```
;;;; Rules of Inference for Logic P-ND

(define uncontrolled-rules
  '(
    (rule (?f (not (not ?a)))
          ;; Double Negation Elimination
          (assert ?a (DNE ?f)))

    (rule (?f (and ?a ?b))
          ;; Simplification
          (assert ?a (-&:L ?f))
          (assert ?b (-&:R ?f)))

    (rule (?f (<-> ?a ?b))
          ;; Biconditional->Conditional
          (assert (-> ?a ?b) (-IFF:L ?f))
          (assert (-> ?b ?a) (-IFF:R ?f)))

    (rule (?f1 (-> ?a ?b))
          (rule (?f2 (-> ?b ?a))
                (assert (<-> ?a ?b)
                        (+IFF ?f1 ?f2))))


    ;; DeMorgan's laws

    (rule (?f (not (or ?a ?b)))
          (assert (and (not ?a) (not ?b))
                  (DM:NV&N ?f)))

    (rule (?f (not (and ?a ?b)))
          (assert (or (not ?a) (not ?b))
                  (DM:N&VN ?f)))

    (rule (?f (and (not ?a) (not ?b)))
          (assert (not (or ?a ?b))
                  (DM:&NNV ?f)))

    (rule (?f (or (not ?a) (not ?b)))
          (assert (not (and ?a ?b))
                  (DM:VNN& ?f)))

    ))
```

```
(define mandatory-controlled-rules
  ;; Must be controlled, to avoid infinite loops
  '(
    (rule (?goal (show (not (not ?a))))
          ;; Double Negation Introduction
          (assert (show ?a) (BC:DNI ?goal))
          (rule (?f ?a)
                (assert (not (not ?a)) (DNI ?f))))

    (rule (?goal (show (and ?a ?b)))
          ;; Adjunction -- introducing a conjunction
          (assert (show ?a) (BC:+& ?goal))
          (rule (?f1 ?a)
                (assert (show ?b) (BC:+& ?goal ?f1))
                (rule (?f2 ?b)
                      (assert (and ?a ?b)
                              (+& ?f1 ?f2)))))

    (rule (?goal (show (or ?a ?b)))
          ;; Addition -- introducing a disjunction
          (assert (show ?a) (BC:+V ?goal))
          (assert (show ?b) (BC:+V ?goal))
          (rule (?f ?a)
                (assert (or ?a ?b) (+V:L ?f)))
          (rule (?f ?b)
                (assert (or ?a ?b) (+V:R ?f))))
    ))
```

```
(define controlled-rules
  '(
    (rule (?goal (show ?b))
          (rule (?i (-> ?a ?b))
                ;; Modus Ponens
                (assert (show ?a)
                        (BC:MP ?goal ?i))
                (rule (?f ?a)
                      (assert ?b
                              (MP ?i ?f))))
          (rule (?v (or ?a ?b))
                ;; Modus Tollendo Ponens (L)
                (assert (show (not ?a))
                        (BC:MTP:L ?goal ?v))
                (rule (?f (not ?a))
                      (assert ?b
                              (MTP:L ?v ?f))))
          (rule (?v (or ?b ?a))
                ;; Modus Tollendo Ponens (r)
                (assert (show (not ?a))
                        (BC:MTP:R ?goal ?v))
                (rule (?f (not ?a))
                      (assert ?b
                              (MTP:L ?v ?f)))))

    (rule (?goal (show (not ?a)))
          (rule (?i (-> ?a ?b))
                ;; Modus Tollens
                (assert (show (not ?b))
                        (BC:MT ?goal ?i))
                (rule (?f (not ?b))
                      (assert (not ?a)
                              (MT ?i ?f)))))
    ))
```

```
;;; Hypothetical introduction needs auxiliary premise,
;;; because we must not retract a premise if it was already
;;; supported independently of its introduction in the
;;; deduction theorem.

(define conditional-proof-rule
  '(
    (rule (?goal (show (-> ?a ?b)))

          ;; Conditional Proof (The Deduction Theorem, see
          ;; Suppes p.20) Note: the deduction theorem is not
          ;; valid if ?a has free variables that are
          ;; generalized in the proof of ?b (See Mendelson
          ;; p.60--61).  However, this is fixed by the P-ND
          ;; rules, using anonymous individuals.

          (assume (hypothesis ?a ?goal))
          (rule (?h (hypothesis ?a ?goal))
                ;; Hypothesis Introduction
                (assert ?a (HI ?h))
                (assert (show ?b) (BC:CP ?goal ?h))
                (rule (?c ?b)
                      (assert (-> ?a ?b) (CP ?h ?c))
                      ;;(retract (hypothesis ?a ?goal))
                      )))
    ))
```

```
(define contradiction-detection
  '(
    ;; Very expensive -- for every supported assertion,
    ;; makes an active rule looking for the negation of that
    ;; assertion.  Probably should use a hack similar to the
    ;; one in the constraint propagator.

    (rule (?f1 ?a)
          (rule (?f2 (not ?a))
                (assert contradiction (CD ?f1 ?f2))))

    (rule (?goal (show (not ?a)))
          ;; Reducto ad absurdum (Proof by Contradiction)

          (assume (hypothesis ?a ?goal))
          (rule (?h (hypothesis ?a ?goal))
                ;; Hypothesis Introduction
                (assert ?a (HI ?h))
                (rule (?c contradiction)
                      (assert (not ?a) (RAA ?h ?c))
                      ;;(retract ?a))))
    ))
```

```scheme
(define separation-of-cases
  '(
    ;; Separation of Cases

    (rule (?i1 (-> ?a ?b))
          (rule (?i2 (-> (not ?a) ?b))
                (assert ?b (SC:A ?i1 ?i2)))
          ;; Must ensure that ?c is not equal to ?a.
          (rule (?i3 (-> ?c ?b))
                (if (not (equal ?a ?c))
                    (assert (-> (or ?a ?c) ?b)
                            (SC:V ?i1 ?i3)))))
    ))
```

```
(define (install-rules rules s)
  (for-each (lambda (rule) (amord rule s)) rules))

(define (full-logic s)
  (for-each (lambda (rule-packet)
               (install-rules rule-packet s))
             (list uncontrolled-rules
                   mandatory-controlled-rules
                   controlled-rules
                   conditional-proof-rule
                   ;; contradiction-detection
                   ;; separation-of-cases
                   ))
  'whew!)
```