have(Cake)
¬have(Cake)
eat(Cake)
eaten(Cake)
¬eaten(Cake)

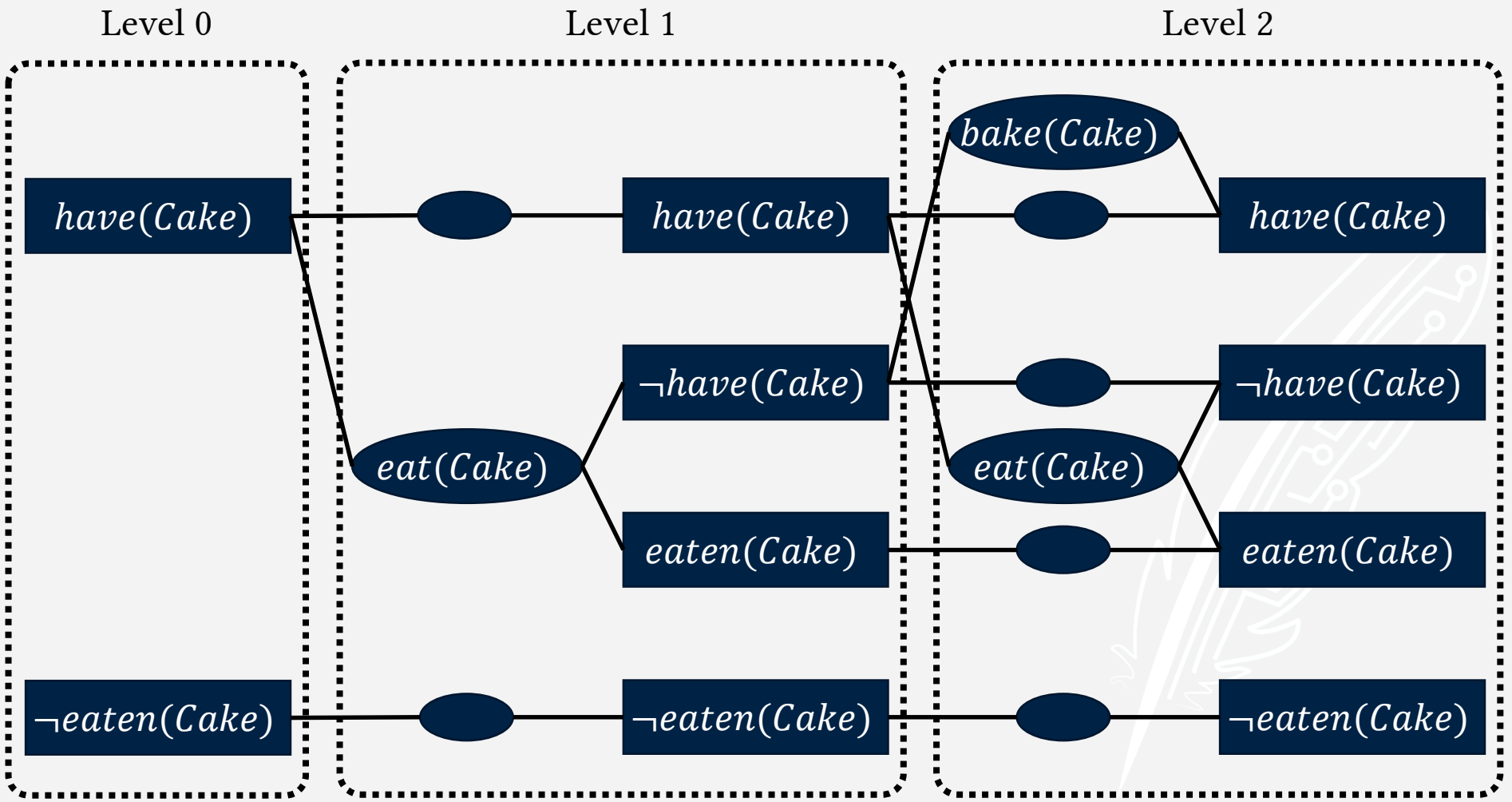
Inconsistent Effects: One action negates an effect of the other.



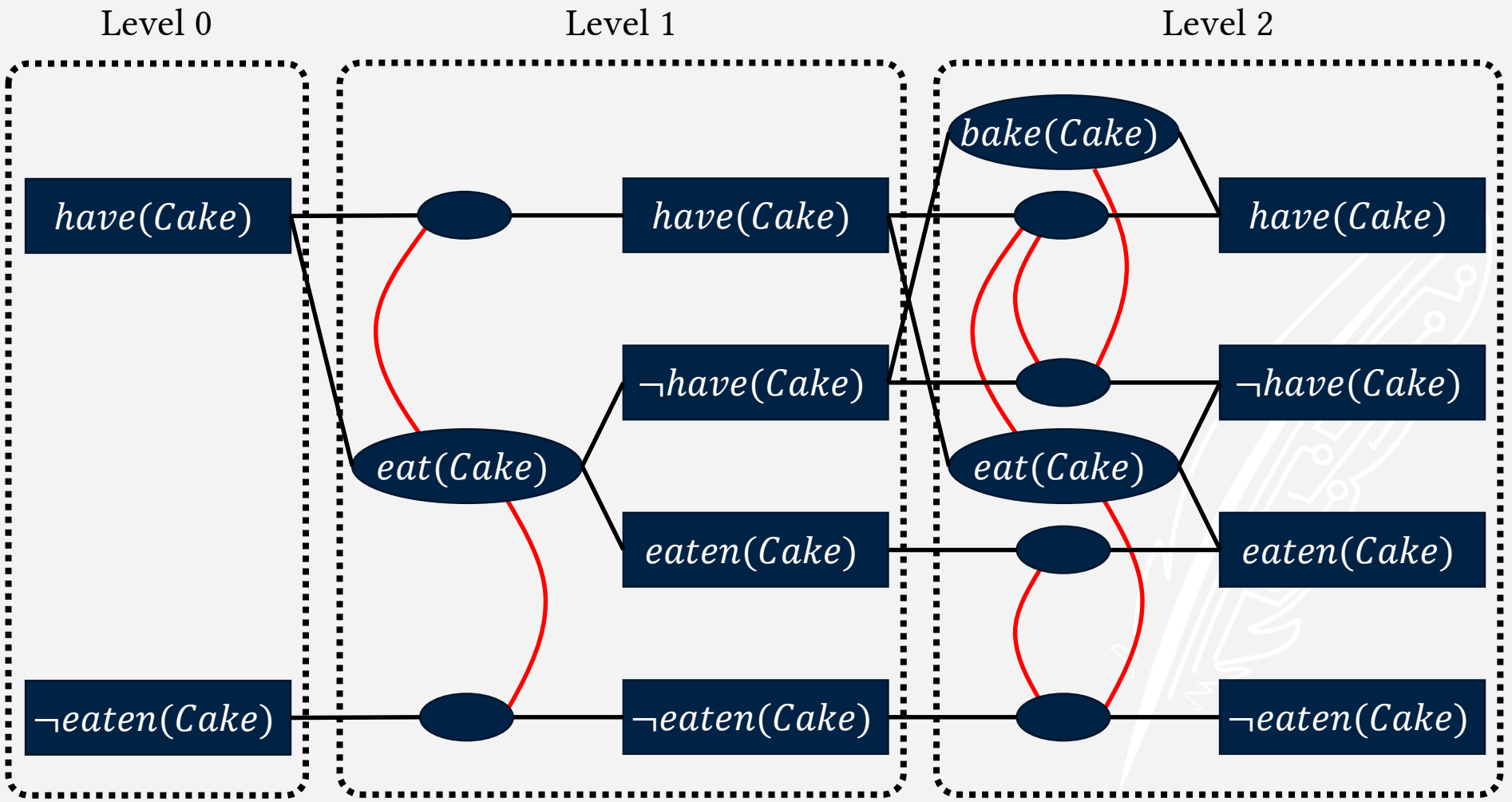
Inconsistent Effects: One action negates an effect of the other.



