Planner: explanation and examples

This explains the bitplanning code and setup.

You must define a **Domain**, which is a set of states and actions. This is like a "world".

There may be variables in the actions in a Domain, but to solve the problems you must create a concrete world by enumerating all the values for the variables. This creates a ConcreteDomain, and explodes out the actions and states.

In [46]:
```
from parsedomain import Domain
```

Defining a domain

A domain contains **states** and **actions**. Each state may be true or false or unknown/unspecified/uninteresting.

A domain is defined with a big string that uses a particular syntax. It's naive and line-based (though not indentation-aware), so the vertical nature of what you see is required. Keywords must be on a line of their own.

States are defined like:

```
state
    StateName(v1, v2)
where [optional]
    v1 is some_variable
    v2 is some_variable
    v1 != v2
```

That is, a line with `state` on it, the name of a state (possibly with variables), and on the next line the name of the state. Names are opaque, and besides leading and trailing whitespace they are whitespace and case sensitive. I.e., `StateName(v1, v2)` and `StateName(v1,v2)` wouldn't match.

The `where` clause is available for most commands. It introduces substitutions. There are two kinds of statements: `v1 is some_variable` which says that the string `v1` will be substituted for all values of `some_variable`. In this case it would be all combinations of `v1` and `v2` (i.e., if `some_variables` had 10 values, there would be 100 states defined). You can also restrict using `v1 != v2`.

To define an action, you use something similar:

```
action
    Move(o, p)
must
    OnTable(o)
    Clear(p)
then
    On(o, p)
    not Clear(p)
where
    o is object
    p is place
```

Note the clauses can be in any order. `must` is all the requirements for the action to be invoked, and `then` is the result of the actions. These each must be states. Note you can add `not` before any action to assert the state must or will be false.

The last kind of definition is a **constraint**. These are optional, but help fill in details later. These are the implications of a state. For instance, if an object is in one place, it can't be in other places:

```
if
    On(o, p)
then
    not On(o, p2)
    not OnTable(o)
where
    o is object
    p is place
    p2 is place
    p != p2
```

This might catch errors elsewhere, and also makes it easier to setup initial state, e.g., if you say `On(A, Location1)` then the system can infer that `not On(A, Location2)`.

```
In [47]: cargo_domain = Domain("cargo_domain", """
         # This is a domain where cargo and planes can be At(thing, airport), or a pi
         # Planes can fly to any other airport. Cargo can be loaded into a plane like
         # unloaded like Unload(cargo, plane, location).
         state
             At(p, a)
         where
             p is plane
             a is airport

         state
             At(c, a)
         where
             c is cargo
             a is airport

         state
             In(c, p)
         where
             c is cargo
             p is plane

         if
             At(p, a)
         then
             not At(p, a2)
         where
             p is plane
             a is airport
             a2 is airport
             a != a2

         if
             At(c, a)
         then
             not At(c, a2)
             not In(c, p)
         where
             c is cargo
             a is airport
             a2 is airport
             a != a2
             p is plane

         if
             In(c, p)
         then
             not At(c, a)
             not In(c, p2)
         where
             c is cargo
             a is airport
             p is plane
             p2 is plane
             p != p2
         to
             Fly(p, a1, a2)
         where
             p is plane
             a1 is airport
             a2 is airport
             a1 != a2
         must
             At(p, a1)
         then
             At(p, a2)
             not At(p, a1)

         to
             Load(c, p, a)
         where
             c is cargo
             p is plane
             a is airport
         must
             At(c, a)
             At(p, a)
         then
             In(c, p)
             not At(c, a)

         to
             Unload(c, p, a)
         where
             c is cargo
             p is plane
             a is airport
         must
             In(c, p)
             At(p, a)
         then
             At(c, a)
             not In(c, p)
         """)
```

Binding variables in a domain

In the previous examples we had unbound variables (e.g., `cargo`). To actually solve problems these must be enumerated, and many states and actions will be created.

Some problems don't have variables, but even then you must bind variables (there's examples of this later), like `a_domain.substitute({})`.

You can bind variables like:

```
simple_cargo = cargo_domain.substitute({"cargo": ["C1", "C2], ...})
```

Or you can use a single string like below:

```
In [48]: simple_cargo = cargo_domain.substitute("""
         cargo: C1 C2
         plane: P1 P2
         airport: JFK SFO
         """)
```

```
In [49]: # Note that the substituted domain has a nice representation (you could als
         simple_cargo
```

Out[49]:

cargo_domain

State name	BitMask
At(C1, JFK)	`A-----------`
At(C1, SFO)	`-B----------`
At(C2, JFK)	`--C---------`
At(C2, SFO)	`---D--------`
At(P1, JFK)	`----E-------`
At(P1, SFO)	`-----F------`
At(P2, JFK)	`------G-----`
At(P2, SFO)	`-------H----`
In(C1, P1)	`--------I---`

Fly(P1, JFK, SFO)
Must: `----Ef------` At(P1, JFK)
Then: `----eF------` At(P1, SFO); not At(P1, JFK)

Defining problems

A problem is a ConcreteDomain with an initial state (`start`) and a `goal`.

The start is defined by a string with one state on each line. You can assert that everything not specified is false using `default_false` on a line by its own, or if you define constraints it may be able to determine all the consequences of your assertions. If not everything is specified then you'll get an error.

The goal is defined similarly, though of course it doesn't have to be fully specified.

```
In [50]: simple_cargo_problem = simple_cargo.problem(
         start="""
         At(C1, SFO)
         At(C2, SFO)
         At(P1, SFO)
         At(P2, JFK)
         """,
         goal="""
         At(C2, JFK)
         At(C1, JFK)
         """)
```

Solving problems

Once you've defined a problem, solving it is as simple as `problem.solve()`. This returns the solution as an ActionSequence, or None if no solution can be found.

```
In [51]: simple_cargo_problem.solve()
```

```
Out[51]:
```

Action sequence:

1. In any order: Load(C1, P1, SFO), Load(C2, P1, SFO)
   - Must: **`aBcDeF--ijkl`**
   - Then: `abcdeF--IjKl`
2. Fly(P1, SFO, JFK)
   - Must: `abcdeF--IjKl`
   - Then: `abcdEf--IjKl`
3. In any order: Unload(C1, P1, JFK), Unload(C2, P1, JFK)
   - Must: `abcdEf--IjKl`
   - Then: `AbCdEf--ijkl`
4. Goal
   - Must: `A-C---------`
   - Then: **`AbCdEf--ijkl`**

Understanding how the solution is found

Along the way to the solution, we try lots of things. It's interesting to watch! The problem saves a `.log`, which you can print or watch in the notebook.

Note that everything shows BitMasks. These are fields of true/false/doesn't-matter. Each state is one item. It's equivalent to a binary number, but we give each digit its own letter so it's easier to follow. A string like `A-C---------` means that A and C are true, and the rest don't matter (`a` means that item is false). Each BitMask is actually two binary numbers: the true/false set, and the matters/doesn't-matter set.

```
In [52]: simple_cargo_problem.log
Out[52]:
```

Start state:
`aBcDeFGhijkl`
At(C1, SFO)
At(C2, SFO)
At(P1, SFO)
At(P2, JFK)
Goal:
`A-C---------`
At(C1, JFK)
At(C2, JFK)

Problem solution log:

Tried: **13**
Skipped: **3** (23%)
Explored: **10**
Time: **0.0069201s**

1. Attempted action of 0 alternatives: (score `(2, 2, 2, 0)`)
   A. Goal
      - Must: **`A-C---------`**
      - Then: `------------`
2. Attempted action of 7 alternatives: (score `(0, 10, 4, 2)`)
   A. In any order: Unload(C1, P2, JFK), Unload(C2, P2, JFK)
      - Must: **`abcd--GhiJkL`**
      - Then: `AbCd--Ghijkl`

   B. Goal

All about the solver

The solver works backwards from the goal.

