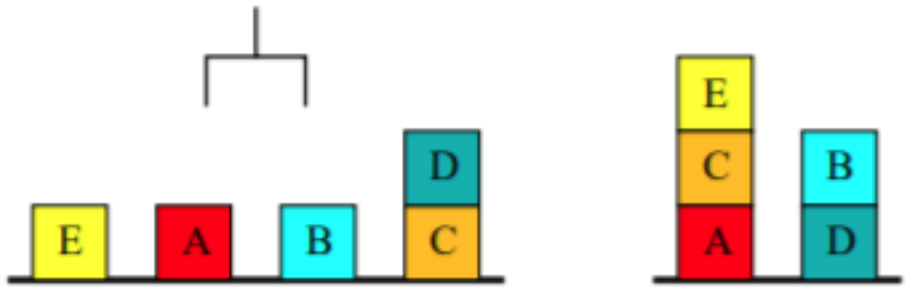
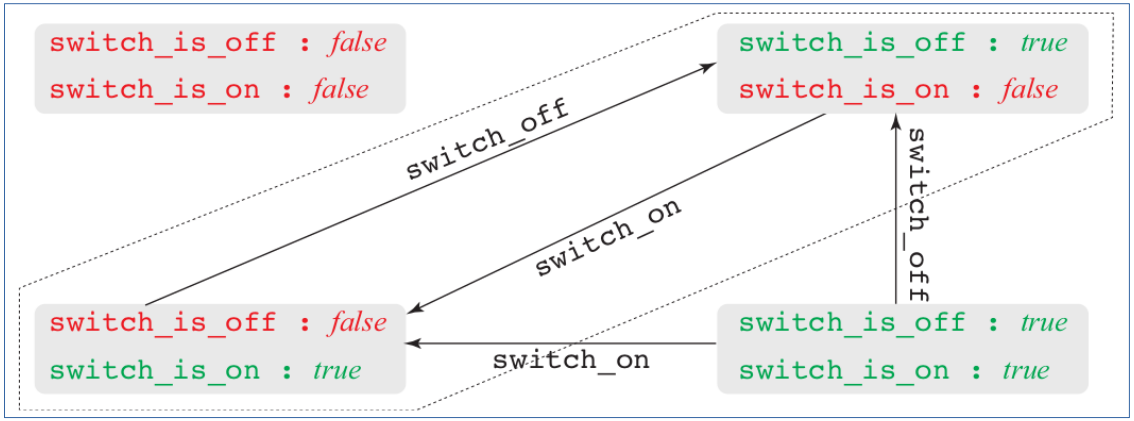


- Facts: $on(x, y)$, $onTable(x)$, $clear(x)$, $holding(x)$, $armEmpty()$.
- Initial state: $\{onTable(E)$, $clear(E)$, $\ldots$, $onTable(C)$, $on(D, C)$, $clear(D)$, $armEmpty()\}$.
- Goal: $\{on(E, C)$, $on(C, A)$, $on(B, D)\}$.
- Actions: $stack(x, y)$, $unstack(x, y)$, $putdown(x)$, $pickup(x)$.
- $stack(x, y)$? $pre : \{holding(x), clear(y)\}$
  $add : \{on(x, y), armEmpty()\}$
  $del : \{holding(x), clear(y)\}$.

Domain.pddl

```
(define (domain switch)
    (:requirements :strips)
    (:predicates    (switch_is_on)
                    (switch_is_off)
    )
    (:action switch_on
        :precondition (switch_is_off)
        :effect (and    (switch_is_on)
                        (not (switch_is_off))
                )
    )
    (:action switch_off
        :precondition (switch_is_on)
        :effect (and    (switch_is_off)
                        (not (switch_is_on))
                )
    )
)
```



Problem.pddl

```
(define (problem turn_it_off)
    (:domain switch)
    (:init
        (switch_is_on)
    )
    (:goal
        (switch_is_off)
    )
)
```

Plan

(switch_off)

Domain.pddl

```
(define (domain BLOCKS)
  (:requirements :strips)
  (:predicates (on ?x ?y)
               (ontable ?x)
               (clear ?x)
               (handempty)
               (holding ?x)
               )

  (:action pick-up
         :parameters (?x)
         :precondition (and (clear ?x)
            (ontable ?x) (handempty))
         :effect
         (and (not (ontable ?x))
           (not (clear ?x))
           (not (handempty))
           (holding ?x)))

  (:action put-down
         :parameters (?x)
         :precondition (holding ?x)
         :effect
         (and (not (holding ?x))
           (clear ?x)
           (handempty)
           (ontable ?x)))

  (:action stack
         :parameters (?x ?y)
         :precondition (and (holding ?x)
            (clear ?y))
         :effect
         (and (not (holding ?x))
           (not (clear ?y))
           (clear ?x)
           (handempty)
           (on ?x ?y)))
  (:action unstack
         :parameters (?x ?y)
         :precondition (and (on ?x ?y)
            (clear ?x) (handempty))
         :effect
         (and (holding ?x)
           (clear ?y)
           (not (clear ?x))
           (not (handempty))
           (not (on ?x ?y)))))
```