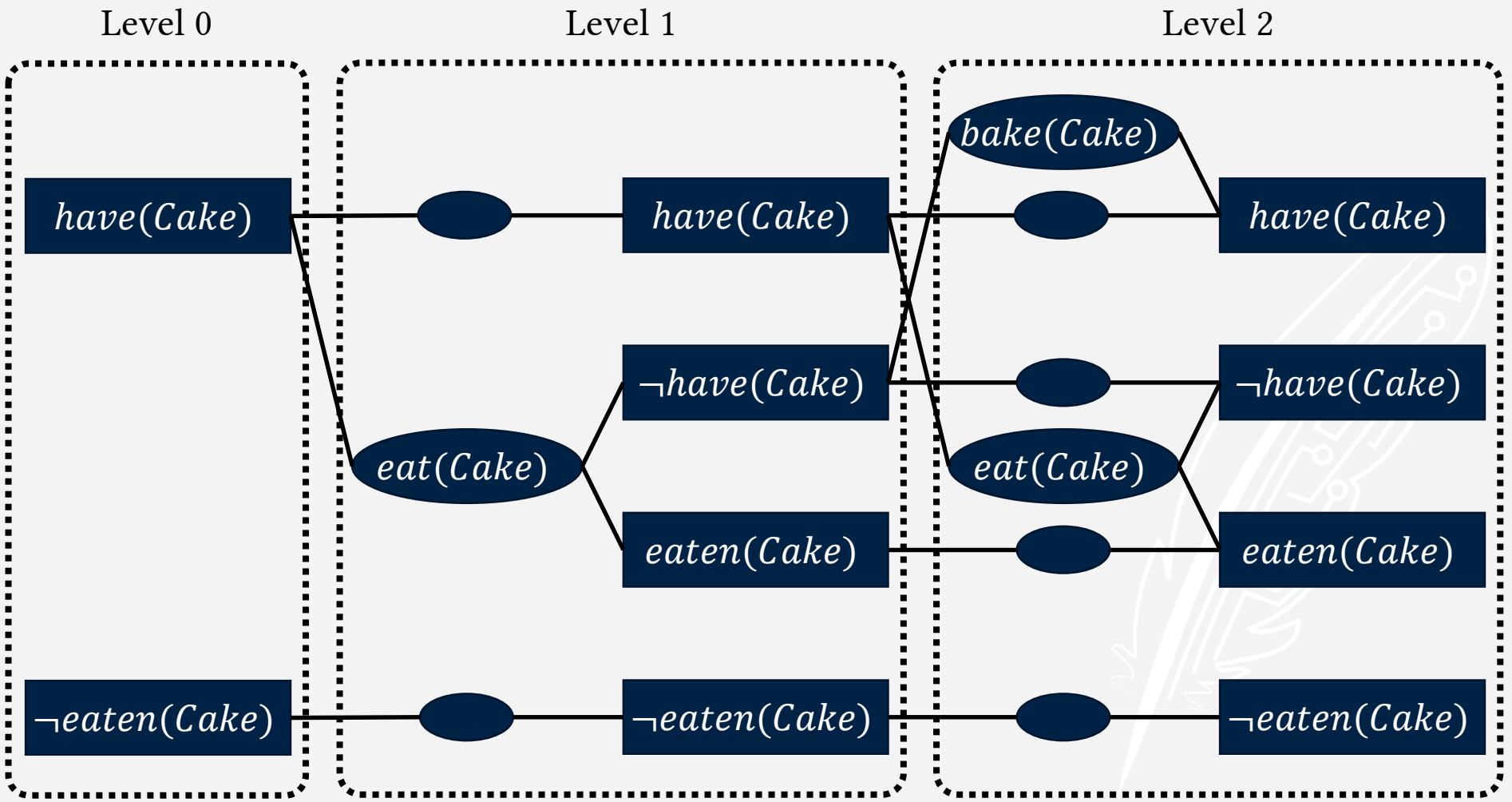
Interference: On action negates a precondition of the other.



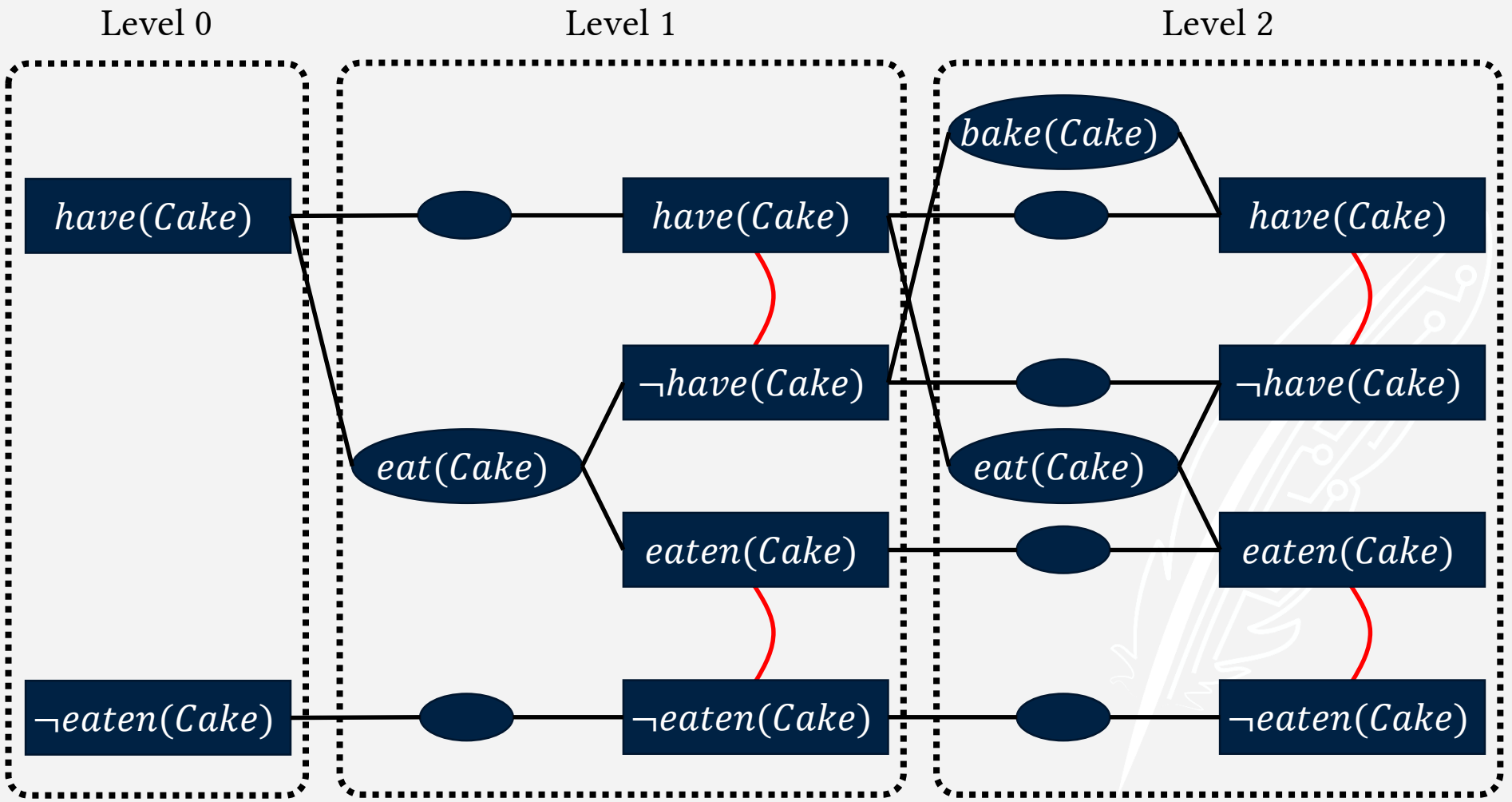
Interference: On action negates a precondition of the other.



