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 [77]:
```
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 [78]:
```
cargo_domain = Domain("cargo_domain", """
# This is a domain where cargo and planes can be At(thing, airport), or a pl
# 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 [79]:
```
simple_cargo = cargo_domain.substitute("""
cargo: C1 C2
plane: P1 P2
airport: JFK SFO
""")
```

In [80]:
```
# Note that the substituted domain has a nice representation (you could also
simple_cargo
```

Out[80]:

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 [81]:
```
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 [82]:
```
simple_cargo_problem.solve()
```

Out[82]:

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 [83]:
```
simple_cargo_problem.log
```

Out[83]:

Problem solution log:

Tried: **17**
Skipped: **9** (52%)
Explored: **8**
Time: **0.0035231s**

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

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

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 [84]:
```
air_cargo_p1 = cargo_domain.substitute("""
cargo: C1 C2
plane: P1 P2
airport: JFK SFO
""")
air_cargo_p1
```

Out[84]:

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 [85]:
```
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 [86]:
```
air_cargo_p1_problem.solve()
```

Out[86]:

Action sequence:

1. Fly(P2, JFK, SFO)
    - Must: **`aBCdeFGhijkl`**
    - Then: `------gH----`
2. Load(C1, P2, SFO)
    - Must: `aBCdeFgHijkl`
    - Then: `ab----gHiJ--`
3. In any order: Fly(P1, SFO, JFK), Fly(P2, SFO, JFK)
    - Must: `abCdeFgHiJkl`
    - Then: `ab--EfGhiJ--`
4. Load(C2, P1, JFK)
    - Must: `abCdEfGhiJkl`
    - Then: `abcdEfGhiJKl`
5. Fly(P1, JFK, SFO)
    - Must: `abcdEfGhiJKl`

In [87]:
```
air_cargo_p1_problem.log
```

Out[87]:

Problem solution log:

Tried: **30**
Skipped: **17** (56%)
Explored: **13**
Time: **0.014333s**

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 19 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 26 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`**

6. Skipped adding action Load(C2, P1, SFO) because must=`AbcDeFGhijkl` has been seen show action
7. Skipped adding action Unload(C1, P2, JFK) 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 Unload(C2, P1, SFO) because must=`abcdeFGhiJKl` has been seen show action
10. Skipped adding action Load(C2, P1, SFO) because must=`abcDeFGhiJkl` has been seen show action
11. Skipped adding action Load(C1, P2, JFK) because must=`AbcdeFGhijKl` has been seen show action
12. Skipped adding action Unload(C1, P2, JFK) because must=`abcDeFGhiJkl` has been seen show action
13. Skipped adding action Unload(C2, P1, SFO) because must=`AbcdeFGhijKl` has been seen show action
14. Skipped adding action Unload(C1, P2, JFK) because must=`abcdeFGhiJKl` has been seen show action
15. Attempted action of 24 alternatives: (score `(0, 12, 6, 3)`)
    A. Fly(P1, JFK, SFO)
    - Must: **`abcdEfGhiJKl`**
    - Then: `----eF------`

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

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

16. Skipped adding action Fly(P1, SFO, JFK) because must=`abcdeFGhiJKl` has been seen show action
17. Attempted action of 30 alternatives: (score `(0, 12, 4, 4)`)
    A. Load(C2, P1, JFK)
    - Must: **`abCdEfGhiJkl`**
    - Then: `--cdEf----Kl`

    B. Fly(P1, JFK, SFO)
    - Must: `abcdEfGhiJKl`
    - Then: `--cdeF----Kl`

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

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

18. Attempted action of 41 alternatives: (score `(0, 12, 2, 5)`)

A. Fly(P1, SFO, JFK)
- Must: **abCdeFGhiJkl**
- Then: `----Ef------`

B. Load(C2, P1, JFK)
- Must: `abCdEfGhiJkl`
- Then: `--cdEf----Kl`

C. Fly(P1, JFK, SFO)
- Must: `abcdEfGhiJKl`
- Then: `--cdeF----Kl`

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

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

19. Attempted action of 49 alternatives: (score `(0, 12, 2, 6)`)

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. Fly(P1, JFK, SFO)
- 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**

