Put Down A

```
(setf (current-problem)
      (create-problem
       (name p1)
       (objects
        (A B C object))
       (state
        (and (holding A)
             (on-table B)
             (clear B)
             (on-table C)
             (clear C)))
       (igoal (arm-empty))))
```

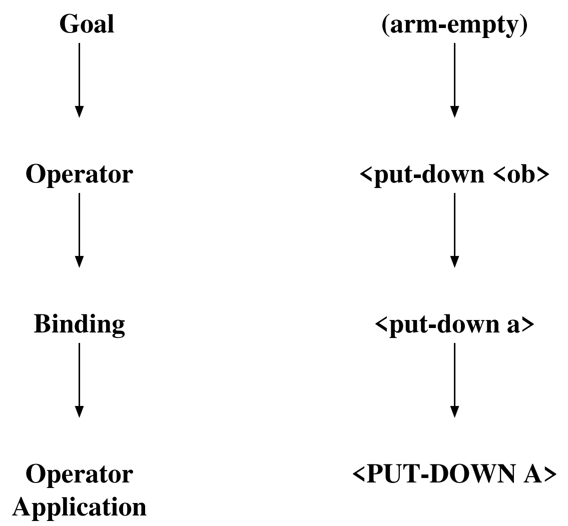
Put Down A

```
>(run)
Creating objects (A B C) of type OBJECT

  2 n2 (done)
  4 n4 <*finish*>
  5   n5 (arm-empty)
  7   n7 <put-down a>
  7   n8 <PUT-DOWN A>
Achieved top-level goals.

Solution:
        <put-down a>
```

Put Down A


Goal
(arm-empty)
Operator
<put-down <ob>
Binding
<put-down a>
Operator Application
<PUT-DOWN A>

Put Down A

State

(holding A)
(on-table B)
(clear B)
(on-table C)
(clear C)

Operators with *(arm-empty)*
PUT-DOWN
STACK

Bindings

<PUT-DOWN A >

Preconditions

(holding A)

Apply Operator

(on-table B), (clear B),(on-table C),(clear C)
(clear A),*(arm-empty)*,*(on-table A)*

Stack A on B

```
(setf (current-problem)
      (create-problem
       (name p1)
       (objects
        (A B C object))
       (state
        (and (holding A)
             (on-table B)
             (clear B)
             (on-table C)
             (clear C)))
       (igoal (on A B))))
```

```
>(run)
Creating objects (A B C) of type OBJECT

  2 n2 (done)
  4 n4 <*finish*>
  5   n5 (on a b)
  7   n7 <stack a b>
  7   n8 <STACK A B>
Achieved top-level goals.

Solution:
        <stack a b>
```

```
(setf (current-problem)
      (create-problem
       (name p3)
       (objects
        (blockA blockB object))
       (state
        (and (clear blockA)
             (clear blockB)
             (on-table blockA)
             (on-table blockB)))
       (igoal (on blockA blockB))))
```

```
 2 n2 (done)
 4 n4 <*finish*>
 5   n5 (on blocka blockb)
 7   n7 <stack blocka blockb>
 8     n8 (holding blocka)
10     n10 <pick-up blocka>
11       n11 (arm-empty)
13       n13 <put-down blockb> [1]
14         n14 (holding blockb)
15         n15 pick-up ...goal loop with node 11

12       n12 put-down
13       n16 <put-down blocka> ...goal loop with node 8

11       n11 (arm-empty)
13       n18 <stack blocka blockb> [1] ...goal loop with node 8
13       n19 <stack blockb blocka>
14         n20 (holding blockb)
15         n21 pick-up ...goal loop with node 11
```

Set the goal to be
n5 (on blocka blockb)

Found Operator to achieve goal (now with bindings)
n7 <stack blocka blockb>

Set Sub-Goal of
n8 (holding blocka)

Choose Operator for Sub-Goal
n10 <pick-up blocka>

In order to pick up something:
n11 (arm-empty)