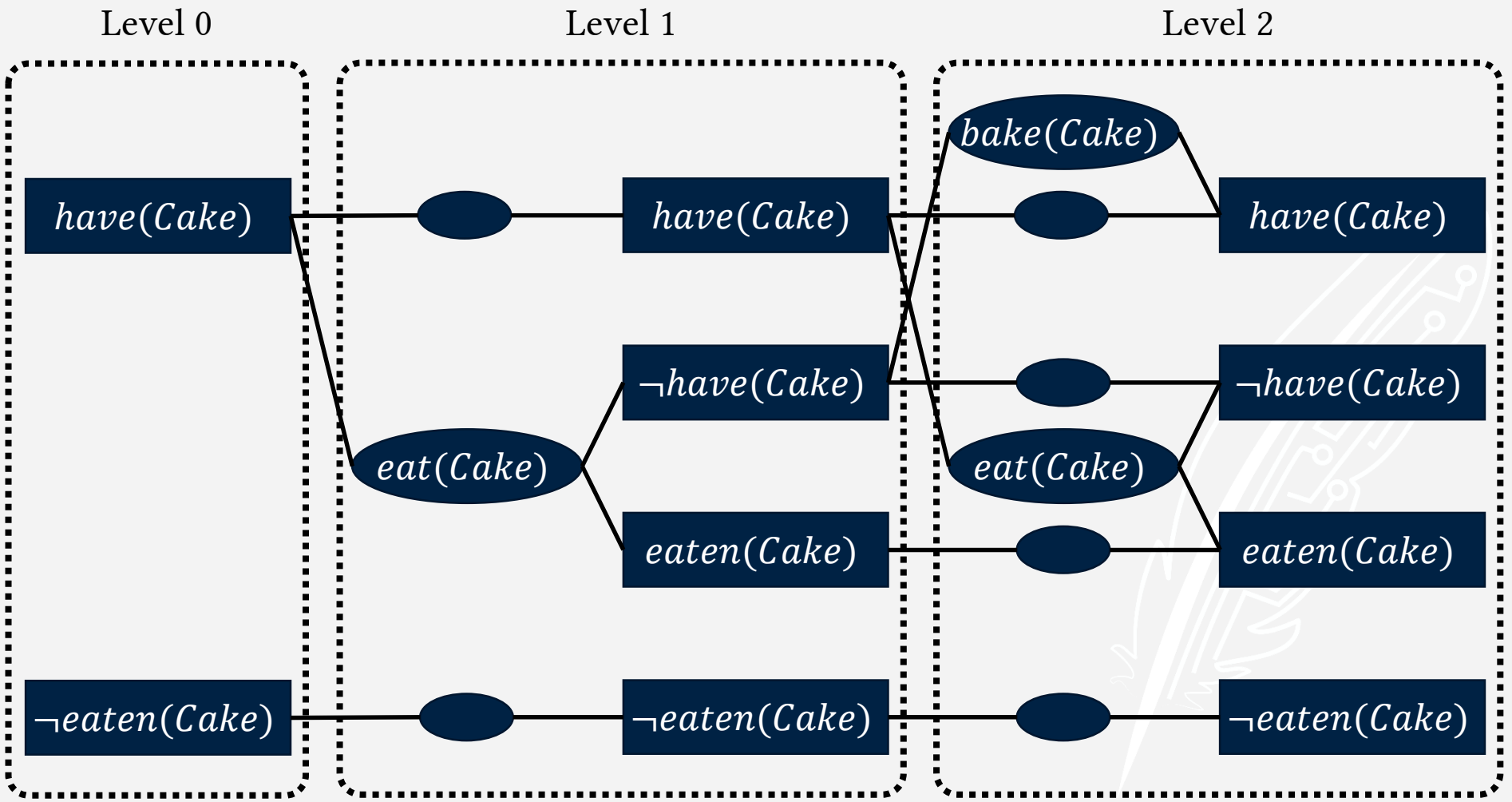
Opposites: One literal is the negation of the other.



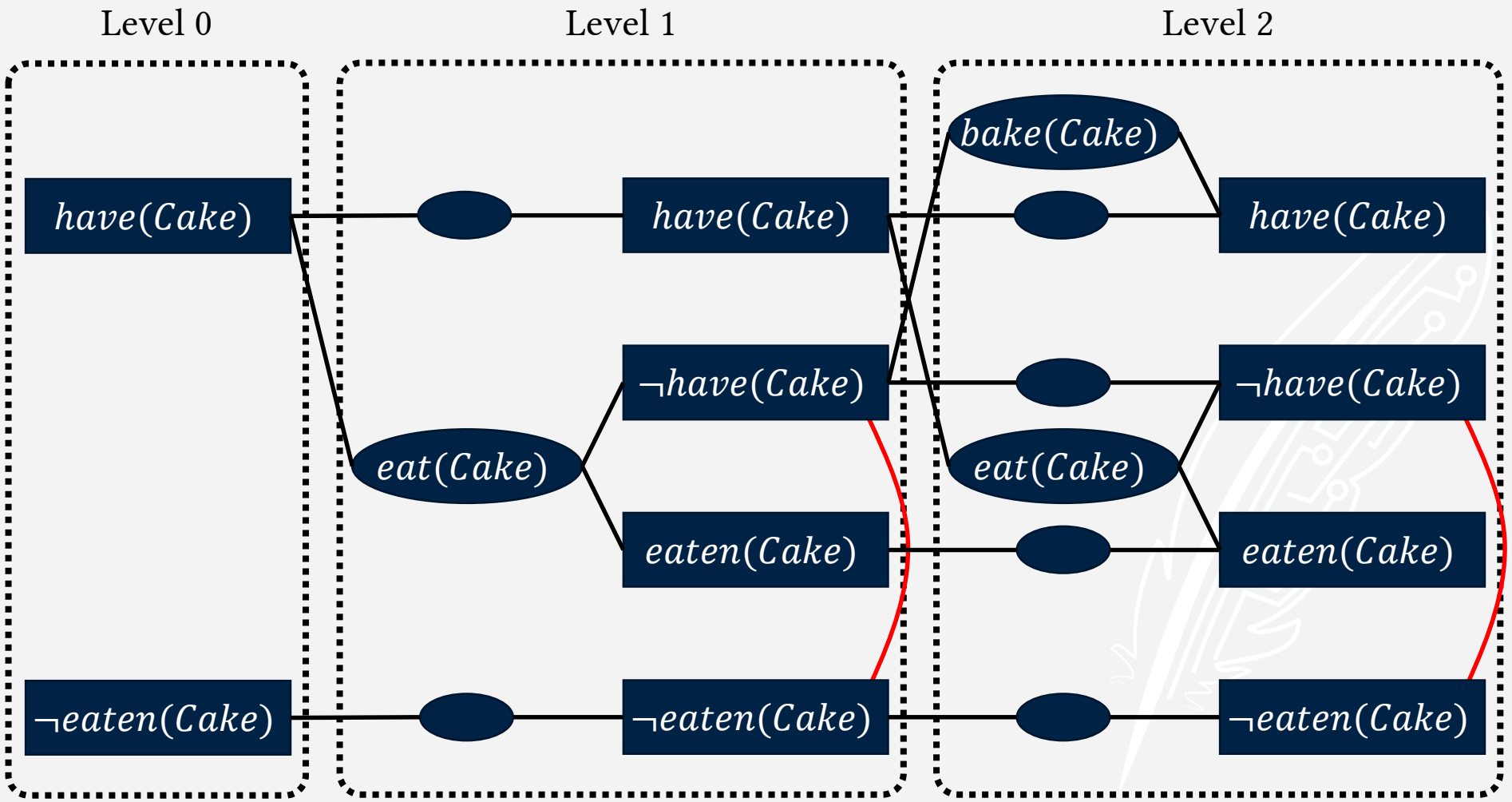
Opposites: One literal is the negation of the other.