The basic theory:

- The solution is false until it is true. Therefore the last action must be something that makes the goal true, and doesn't undo any part of the goal.
- We can think about sets of actions instead of individual actions. These sets of actions (called ActionPool) can all co-exist:
  - No action can have an effect that invalidates the prerequisite of another action in the pool
  - No action can have a prerequisite that conflicts with the prerequisite of another action in the pool
  - No action can have an effect that is in conflict with the prerequisite of another action in the pool
- When we pick a "last" action, then we have a new goal, which is the requirements of the last action, and any goal requirements that the last action didn't satisfy.
- There are many possible "last" actions, so we create a set of options (called the "frontier" in the code). We pick what we consider the "best" option, using east option's "score" (this is done in `Domain.score_accomplishment_pool()`).
- When we pick the best-looking option, we remove it from the frontier, we test whether it is a solution, and if not then we consider actions that could happen before it which could make it part of a full solution. All these options are scored and added to the frontier, and may be picked later.
- If some sequence of actions has a prerequisite (`must`) that is identical to something else we've seen, then we throw it away. This avoids loops.

A bunch more examples

Below are a bunch of examples. `fixit` is the one we can't solve, *sadface*

In [53]:
```
air_cargo_p1 = cargo_domain.substitute("""
cargo: C1 C2
plane: P1 P2
airport: JFK SFO
""")
air_cargo_p1
```

Out[53]:

cargo_domain

State name	BitMask
At(C1, JFK)	`A-----------`
At(C1, SFO)	`-B----------`
At(C2, JFK)	`--C---------`
At(C2, SFO)	`---D--------`
At(P1, JFK)	`----E-------`
At(P1, SFO)	`-----F------`
At(P2, JFK)	`------G-----`
At(P2, SFO)	`-------H----`
In(C1, P1)	`--------I---`

Fly(P1, JFK, SFO)
Must: `----Ef------` At(P1, JFK)
Then: `----eF------` At(P1, SFO); not At(P1, JFK)

In [54]:
```
air_cargo_p1_problem = air_cargo_p1.problem(
start="""
At(C1, SFO)
At(C2, JFK)
At(P1, SFO)
At(P2, JFK)
""",
goal="""
At(C1, JFK)
At(C2, SFO)
""")
```

```
In [55]: air_cargo_p1_problem.solve()
```

```
Out[55]:
```

Action sequence:

1. Load(C1, P1, SFO)
   - Must: **`aBCdeFGhijkl`**
   - Then: `ab--eF--Ij--`
2. Fly(P1, SFO, JFK)
   - Must: `abCdeFGhIjkl`
   - Then: `ab--Ef--Ij--`
3. In any order: Load(C2, P1, JFK), Unload(C1, P1, JFK)
   - Must: `abCdEfGhIjkl`
   - Then: `AbcdEf--ijKl`
4. Load(C1, P2, JFK)
   - Must: `AbcdEfGhijKl`
   - Then: `abcdEfGhiJKl`
5. In any order: Fly(P1, JFK, SFO), Unload(C1, P2, JFK)
   - Must: `abcdEfGhiJKl`

```
In [56]: air_cargo_p1_problem.log
```

```
Out[56]:
```

Problem solution log:

Tried: **22**
Skipped: **6** (27%)
Explored: **16**
Time: **0.015646s**

Start state:
`aBCdeFGhijkl`
At(C1, SFO)
At(C2, JFK)
At(P1, SFO)
At(P2, JFK)
Goal:
`A--D--------`
At(C1, JFK)
At(C2, SFO)

1. Attempted action of 0 alternatives: (score `(2, 2, 2, 0)`)
   A. Goal
      - Must: **`A--D--------`**
      - Then: **`------------`**
2. Attempted action of 5 alternatives: (score `(0, 12, 4, 2)`)
   A. In any order: Unload(C1, P2, JFK), Unload(C2, P1, SFO)
      - Must: **`abcdeFGhiJKl`**
      - Then: `AbcDeFGhijkl`
   B. Goal
      - Must: `A--D--------`
      - Then: **`AbcDeFGhijkl`**
3. Attempted action of 12 alternatives: (score `(0, 12, 4, 3)`)
   A. Load(C1, P2, JFK)
      - Must: **`AbcdeFGhijKl`**
      - Then: `ab----GhiJ--`
   B. In any order: Unload(C1, P2, JFK), Unload(C2, P1, SFO)
      - Must: `abcdeFGhiJKl`
      - Then: `AbcDeFGhijkl`
   C. Goal
      - Must: `A--D--------`
      - Then: **`AbcDeFGhijkl`**
4. Attempted action of 18 alternatives: (score `(0, 12, 4, 3)`)
   A. Load(C2, P1, SFO)
      - Must: **`abcDeFGhiJKl`**
      - Then: `--cdeF----Kl`
   B. In any order: Unload(C1, P2, JFK), Unload(C2, P1, SFO)
      - Must: `abcdeFGhiJKl`
      - Then: `AbcDeFGhijkl`
   C. Goal
      - Must: `A--D--------`
      - Then: **`AbcDeFGhijkl`**
5. Attempted action of 23 alternatives: (score `(0, 12, 6, 5)`)
   A. In any order: Fly(P1, JFK, SFO), Unload(C1, P2, JFK)
      - Must: **`abcdEfGhiJKl`**
      - Then: `Ab--eFGhij--`
   B. Load(C1, P2, JFK)
      - Must: `AbcdeFGhijKl`
      - Then: `ab--eFGhiJ--`
   C. In any order: Unload(C1, P2, JFK), Unload(C2, P1, SFO)

        - Must: `abcdeFGhiJKl`
        - Then: `AbcDeFGhijkl`
    D. Goal
        - Must: `A--D--------`
        - Then: **`AbcDeFGhijkl`**
6. Attempted action of 28 alternatives: (score `(0, 12, 4, 4)`)
    A. In any order: Load(C1, P2, JFK), Load(C2, P1, SFO)
        - Must: **`AbcDeFGhijkl`**
        - Then: `abcdeFGhiJKl`
    B. In any order: Unload(C1, P2, JFK), Unload(C2, P1, SFO)
        - Must: `abcdeFGhiJKl`
        - Then: `AbcDeFGhijkl`
    C. Goal
        - Must: `A--D--------`
        - Then: **`AbcDeFGhijkl`**
7. Skipped adding action Load(C2, P1, SFO) because must=`AbcDeFGhijkl` has been seen

    show action
8. Skipped adding action Load(C1, P2, JFK) because must=`AbcDeFGhijkl` has been seen

    show action
9. Skipped adding action Load(C2, P1, SFO) because must=`abcDeFGhiJkl` has been seen

    show action
10. Attempted action of 29 alternatives: (score `(0, 12, 6, 6)`)
    A. Load(C1, P2, JFK)
        - Must: **`AbcdEfGhijKl`**
        - Then: `ab----GhiJ--`
    B. In any order: Fly(P1, JFK, SFO), Unload(C1, P2, JFK)
        - Must: `abcdEfGhiJKl`
        - Then: `Ab--eFGhij--`
    C. Load(C1, P2, JFK)
        - Must: `AbcdeFGhijKl`
        - Then: `ab--eFGhiJ--`
    D. In any order: Unload(C1, P2, JFK), Unload(C2, P1, SFO)
        - Must: `abcdeFGhiJKl`
        - Then: `AbcDeFGhijkl`
    E. Goal
        - Must: `A--D--------`
        - Then: **`AbcDeFGhijkl`**
11. Attempted action of 37 alternatives: (score `(0, 12, 4, 6)`)
    A. Load(C2, P1, JFK)
        - Must: **`abCdEfGhiJkl`**
        - Then: `--cdEf----Kl`
    B. In any order: Fly(P1, JFK, SFO), Unload(C1, P2, JFK)
        - Must: `abcdEfGhiJKl`
        - Then: `AbcdeFGhijKl`
    C. Load(C1, P2, JFK)
        - Must: `AbcdeFGhijKl`
        - Then: `abcdeFGhiJKl`
    D. In any order: Unload(C1, P2, JFK), Unload(C2, P1, SFO)