But will now run into a problem:
It was not explicitely stated this time that the arm was empty

Inference Rules

```
(Inference-Rule Infer-ArmEmpty
 (params)
 (preconds ()
      (~ (exists ((<ob> Object)) (holding <ob>))))
 (effects ()
      ((add (arm-empty)))))
```

>

After Inference Rule

```
>(run)
Creating objects (BLOCKA BLOCKB) of type OBJECT

  2 n2 (done)
  4 n4 <*finish*>
  5   n5 (on blocka blockb)
  7   n7 <stack blocka blockb>
  8     n8 (holding blocka)
 10     n10 <pick-up blocka>
 11       n11 (arm-empty)
 13       n13 <infer-armempty>
 14     n14 <PICK-UP BLOCKA>
 14   n15 <STACK BLOCKA BLOCKB>
Achieved top-level goals.

Solution:
        <infer-armempty>
        <pick-up blocka>
        <stack blocka blockb>
```

Stacking Three Objects

```
(setf (current-problem)
      (create-problem
       (name aonbonc1)
       (objects
        (objects-are blockA blockB blockC Object))
       (state
        (and
         (arm-empty)
         (clear blockA)
         (clear blockB)
         (clear blockC)
         (on-table blockA)
         (on-table blockB)
         (on-table blockC)))
       (igoal (and (on blockB blockC)
                   (on blockA blockB)
                   ))))
```

Stacking Three Objects

```
  2 n2 (done)
  4 n4 <*finish*>
  5   n5 (on blockb blockc) [1]
  7   n7 <stack blockb blockc>
  8     n8 (holding blockb) [1]
 10     n10 <pick-up blockb>
 11     n11 <PICK-UP BLOCKB> [1]
 12   n12 <STACK BLOCKB BLOCKC> [1]
 13   n13 (on blocka blockb)
 15   n15 <stack blocka blockb>
 16     n16 (holding blocka)
 18     n18 <pick-up blocka>
 19     n19 <PICK-UP BLOCKA>
 19   n20 <STACK BLOCKA BLOCKB>
Achieved top-level goals.

Solution:
        <pick-up blockb>
        <stack blockb blockc>
        <pick-up blocka>
        <stack blocka blockb>
```

Stacking 3 Blocks: Rearranged

```
(setf (current-problem)
      (create-problem
       (name aonbonc1)
       (objects
        (objects-are blockA blockB blockC Object))
       (state
        (and
         (arm-empty)
         (clear blockA)
         (clear blockB)
         (clear blockC)
         (on-table blockA)
         (on-table blockB)
         (on-table blockC)))
       (igoal (and (on blockA blockB)
                   (on blockB blockC)))))
```

Stacking 3 Blocks: Rearranged

```
  2 n2 (done)
  4 n4 <*finish*>
  5   n5 (on blocka blockb) [1]
  7   n7 <stack blocka blockb>
  8     n8 (holding blocka) [1]
 10     n10 <pick-up blocka>
 11     n11 <PICK-UP BLOCKA> [1]
 12   n12 <STACK BLOCKA BLOCKB> [1]
 13   n13 (on blockb blockc)
 15   n15 <stack blockb blockc>
    .
    .
    .
 16     n136 <pick-up blockb>
 17     n137 <PICK-UP BLOCKB>
 18   n138 <STACK BLOCKB BLOCKC> [2]
 19     n139 <PICK-UP BLOCKA>
 19   n140 <STACK BLOCKA BLOCKB>
Achieved top-level goals.

Solution:
        <pick-up blockb>
        <stack blockb blockc>
        <pick-up blocka>
        <stack blocka blockb>
```

Introduction of a Control Rule

```
(Control-rule Prefer-right-goal
              (IF (and (candidate-goal (on <x> <y>))
                       (candidate-goal (on <y> <z>))
                       (true-in-state (clear <x>))))
              (THEN prefer goal (on <y> <z>) (on <x> <y>)))
```