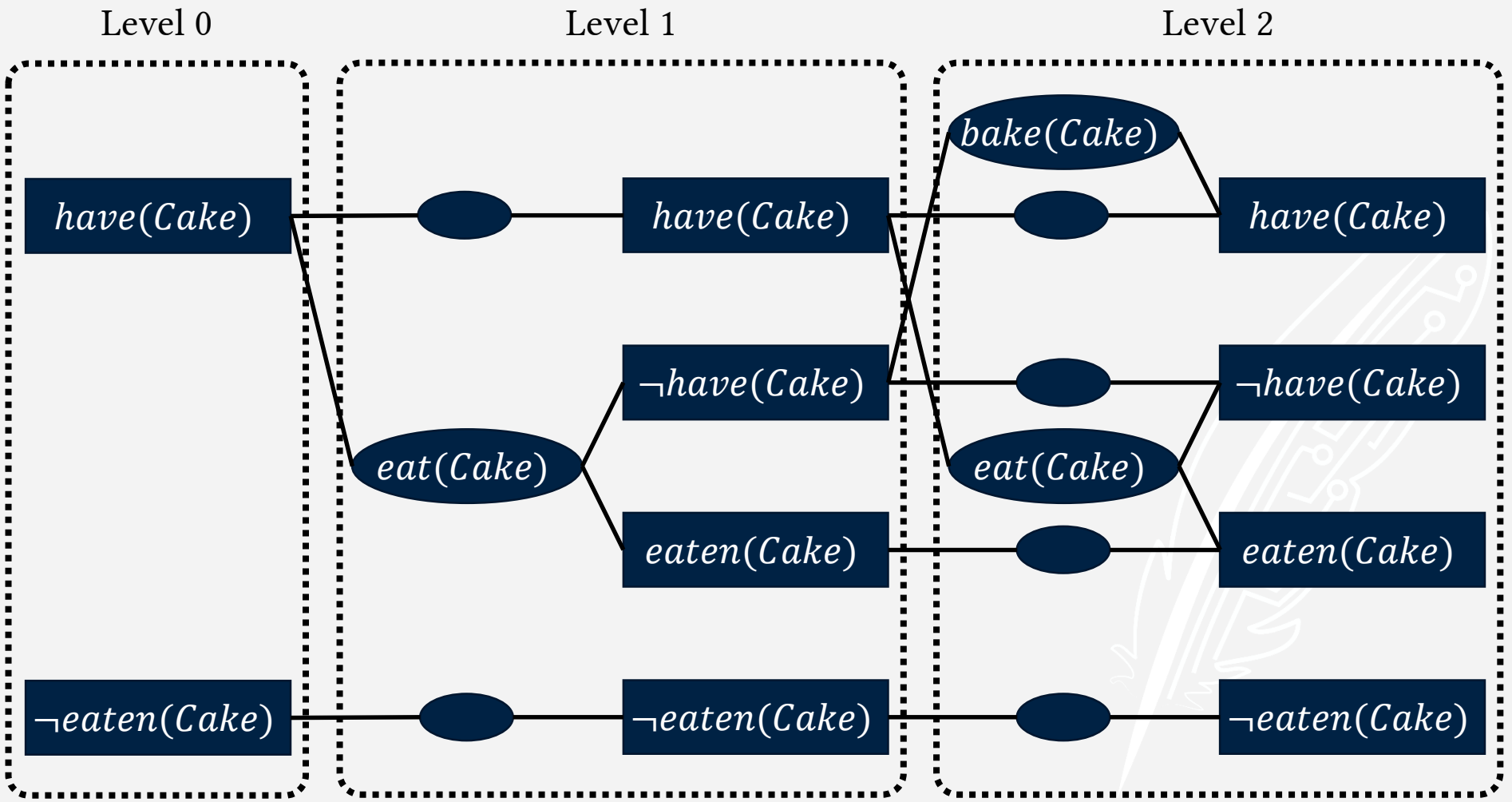
Inconsistent Support: Every pair of actions that achieves the literals is mutually exclusive.



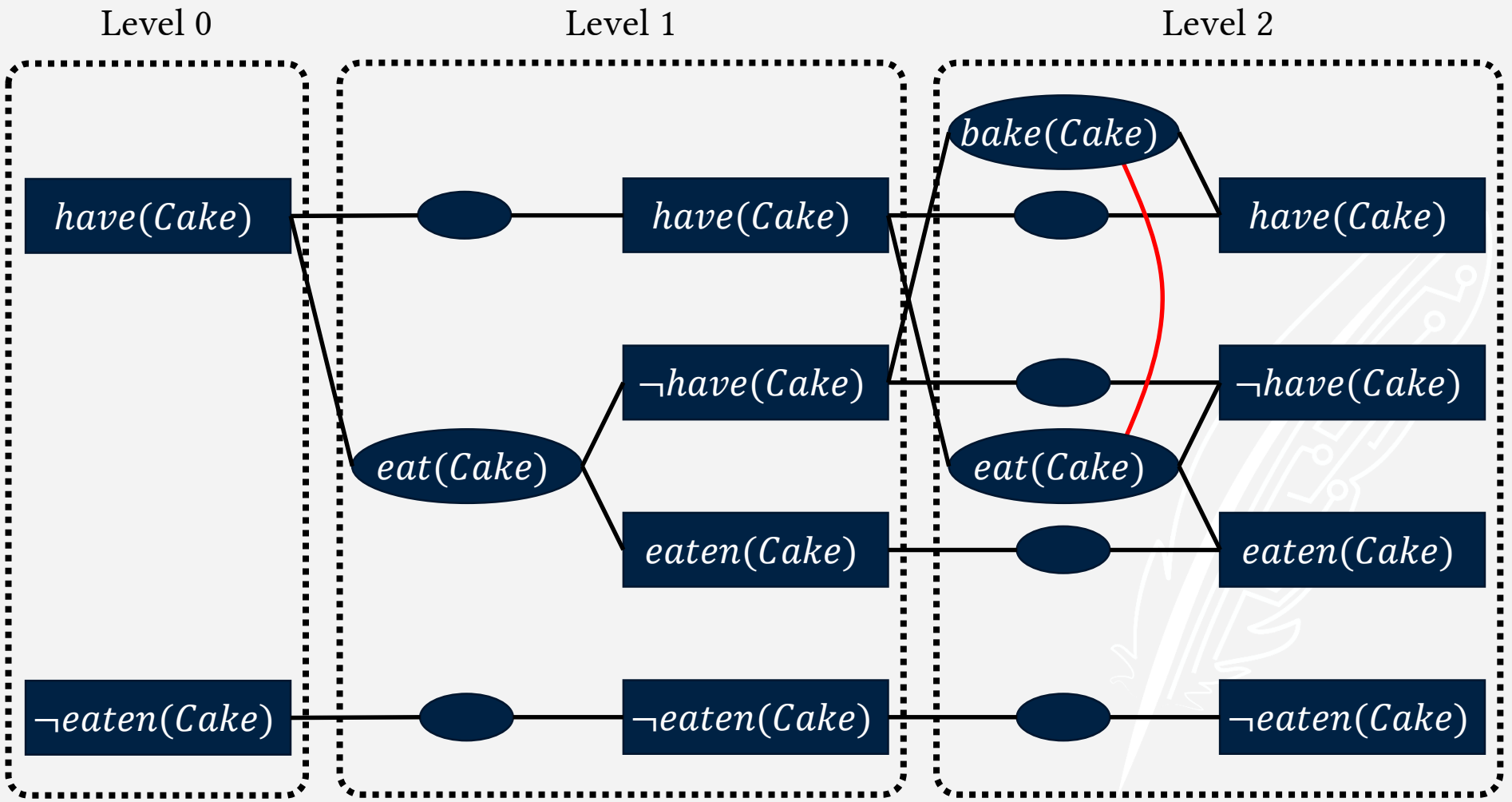
Inconsistent Support: Every pair of actions that achieves the literals is mutually exclusive.

