1 Classical Planning Overview

1.1 Background

1.2 PDDL

1.3 FF Planner

1.4 Planning Domain Examples

1.4.1 Blocks World

Listing 1: Blocks World Domain Description in PDDL

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

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

  (:action put-down
             :parameters (?x - block)
             :precondition (holding ?x)
             :effect
             (and (not (holding ?x))
                  (clear ?x)
                  (handempty)
                  (ontable ?x)))
  (:action stack
             :parameters (?x - block ?y - block)
             :precondition (and (holding ?x) (clear ?y))
             :effect
             (and (not (holding ?x))
                  (not (clear ?y))
                  (clear ?x)
                  (handempty)
                  (on ?x ?y)))
  (:action unstack
             :parameters (?x - block ?y - block)
             :precondition (and (on ?x ?y) (clear ?x) (handempty))
             :effect
             (and (holding ?x)
                  (clear ?y)
                  (not (clear ?x))
                  (not (handempty))
                  (not (on ?x ?y)))))
```

1.4.2 Towers of Hanoi

Listing 2: Towers of Hanoi Domain Description in PDDL

```
;; towers of hanoi

(define (domain HANOI)
    (:requirements :typing)
    (:types disc peg)
    (:predicates
        (clear ?x)
        (on ?x - disc ?y)
        (larger ?d - disc ?e - disc)
    )

    (:action stack-d
        :parameters (?d - disc ?e - disc)
        :vars (?l)
        :precondition (and
            (on ?d ?l)
            (not (on ?d ?e))
            (not (= ?d ?e))
            (not (= ?e ?l))
            (larger ?e ?d)
            (clear ?d)
            (clear ?e)
        )
        :effect (and
            (not (on ?d ?l))
            (not (clear ?e))
            (on ?d ?e)
            (clear ?l)
        )
    )

    (:action stack-p
        :parameters (?d - disc ?p - peg)
        :vars (?l)
        :precondition (and
            (on ?d ?l)
            (clear ?p)
            (clear ?d)
            (not (= ?p ?l))
        )
        :effect (and
            (not (clear ?p))
            (not (on ?d ?l))
            (on ?d ?p)
            (clear ?l)
        )
    )
)
```

1.4.3 Lin's Briefcase

Listing 3: Lin's Briefcase Domain Description in PDDL

```
;; briefcase domain

(define (domain BRIEFCASE)
    (:requirements :typing)
```

```
    (:types latch)
    (:predicates
        (open)
        (latched ?l - latch)
    )

    (:action flip-open
        :parameters (?l - latch)
        :precondition (latched ?l)
        :effect (not (latched ?l))
    )

    (:action flip-closed
        :parameters (?l - latch)
        :precondition (not (latched ?l))
        :effect (latched ?l)
    )

    (:action open
        :parameters ()
        :precondition (and
            (forall (?l - latch)
                (not (latched ?l))
            )
            (not (open))
        )
        :effect (open)
    )
)
```

1.4.4 Electrical Circuit

Listing 4: Electrical Circuit Domain Description in PDDL

```
;; circuit domain

(define (domain CIRCUIT)
    (:requirements :typing :conditional-effects)
    (:types wire gate level)
    (:predicates
        (wire-high ?w - wire)
        (gate-active ?g - gate)
        (gate-level ?g - gate ?l - level)
        (and-gate ?g - gate)
        (or-gate ?g - gate)
        (inv-gate ?g - gate)
        (input-to ?w - wire ?g - gate)
        (output-from ?w - wire ?g - gate)
    )

    (:action activate-wire
        :parameters (?w - wire)
        :vars (?g2 - gate)
        :precondition (and
            (not (wire-high ?w))
            (input-to ?w ?g2)
            (forall (?g - gate)
                (not (output-from ?w ?g))
            )
        )
        :effect (and
```

```
            (wire-high ?w)
        )
    )

    (:action deactivate-wire
        :parameters (?w - wire)
        :vars (?g2 - gate)
        :precondition (and
            (wire-high ?w)
            (input-to ?w ?g2)
            (forall (?g - gate)
                (not (output-from ?w ?g))
            )
        )
        :effect (and
            (not (wire-high ?w))
        )
    )
    (:action activate-and-gate
        :parameters (?g - gate)
        :vars (?w1 - wire ?w2 - wire ?w3 - wire)
        :precondition (and
            (and-gate ?g)
            (not (gate-active ?g))
            (input-to ?w1 ?g)
            (input-to ?w2 ?g)
            (output-from ?w3 ?g)
            (wire-high ?w1)
            (wire-high ?w2)
            (not (= ?w1 ?w2))
            (not (= ?w1 ?w3))
            (not (= ?w2 ?w3))
        )
        :effect (and
            (wire-high ?w3)
            (gate-active ?g)
        )
    )

    (:action activate-inv-gate
        :parameters (?g - gate)
        :vars (?w1 - wire ?w2 - wire)
        :precondition (and
            (inv-gate ?g)
            (not (gate-active ?g))
            (input-to ?w1 ?g)
            (output-from ?w2 ?g)
            (not (wire-high ?w1))
        )
        :effect (and
            (wire-high ?w2)
            (gate-active ?g)
        )
    )

    (:action activate-or-gate
        :parameters (?g - gate)
        :vars (?w1 - wire ?w2 - wire ?w3 - wire)
        :precondition (and
            (or-gate ?g)
            (not (gate-active ?g))
```

```
            (input-to ?w1 ?g)
            (input-to ?w2 ?g)
            (output-from ?w3 ?g)
            (or (wire-high ?w1) (wire-high ?w2))
        )
        :effect (and
            (wire-high ?w3)
            (gate-active ?g)
        )
    )
)
```