With Control Rule

```
  2 n2 (done)
  4 n4 <*finish*>
Firing pref rule PREFER-RIGHT-GOAL for (#<ON BLOCKB BLOCKC>) over #<ON BLOCKA BLOCKB>
  5   n5 (on blockb blockc) [1]
  7   n7 <stack blockb blockc>
  8     n8 (holding blockb) [1]
Firing select op rule SELECT-PICKUP-FOR-HOLDING-IF-ON-TABLE to get (#<OP: PICK-UP>)
 10     n10 <pick-up blockb>
 11     n11 <PICK-UP BLOCKB> [1]
 12   n12 <STACK BLOCKB BLOCKC> [1]
 13   n13 (on blocka blockb)
 15   n15 <stack blocka blockb>
 16     n16 (holding blocka)
Firing select op rule SELECT-PICKUP-FOR-HOLDING-IF-ON-TABLE to get (#<OP: PICK-UP>)
 18     n18 <pick-up blocka>
 19     n19 <PICK-UP BLOCKA>
 19   n20 <STACK BLOCKA BLOCKB>
Achieved top-level goals.
Solution:
        <pick-up blockb>
        <stack blockb blockc>
        <pick-up blocka>
        <stack blocka blockb>
```

Trace Node 4

```
>(pshow 'node 4)

Structure of type BINDING-NODE
Byte:[Slot Type]Slot Name   :Slot Value
   0:[FIXNUM] NAME                :4
   4:[FIXNUM] ABS-LEVEL           :0
   8:ABS-PARENT          :NIL
  12:ABS-CHILDREN        :NIL
  16:OPEN-NODE-LINK      :#<Link between: 7 & DUMMY>
  20:PARENT              :#<OPERATOR-NODE 3 #<OP: *FINISH*>>
  24:CHILDREN            :(#<GOAL-NODE 5 #<ON BLOCKB BLOCKC>>)
  28:PLIST               :(PRODIGY4::GOAL-NODES
                               (#<GOAL-NODE 13 #<ON BLOCKA BLOCKB>>
                                #<GOAL-NODE 5 #<ON BLOCKB BLOCKC>>)
                               :CONSP-NUM 0)
  32:PENDING-GOALS       :(#<ON BLOCKA BLOCKB>)
  36:GOALS-LEFT          :(#<ON BLOCKA BLOCKB>)
  40:APPLICABLE-OPS-LEFT :NIL
  44:APPLIED             :NIL
  48:INSTANTIATED-OP     :#<*FINISH*>
  52:JUSTIFICATION-TABLE :NIL
  56:INSTANTIATED-PRECONDS:(AND #<ON BLOCKA BLOCKB>
                                 #<ON BLOCKB BLOCKC>)
  60:DISJUNCTION-PATH    :(NIL NIL)
```