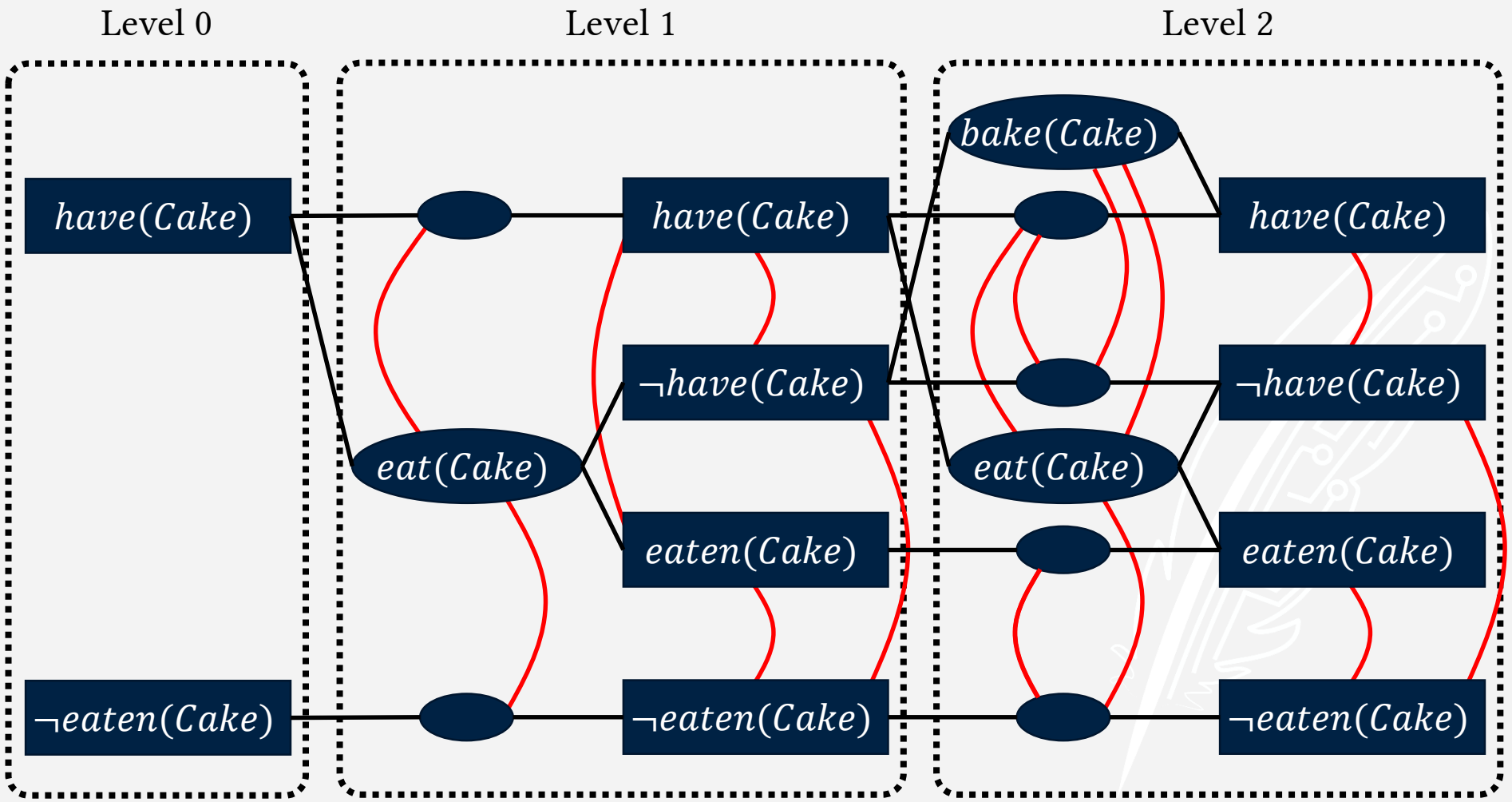


Competing Needs: Actions have mutually exclusive preconditions.



Competing Needs: Actions have mutually exclusive preconditions.




Level 0
Level 1
Level 2
have(Cake)
¬eaten(Cake)
have(Cake)
¬have(Cake)
eat(Cake)
eaten(Cake)
¬eaten(Cake)
bake(Cake)
have(Cake)
¬have(Cake)
eat(Cake)
eaten(Cake)
¬eaten(Cake)

Mutexes

Do these mutex relations cover every possible set of nodes which are mutex?

No. Consider this Block's World problem:

Goal: $on(A, B) \wedge on(B, C) \wedge on(C, A)$

The three goal literals are mutually exclusive, but no *pair* of them is mutually exclusive.

Plan Graph Algorithm

Plan graphs are not only useful for heuristics. There exists a family of algorithms that search the plan graph for a plan.

Graphplan Algorithm

Extend the plan graph until all goals are non-mutex.

Let G be the current goals, initially the problem's goals.

Let n be the current level, initially the highest level.

To satisfy the goals in G at level n:

    If n = 0, return the plan as a solution.

    Choose a set of steps S which achieve all the goals in G.

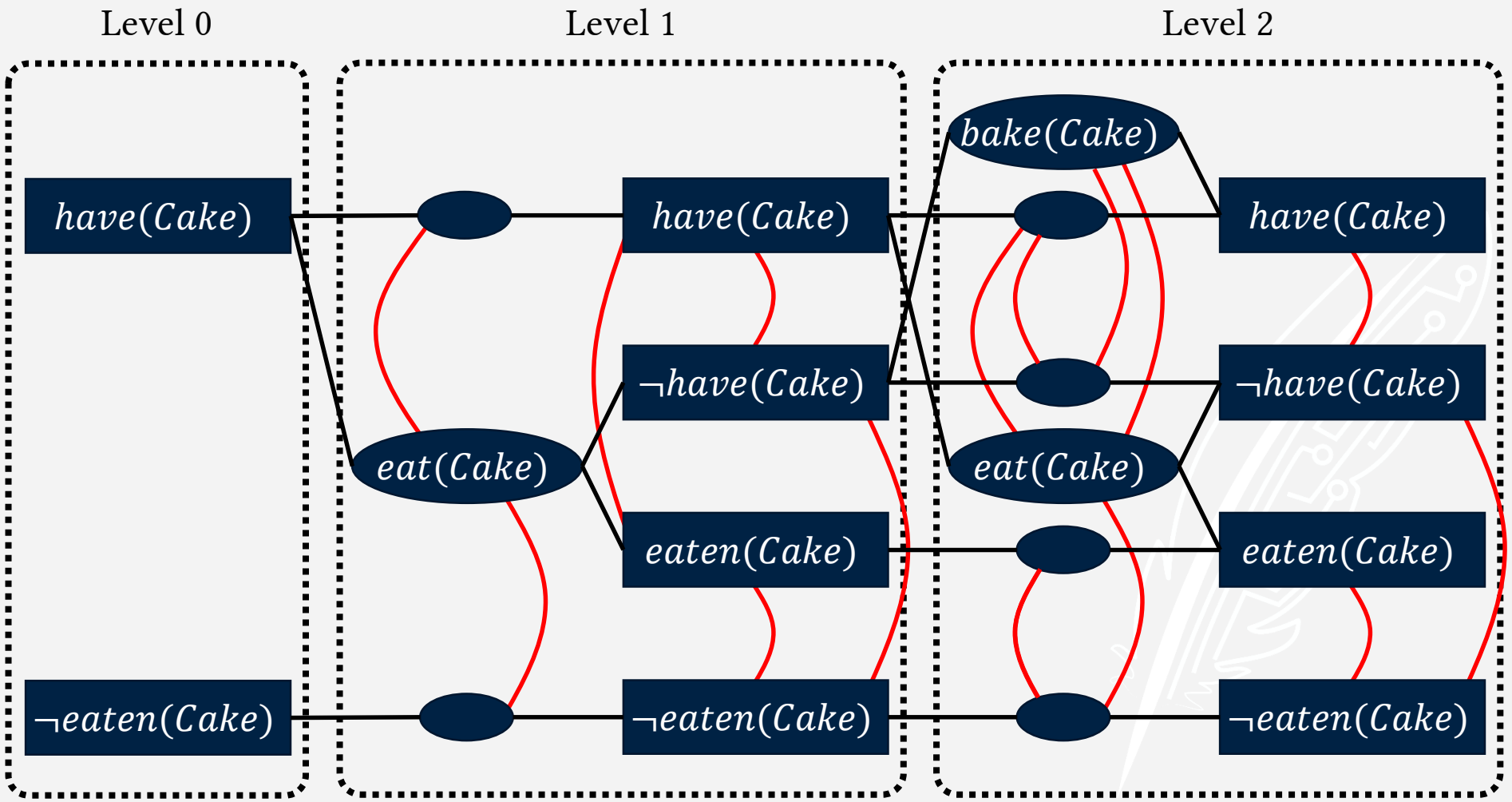
    (Every pair of steps in S must be non-mutext).