        - Must: `abcdeFGhiJKl`
        - Then: `AbcDeFGhijkl`
    E. Goal
        - Must: `A--D--------`
        - Then: **`AbcDeFGhijkl`**
12. Attempted action of 46 alternatives: (score `(0, 12, 2, 7)`)
    A. Fly(P1, SFO, JFK)
        - Must: **`abCdeFGhiJkl`**
        - Then: `----Ef------`
    B. Load(C2, P1, JFK)
        - Must: `abCdEfGhiJkl`
        - Then: `--cdEf----Kl`
    C. In any order: Fly(P1, JFK, SFO), Unload(C1, P2, JFK)
        - Must: `abcdEfGhiJKl`
        - Then: `AbcdeFGhijKl`
    D. Load(C1, P2, JFK)
        - Must: `AbcdeFGhijKl`
        - Then: `abcdeFGhiJKl`
    E. In any order: Unload(C1, P2, JFK), Unload(C2, P1, SFO)
        - Must: `abcdeFGhiJKl`
        - Then: `AbcDeFGhijkl`
    F. Goal
        - Must: `A--D--------`
        - Then: **`AbcDeFGhijkl`**
13. Attempted action of 53 alternatives: (score `(0, 12, 2, 8)`)
    A. In any order: Fly(P1, SFO, JFK), Load(C1, P2, JFK)
        - Must: **`AbCdeFGhijkl`**
        - Then: `ab--EfGhiJ--`
    B. Load(C2, P1, JFK)
        - Must: `abCdEfGhiJkl`
        - Then: `abcdEfGhiJKl`
    C. In any order: Fly(P1, JFK, SFO), Unload(C1, P2, JFK)
        - Must: `abcdEfGhiJKl`
        - Then: `AbcdeFGhijKl`
    D. Load(C1, P2, JFK)
        - Must: `AbcdeFGhijKl`
        - Then: `abcdeFGhiJKl`
    E. In any order: Unload(C1, P2, JFK), Unload(C2, P1, SFO)
        - Must: `abcdeFGhiJKl`
        - Then: `AbcDeFGhijkl`
    F. Goal
        - Must: `A--D--------`
        - Then: **`AbcDeFGhijkl`**
14. Skipped adding action Load(C1, P2, JFK) because must=`AbCdeFGhijkl` has been seen

    show action
15. Attempted action of 58 alternatives: (score `(0, 12, 6, 6)`)
    A. In any order: Fly(P1, JFK, SFO), Unload(C1, P2, JFK)
        - Must: **`abcDEfGhiJkl`**

        - Then: `Ab--eFGhij--`
    B. In any order: Load(C1, P2, JFK), Load(C2, P1, SFO)
        - Must: `AbcDeFGhijkl`
        - Then: `abcdeFGhiJKl`
    C. In any order: Unload(C1, P2, JFK), Unload(C2, P1, SFO)
        - Must: `abcdeFGhiJKl`
        - Then: `AbcDeFGhijkl`
    D. Goal
        - Must: `A--D--------`
        - Then: **`AbcDeFGhijkl`**
16. Attempted action of 60 alternatives: (score `(0, 12, 4, 7)`)
    A. In any order: Load(C1, P2, JFK), Load(C2, P1, JFK)
        - Must: **`AbCdEfGhijkl`**
        - Then: `abcdEfGhiJKl`
    B. In any order: Fly(P1, JFK, SFO), Unload(C1, P2, JFK)
        - Must: `abcdEfGhiJKl`
        - Then: `AbcdeFGhijKl`
    C. Load(C1, P2, JFK)
        - Must: `AbcdeFGhijKl`
        - Then: `abcdeFGhiJKl`
    D. In any order: Unload(C1, P2, JFK), Unload(C2, P1, SFO)
        - Must: `abcdeFGhiJKl`
        - Then: `AbcDeFGhijkl`
    E. Goal
        - Must: `A--D--------`
        - Then: **`AbcDeFGhijkl`**
17. Skipped adding action Load(C2, P1, JFK) because must=`AbCdEfGhijkl` has been seen
    show action
18. Skipped adding action Load(C1, P2, JFK) because must=`AbCdEfGhijkl` has been seen
    show action
19. Attempted action of 69 alternatives: (score `(0, 12, 4, 8)`)
    A. In any order: Load(C2, P1, JFK), Unload(C1, P1, JFK)
        - Must: **`abCdEfGhIjkl`**
        - Then: `AbcdEf--ijKl`
    B. Load(C1, P2, JFK)
        - Must: `AbcdEfGhijKl`
        - Then: `abcdEfGhiJKl`
    C. In any order: Fly(P1, JFK, SFO), Unload(C1, P2, JFK)
        - Must: `abcdEfGhiJKl`
        - Then: `AbcdeFGhijKl`
    D. Load(C1, P2, JFK)
        - Must: `AbcdeFGhijKl`
        - Then: `abcdeFGhiJKl`
    E. In any order: Unload(C1, P2, JFK), Unload(C2, P1, SFO)
        - Must: `abcdeFGhiJKl`
        - Then: `AbcDeFGhijkl`
    F. Goal
        - Must: `A--D--------`

    - Then: **`AbcDeFGhijkl`**

20. Attempted action of 78 alternatives: (score `(0, 12, 6, 8)`)
    A. In any order: Fly(P1, SFO, JFK), Fly(P2, SFO, JFK)
    - Must: **`AbcdeFgHijKl`**
    - Then: `----EfGh----`

    B. Load(C1, P2, JFK)
    - Must: `AbcdEfGhijKl`
    - Then: `ab--EfGhiJ--`

    C. In any order: Fly(P1, JFK, SFO), Unload(C1, P2, JFK)
    - Must: `abcdEfGhiJKl`
    - Then: `Ab--eFGhij--`

    D. Load(C1, P2, JFK)
    - Must: `AbcdeFGhijKl`
    - Then: `ab--eFGhiJ--`

    E. In any order: Unload(C1, P2, JFK), Unload(C2, P1, SFO)
    - Must: `abcdeFGhiJKl`
    - Then: `AbcDeFGhijkl`

    F. Goal
    - Must: `A--D--------`
    - Then: **`AbcDeFGhijkl`**

21. Attempted action of 79 alternatives: (score `(0, 12, 2, 9)`)
    A. Fly(P1, SFO, JFK)
    - Must: **`abCdeFGhIjkl`**
    - Then: `----Ef------`

    B. In any order: Load(C2, P1, JFK), Unload(C1, P1, JFK)
    - Must: `abCdEfGhIjkl`
    - Then: `AbcdEf--ijKl`

    C. Load(C1, P2, JFK)
    - Must: `AbcdEfGhijKl`
    - Then: `abcdEfGhiJKl`

    D. In any order: Fly(P1, JFK, SFO), Unload(C1, P2, JFK)
    - Must: `abcdEfGhiJKl`
    - Then: `AbcdeFGhijKl`

    E. Load(C1, P2, JFK)
    - Must: `AbcdeFGhijKl`
    - Then: `abcdeFGhiJKl`

    F. In any order: Unload(C1, P2, JFK), Unload(C2, P1, SFO)
    - Must: `abcdeFGhiJKl`
    - Then: `AbcDeFGhijkl`

    G. Goal
    - Must: `A--D--------`
    - Then: **`AbcDeFGhijkl`**

22. Attempted action of 85 alternatives: (score `(0, 12, 0, 10)`)
    A. Load(C1, P1, SFO)
    - Must: **`aBCdeFGhijkl`**
    - Then: `ab--eF--Ij--`

    B. Fly(P1, SFO, JFK)
    - Must: `abCdeFGhIjkl`
    - Then: `ab--Ef--Ij--`

    C. In any order: Load(C2, P1, JFK), Unload(C1, P1, JFK)
        - Must: `abCdEfGhIjkl`
        - Then: `AbcdEf--ijKl`
    D. Load(C1, P2, JFK)
        - Must: `AbcdEfGhijKl`
        - Then: `abcdEfGhiJKl`
    E. In any order: Fly(P1, JFK, SFO), Unload(C1, P2, JFK)
        - Must: `abcdEfGhiJKl`
        - Then: `AbcdeFGhijKl`
    F. Load(C1, P2, JFK)
        - Must: `AbcdeFGhijKl`
        - Then: `abcdeFGhiJKl`
    G. In any order: Unload(C1, P2, JFK), Unload(C2, P1, SFO)
        - Must: `abcdeFGhiJKl`
        - Then: `AbcDeFGhijkl`
    H. Goal
        - Must: `A--D--------`
        - Then: **`AbcDeFGhijkl`**