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

21. Skipped adding action Unload(C1, P2, JFK) because must=`abCdeFGhiJkl` has been seen show action

22. Skipped adding action Fly(P1, SFO, JFK) because must=`AbcdeFGhijKl` has been seen show action

23. Attempted action of 54 alternatives: (score `(0, 12, 4, 5)`)

A. In any order: Load(C1, P2, JFK), Load(C2, P1, JFK)
- Must: **AbCdEfGhijkl**
- Then: `abcdEfGhiJKl`

B. Fly(P1, JFK, 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`**

24. Skipped adding action Fly(P1, SFO, JFK) because must=`AbCdeFGhijkl` has been seen
    show action
25. Skipped adding action Fly(P1, SFO, JFK) because must=`abCdeFGhiJkl` has been seen
    show action
26. Skipped adding action Load(C1, P2, JFK) because must=`AbCdEfGhijkl` has been seen
    show action
27. Attempted action of 67 alternatives: (score `(0, 12, 4, 6)`)
    A. In any order: Fly(P1, SFO, JFK), Fly(P2, SFO, JFK)
    - Must: **`abCdeFgHiJkl`**
    - Then: `----EfGh----`

    B. Load(C2, P1, JFK)
    - Must: `abCdEfGhiJkl`
    - Then: `--cdEfGh--Kl`

    C. Fly(P1, JFK, SFO)
    - Must: `abcdEfGhiJKl`
    - Then: `--cdeFGh--Kl`

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

    E. Goal
    - Must: `A--D--------`
    - Then: **`AbcDeFGhijkl`**
28. Skipped adding action Fly(P2, JFK, SFO) because must=`abCdeFGhiJkl` has been seen
    show action
29. Attempted action of 70 alternatives: (score `(0, 12, 2, 7)`)
    A. Load(C1, P2, SFO)
    - Must: **`aBCdeFgHijkl`**
    - Then: `ab----gHiJ--`

    B. In any order: Fly(P1, SFO, JFK), Fly(P2, SFO, JFK)
    - Must: `abCdeFgHiJkl`
    - Then: `ab--EfGhiJ--`

    C. Load(C2, P1, JFK)
    - Must: `abCdEfGhiJkl`
    - Then: `abcdEfGhiJKl`

    D. Fly(P1, JFK, SFO)
    - 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`**
30. Attempted action of 76 alternatives: (score `(0, 12, 0, 8)`)
    A. Fly(P2, JFK, SFO)
    - Must: **`aBCdeFGhijkl`**

- Then: `------gH----`

B. Load(C1, P2, SFO)
- Must: `aBCdeFgHijkl`
- Then: `ab----gHiJ--`

C. In any order: Fly(P1, SFO, JFK), Fly(P2, SFO, JFK)
- Must: `abCdeFgHiJkl`
- Then: `ab--EfGhiJ--`

D. Load(C2, P1, JFK)
- Must: `abCdEfGhiJkl`
- Then: `abcdEfGhiJKl`

E. Fly(P1, JFK, SFO)
- 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`**

31. Found solution with 76 alternatives unexplored:

A. Fly(P2, JFK, SFO)
- Must: **`aBCdeFGhijkl`**
- Then: `------gH----`

B. Load(C1, P2, SFO)
- Must: `aBCdeFgHijkl`
- Then: `ab----gHiJ--`

C. In any order: Fly(P1, SFO, JFK), Fly(P2, SFO, JFK)
- Must: `abCdeFgHiJkl`
- Then: `ab--EfGhiJ--`

D. Load(C2, P1, JFK)
- Must: `abCdEfGhiJkl`
- Then: `abcdEfGhiJKl`

E. Fly(P1, JFK, SFO)
- 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`**

In [88]:
```
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[88]:

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 [89]:
```
air_cargo_p2_problem.log
```

Out[89]:

Problem solution log:

Tried: **112**
Skipped: **84** (75%)
Explored: **28**
Time: **0.27987s**

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 [90]:
```
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[90]:

Action sequence:

1. Fly(P2, JFK, SFO)
   - Must: **abcDeFghIjklmnOpqrsTuVwxyzabcdef**
   - Then: ----------uvwX-------
2. Load(C1, P2, SFO)
   - Must: abcDeFghIjklmnOpqrsTuvwXyzabcdef
   - Then: abcd----------uvwXyZ------
3. Fly(P2, SFO, ATL)
   - Must: abcdeFghIjklmnOpqrsTuvwXyZabcdef
   - Then: abcd----------UvwxyZ------
4. Load(C3, P2, ATL)
   - Must: abcdeFghIjklmnOpqrsTUvwxyZabcdef
   - Then: abcd----ijkl--------UvwxyZ--cD--
5. In any order: Fly(P1, SFO, ORD), Fly(P2, ATL, JFK)
   - Must: abcdeFghijklmnOpqrsTUvwxyZabcDef

In [91]:
```
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[91]:

Action sequence:

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

In [92]:
```
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[92]:

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 [93]:
```
spare_tire_problem = spare_tire.problem(
start="""
At(Flat, Axle)
At(Spare, Trunk)
""",
goal="""
At(Spare, Axle)
""")
spare_tire_problem.solve()
```

Out[93]:

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 [94]:
```
spare_tire_problem.log
```

Out[94]:

Problem solution log:

Tried: **6**
Skipped: **1** (16%)
Explored: **5**
Time: **0.0014572s**

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

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

In [95]:
```
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[95]:

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 [96]:
```
dinner_date_problem = dinner_date.problem(
start="""
GarbageInside
HandsClean
IsQuiet
not DinnerCooked
not PresentWrapped
""",
goal="""
DinnerCooked
PresentWrapped
not GarbageInside
""")
dinner_date_problem.solve()
```

Out[96]:

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 [97]:
```
dinner_date_problem.log
```

- Then: `A-C--`

B. In any order: CarryOutGarbage, Wrap
- Must: `A--D-`
- Then: `AbcDE`

C. Goal
- Must: `Ab--E`
- Then: **`AbcDE`**

4. Found solution with 8 alternatives unexplored:

A. Cook
- Must: **`--CD-`**
- Then: `A-C--`

B. In any order: CarryOutGarbage, Wrap
- Must: `A--D-`
- Then: `AbcDE`

C. Goal
- Must: `Ab--E`
- Then: **`AbcDE`**

Blocks

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

In [98]:
```
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 [99]:
```
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[99]:

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 [100]:
```
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[100]:

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 [101]:
```
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[101]:

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. PutDown(D)
   - 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 [102]:
```
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 [103]:
```
# 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[103]:

Problem solution log:

Tried: **1**
Skipped: **0** (0%)
Explored: **1**
Time: **0.00015116s**

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 [104]:
```
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 [105]:
```
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[105]:

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 [106]:
```
tsp_world.log
```

Out[106]:

:e:

```
;HIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUvwxyzabcdefghijkl
NNECTED)
NNECTED)
NNECTED)
NNECTED)
INECTED)
NNECTED)
NNECTED)
NECTED)
NNECTED)
NNECTED)
NNECTED)
NNECTED)
NNECTED)
NNECTED)
```

In [107]:
```
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[107]:

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 [108]:
```
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[108]:

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 [109]:
```
fixit_domain = Domain("fixit", """
to
    (open c)
where
    c is container
must
    (unlocked c)
    (closed c)
then
    not (closed c)
    (open c)