Problem.pddl

```
(define (problem BLOCKS-5-0)
    (:domain BLOCKS)
    (:objects B E A C D )
    (:init
        (CLEAR D)
        (CLEAR C)
        (ONTABLE D)
        (ONTABLE A)
        (ON C E)
        (ON E B)
        (ON B A)
        (HANDEMPTY)
    )
    (:goal
        (AND
            (ON A E)
            (ON E B)
            (ON B D)
            (ON D C)
        )
    )
)
```

Plan >>

- (unstack c e)
- (put-down c)
- (pick-up d)
- (stack d c)
- (unstack e b)
- (put-down e)
- (unstack b a)
- (stack b d)
- (pick-up e)
- (stack e b)
- (pick-up a)
- (stack a e)

```
(:action unstack
  :parameters (c e)
  :precondition
    (and
      (on c e)
      (clear c)
      (handempty)
    )
  :effect
    (and
      (holding c)
      (clear e)
      (not
        (clear c)
      )
      (not
        (handempty)
      )
      (not
        (on c e)
      )
    )
)
```

Domain.pddl (3 operators)

```
(define (domain blocks_world)
  (:requirements :strips)
  (:predicates (on-table ?x) (on ?x ?y) (clear ?x))

  (:action Move_from_Stack_to_Table
    :parameters (?obj ?src)
    :precondition (and (clear ?obj) (on ?obj ?src))
    :effect (and (clear ?src) (on-table ?obj) (not (on ?obj ?src))))

  (:action Move_from_Stack_to_Stack
    :parameters (?obj ?src ?dst)
    :precondition (and (clear ?obj) (clear ?dst) (on ?obj ?src))
    :effect (and (clear ?src) (on ?obj ?dst) (not (clear ?dst)) (not (on ?obj ?src))))

  (:action Move_from_Table_to_Stack
    :parameters (?obj ?dst)
    :precondition (and (clear ?obj) (clear ?dst) (on-table ?obj))
    :effect (and (on ?obj ?dst) (not (clear ?dst)) (not (on-table ?obj))))
)
```

Problem.pddl

```
(define (problem bw-12step)
    (:domain blocks_world)
    (:objects A B C D E F G)
    (:init  (on-table A) (clear A) (on-table B) (clear B) (on-table G) (on F G)
            (on E F) (on D E) (on C D) (clear C))
    (:goal (and (on B C) (on-table A) (on F A) (on C D)))
)
```

Domain.pddl (2 operators)

```
(define (domain simple-blocks)
  (:requirements :strips :equality)
  (:predicates (on ?x ?y)
               (clear ?x))
  (:constants Table)

  (:action Move_to_table
         :parameters (?obj ?src)
         :precondition (and (on ?obj ?src) (clear ?obj)
                            (not (= ?obj Table)) (not (= ?src Table)))
         :effect (and (on ?obj Table) (clear ?src) (not (on ?obj ?src))))
  (:action Move_to_Stack
         :parameters (?obj ?src ?dst)
         :precondition (and (on ?obj ?src) (clear ?obj) (clear ?dst)
                            (not (= ?obj Table)) (not (= ?dst Table)) (not (= ?obj ?dst)))
         :effect (and (on ?obj ?dst) (clear ?src)
                      (not (on ?obj ?src)) (not (clear ?dst)))))
```

Problem.pddl

```
(define (problem simple-block1)
    (:domain simple-blocks)
  (:objects A B C)
  (:goal (and (on a b) (on b c)))
  (:init (on a table) (on c a) (clear c) (on b table) (clear b)))
```

Domain.pddl (1 operator)

```
(define (domain blocks-world-domain)
  (:requirements :strips :equality :conditional-effects)
  (:constants Table)
  (:predicates (on ?x ?y)
               (clear ?x)
               (block ?b)
               )

  (:action move
         :parameters (?obj ?src ?dst)
         :precondition (and (on ?obj ?src) (clear ?obj) (clear ?dst)
                  (not (= ?dst ?src)) (not (= ?obj ?src))
                  (not (= ?obj ?dst)) (not (= ?obj Table)))
         :effect
         (and (on ?obj ?dst) (not (on ?obj ?src))
           (when (not (= ?src Table)) (clear ?src))
           (when (not (= ?dst Table)) (not (clear ?dst))))))
```