23. Found solution with 85 alternatives unexplored:
    A. Load(C1, P1, SFO)
        - Must: **`aBCdeFGhijkl`**
        - Then: `ab--eF--Ij--`
    B. Fly(P1, SFO, JFK)
        - Must: `abCdeFGhIjkl`
        - Then: `ab--Ef--Ij--`
    C. In any order: Load(C2, P1, JFK), Unload(C1, P1, JFK)
        - Must: `abCdEfGhIjkl`
        - Then: `AbcdEf--ijKl`
    D. Load(C1, P2, JFK)
        - Must: `AbcdEfGhijKl`
        - Then: `abcdEfGhiJKl`
    E. In any order: Fly(P1, JFK, SFO), Unload(C1, P2, JFK)
        - Must: `abcdEfGhiJKl`
        - Then: `AbcdeFGhijKl`
    F. Load(C1, P2, JFK)
        - Must: `AbcdeFGhijKl`
        - Then: `abcdeFGhiJKl`
    G. In any order: Unload(C1, P2, JFK), Unload(C2, P1, SFO)
        - Must: `abcdeFGhiJKl`
        - Then: `AbcDeFGhijkl`
    H. Goal
        - Must: `A--D--------`
        - Then: **`AbcDeFGhijkl`**

In [57]:
```
air_cargo_p2_problem = cargo_domain.substitute("""
cargo: C1 C2 C3
plane: P1 P2 P3
airport: JFK SFO ATL
""").problem(
start="""
At(C1, SFO)
At(C2, JFK)
At(C3, ATL)
At(P1, SFO)
At(P2, JFK)
At(P3, ATL)
""",
goal="""
At(C1, JFK)
At(C2, SFO)
At(C3, SFO)
""")
air_cargo_p2_problem.solve()
```

Out[57]:

Action sequence:

1. Fly(P3, ATL, JFK)
    - Must: **`abCdEfGhi---mNoPqrstuvwxyza`**
    - Then: `---------------pQr---------`
2. Load(C2, P3, JFK)
    - Must: `abCdEfGhi---mNopQrstuvwxyza`
    - Then: `---def---------pQr---vwX---`
3. In any order: Fly(P2, JFK, SFO), Fly(P3, JFK, ATL)
    - Must: `abCdefGhi---mNopQrstuvwXyza`
    - Then: `---def------mnOPqr---vwX---`
4. Load(C1, P2, SFO)
    - Must: `abCdefGhi---mnOPqrstuvwXyza`
    - Then: `abcdef------mnOPqrsTuvwX---`
5. In any order: Fly(P2, SFO, JFK), Load(C3, P3, ATL)
    - Must: `abcdefGhi---mnOPqrsTuvwXyza`

In [58]:
```
air_cargo_p2_problem.log
```

Out[58]:

Problem solution log:

Tried: **72**
Skipped: **30** (41%)
Explored: **42**
Time: **0.39876s**

1. Attempted action of 0 alternatives: (score `(3, 3, 3, 0)`)
   A. Goal
   - Must: **`-B---F--I------------------`**
   - Then: **`---------------------------`**
2. Attempted action of 41 alternatives: (score `(0, 24, 6, 3)`)
   A. In any order: Unload(C1, P2, JFK), Unload(C2, P1, SFO), Unload(C3, P1, SFO)
   - Must: **`abcdefghijkLmNo---sTuVwxYza`**

Start state:
`abCdEfGhijkLmNoPqrstuvwxyza`
At(C1, SFO)
At(C2, JFK)
At(C3, ATL)
At(P1, SFO)
At(P2, JFK)
At(P3, ATL)
Goal:
`-B---F--I------------------`
At(C1, JFK)
At(C2, SFO)
At(C3, SFO)

In [59]:
```
air_cargo_p3_problem = cargo_domain.substitute("""
cargo: C1 C2 C3 C4
plane: P1 P2
airport: JFK SFO ATL ORD
""").problem(
start="""
At(C1, SFO)
At(C2, JFK)
At(C3, ATL)
At(C4, ORD)
At(P1, SFO)
At(P2, JFK)
""",
goal="""
At(C1, JFK)
At(C2, SFO)
At(C3, JFK)
At(C4, SFO)
""")
air_cargo_p3_problem.solve()
```

Out[59]:

Action sequence:

1. Load(C1, P1, SFO)
   - Must: **`abcDeFghIjklmnOpqrsTuVwxyzabcdef`**
   - Then: `abcd------------qrsT----Yz------`
2. Fly(P1, SFO, JFK)
   - Must: `abcdeFghIjklmnOpqrsTuVwxYzabcdef`
   - Then: `abcd------------qRst----Yz------`
3. In any order: Load(C2, P1, JFK), Unload(C1, P1, JFK)
   - Must: `abcdeFghIjklmnOpqRstuVwxYzabcdef`
   - Then: `aBcdefgh--------qRst----yzAb----`
4. Fly(P1, JFK, ORD)
   - Must: `aBcdefghIjklmnOpqRstuVwxyzAbcdef`
   - Then: `aBcdefgh--------qrSt----yzAb----`
5. In any order: Fly(P2, JFK, ATL), Load(C4, P1, ORD)
   - Must: `aBcdefghIjklmnOpqrStuVwxyzAbcdef`

In [60]:
```
have_cake_problem = Domain("have_cake", """
state
  HaveCake
state
  EatenCake

to
  Eat
must
  HaveCake
then
  not HaveCake
  EatenCake

to
  MakeCake
then
  HaveCake
""").substitute({}).problem(
start="""
HaveCake
not EatenCake
""",
goal="""
HaveCake
EatenCake
""")
have_cake_problem.solve()
```

Out[60]:

Action sequence:

1. Eat
   - Must: **`-B`**
   - Then: `Ab`
2. MakeCake
   - Must: `A-`
   - Then: `AB`
3. Goal
   - Must: `AB`
   - Then: **`AB`**

In [61]:
```
spare_tire = Domain("spare_tire", """
state
    At(t, p)
where
    t is tire
    p is place

to
    Remove(t, r)
where
    t is tire
    r is removable
must
    At(t, r)
then
    At(t, Ground)

to
    PutOn(t, Axle)
where
    t is tire
    t2 is tire
    t != t2
must
    At(t, Ground)
    not At(t2, Axle)
then
    At(t, Axle)
    not At(t, Ground)

to
    LeaveOvernight
then
    not At(Spare, Ground)
    not At(Spare, Trunk)
    not At(Spare, Axle)
    not At(Flat, Ground)
    not At(Flat, Trunk)
    not At(Flat, Axle)

if
    At(t, p)
then
    not At(t, p2)
where
    t is tire
    p is place
    p2 is place
    p != p2
""").substitute("""
tire: Flat Spare
place: Axle Ground Trunk
removable: Axle Trunk
""")
spare_tire
```

Out[61]:

spare_tire

State name	BitMask
At(Flat, Axle)	`A-----`
At(Flat, Ground)	`-B----`
At(Flat, Trunk)	`--C---`
At(Spare, Axle)	`---D--`
At(Spare, Ground)	`----E-`
At(Spare, Trunk)	`-----F`

LeaveOvernight
Must: `------`
Then: `abcdef` not At(Spare, Ground); not At(Spare, Trunk); not At(Spare, Axle); not At(Flat, Ground);

In [62]:
```
spare_tire_problem = spare_tire.problem(
start="""
At(Flat, Axle)
At(Spare, Trunk)
""",
goal="""
At(Spare, Axle)
""")
spare_tire_problem.solve()
```