    Add all steps in S to the plan.

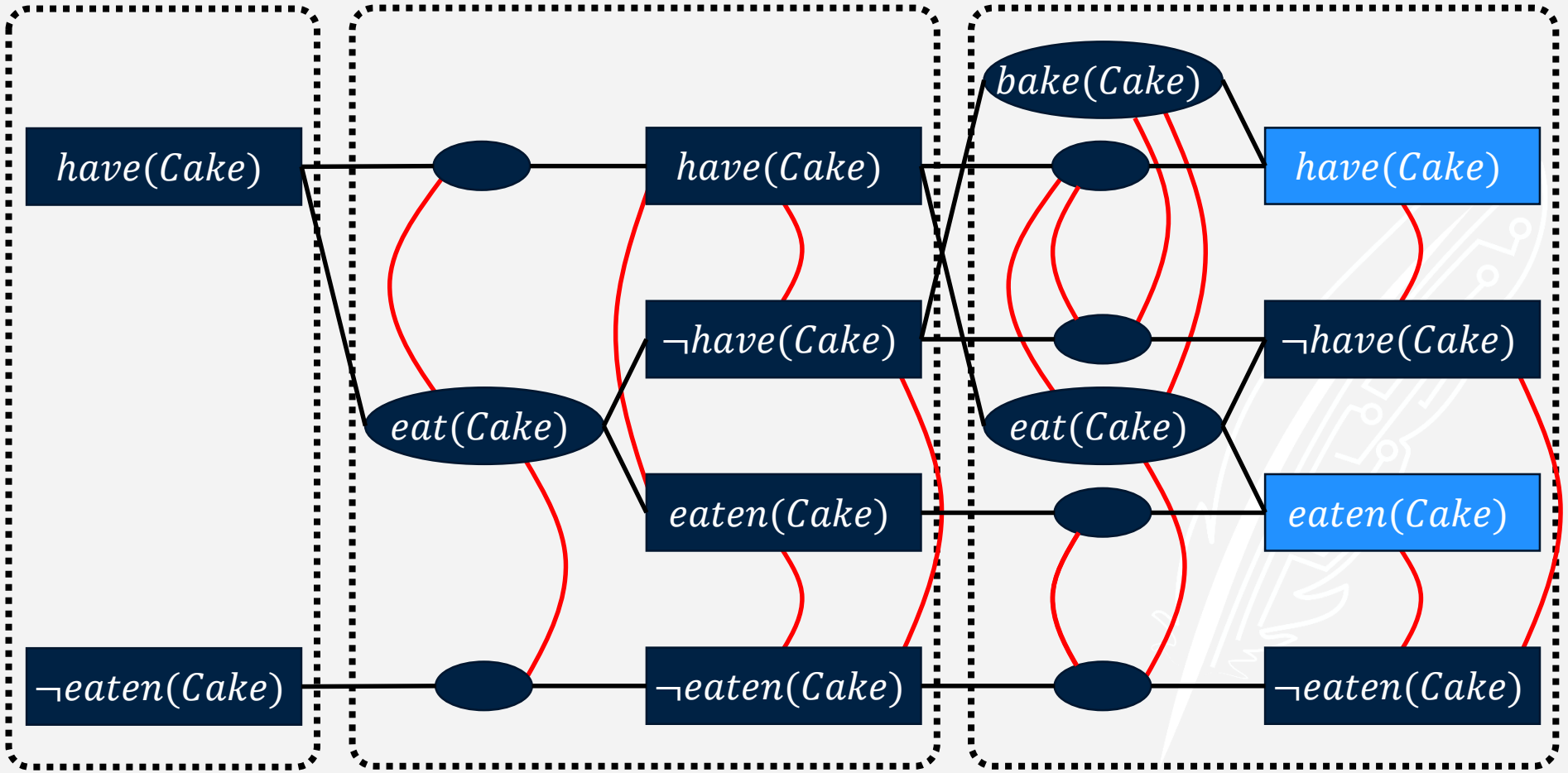
    Let G = all the preconditions of the steps in S.

    Recursively satisfy all the goals in G at level n – 1.