Problem.pddl

```
(define (problem sussman-anomaly)
    (:domain blocks-world-domain)
  (:objects A B C)
  (:init (block A) (block B) (block C) (block Table)
     (on C A) (on A Table) (on B Table) (clear C) (clear B) (clear Table))
  (:goal (and (on B C) (on A B))))
```

Domain.pddl (1 operator)

```
(define (domain mcd-blocksworld)
  (:requirements :adl :universal-preconditions :disjunctive-preconditions)
  (:constants Table)
  (:predicates (on ?x ?y)
               (clear ?x)
               (block ?b)
               (above ?x ?y))
  (:action puton
           :parameters (?obj ?dst ?src)
           :precondition (and (not (= ?obj ?dst)) (not (= ?obj table)) (not (= ?src ?dst))
                              (on ?obj ?src)
                              (or (= ?obj Table)
                                   (forall (?b) (imply (block ?b) (not (on ?b ?obj)))))
                              (or (= ?dst Table)
                                   (forall (?b) (imply (block ?b) (not (on ?b ?dst))))))
           :effect
           (and (on ?obj ?dst) (not (on ?obj ?src))
             (forall (?c)
                  (when (or (= ?dst ?c) (above ?dst ?c))
                     (above ?obj ?c)))
             (forall (?e)
                  (when (and (above ?obj ?e) (not (= ?dst ?e))
                             (not (above ?dst ?e)))
                     (not (above ?obj ?e)))))))
```

Problem.pddl

```
(define (problem mcd-sussman-anomaly)
    (:domain mcd-blocksworld)
  (:objects A B C)
  (:init (block A) (block B) (block C) (block Table)
     (on c a) (on b table) (on a table))
  (:goal (and (on b c) (on a b))))
```

```
(define (domain knights-tour)
  (:requirements :negative-preconditions)

  (:predicates
    (at ?col ?row)
    (visited ?col ?row)
    (diff_by_one ?x ?y)
    (diff_by_two ?x ?y)
   )

  (:action move_2col_1row
   :parameters (?from_col ?from_row ?to_col ?to_row)
   :precondition (and (at ?from_col ?from_row)
              (diff_by_two ?from_col ?to_col) ; col +/- 2
              (diff_by_one ?from_row ?to_row) ; row +/- 1
              (not (visited ?to_col ?to_row)))
   :effect (and (not (at ?from_col ?from_row))
       (at ?to_col ?to_row)
       (visited ?to_col ?to_row))
   )

  (:action move_2row_1col
   :parameters (?from_col ?from_row ?to_col ?to_row)
   :precondition (and (at ?from_col ?from_row)
              (diff_by_two ?from_row ?to_row) ; row +/- 2
              (diff_by_one ?from_col ?to_col) ; col +/- 1
              (not (visited ?to_col ?to_row)))
   :effect (and (not (at ?from_col ?from_row))
       (at ?to_col ?to_row)
       (visited ?to_col ?to_row))
   )
)
```

♞	14	9	20	3
24	19	2	15	10
13	8	23	4	21
18	25	6	11	16
7	12	17	22	5

```
(define (problem knights-tour-problem-5x5)
  (:domain knights-tour)

  ; Define a set of "numbers" 1..8:
  (:objects n1 n2 n3 n4 n5)

  (:init
    ; Initial position of the Knight piece
    ; (upper left corner):
    (at n1 n5)
    (visited n1 n5)

    ; Here, we have to
    ; of the static pre
    (diff_by_one n1 n2)
    (diff_by_one n2 n1)
    (diff_by_one n2 n3)
    (diff_by_one n3 n2)
    (diff_by_one n3 n4)
    (diff_by_one n4 n3)
    (diff_by_one n4 n5)
    (diff_by_one n5 n4)

    (diff_by_two n1 n3)
    (diff_by_two n3 n1)
    (diff_by_two n2 n4)
    (diff_by_two n4 n2)
    (diff_by_two n3 n5)
    (diff_by_two n5 n3)
   )
```

```
  (:goal (and (visited n1 n1)
              (visited n1 n2)
              (visited n1 n3)
              (visited n1 n4)
              (visited n1 n5)
              (visited n2 n1)
              (visited n2 n2)
              (visited n2 n3)
              (visited n2 n4)
              (visited n2 n5)
              (visited n3 n1)
              (visited n3 n2)
              (visited n3 n3)
              (visited n3 n4)
              (visited n3 n5)
              (visited n4 n1)
              (visited n4 n2)
              (visited n4 n3)
              (visited n4 n4)
              (visited n4 n5)
              (visited n5 n1)
              (visited n5 n2)
              (visited n5 n3)
              (visited n5 n4)
              (visited n5 n5)))
)
```