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

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

to
    (put-away o c)
where
    o is object
    c is container
must
    (have o)
    (open 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 (on-ground h)
    not (have jack)

to
    (jack-down h)
where
    h is hub
must
    (not-on-ground h)
then
    not (not-on-ground h)
    (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)
    (unfastened h)
    not (fastened h)
    not (loose n h)

to
    (do-up n h)
where
```

```
    n is nut
    h is hub
must
    (have wrench)
    (unfastened h)
    (not-on-ground h)
    (have n)
then
    (loose n h)
    (fastened h)
    not (unfastened h)
    not (have n)

to
    (remove-wheel w h)
where
    w is wheel
    h is hub
must
    (not-on-ground h)
    (on w h)
    (unfastened 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)
    (unfastened 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
    not (not-inflated w)
    (inflated w)

to
    cuss
then
```

```
    not annoyed
""")
```

In [110]:
```
fixit_1 = fixit_domain.substitute("""
object: wrench jack pump the-hub nuts wheel1 wheel2
hub: the-hub
nut: nuts
container: boot
wheel: wheel1 wheel2
""")
fixit_1
```

Out[110]:

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 the-hub)	------G-----------------------
(have wheel1)	-------H-----------------------
(have wheel2)	--------I-----------------------
(have wrench)	---------J----------------------
(in jack boot)	----------K----------------------
(in nuts boot)	-----------L---------------------
(in pump boot)	------------M--------------------
(in the-hub boot)	-------------N-------------------
(in wheel1 boot)	--------------O------------------
(in wheel2 boot)	---------------P-----------------
(in wrench boot)	----------------Q----------------
(inflated wheel1)	-----------------R---------------
(inflated wheel2)	------------------S--------------
(intact wheel1)	-------------------T-------------
(intact wheel2)	--------------------U------------
(loose nuts the-hub)	---------------------V-----------
(not-inflated wheel1)	----------------------W----------
(not-inflated wheel2)	-----------------------X---------
(not-on-ground the-hub)	------------------------Y--------
(on wheel1 the-hub)	-------------------------Z-------

(on wheel2 the-hub)	----------A------
(on-ground the-hub)	----------B-----
(open boot)	----------C----
(tight nuts the-hub)	----------D---
(unfastened the-hub)	----------E--
(unlocked boot)	----------F-
annoyed	----------G

(close boot)
Must: ----------C---- (open boot)
Then: A----------c---- not (open boot); (closed boot)

(do-up nuts the-hub)
Must: ----E----J----------Y-----E-- (have wrench); (unfastened the-hub); (not-on-g
Then: -B--e----J----------V--Y-----e-- (loose nuts the-hub); (fastened the-hub); not (u

(fetch jack boot)
Must: ----------K----------C---- (in jack boot); (open boot)
Then: ---D------k----------C---- not (in jack boot); (have jack)

(fetch nuts boot)
Must: ----------L----------C---- (in nuts boot); (open boot)
Then: ----E------l----------C---- not (in nuts boot); (have nuts)

(fetch pump boot)
Must: ----------M----------C---- (in pump boot); (open boot)
Then: -----F------m----------C---- not (in pump boot); (have pump)

(fetch the-hub boot)
Must: ----------N----------C---- (in the-hub boot); (open boot)
Then: ------G------n----------C---- not (in the-hub boot); (have the-hub)

(fetch wheel1 boot)
Must: ----------O----------C---- (in wheel1 boot); (open boot)
Then: -------H------o----------C---- not (in wheel1 boot); (have wheel1)

(fetch wheel2 boot)
Must: ----------P----------C---- (in wheel2 boot); (open boot)
Then: --------I------p----------C---- not (in wheel2 boot); (have wheel2)

(fetch wrench boot)

Must: ----------------Q-----------C---- (in wrench boot); (open boot)
Then: ---------J------q-----------C---- not (in wrench boot); (have wrench)

(inflate wheel1)
Must: -----F-------------T--W---------- (have pump); (not-inflated wheel1); (intact whe