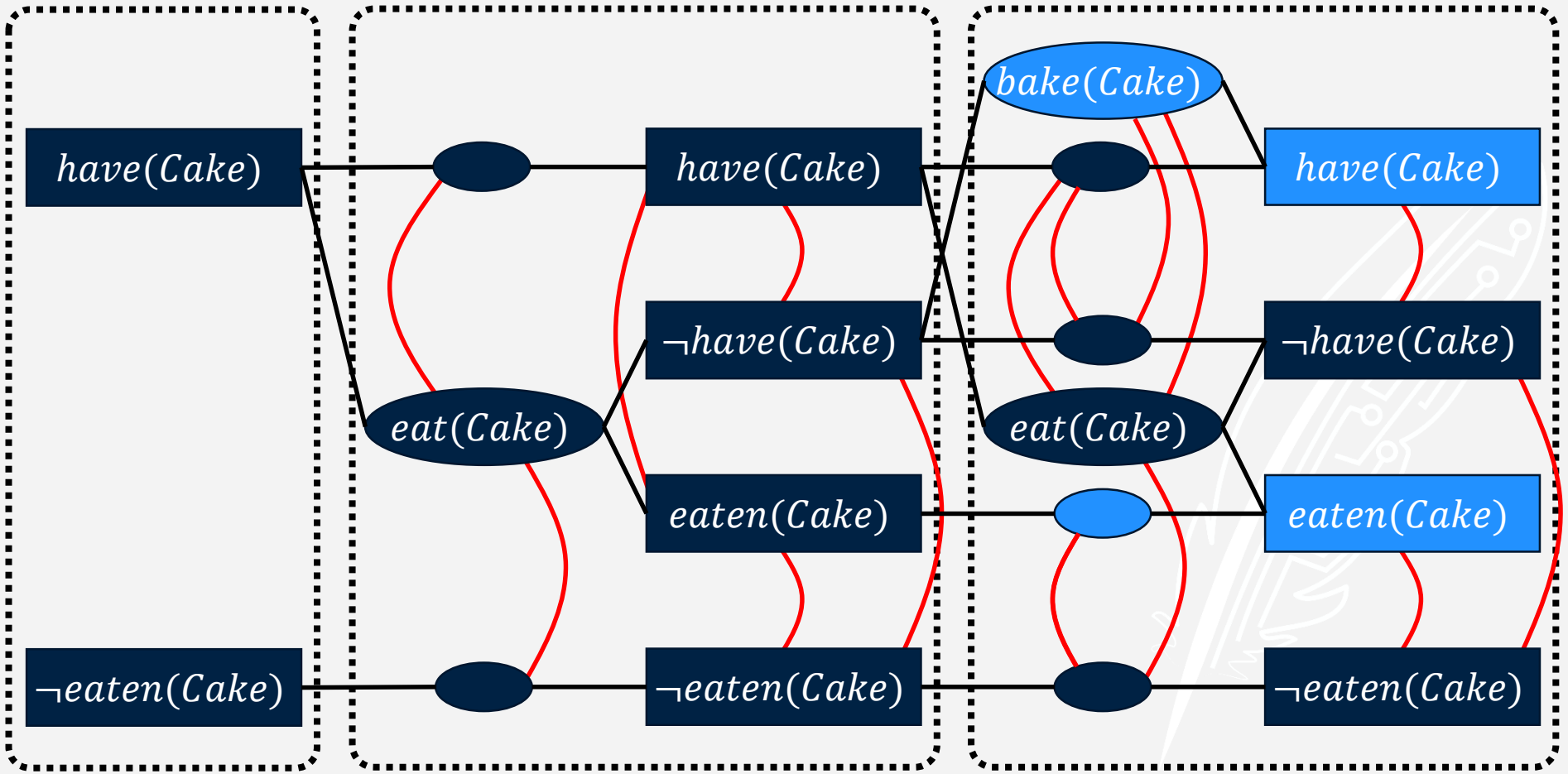
If satisfying G fails, add a level to the graph and try again.


Level 0
Level 1
Level 2
have(Cake)
¬eaten(Cake)
have(Cake)
¬have(Cake)
eat(Cake)
eaten(Cake)
¬eaten(Cake)
bake(Cake)
have(Cake)
¬have(Cake)
eat(Cake)
eaten(Cake)
¬eaten(Cake)