Out[62]:

Action sequence:

1. In any order: Remove(Flat, Axle), Remove(Spare, Trunk)
   - Must: **`AbcdeF`**
   - Then: `aBcdEf`
2. PutOn(Spare, Axle)
   - Must: `a--dEf`
   - Then: `aBcDef`
3. Goal
   - Must: `---D--`
   - Then: **`aBcDef`**

```
In [63]: spare_tire_problem.log
```

```
Out[63]:
```

Problem solution log:

Tried: **6**
Skipped: **0** (0%)
Explored: **6**
Time: **0.0012329s**

1. Attempted action of 0 alternatives: (score `(1, 1, 1, 0)`)
   A. Goal
      - Must: **`---D--`**
      - Then: **`------`**
2. Attempted action of 0 alternatives: (score `(0, 4, 3, 1)`)
   A. PutOn(Spare, Axle)
      - Must: **`a--dEf`**
      - Then: `a--Def`

   B. Goal

Start state:
`AbcdeF`
At(Flat, Axle)
At(Spare, Trunk)
Goal:
`---D--`
At(Spare, Axle)

```
In [64]: dinner_date = Domain("dinner_date", """
         state
             GarbageInside
         state
             HandsClean
         state
             IsQuiet
         state
             DinnerCooked
         state
             PresentWrapped

         to
             Cook
         must
             HandsClean
         then
             DinnerCooked

         to
             Wrap
         must
             IsQuiet
         then
             PresentWrapped

         to
             CarryOutGarbage
         then
             not GarbageInside
             not HandsClean

         to
             DollyOutGarbage
         then
             not GarbageInside
             not IsQuiet
         """).substitute({})
         dinner_date
```

Out[64]:

dinner_date

State name	BitMask
DinnerCooked	`A----`
GarbageInside	`-B---`
HandsClean	`--C--`
IsQuiet	`---D-`
PresentWrapped	`----E`

CarryOutGarbage
Must: `-----`
Then: `-bc--` not GarbageInside; not HandsClean

Cook
Must: `--C--` HandsClean
Then: `A-C--` DinnerCooked

DollyOutGarbage
Must: `-----`
Then: `-b-d-` not GarbageInside; not IsQuiet

Wrap
Must: `---D-` IsQuiet
Then: `---DE` PresentWrapped

```
In [65]: dinner_date_problem = dinner_date.problem(
start="""
GarbageInside
HandsClean
IsQuiet
not DinnerCooked
not PresentWrapped
""",
goal="""
DinnerCooked
PresentWrapped
not GarbageInside
""")
dinner_date_problem.solve()
```

Out[65]:

Action sequence:

1. Cook
   - Must: **`--CD-`**
   - Then: `A-C--`
2. In any order: CarryOutGarbage, Wrap
   - Must: `A--D-`
   - Then: `AbcDE`
3. Goal
   - Must: `Ab--E`
   - Then: **`AbcDE`**

In [66]:
```
dinner_date_problem.log
```
Out[66]:

Problem solution log:

Tried: **3**
Skipped: **0** (0%)
Explored: **3**
Time: **0.0008893s**

1. Attempted action of 0 alternatives: (score `(3, 3, 3, 0)`)
   A. Goal
      - Must: **`Ab--E`**
      - Then: **`-----`**
2. Attempted action of 6 alternatives: (score `(1, 2, 1, 2)`)
   A. In any order: CarryOutGarbage, Wrap
      - Must: **`A--D-`**
      - Then: `-bcDE`
   B. Goal

Start state:
`aBCDe`
GarbageInside
HandsClean
IsQuiet
Goal:
`Ab--E`
DinnerCooked
PresentWrapped

Blocks

From the blocks descriptions in this graphplan directory (http://www.cs.cmu.edu/afs/cs.cmu.edu/usr/avrim/Planning/Graphplan/)

In [67]:
```
block_domain = Domain("block_domain", """
state
    On(o, under)
where
    o is object
    under is object
    o != under

state
    OnTable(o)
where
    o is object

state
    Clear(o)
where
    o is object

state
    Holding(o)
where
    o is object

state
    ArmEmpty

to
    PickUp(o)
where
    o is object
must
    Clear(o)
    OnTable(o)
    ArmEmpty
then
    Holding(o)
    not ArmEmpty

to
    PutDown(o)
where
    o is object
must
    Holding(o)
then
    Clear(o)
    not Holding(o)
    ArmEmpty
    OnTable(o)

to
    Stack(o, under)
where
    o is object
    under is object
    o != under
```

```
must
    Clear(under)
    Holding(o)
then
    not Holding(o)
    ArmEmpty
    Clear(o)
    On(o, under)

to
    Unstack(o, under)
where
    o is object
    under is object
    o != under
must
    On(o, under)
    Clear(o)
    ArmEmpty
then
    Holding(o)
    not ArmEmpty
    Clear(under)

if
    On(o, under)
where
    o is object
    under is object
    o != under
    other is object
    other != under
    other != o
then
    not Clear(under)
    not OnTable(o)
    not Holding(o)
    not Holding(under)
    not On(o, other)

if
    OnTable(o)
where
    o is object
    under is object
    o != under
then
    not Holding(o)
    not On(o, under)

if
    Holding(o)
where
    o is object
    under is object
    o != under
then
```

```
    not ArmEmpty
    not On(o, under)
    not OnTable(o)

if
    ArmEmpty
where
    o is object
then
    not Holding(o)

if
    Clear(o)
where
    o is object
    over is object
    o != over
then
    not On(over, o)
""")
```

In [68]:

```
block_suss = block_domain.substitute("""object: A B C""").problem(
start="""
OnTable(A)
OnTable(B)
On(C, A)
Clear(B)
Clear(C)
ArmEmpty
""",
goal="""
On(A, B)
On(B, C)
""")
block_suss.solve()
```

Out[68]:

Action sequence:

1. Unstack(C, A)
    - Must: **`AbCDefghi-kLmNOp`**
    - Then: `aB-DefG-ijklm--p`
2. PutDown(C)
    - Must: `aBC---Gh-j-lmNOp`
    - Then: `AB-Defg-ijklm--P`
3. PickUp(B)
    - Must: `ABCDefghijklmNO-`
    - Then: `aBCDeFghijklm-oP`
4. Stack(B, C)
    - Must: `aB-D-F--ijkl-No-`
    - Then: `ABCdefghijKlm-oP`
5. PickUp(A)
    - Must: `ABC-efghijKlmN--`

In [69]:
```
blocks_4 = block_domain.problem(
bindings="""
object: A B C D
""",
start="""
OnTable(A)
On(B, A)
On(C, B)
On(D, C)
Clear(D)
ArmEmpty
""",
goal="""
On(B, A)
On(C, B)
On(A, D)
""")
blocks_4.solve()
```

Out[69]:

Action sequence:

1. Unstack(D, C)
    - Must: **`AbcdEfghi--lM-opQrstUVwxy`**
    - Then: `a--DEfghI-kl-no--rstu---y`
2. PutDown(D)
    - Must: `abcD----I-k-Mn-pQ-stuVwxy`
    - Then: `A--DEfghi-kl-no--rstu---Y`
3. Unstack(C, B)
    - Must: `AbcDEfghi-klMnopQr--uVwx-`
    - Then: `a-CDEfgHijkl-nopqrstu--xY`
4. PutDown(C)
    - Must: `abC-E--H-j-lM-opqr-t-Vwx-`
    - Then: `A-CDEfghijkl-nopqrstu--XY`
5. Unstack(B, A)
    - Must: `AbCDEfghijklMno-qr-tuVwX-`

In [70]:
```
blocks_5 = block_domain.problem(
bindings="""
object: A B C D E
""",
start="""
OnTable(A)
On(B, A)
On(C, B)
On(D, C)
On(E, D)
Clear(E)
ArmEmpty
""",
goal="""
On(B, A)
On(C, B)
On(D, C)
On(A, E)
""")
blocks_5.solve()
```

Out[70]:

Action sequence:

1. Unstack(E, D)
   - Must: **`AbcdeFghijk---oP--stU-wxyZabcdEFghij`**
   - Then: `a---EFghijK--no--rs--vw---abcde----j`
2. PutDown(E)
   - Must: `abcdE-----K--n-P-r-tUv-xyZ-bcdeFghij`
   - Then: `A---EFghijk--no--rs--vw---abcde----J`
3. Unstack(D, C)
   - Must: `AbcdEFghijk--noP-rstUvwxyZa---eFghi-`
   - Then: `a--DEFghiJk-mno-qrs--vwxyzabcde---iJ`
4. Stack(D, E)
   - Must: `abcD-F---J--m-oPq-stU-wxyza--d-Fghi-`
   - Then: `A--DEfghijk-mno-qrs--vwxyzAbcde---iJ`
5. Unstack(C, B)
   - Must: `AbcDEfghijk-mnoPqrstUvw--zA--deFghi-`

In [71]:
```
blocks_impossible = block_domain.problem(
bindings="""
object: A B C
""",
start="""
OnTable(A)
On(B, A)
On(C, B)
Clear(C)
ArmEmpty
""",
goal="""
On(C, B)
On(B, A)
On(A, C)
""")
blocks_impossible.solve()
```

In [72]:
```
# The log looks very minimal, but we can detect that the solution is impossi
# there's no "last" action that solves a goal state and doesn't invalidate a
blocks_impossible.log
```

Out[72]:

Problem solution log:

Tried: **1**
Skipped: **0** (0%)
Explored: **1**
Time: **0.000139s**

1. Attempted action of 0 alternatives: (score `(3, 3, 1, 0)`)
   A. Goal
      - Must: **`---------IJ--M---`**
      - Then: `----------------`
2. No actions can accomplish the prerequisite `--------IJ--M---` from:
   A. Goal
      - Must: **`---------IJ--M---`**
      - Then: `----------------`
3. Found no solution

Start state:
`AbcDefghiJklMNop`
ArmEmpty
Clear(C)
On(B, A)
On(C, B)
OnTable(A)
Goal:
`--------IJ--M---`
On(A, C)
On(B, A)
On(C, B)

```
In [73]: tsp_domain = Domain("tsp", """
         state
             At(l)
         where
             l is location

         state
             Visited(l)
         where
             l is location

         state
             (l1 l2 CONNECTED)
         where
             l1 is location
             l2 is location
             l1 != l2

         to
             Move(start, end)
         where
             start is location
             end is location
             start != end
         must
             At(start)
             (start end CONNECTED)
         then
             At(end)
             Visited(end)

         if
             At(l)
         where
             l is location
             l2 is location
             l != l2
         then
             not At(l2)
         """)
```

```
In [74]: tsp_world = tsp_domain.problem(
         bindings="""
         location: A B C D E F G H I
         """,
         start="""
         default_false

         (A B CONNECTED)
         (A C CONNECTED)
         (A D CONNECTED)
         (A E CONNECTED)
         (A F CONNECTED)
         (A G CONNECTED)
         (A H CONNECTED)
         (A I CONNECTED)

         (B A CONNECTED)
         (B C CONNECTED)
         (B D CONNECTED)
         (B E CONNECTED)
         (B F CONNECTED)
         (B G CONNECTED)
         (B H CONNECTED)
         (B I CONNECTED)

         (C A CONNECTED)
         (C B CONNECTED)
         (C D CONNECTED)
         (C E CONNECTED)
         (C F CONNECTED)
         (C G CONNECTED)
         (C H CONNECTED)
         (C I CONNECTED)

         (D A CONNECTED)
         (D B CONNECTED)
         (D C CONNECTED)
         (D E CONNECTED)
         (D F CONNECTED)
         (D G CONNECTED)
         (D H CONNECTED)
         (D I CONNECTED)

         (E A CONNECTED)
         (E B CONNECTED)
         (E C CONNECTED)
         (E D CONNECTED)
         (E F CONNECTED)
         (E G CONNECTED)
         (E H CONNECTED)
         (E I CONNECTED)

         (F A CONNECTED)
         (F B CONNECTED)
         (F C CONNECTED)
         (F D CONNECTED)
         (F E CONNECTED)
         (F G CONNECTED)
         (F H CONNECTED)
         (F I CONNECTED)

         (G A CONNECTED)
         (G B CONNECTED)
         (G C CONNECTED)
         (G D CONNECTED)
         (G E CONNECTED)
         (G F CONNECTED)
         (G H CONNECTED)
         (G I CONNECTED)

         (H A CONNECTED)
         (H B CONNECTED)
         (H C CONNECTED)
         (H D CONNECTED)
         (H E CONNECTED)
         (H F CONNECTED)
         (H G CONNECTED)
         (H I CONNECTED)

         (I A CONNECTED)
         (I B CONNECTED)
         (I C CONNECTED)
         (I D CONNECTED)
         (I E CONNECTED)
         (I F CONNECTED)
         (I G CONNECTED)
         (I H CONNECTED)

         At(A)

         """,
         goal="""
         At(A)
         Visited(A)
         Visited(B)
         Visited(C)
         Visited(D)
         Visited(E)
         Visited(F)
         Visited(G)
         Visited(H)
         Visited(I)
         """)
         tsp_world.solve()
```

Out[74]:

Action sequence:

1. Move(A, I)
    - Must: **`-------HI--------R--------A--------J--------S--------B--------K--------TUvwxyzabc---------`**
    - Then: `-------H-----------------------------------------------------------------uvwxyzabC--------L`

2. Move(I, H)
    - Must: `--------I--------R--------A--------J--------S--------B---------K--------TuvwxyzabC--------L`
    - Then: `-------H-------------------------------------------------------------------TuvwxyzaBc-------KL`
3. Move(H, G)
    - Must: `--------I--------R--------A--------J--------S--------B---------K---------uvwxyzaBc-------KL`
    - Then: `-------H--------------------------------------------------`

```
In [75]: tsp_world.log
```

Out[75]:

te:

`}HIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUvwxyzabcdefghijkl`

NNECTED)
NNECTED)
NNECTED)
NNECTED)
NNECTED)
NNECTED)
NNECTED)
NECTED)
NNECTED)
NNECTED)
NNECTED)
NNECTED)
NNECTED)
NNECTED)

In [76]:
```
tsp_peterson = tsp_domain.problem(
bindings="""
location: A B C D E F G H I J
""",
start="""
default_false
(A B CONNECTED)
(B A CONNECTED)
(A C CONNECTED)
(C A CONNECTED)
(A I CONNECTED)
(I A CONNECTED)
(B F CONNECTED)
(F B CONNECTED)
(B H CONNECTED)
(H B CONNECTED)
(C D CONNECTED)
(D C CONNECTED)
(C E CONNECTED)
(E C CONNECTED)
(D H CONNECTED)
(H D CONNECTED)
(D J CONNECTED)
(J D CONNECTED)
(E F CONNECTED)
(F E CONNECTED)
(E G CONNECTED)
(G E CONNECTED)
(F J CONNECTED)
(J F CONNECTED)
(G H CONNECTED)
(H G CONNECTED)
(G I CONNECTED)
(I G CONNECTED)
(I J CONNECTED)
(J I CONNECTED)

At(A)
""",
goal="""
At(A)
Visited(A)
Visited(B)
Visited(C)
Visited(D)
Visited(E)
Visited(F)
Visited(G)
Visited(H)
Visited(I)
Visited(J)
""")
tsp_peterson.solve()
```

Out[76]:

Action sequence:

1. Move(A, I)
   - Must: **`-------H-J-----------V-------D----------O-----U--------`**
     `-----I-----O-----------A-C--------LMnopqrstuv----------`
   - Then: `-------H-----------------------------------------------`
     `-----------------------------------mnopqrstUv--------E-`
2. Move(I, J)
   - Must: `---------J-----------V-------D----------O-----U--------`
     `-----I-----O-----------A-C--------LmnopqrstUv----------`
   - Then: `-------H-----------------------------------------------`
     `-------------------------C---------mnopqrstuV--------EF`
3. Move(J, I)
   - Must: `---------J-----------V-------D----------O-----U--------`
     `-----I-----O-----------A----------LmnopqrstuV---------F`
   - Then: `-------H-----------------------------------------------`