Node 5

```
Structure of type GOAL-NODE
Byte:[Slot Type]Slot Name   :Slot Value
   0:[FIXNUM] NAME                :5
   4:[FIXNUM] ABS-LEVEL           :0
   8:ABS-PARENT          :NIL
  12:ABS-CHILDREN        :NIL
  16:OPEN-NODE-LINK      :#<Link between: DUMMY & 4>
  20:PARENT              :#<BINDING-NODE 4 #<*FINISH*>>
  24:CHILDREN            :(#<OPERATOR-NODE 6 #<OP: STACK>>)
  28:PLIST               :(:TERMINATION-REASON :EXHAUSTED
                              :UNDER-DELETION NIL :CONSP-NUM 0)
  32:GOAL                :#<ON BLOCKB BLOCKC>
  36:GOALS-LEFT          :(#<ON BLOCKA BLOCKB>)
  40:APPLICABLE-OPS-LEFT :NIL
  44:INTRODUCING-OPERATORS:(#<BINDING-NODE 4 #<*FINISH*>>)
  48:OPS-LEFT            :NIL
```

```
Structure of type OPERATOR-NODE
Byte:[Slot Type]Slot Name   :Slot Value
   0:[FIXNUM] NAME                 :6
   4:[FIXNUM] ABS-LEVEL            :0
   8:ABS-PARENT         :NIL
  12:ABS-CHILDREN       :NIL
  16:OPEN-NODE-LINK     :#<Link between: DUMMY & 4>
  20:PARENT             :#<GOAL-NODE 5 #<ON BLOCKB BLOCKC>>
  24:CHILDREN           :(#<BINDING-NODE 7 #<STACK [<OB> BLOCKB] [<UNDEROB> BLOCKC]>>)
  28:PLIST              :(:TERMINATION-REASON :EXHAUSTED
                             :UNDER-DELETION NIL :REGRESSED-PRECONDS
                             ((AND #<CLEAR BLOCKC>
                                   #<HOLDING BLOCKB>))
                             :CONSP-NUM 0)
  32:OPERATOR           :#<OP: STACK>
  36:OPS-LEFT           :NIL
  40:BINDINGS-LEFT      :NIL
```

```
>(pshow 'node 7)

Structure of type BINDING-NODE
Byte:[Slot Type]Slot Name   :Slot Value
   0:[FIXNUM] NAME               :7
   4:[FIXNUM] ABS-LEVEL          :0
   8:ABS-PARENT          :NIL
  12:ABS-CHILDREN        :NIL
  16:OPEN-NODE-LINK      :#<Link between: 10 & 4>
  20:PARENT              :#<OPERATOR-NODE 6 #<OP: STACK>>
  24:CHILDREN            :(#<GOAL-NODE 8 #<HOLDING BLOCKB>>)
  28:PLIST               :(PRODIGY4::GOAL-NODES
                               (#<GOAL-NODE 8 #<HOLDING BLOCKB>>)
                               :CONSP-NUM 0)
  32:PENDING-GOALS       :(#<HOLDING BLOCKB>
                           #<ON BLOCKA BLOCKB>)
  36:GOALS-LEFT          :(#<ON BLOCKA BLOCKB>)
  40:APPLICABLE-OPS-LEFT :NIL
  44:APPLIED             :NIL
  48:INSTANTIATED-OP     :#<STACK [<OB> BLOCKB] [<UNDEROB> BLOCKC]>
  52:JUSTIFICATION-TABLE :NIL
  56:INSTANTIATED-PRECONDS:(AND #<CLEAR BLOCKC>
                                 #<HOLDING BLOCKB>)
  60:DISJUNCTION-PATH    :(NIL NIL)
```

```
>(pshow 'node 8)

Structure of type GOAL-NODE
Byte:[Slot Type]Slot Name   :Slot Value
   0:[FIXNUM] NAME                :8
   4:[FIXNUM] ABS-LEVEL           :0
   8:ABS-PARENT          :NIL
  12:ABS-CHILDREN        :NIL
  16:OPEN-NODE-LINK      :#<Link between: DUMMY & 7>
  20:PARENT              :#<BINDING-NODE 7 #<STACK [<OB> BLOCKB] [<UNDEROB> BLOCKC]>>
  24:CHILDREN            :(#<OPERATOR-NODE 9 #<OP: PICK-UP>>)
  28:PLIST               :(:TERMINATION-REASON :EXHAUSTED
                               :UNDER-DELETION NIL :CONSP-NUM 0)
  32:GOAL                :#<HOLDING BLOCKB>
  36:GOALS-LEFT          :(#<ON BLOCKA BLOCKB>)
  40:APPLICABLE-OPS-LEFT :NIL
  44:INTRODUCING-OPERATORS:(#<BINDING-NODE 7 #<STACK [<OB> BLOCKB] [<UNDEROB> BLOCKC]>>)
  48:OPS-LEFT            :NIL
```

```
(setf (current-problem)
      (create-problem
       (name aonbonc1)
       (objects
        (objects-are blockA blockB blockC Object))
       (state
        (and
         (arm-empty)
         (clear blockA)
         (on blockA blockB)
         (on-table blockB)))
       (igoal (on blockB blockA))))
```

```
  2 n2 (done)
  4 n4 <*finish*>
  5   n5 (on blockb blocka)
  7   n7 <stack blockb blocka>
  8     n8 (holding blockb)
Firing select op rule SELECT-PICKUP-FOR-HOLDING-IF-ON-TABLE to get (#<OP: PICK-UP>)
 10     n10 <pick-up blockb>
 11       n11 (clear blockb)
Firing select op rule SELECT-OP-UNSTACK-FOR-CLEAR to get (#<OP: UNSTACK>)
Firing select binding rule SELECT-BINDINGS-UNSTACK-CLEAR at node 12
 13       n13 <unstack blocka blockb>
 14       n14 <UNSTACK BLOCKA BLOCKB>
 15       n15 (arm-empty) [1]
 17       n17 <infer-armempty>
 18     n18 (clear blocka)
Firing select op rule PUTDOWN-CLEAR-IF-HOLDING to get (#<OP: PUT-DOWN>)
 20     n20 <put-down blocka>
 21     n21 <PUT-DOWN BLOCKA>
 21     n21 <INFER-ARMEMPTY>
 22     n22 <PICK-UP BLOCKB>
 22   n23 <STACK BLOCKB BLOCKA>
Achieved top-level goals.
Solution:
        <unstack blocka blockb>
        <put-down blocka>
        <infer-armempty>
        <pick-up blockb>
        <stack blockb blocka>
```

```
(CONTROL-RULE SELECT-PICKUP-FOR-HOLDING-IF-ON-TABLE
              (IF (and (current-goal (holding <ob>))
                       (true-in-state (on-table <ob>))))
              (THEN select operator PICK-UP))
(CONTROL-RULE SELECT-BINDINGS-UNSTACK-CLEAR
              (IF (and (current-goal (clear <y>))
                       (current-ops (UNSTACK))
                       (true-in-state (on <x> <y>))))
              (THEN select bindings ((<ob> . <x>) (<underob> . <y>))))

(CONTROL-RULE PUTDOWN-CLEAR-IF-HOLDING
              (IF (and (current-goal (clear <ob>))
                       (true-in-state (holding <other-ob>))))
              (THEN select operator PUT-DOWN))
```

```
(setf (current-problem)
      (create-problem
       (name p5)
       (objects
        (objects-are blockA blockB blockC blockD Object))
       (state
        (and
         (arm-empty)
         (clear blockA)
         (clear blockB)
         (clear blockD)
         (on-table blockA)
         (on-table blockC)
         (on blockD blockC)
         (on-table blockB)))
       (igoal (and (on blockA blockB)
                   (on blockB blockC)))))
```

```
  2 n2 (done)
  4 n4 <*finish*>
  5   n5 (on blockb blockc) [1]
  7   n7 <stack blockb blockc>
  8     n8 (clear blockc) [2]
 10     n10 <unstack blockd blockc>
 11     n11 <UNSTACK BLOCKD BLOCKC> [2]
 12     n12 (holding blockb) [1]
 14     n14 <pick-up blockb>
 15       n15 (arm-empty) [1]
 17       n17 <put-down blockd>
 18       n18 <PUT-DOWN BLOCKD> [1]
 19     n19 <PICK-UP BLOCKB> [1]
 20   n20 <STACK BLOCKB BLOCKC> [1]
 21   n21 (on blocka blockb)
 23   n23 <stack blocka blockb>
 24     n24 (holding blocka)
 26     n26 <pick-up blocka>
 27     n27 <PICK-UP BLOCKA>
 27   n28 <STACK BLOCKA BLOCKB>
Achieved top-level goals.
Solution:
        <unstack blockd blockc>
        <put-down blockd>
        <pick-up blockb>
        <stack blockb blockc>
        <pick-up blocka>
        <stack blocka blockb>
```