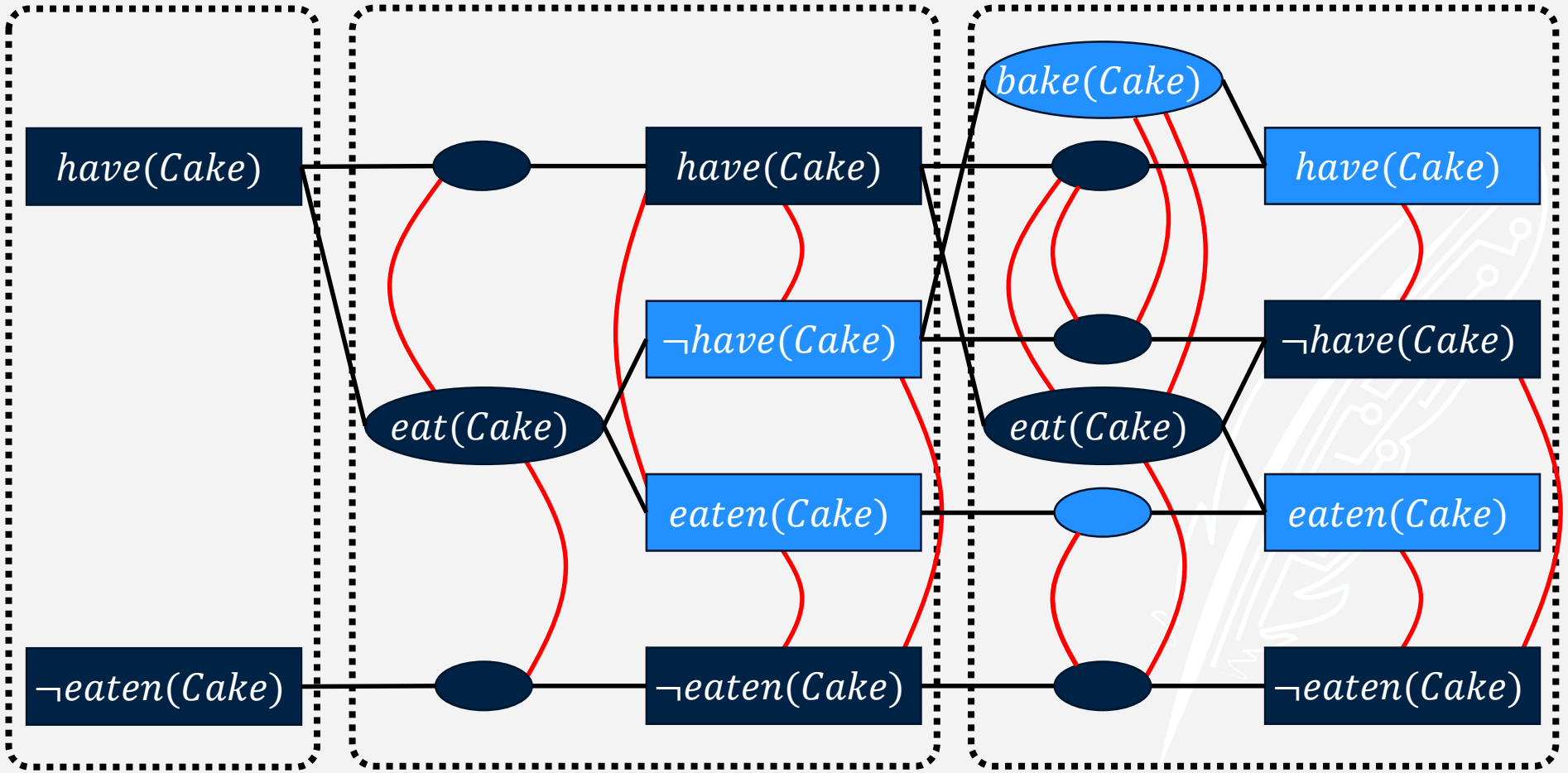
Goals: $have(Cake) \wedge eaten(Cake)$



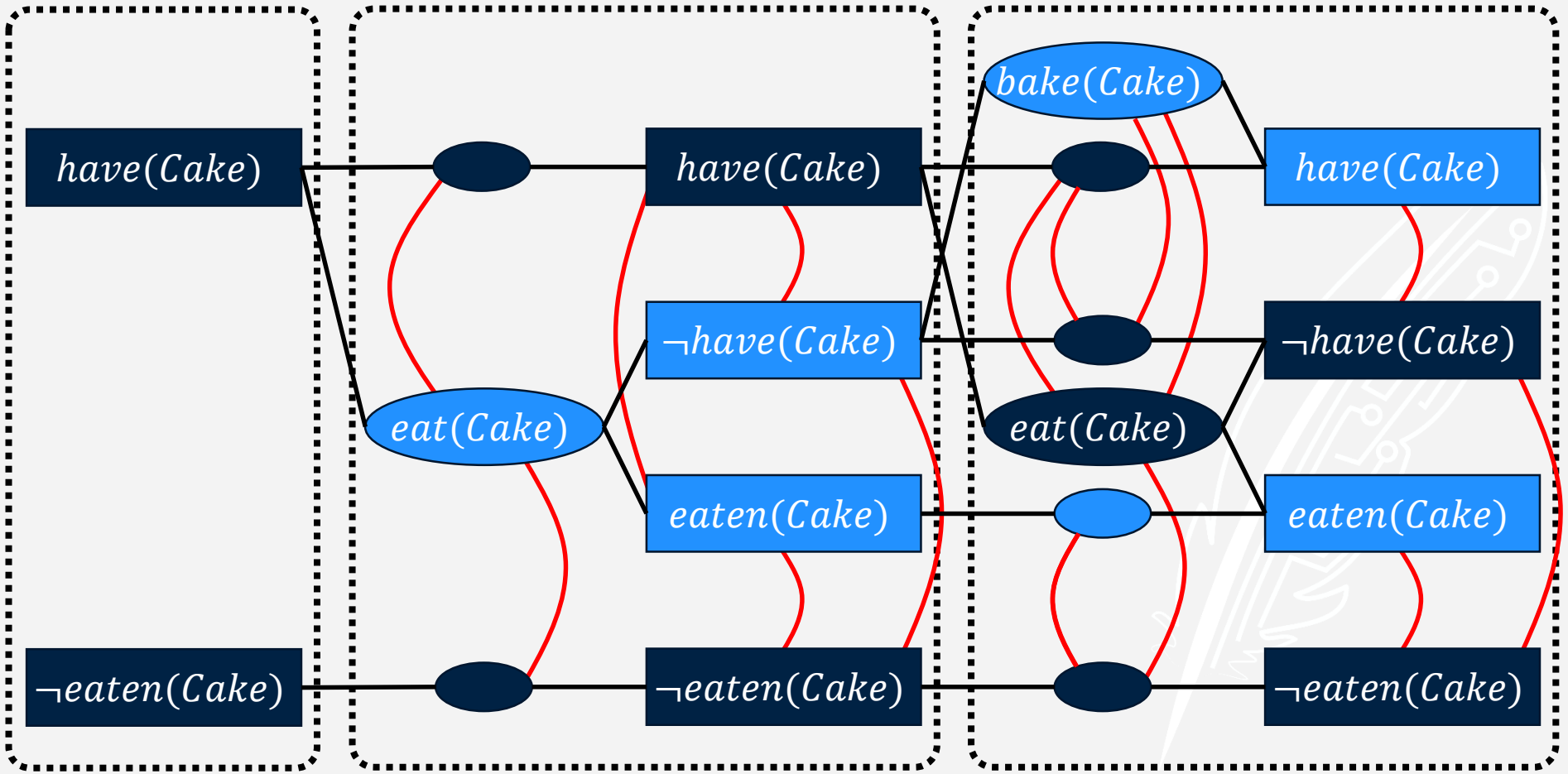
Goals: $have(Cake) \wedge eaten(Cake)$



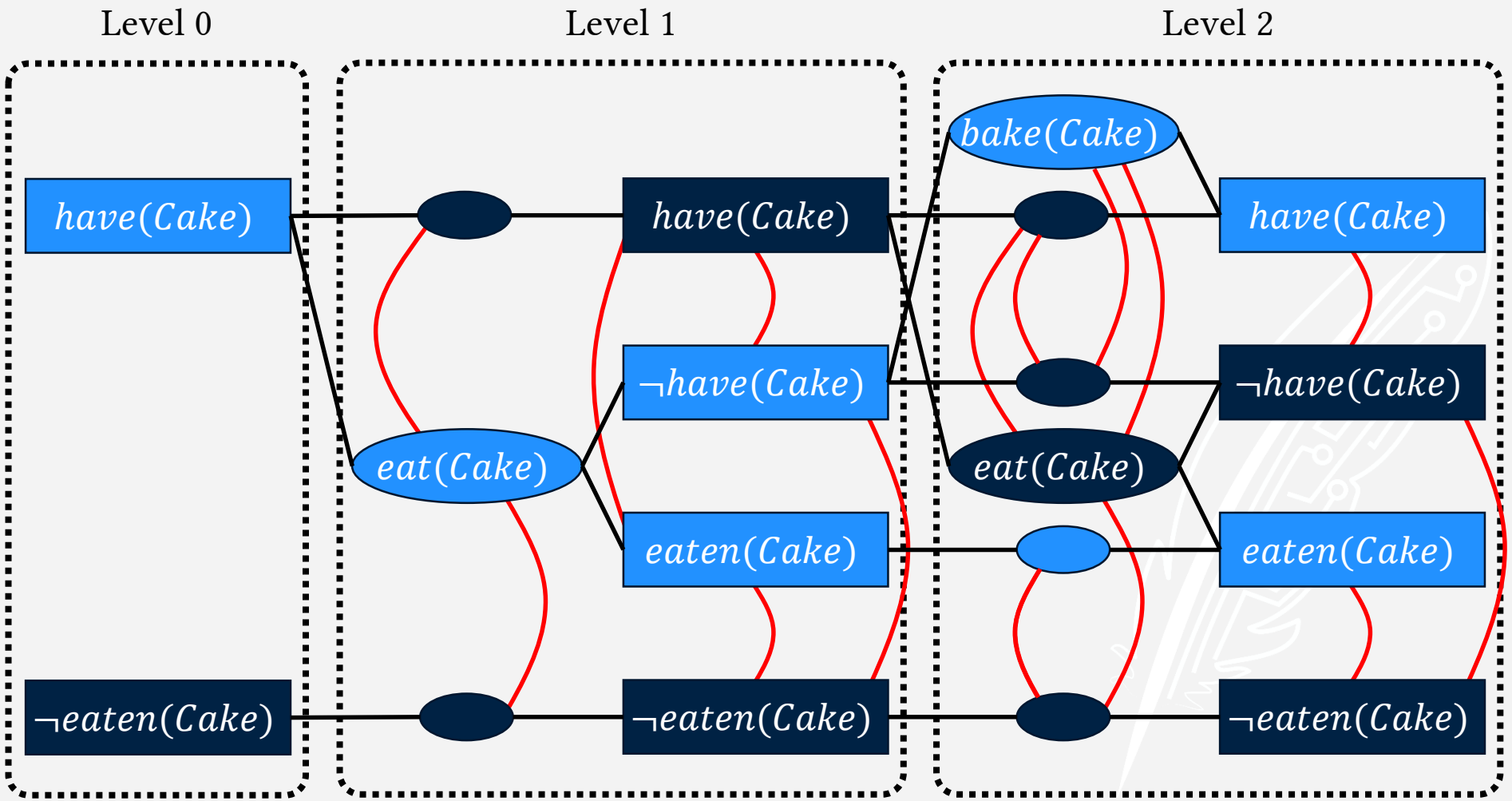
Goals: $\neg have(Cake) \wedge eaten(Cake)$



Goals: $\neg have(Cake) \wedge eaten(Cake)$



Goals: $have(Cake)$



Backtracking Search

This search procedure is a backtracking search. In other words, if we can't find any way to satisfy the goals at some low level, we have to go back to the higher levels and try to solve them differently.

As you might have guessed, since building the plan graph only P-TIME-hard, this search is P-SPACE hard.

No-Goods

When a complete backtracking search fails to satisfy a set of goals at some level, it means that set cannot be achieved at that level.

We can record this set as a **no-good**. Later, if we are asked to solve that set of goals at that level again, we can automatically fail without doing the search again.

No-goods are just many-node mutexes.

Extending the Plan Graph

Once a literal appears at some level, it will appear at all higher levels.

Once a step appears at some level, it will appear at all higher levels.

Once a mutex appears, it might disappear at higher levels. (Things which were mutex early on might become non-mutex later.)

Once a no-good appears, it might disappear at a higher level.

Leveling Off and Termination

Eventually, the literals and goals at each level will stop increasing.

Eventually, the mutexes and no-goods will stop decreasing.

When both of these conditions have occurred, the graph has **leveled off**. If search fails once the graph has leveled off, we know no solution to the problem exists.