Reason-Schemas for Planning
Non-Linear Planner 43

(def-backwards-reason PROTOPLAN
  :conclusions "(plan-for plan goal)"
  :condition (interest-variable plan)
  :backwards-premises
  "(protoplan-for plan goal nil nil nil nil)"
  :defeasible? t
  :strength .99
  :variables goal plan)

(def-backwards-reason NULL-PLAN
  :conclusions "(protoplan-for plan goal goals nodes nodes-used links bad-link)"
  :condition (and (interest-variable plan) (not (conjunctionp goal))
    (temporally-projectible goal)
    (or (null bad-link) (not (eq (causal-link-goal bad-link) *start*))
      (not (equal goal (causal-link-goal bad-link))))
    (or nodes nodes-used (not (mem goal goals))))
  :backwards-premises
  "goal"
  "(define plan (null-plan goal))"
  :variables goal plan goals nodes nodes-used links bad-link)

(def-backwards-reason GOAL-REGRESSION
  :conclusions "(protoplan-for plan goal goals nodes nodes-used links bad-link)"
  :condition (and (interest-variable plan) (null nodes-used)
    (not (conjunctionp goal))
    (not (mem goal goals))
    (or (null bad-link) (eq (causal-link-goal bad-link) goal)
      (not (some #'(lambda (L) (mem L goal)) links))))
  :backwards-premises
  "(define new-goals (cons goal goals))"
  "((precondition & action) => goal)"
  (:condition (and (not (mem precondition goals))
    (temporally-projectible precondition)
    (not (some #'(lambda (c) (mem c goals)) (conjuncts precondition))))
    "(protoplan-for subplan precondition new-goals nodes nodes-used links bad-link)"
    "(define plan (extend-plan action goal subplan bad-link))"
    (:condition (not (null plan)))
  :variables precondition action plan subplan goals new-goals nodes
    nodes-used links bad-link)

(def-backwards-reason PROTOPLAN-FOR-GOAL
  :conclusions
    "(protoplan-for plan goal goals nil nil nil nil)"
  :condition (interest-variable plan)
  :forwards-premises
"(protoplan-for plan goal goals0 nil nil nil nil)"
(:condition (every #'(lambda (L) (not (mem (causal-link-goal L) goals))) (causal-links plan))
:variables plan goal goals goals0)

(def-backwards-reason SPLIT-CONJUNCTIVE-GOAL
 :conclusions
 "(protoplan-for plan& (goal1 & goal2) goals nodes nodes-used links bad-link)"
 :condition (and (interest-variable plan&) (temporally-projectible goal1)
                 (temporally-projectible goal2))
 :backwards-premises
 "(protoplan-for plan1 goal1 goals nodes nodes-used links bad-link)"
 "(protoplan-for plan2 goal2 goals nodes nodes-used links bad-link)"
 (:condition
  (not (some #'(lambda (L1) (some #'(lambda (L2) ...
  (:condition (not (null plan&)))
 :variables goal1 goal2 plan1 plan2 goals plan& nodes nodes-used links bad-link)

:: UNDERMINING CAUSAL-LINKS

(def-backwards-undercutter UNDERMINE-CAUSAL-LINKS
 :defeatee protoplan
 :backwards-premises
 "((define links (if (live-links? plan) (live-causal-links plan) (causal-links plan)))"
 "((plan-undermines-causal-links plan links))"
 :variables plan links)

(def-backwards-reason PLAN-UNDERMINES-FIRST-CAUSAL-LINK
 :conclusions "((plan-undermines-causal-links plan links))"
 :condition (car links)
 :backwards-premises
 "((define first-link (car links)))"
 "((plan-undermines-causal-link plan R node first-link))"
 :variables plan node links first-link R)

(def-backwards-reason PLAN-UNDERMINES-ANOTHER-CAUSAL-LINK
 :conclusions "((plan-undermines-causal-links plan links))"
 :condition (cdr links)
 :backwards-premises
 "((define rest-of-links (cdr links)))"
 "((plan-undermines-causal-links plan rest-of-links))"
 :variables plan links rest-of-links)

(def-backwards-reason PLAN-UNDERMINES-CAUSAL-LINK

2
SEARCHING FOR EMBELLISHED-PLANS

(def-backwards-reason EMBELLISHED-PROTOPLAN
  :conclusions "(embellished-plan-for plan plan+ -goal node1 node2 before not-between)"
  :condition (interest-variable plan)
  :backwards-premises
  "(embellished-protoplan-for plan plan+ -goal node1 node2 before not-between)"
  :defeasible? t
  :strength .99
  :variables plan plan+ -goal node1 node2 before not-between)

(def-backwards-undercutter UNDERMINE-EMBEDDED-CAUSAL-LINKS
  :defeatee embellished-protoplan
  :backwards-premises
  "(define links (set-difference (causal-links plan) (causal-links plan+)))"
  "(plan-undermines-causal-links plan links)"
  :variables plan plan+ links)