Extended Strips

```
(create-problem-space 'extended-strips :current t)

(ptype-of OBJECT :top-type)
(ptype-of BOX    OBJECT)
(ptype-of KEY    OBJECT)
(ptype-of ROOM   :top-type)
(ptype-of DOOR   :top-type)
(ptype-of LOCX   :top-type)
(ptype-of LOCY   :top-type)
(ptype-of Robot  :top-type)
(ptype-of Status :top-type)

(pinstance-of ROBOT Robot)
(pinstance-of OPEN  Status)
(pinstance-of CLOSED Status)
```

```
(setf (getf (p4::problem-space-plist *current-problem-space*) :depth-bound) 50)
(output-level 3)

(setf (current-problem)
      (create-problem
       (name strips-prob)
       (objects
        (room1 room2 room3 room4 Room)
        (door12 door13 door34 door24 Door)
        (box1 box2 box3 Object)
        )
```

```
(state
 (and
  (dr-to-rm door12 room1) (dr-to-rm door12 room2)
  (dr-to-rm door13 room1) (dr-to-rm door13 room3)
  (dr-to-rm door34 room3) (dr-to-rm door34 room4)
  (dr-to-rm door24 room2) (dr-to-rm door24 room4)
  (connects door12 room1 room2) (connects door12 room2 room1)
  (connects door13 room1 room3) (connects door13 room3 room1)
  (connects door34 room3 room4) (connects door34 room4 room3)
  (connects door24 room2 room4) (connects door24 room4 room2)
  (dr-open door12) (dr-open door13)
  (dr-closed door34) (dr-closed door24)
  (unlocked door34) (locked door24)
  (arm-empty)
  (inroom robot room1)
  (inroom box1 room3)
  (inroom box2 room3)
  (inroom box3 room4)
  (pushable box1)
  (pushable box2)
  (pushable box3)))
(igoal
 (and (dr-closed door34)(next-to robot box3))
 )))
```