Then: -----F-----------R-T--w---------- not (not-inflated wheel1); (inflated wheel1)

(inflate wheel2)
Must: -----F--------------U--X--------- (have pump); (not-inflated wheel2); (intact whe
Then: -----F------------S-U--x--------- not (not-inflated wheel2); (inflated wheel2)

(jack-down the-hub)
Must: ------------------------Y-------- (not-on-ground the-hub)
Then: ---D--------------------y--B----- not (not-on-ground the-hub); (on-ground the-h

(jack-up the-hub)
Must: ---D----------------------B----- (on-ground the-hub); (have jack)
Then: ---d--------------------Y--b----- (not-on-ground the-hub); not (on-ground the-h

(loosen nuts the-hub)
Must: ---------J-----------------B-D--- (have wrench); (tight nuts the-hub); (on-ground
Then: ---------J-----------V-----B-d--- (loose nuts the-hub); not (tight nuts the-hub)

(open boot)
Must: A----------------------------F- (unlocked boot); (closed boot)
Then: a--------------------------C--F- not (closed boot); (open boot)

(put-away jack boot)
Must: ---D----------------------C---- (have jack); (open boot)
Then: ---d------K----------------C---- (in jack boot); not (have jack)

(put-away nuts boot)
Must: ----E----------------------C---- (have nuts); (open boot)
Then: ----e------L---------------C---- (in nuts boot); not (have nuts)

(put-away pump boot)
Must: -----F--------------------C---- (have pump); (open boot)
Then: -----f------M--------------C---- (in pump boot); not (have pump)

(put-away the-hub boot)
Must: ------G--------------------C---- (have the-hub); (open boot)
Then: ------g------N-------------C---- (in the-hub boot); not (have the-hub)

(put-away wheel1 boot)
Must: -------H--------------------C---- (have wheel1); (open boot)
Then: -------h------O-------------C---- (in wheel1 boot); not (have wheel1)

(put-away wheel2 boot)
Must: --------I--------------------C---- (have wheel2); (open boot)
Then: --------i------P------------C---- (in wheel2 boot); not (have wheel2)

(put-away wrench boot)
Must: ---------J--------------------C---- (have wrench); (open boot)
Then: ---------j------Q-----------C---- (in wrench boot); not (have wrench)

(put-on-wheel wheel1 the-hub)
Must: --C----H----------------Y-----E-- (have wheel1); (free the-hub); (unfastened the-
Then: --c----h----------------YZ----E-- (on wheel1 the-hub); not (free the-hub); not (ha

(put-on-wheel wheel2 the-hub)
Must: --C-----I---------------Y-----E-- (have wheel2); (free the-hub); (unfastened the-
Then: --c-----i---------------Y-A---E-- (on wheel2 the-hub); not (free the-hub); not (ha

(remove-wheel wheel1 the-hub)
Must: ------------------------YZ----E-- (not-on-ground the-hub); (on wheel1 the-hub);
Then: --C----H----------------Yz----E-- (have wheel1); (free the-hub); not (on wheel1 th

(remove-wheel wheel2 the-hub)
Must: ------------------------Y-A---E-- (not-on-ground the-hub); (on wheel2 the-hub);
Then: --C-----I---------------Y-a---E-- (have wheel2); (free the-hub); not (on wheel2 th

(tighten nuts the-hub)
Must: ---------J-----------V-----B----- (have wrench); (loose nuts the-hub); (on-groun
Then: ---------J-----------v-----B-D--- (tight nuts the-hub); not (loose nuts the-hub)

(undo nuts the-hub)
Must: -B-------J-----------V--Y-------- (not-on-ground the-hub); (fastened the-hub); (h
Then: -b--E----J-----------v--Y-----E-- (have nuts); (unfastened the-hub); not (fastene

cuss
Must: --------------------------------
Then: --------------------------------g not annoyed

In [111]:
```
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 [112]:
```
# This doesn't work :(
# fixit_1_problem.solve()
# Should be something like:
# (open boot)
# (fetch jack boot) (fetch wrench boot) (fetch wheel2 boot)
# (loosen nuts the-hub) (inflate wheel2)
# (jack-up the-hub)
# (undo nuts the-hub)
# (remove-wheel wheel1 the-hub)
# (put-away wheel1 the-hub) (put-on-wheel wheel2 the-hub)
# (put-on nuts the-hub)
# (jack-down the-hub)
# (tighten nuts the-hub)
# (put-away jack boot) (put-away wrench)
# (close boot)
```