(def-backwards-reason EMBELLISHED-PROTOPLAN-for-GOAL
  :conclusions "(embellished-protoplan-for plan plan+ -goal node1 node2 before not-between)"
  :condition (interest-variable plan)
  :forwards-premises
  "(protoplan-for plan0 -goalgoals nil nil nil)"
  (:condition (subplan plan0 plan+))
  "((define p-nodes (penultimate-nodes plan0))"
  :condition
  (if node1 (subsetq p-nodes
    (possibly-intermediate-nodes
    ;; R is used for CONFRONTATION
    :variables plan plan+ link -goal node node1 node2 R before not-between)"
node1 node2 plan+ (plan-steps plan+) before not-between)
(subsetp p-nodes
  (possibly-preceding-nodes node2 plan+ (plan-steps plan+) before)))))
"(define new-order
  (let ((before0 (remove-finish before))
    (not-between0 (remove-not-between-finish before not-between)))
    (dolist (L (causal-links plan0))
      (when (eq (causal-link-target L) "finish")
        (push (cons (causal-link-root L) "finish") before0)))
    (dolist (penultimate-node p-nodes)
      (dolist (n (possibly-succeeding-nodes
        penultimate-node plan+ (plan-steps plan+) before0))
        (multiple-value-bind
          (before-nodes* not-between*)
          (add-before "finish" n plan+ before0 not-between0)
          (setf before0 before-nodes* not-between0 not-between*)))))
  (list before0 not-between0)))")
 (:condition (not (null new-order)))
 "(define plan
   (build-plan
    (plan-steps plan+) -goal (causal-links plan0) (car new-order) (cadr new-order)))" :variables plan plan0 plan+ -goal node node1 node2 p-nodes
  goals before not-between new-order)
(def-backwards-reason EMBEDDED-GOAL-REGRESSION
  :conclusions "(embellished-protoplan-for plan plan+ goal node1 node2 before not-between)"
  :condition (interest-variable plan)
  :forwards-premises
  "((& precondition action) => goal)"
 (:condition (temporally-projectible precondition))
 "(define possible-nodes
   (if node1
     (possibly-intermediate-nodes
      node1 node2 plan+ (plan-steps plan+) before not-between)
     (possibly-preceding-nodes node2 plan+ (plan-steps plan+) before)))))")
 (:condition (not (null possible-nodes)))
 "(plan-node new-node action)"
 (:condition (member new-node possible-nodes))
 "(define new-order
   (multiple-value-bind
     (before* not-between*)
     (catch 'merge-plans
       (add-befores (if node1 (list (cons node1 new-node) (cons new-node node2))
         (list (cons new-node node2)))
       before not-between plan+))
     (list before* not-between*)))")
 (:condition (car new-order))
 "(define new-before (mem1 new-order))"
 "(define new-between (mem2 new-order))"
 :backwards-premises
 "(embellished-proto-plan-for
(define plan
  (extend-embellished-plan new-node goal subplan plan+))
(:condition (not (null plan)))
:variables plan plan+ subplan goal node1 node2 new-node precondition before not-between
  new-order new-before new-between possible-nodes action)

(def-backwards-reason EMBEDDED-NULL-PLAN
 :conclusions
  "((embellished-protoplan-for plan plan+ goal node1 node2 before not-between)"
 :condition (and (interest-variable plan) (null node1) (not (conjunctionp goal))
    (temporally-projectible goal))
 :backwards-premises
  "goal"
  "((define plan (embedded-null-plan goal plan+ before not-between)))"
  (:condition (not (null plan)))
:variables plan+ goal plan node node1 node2 before not-between)

(def-backwards-reason SPLIT-EMBEDDED-CONJUNCTIVE-GOAL
 :conclusions
  "((embellished-protoplan-for plan& plan+ (goal1 & goal2) node1 node2 before not-between)"
 :condition (and (interest-variable plan&) (null node1) (temporally-projectible goal1)
    (temporally-projectible goal2))
 :backwards-premises
  "((embellished-protoplan-for plan1 plan+ goal1 node1 node2 before not-between)"
  "((define before1 (before-nodes plan1))"
  "((define not-between1 (not-between1 plan1))"
  "((embellished-protoplan-for plan2 plan+ goal2 node1 node2 before1 not-between1))"
  "((define plan& (merge-embellished-plans plan1 plan2 goal1 goal2))"
  (:condition (not (null plan&)))
:variables plan+ plan& plan1 plan2 nodes goal1 goal2 node1 node2 before
  not-between before1 not-between1)

;;   ADDING ORDERING-CONSTRAINTS

(def-forwards-reason ADD-ORDERING-CONSTRAINTS
 :conclusions
  "((protoplan-for plan goal goals nil nil nil)"
 :forwards-premises
  "((plan-undermines-causal-link plan- R node link))"
  (:clue? t)
  "((protoplan-for plan- goal goals nil nil nil)"
  "((define plan (add-not-between node link plan- t))"
  (:condition (not (null plan)))
:variables plan plan- node link goal goals R)