Pick up and Put down objects

```
(operator PICKUP-OBJ
 (params <o1>)
 (preconds
  ((<o1> OBJECT))
  (and (arm-empty)
       (next-to robot <o1>)
       (carriable <o1>)))
 (effects
  ((<other> (or OBJECT DOOR))
   (<other2> (or OBJECT DOOR)))
  ((del (arm-empty))
   (del (next-to <o1> <other>))
   (del (next-to <other2> <o1>))
   (add (holding <o1>)))))

(operator PUTDOWN
 (params <o2>)
 (preconds
  ((<o2> OBJECT))
  (holding <o2>))
 (effects ()
         ((del (holding <o2>))
          (add (next-to robot <o2>))
          (add (arm-empty)))))
```

Put next to

```
(operator PUTDOWN-NEXT-TO
 (params <o3> <other-ob> <o3-rm>)
 (preconds
  ((<o3> OBJECT)
   (<other-ob> OBJECT)
   (<o3-rm> ROOM))
  (and (holding <o3>)
       (inroom <other-ob> <o3-rm>)
       (inroom <o3> <o3-rm>)
       (next-to robot <other-ob>)))
 (effects
  ()
  ((del (holding <o3>))
   (add (next-to <o3> <other-ob>))
   (add (next-to robot <o3>))
   (add (next-to <other-ob> <o3>))
   (add (arm-empty)))))
```

Push Through Door

```
(operator PUSH-THRU-DR
 (params <b-x> <d-x> <r-x> <r-y>)
 (preconds
  ((<b-x> BOX) (<d-x> DOOR) (<r-x> ROOM) (<r-y> ROOM))
  (and (dr-to-rm <d-x> <r-x>)
       (dr-open <d-x>)
       (next-to <b-x> <d-x>)
       (next-to robot <b-x>)
       (pushable <b-x>)
       (connects <d-x> <r-x> <r-y>)
       (inroom <b-x> <r-x>)))
 (effects
  ((<other>  (and (or OBJECT DOOR) (diff <other> <b-x>)))
   (<other2> (or OBJECT DOOR))
   (<other3> (and (or OBJECT DOOR) (diff <other3> robot)))
   (<other4> OBJECT))
  ((del (next-to robot <other>))
   (del (next-to <b-x> <other2>))
   (del (next-to <other3> <b-x>))
   (if (holding <other4>)
       ((del (inroom <other4> <r-x>))
        (add (inroom <other4> <r-y>))))
   (del (inroom robot <r-x>))
   (del (inroom <b-x> <r-x>))
   (add (inroom robot <r-y>))
   (add (inroom <b-x> <r-y>)))))
```