In [77]:
```
tsp_facts = tsp_domain.problem(
bindings="""
location: Boston NewYork Pittsburgh Toronto Albany
""",
start="""
default_false
(Boston NewYork CONNECTED)
(NewYork Boston CONNECTED)
(Pittsburgh Boston CONNECTED)
(Boston Pittsburgh CONNECTED)
(Pittsburgh NewYork CONNECTED)
(NewYork Pittsburgh CONNECTED)
(Toronto Pittsburgh CONNECTED)
(Toronto NewYork CONNECTED)
(NewYork Toronto CONNECTED)
(NewYork Albany CONNECTED)
(Albany NewYork CONNECTED)
(Albany Toronto CONNECTED)
(Toronto Albany CONNECTED)
At(Pittsburgh)
""",
goal="""
Visited(Boston)
Visited(NewYork)
Visited(Pittsburgh)
Visited(Toronto)
Visited(Albany)
At(Pittsburgh)
""")
tsp_facts.solve()
```

Out[77]:

Action sequence:

1. Move(Pittsburgh, NewYork)
   - Must: **`-B----G--J-L--O-Q---uvwXy-----`**
   - Then: `--------------O-----uvWxy--B--`
2. Move(NewYork, Toronto)
   - Must: `-B----G--J-L----Q---uvWxy-----`
   - Then: `-----------L--O-----uvwxY--B-D`
3. Move(Toronto, Albany)
   - Must: `-B----G--J------Q---uvwxY----D`
   - Then: `-----------L--O-Q---UvwxyZ-B-D`
4. Move(Albany, NewYork)
   - Must: `-B----G--J----------UvwxyZ---D`
   - Then: `-B---------L--O-Q---uvWxyZ-B-D`
5. Move(NewYork, Boston)
   - Must: `------G--J----------uvWxyZ-B-D`

Fixit, our nemesis

This example (from here (http://www.cs.cmu.edu/afs/cs.cmu.edu/usr/avrim/Planning/Graphplan/)) is hard, and the planner makes very little progress on it. `python fixit_exampe.py` also runs this. Because it doesn't come to a solution it can be easier to run it from the command-line.

In the domain we've changed a few things: the fixit ops seem to be setup for a system where you can never require a state be false. That's not a problem for this solver, so we remove some states (like `(not-fastened hub)` is replaced with `not (fastened hub)`. This doesn't change the problem, just rephrase it.

Some constraints have been added to the end, which seem to help the algorithm infer more about the actions.

Lastly we add an action `(put-away-all-tools)` which cleans everything, which is a particular sticking point for this algorithm. Because the fixit problem requires that you do something like `(put-away jack boot) (put-away pump boot) (put-away wrench boot) (close boot)` at the end, but our search prefers to do more work at the end (since it is working backwards) and so it heavily prefers impossible search branches.

This still remains by far the hardest problem for this algorithm.

```
In [78]: fixit_domain = Domain("fixit", """
         to
             (open c)
         where
             c is container
         must
             (unlocked c)
             (closed c)
         then
             not (closed c)

         to
             (close c)
         where
             c is container
         must
             not (closed c)
         then
             (closed c)

         to
             (fetch o c)
         where
             o is object
             c is container
         must
             (in o c)
             not (closed c)
         then
             not (in o c)
             (have o)

         to
             (put-away o c)
         where
             o is object
             c is container
         must
             (have o)
             not (closed c)
         then
             (in o c)
             not (have o)

         to
             (loosen n h)
         where
             n is nut
             h is hub
         must
             (have wrench)
             (tight n h)
             (on-ground h)
         then
             (loose n h)
             not (tight n h)
```

```
to
    (tighten n h)
where
    n is nut
    h is hub
must
    (have wrench)
    (loose n h)
    (on-ground h)
then
    (tight n h)
    not (loose n h)

to
    (jack-up h)
where
    h is hub
must
    (on-ground h)
    (have jack)
then
    not (on-ground h)
    not (have jack)

to
    (jack-down h)
where
    h is hub
must
    not (on-ground h)
then
    (on-ground h)
    (have jack)

to
    (undo n h)
where
    n is nut
    h is hub
must
    not (on-ground h)
    (fastened h)
    (have wrench)
    (loose n h)
then
    (have n)
    not (fastened h)
    not (loose n h)

to
    (do-up n h)
where
    n is nut
    h is hub
must
    (have wrench)
    not (fastened h)
    not (on-ground h)
    (have n)
then
    (loose n h)
    (fastened h)
    not (have n)

to
    (remove-wheel w h)
where
    w is wheel
    h is hub
must
    not (on-ground h)
    (on w h)
    not (fastened h)
then
    (have w)
    (free h)
    not (on w h)

to
    (put-on-wheel w h)
where
    w is wheel
    h is hub
must
    (have w)
    (free h)
    not (fastened h)
    not (on-ground h)
then
    (on w h)
    not (free h)
    not (have w)

to
    (inflate w)
where
    w is wheel
must
    (have pump)
    not (inflated w)
    (intact w)
then
    (inflated w)

# Encompasses:
# optionally open/close boot, (put-away wrench boot) (put-away jack boot) (p
# As mentioned above, this helps our algorithm with the final steps, which a
# for it
to
    (put-away-tools)
must
    (have jack)
    (have wrench)
    (have pump)
then
    not (have jack)
    not (have pump)
    not (have wrench)
    (in jack boot)
    (in pump boot)
    (in wrench boot)

# Constraints:

if
    (have o)
where
    o is object
    c is container
then
    not (in o c)

if
    (in o c)
where
    o is object
    c is container
then
    not (have o)

if
    (fastened h)
where
    h is hub
then
    not (free h)

if
    (loose n h)
where
    n is nut
    h is hub
then
    not (free h)
    (fastened h)

if
    (tight n h)
where
    n is nut
    h is hub
then
    not (loose n h)
    not (free h)
    (fastened h)

if
    (have n)
where
    n is nut
    h is hub
then
    not (loose n h)
    not (tight n h)
""")
```

```
In [79]: fixit_1 = fixit_domain.substitute("""
         object: wrench jack pump nuts wheel1 wheel2
         hub: the-hub
         nut: nuts
         container: boot
         wheel: wheel1 wheel2
         """)
         fixit_1
```

Out[79]:

fixit

State name	BitMask
(closed boot)	A------------------------
(fastened the-hub)	-B-----------------------
(free the-hub)	--C----------------------
(have jack)	---D---------------------
(have nuts)	----E--------------------
(have pump)	-----F-------------------
(have wheel1)	------G------------------
(have wheel2)	-------H-----------------
(have wrench)	--------I----------------
(in jack boot)	---------J---------------
(in nuts boot)	----------K--------------
(in pump boot)	-----------L-------------
(in wheel1 boot)	------------M------------
(in wheel2 boot)	-------------N-----------
(in wrench boot)	--------------O----------
(inflated wheel1)	---------------P---------
(inflated wheel2)	----------------Q--------
(intact wheel1)	-----------------R-------
(intact wheel2)	------------------S------
(loose nuts the-hub)	-------------------T-----
(on wheel1 the-hub)	--------------------U----
(on wheel2 the-hub)	---------------------V---
(on-ground the-hub)	----------------------W--
(tight nuts the-hub)	-----------------------X-
(unlocked boot)	------------------------Y

(close boot)
Must: `a------------------------` not (closed boot)
Then: `A------------------------` (closed boot)

(do-up nuts the-hub)
Must: `-b--E---I-k---o----t--wx-` (have wrench); not (fastened the-hub); not (on-ground the
Then: `-Bc-e---I-k---o----T--wx-` (loose nuts the-hub); (fastened the-hub); not (have nuts)

(fetch jack boot)
Must: `a--d-----J---------------` (in jack boot); not (closed boot)
Then: `a--D-----j---------------` not (in jack boot); (have jack)