(def-forwards-reason ADD-EMBEDDED-ORDERING-CONSTRAINTS
 :conclusions "((embellished-protoplan-for plan plan+ goal node1 node2 before not-between)"
(def-forwards-reason REUSE-NODES
  :conclusions
  "(protoplan-for plan goal nil nil nil nil nil)"
  :forwards-premises
  "(plan-undermines-causal-link plan+ R node link)"
  (:clue? t)
  "(embellished-protoplan-for plan+ goal node1 node2 before not-between)"
  "(define plan (add-not-between node link plan- nil))"
  (:condition (not (null plan)))
  :variables plan- plan+ plan goal node1 node2 before not-between R node link)

;;

(def-backwards-reason REUSE-PLANS
  :conclusions
  (protoplan-for plan goal goals nodes-used links bad-link)
  :condition (and (interest-variable plan) (not (null nodes)))
  :forwards-premises
  (protoplan-for new-plan0 goal0 goals new-nodes nil links0 bad-link)
  (:condition (and (not (some
    #(lambda (L) (and (eq (causal-link-target L) *finish*) (eq (causal-link-root L) (causal-link-root bad-link))))
    (causal-links new-plan0)))
    (goals-used (cons goal0 goals) plan+ bad-link)))
  (:clue? t)
  "(define new-nodes
    (cons node (possibly-preceding-nodes node plan+ (plan-steps plan+) (before-nodes plan+)))")
  "(define links (remove bad-link (causal-links plan+)))"
  :backwards-premises
  (protoplan-for plan goal nil nil nil nil nil nil)
  (:condition (and (not (some
    #(lambda (L) (and (eq (causal-link-target L) *finish*) (eq (causal-link-root L) (causal-link-root bad-link))))
    (causal-links plan0)))
    (goals-used (cons goal0 goals) plan+ bad-link)))
  (:clue? t)
  "(define new-nodes
    (cons node (possibly-preceding-nodes node plan+ (plan-steps plan+) (before-nodes plan+)))")
  "(define links (remove bad-link (causal-links plan+)))"
  :variables plan goal goal0 goals nodes plan+ R node new-nodes links bad-link plan0 new-plan0 links0 link0 node1 node2)
"(protoplan-for plan goal goals0 nodes0 nodes-used0 links0 bad-link0)"
:condition (and (subsetp (plan-steps plan) nodes)
  (not (member bad-link (causal-links plan)))
  (or (not (equal goal (causal-link-goal bad-link)))
    (mem goal goals)
    (not (some #'(lambda (L)
      (and (equal goal (causal-link-goal bad-link))
        (eq (causal-link-root L) (causal-link-root bad-link))
        (eq (causal-link-target L) "finish"))
      (causal-links plan))))
  (or (plan-steps plan)
    (null bad-link)
    (not (eq (causal-link-target bad-link) "start"))
    (not (equal goal (causal-link-goal bad-link))))))
:variables plan goal goals nodes nodes-used links bad-link
  goals0 nodes0 nodes-used0 links0 bad-link0)

(def-backwards-reason REUSE-NODE
  :conclusions "(protoplan-for plan goal goals nodes nodes-used links bad-link)"
  :condition (and (interest-variable plan) (not (null nodes)) (not (conjunctionp goal)))
  :forwards-premises
  "(=> (& R action) goal)"
  "(plan-node node action)"
  :condition
  (and (member node nodes)
    (or (null bad-link)
      (not (equal goal (causal-link-goal bad-link)))
      (mem goal goals)
      (not (equal (plan-node-action (causal-link-root bad-link)) action))))
  "(define new-nodes (remove node nodes))"
  "(define new-nodes-used (cons node nodes-used))"
  :backwards-premises
  "(protoplan-for subplan R goals new-nodes new-nodes-used links bad-link)"
  "(define plan (extend-plan-with-node goal subplan bad-link))"
  :condition (not (null plan))
:variables R action plan goal goals nodes node new-nodes
  subplan nodes-used new-nodes-used links bad-link)

;;;;

(def-forwards-reason CONFRONTATION
  :conclusions
  "(protoplan-for plan goal goals nodes nodes-used links bad-link)"
  :forwards-premises
  "(plan-undermines-causal-link plan- R node link)"
  :condition (not (null R))
  (:clue? t)
  "(protoplan-for plan- goal goals nodes nodes-used links bad-link)"
  (:clue? t)
  "(protoplan-for subplan R goals new-nodes new-nodes-used links bad-link)"
  :condition (not (null plan))
:variables R action plan goal goals nodes node new-nodes
  subplan nodes-used new-nodes-used links bad-link)

;;;;
(def-forwards-reason EMBEDDED-CONFRONTATION
  :conclusions "((embellished-protoplan-for plan plan+ goal node1 node2 before not-between)"
  :forwards-premises
  "((plan-undermines-causal-link plan+ R node link)"
   (:condition (not (null R)))
   (:clue? t)
  "((embellished-protoplan-for subplan plan+ goal node1 node2 before not-between)"
   (:clue? t)
  :backwards-premises
  "((define -R (neg R)))"
  "((embellished-plan-for repair-plan plan+ -R node1* node2* new-before new-not-between)"
  "((define plan (make-confrontation-plan repair-plan subplan -R node (list link))))"
  (:condition (not (null plan)))
  :variables plan plan+ goal node1 node2 before not-between R node link subplan
  precondition new-node new-before new-not-between -R node1* node2* repair-plan)