Go Thru Door

```
(operator GO-THRU-DR
 (params <ddx> <rrx> <rry>)
 (preconds
  ((<ddx> DOOR)
   (<rrx> ROOM)
   (<rry> ROOM))
  (and (arm-empty)
       (dr-to-rm <ddx> <rrx>)
       (dr-open <ddx>)
       (next-to robot <ddx>)
       (connects <ddx> <rrx> <rry>)
       (inroom robot <rrx>)))
 (effects
  ((<other> OBJECT)
   (<other1> OBJECT))
  (
  ;; this should not be here.  GO-THRU-DR should not allow the robot
  ;; holding anything.
   (if (holding <other1>)
       ((del (inroom <other1> <rrx>))
        (add (inroom <other1> <rry>))))
   (del (next-to robot <other>))
   (del (inroom robot <rrx>))
   (add (inroom robot <rry>)))))
```

Go to door

```
(operator GOTO-DR
 (params <dx> <rx>)
 (preconds
  ((<dx> DOOR)
   (<rx> ROOM))
  (and (dr-to-rm <dx> <rx>)
       (inroom robot <rx>)))
 (effects
  ((<other> (or OBJECT DOOR)))
  ((del (next-to robot <other>))
   (add (next-to robot <dx>)))))
```

Open and Close Door

```
(Operator OPEN-DOOR
 (params <door>)
 (preconds
  ((<door> Door))
  (and (next-to robot <door>)
       (unlocked <door>)
       (dr-closed <door>)))
 (effects
  ()
  ((del (dr-closed <door>))
   (add (dr-open <door>)))))
```

```
(operator CLOSE-DOOR
 (params <door1>)
 (preconds
  ((<door1> DOOR))
  (and (next-to robot <door1>)
       (dr-open <door1>)))
 (effects
  ()
  ((del (dr-open <door1>))
   (add (dr-closed <door1>)))))
```

Lock and Unlock Door

```
(operator LOCK
 (params <door2> <k1> <rm-b>)
 (preconds
  ((<door2> DOOR)
   (<k1> KEY)
   (<rm-b> ROOM))
  (and (is-key <door2> <k1>)
       (holding <k1>)
       (dr-to-rm <door2> <rm-b>)
       (inroom <k1> <rm-b>)
       (next-to robot <door2>)
       (dr-closed <door2>)
       (unlocked <door2>)))
 (effects
  ()
  ((del (unlocked <door2>))
   (add (locked <door2>)))))
```

Lock and Unlock Door

```
(operator UNLOCK
 (params <door3> <k2> <rm-a>)
 (preconds
  ((<door3> DOOR)
   (<k2> KEY)
   (<rm-a> ROOM))
  (and (is-key <door3> <k2>)
       (holding <k2>)
       (dr-to-rm <door3> <rm-a>)
       (inroom <k2> <rm-a>)
       (inroom robot <rm-a>)
       (next-to robot <door3>)
       (locked <door3>)))
 (effects
  ()
  ((del (locked <door3>))
   (add (unlocked <door3>)))))
```

Doors are symmetric

```
(Inference-Rule
 Connects-is-symmetric
 (mode eager)
 (params <door> <room1> <room2>)
 (preconds
  ((<door> Door)
   (<room1> Room)
   (<room2> Room))
  (connects <door> <room1> <room2>))
 (effects () ((add (connects <door> <room2> <room1>)))))
```