(fetch nuts boot)
Must: `a---e-----K--------------` (in nuts boot); not (closed boot)
Then: `a---E-----k--------t---x-` not (in nuts boot); (have nuts)

(fetch pump boot)
Must: `a----f-----L-------------` (in pump boot); not (closed boot)
Then: `a----F-----l-------------` not (in pump boot); (have pump)

(fetch wheel1 boot)
Must: `a-----g-----M------------` (in wheel1 boot); not (closed boot)
Then: `a-----G-----m------------` not (in wheel1 boot); (have wheel1)

(fetch wheel2 boot)
Must: `a------h-----N-----------` (in wheel2 boot); not (closed boot)
Then: `a------H-----n-----------` not (in wheel2 boot); (have wheel2)

(fetch wrench boot)
Must: `a-------i-----O----------` (in wrench boot); not (closed boot)
Then: `a-------I-----o----------` not (in wrench boot); (have wrench)

(inflate wheel1)
Must: `-----F-----l---p-R-------` (have pump); not (inflated wheel1); (intact wheel1)
Then: `-----F-----l---P-R-------` (inflated wheel1)

(inflate wheel2)
Must: `-----F-----l----q-S------` (have pump); not (inflated wheel2); (intact wheel2)
Then: `-----F-----l----Q-S------` (inflated wheel2)

(jack-down the-hub)
Must: `----------------------w--` not (on-ground the-hub)
Then: `---D-----j------------W--` (on-ground the-hub); (have jack)

(jack-up the-hub)
Must: `---D-----j------------W--` (on-ground the-hub); (have jack)
Then: `---d-----j------------w--` not (on-ground the-hub); not (have jack)

(loosen nuts the-hub)
Must: `-Bc-----I-----o----t--WX-` (have wrench); (tight nuts the-hub); (on-ground the-hub)
Then: `-Bc-----I-----o----T--Wx-` (loose nuts the-hub); not (tight nuts the-hub)

(open boot)
Must: `A-----------------------Y` (unlocked boot); (closed boot)
Then: `a-----------------------Y` not (closed boot)

(put-away jack boot)
Must: `a--D-----j---------------` (have jack); not (closed boot)
Then: `a--d-----J---------------` (in jack boot); not (have jack)

(put-away nuts boot)
Must: `a---E-----k--------t---x-` (have nuts); not (closed boot)
Then: `a---e-----K--------t---x-` (in nuts boot); not (have nuts)

(put-away pump boot)
Must: `a----F-----l-------------` (have pump); not (closed boot)
Then: `a----f-----L-------------` (in pump boot); not (have pump)

(put-away wheel1 boot)
Must: `a-----G-----m------------` (have wheel1); not (closed boot)
Then: `a-----g-----M------------` (in wheel1 boot); not (have wheel1)

(put-away wheel2 boot)
Must: `a------H-----n-----------` (have wheel2); not (closed boot)
Then: `a------h-----N-----------` (in wheel2 boot); not (have wheel2)

(put-away wrench boot)
Must: `a-------I-----o----------` (have wrench); not (closed boot)
Then: `a-------i-----O----------` (in wrench boot); not (have wrench)

(put-away-tools)
Must: `---D-F--Ij-l--o----------` (have jack); (have wrench); (have pump)
Then: `---d-f--iJ-L--O----------` not (have jack); not (have pump); not (have wrench); (in ja

(put-on-wheel wheel1 the-hub)
Must: `-bC---G-----m---------w--` (have wheel1); (free the-hub); not (fastened the-hub); not
Then: `-bc---g-----m-------U-w--` (on wheel1 the-hub); not (free the-hub); not (have wheel1)

(put-on-wheel wheel2 the-hub)
Must: `-bC----H-----n--------w--` (have wheel2); (free the-hub); not (fastened the-hub); not
Then: `-bc----h-----n-------Vw--` (on wheel2 the-hub); not (free the-hub); not (have wheel2)

(remove-wheel wheel1 the-hub)
Must: `-b-----------------U-w--` not (on-ground the-hub); (on wheel1 the-hub); not (fasten
Then: `-bC---G-----m-------u-w--` (have wheel1); (free the-hub); not (on wheel1 the-hub)

(remove-wheel wheel2 the-hub)
Must: `-b------------------Vw--` not (on-ground the-hub); (on wheel2 the-hub); not (fasten
Then: `-bC----H-----n-------vw--` (have wheel2); (free the-hub); not (on wheel2 the-hub)

(tighten nuts the-hub)
Must: `-Bc-----I-----o----T--W--` (have wrench); (loose nuts the-hub); (on-ground the-hub)
Then: `-Bc-----I-----o----t--WX-` (tight nuts the-hub); not (loose nuts the-hub)

(undo nuts the-hub)
Must: `-Bc-----I-----o----T--w--` not (on-ground the-hub); (fastened the-hub); (have wrenc
Then: `-bc-E---I-k---o----t--wx-` (have nuts); not (fastened the-hub); not (loose nuts the-hu

In [80]:
```
fixit_1_problem = fixit_1.problem(
start="""
default_false
(intact wheel2)
(in jack boot)
(in pump boot)
(in wheel2 boot)
(in wrench boot)
(on wheel1 the-hub)
(on-ground the-hub)
(tight nuts the-hub)
not (inflated wheel2)
(unlocked boot)
(fastened the-hub)
(closed boot)
""",
goal="""
(on wheel2 the-hub)
(in wheel1 boot)
(inflated wheel2)
(in wrench boot)
(in jack boot)
(in pump boot)
(tight nuts the-hub)
(closed boot)
""")
```

In [81]:
```
# Note this takes about a minute to run
fixit_1_problem.solve()
```

6. In any order: (fetch wheel2 boot), (put-away nuts boot), (put-away wrench boot), (remove-wheel wheel1 the-hub)
   - Must: `ab--Ef-hI-kL-No-q-StU-wx-`
   - Then: `abCde-GHijK-mnO----tu-wxY`
7. In any order: (fetch nuts boot), (fetch wrench boot), (put-on-wheel wheel2 the-hub)
   - Must: `abC-efGHi-KLmnO-q-S---w--`
   - Then: `abcdE-GhIjk-mno----tuVwxY`
8. (do-up nuts the-hub)
   - Must: `ab--EfG-I-kLm-o-q-St-Vwx-`
   - Then: `aBcde-GhIjk-mno----TuVwxY`
9. (jack-down the-hub)
   - Must: `aBc--fG-I--Lm-o-q-ST-Vw--`
   - Then: `aBcDe-GhIjk-mno----TuVWxY`
10. In any order: (fetch pump boot), (put-away jack boot)
    - Must: `aBcD-fG-Ij-Lm-o-q-ST-VW--`
    - Then: `aBcdeFGhIJklmno----TuVWxY`
11. In any order: (fetch jack boot), (inflate wheel2), (put-away wheel1 boot), (tighten nuts the-

In [82]:
```
# The log is really long, so we'll truncate
fixit_1_problem.log.activity = fixit_1_problem.log.activity[-4:]
fixit_1_problem.log
```

Out[82]:

Problem solution log:

Tried: **23282**
Skipped: **20980** (90%)
Explored: **2302**
Time: **15.353s**

1. Skipped adding action (fetch nuts boot) because must=`abCDeF-HIjKl-no-q-S-U-w--` has been seen
   show action
2. Attempted action of 15867 alternatives: (score `(0, 18, 0, 23)`)
   A. (open boot)
      - Must: **`ABcd-f-hiJ-L-NO-q-StU-WXY`**
      - Then: a`-----------------------Y`
   B. (fetch wrench boot)

Start state:
`ABcdefghiJkLmNOpqrStUvWXY`
(closed boot)
(fastened the-hub)
(in jack boot)
(in pump boot)
(in wheel2 boot)
(in wrench boot)
(intact wheel2)
(on wheel1 the-hub)
(on-ground the-hub)
(tight nuts the-hub)
(unlocked boot)
Goal:
`A--------J-LM-O-Q----V-X-`
(closed boot)

In [ ]: