Loop Notes

**Input:** Minimal annotated consistent partial order $\mathcal{P}$

**Output:** Template $T$ representing $\mathcal{P}$

**procedure** `Convert_To_Template($\mathcal{P}$):

$\quad T$.conditions ← `Find_Relevant_Current($\mathcal{P}$) + `Find_Relevant_Goal($\mathcal{P}$)

$\quad T$.body ← $\mathcal{P}$

`Identify_Loops($T$)

**procedure** `Identify_Loops($T$):

change ← true

while (change) do

change ← false

∀ fans outs

∀ fans with same sequences:

identify varying parameter(s),
introduce new variable for them

if fans then have same init conds & results then

newloop ← empty while loop

newloop.conditions ← fans.init conds & results

newloop.body ← fan.sequence

replace similar fans with newloop
reconnect condition and result arcs

change ← true

∀ sequences

use string-matching alg to find repeated seqs
∀ repeated sequences

if last repetition has different outcome then

identify varying parameter(s) (if any)
introduce new variable for them

newloop ← empty while loop

newloop.conditions ← not last outcome and any common conditions

newloop.body ← sequence

change ← true

---

Table 1: Converting a plan into a template with loops

Worrisome issues:
while ((in_current_state (at(v?1:obj v?2:loc)) or 
in_current_state (inside(v?1:obj v?4:rocket))) and 
in_goal_state (at(v?1:obj v?3:loc)) or 
in_goal_state (inside(v?1:obj v?4:rocket))) do 
if (in_current_state (at(?1:obj ?2:loc)) and 
in_current_state (at(?4:rocket ?5:loc))) then 
while (in_current_state (at(v?6:obj ?2:loc)) and 
in_current_state (at(?4:rocket ?2:loc)) and 
in_goal_state (at(v?6:obj ?7:loc)) or 
in_goal_state (inside(v?6:obj ?4:rocket))) do 
while (in_current_state (inside(v?6:obj ?4:rocket)) and 
in_current_state (at(?4:rocket ?2:loc)) and 
in_goal_state (at(v?6:obj ?2:loc))) do 
while (in_current_state (inside(v?1:obj v?2:rocket)) and 
in_current_state (at(?2:rocket v?3:loc)) and 
in_goal_state (at(v?1:obj v?4:loc))) do 
while (in_current_state (inside(v?5:obj ?2:rocket)) and 
in_current_state (at(?5:obj ?4:loc))) do 
while (in_current_state (at(v?1:rocket v?2:loc)) and 
in_goal_state (at(v?1:rocket v?3:loc))) do 

Table 2: Rocket domain template

- How to handle whiles with not-quite-matching bodies (whiles with nested if's)? — there's a line somewhere about how aggressively we should merge things, but not sure where it is.
- If this is going to work as an watch-and-learn system WITHOUT guaranteed super-nice-and-wise teachers, we have to relax requirements on form of observed examples. How can we learn templates from inconsistent examples, examples with messy/difficult orderings, optimal examples, etc?
- Class of problems/domains attacking?
- Will we be able to say ANYTHING about efficiency?

Professed goals:
- Learn dom-spec planner from super-nice-and-wise teacher
- Learn to solve “some” problems faster than g-p planning
• Learn to solve more of “some” problems than g-p planning (horizon!)
• Use for agent modelling (develop example domain(s))
• Any impact/help in multi-agent situation(s)? (not sure how)
• Can we learn dom-spec planner from non-super-nice-and-wise teacher?

Issues to remind folks of:
• Folks have done work on revealing domain operators (mei wang)
• Folks have done work on speeding up subop plans (knoblock)

Schedule (a.k.a. To Do list):
• Rewrite SPRAWL paper to include poly solution (almost done!)
• Hand-write templates for all ’00 & 02 domains
• Decide what to do about difficult issue #1 (handling loops of non-identical steps (whiles with nested ifs)) &/or if it needs to be addressed right now
• Lure unsuspecting committee members (Manuela (hah! already trapped!), Reid, Avrim or Steve Smith, Craig) with appealing extended abstract
• Write up proposal
• Read a bunch more papers! (including the ones suggested by Dan Weld & by Avrim)
• Give proposal talk
• Clean up & mass-market SPRAWL
  – Detach from FF
  – Implement poly solution
  – Add saving needs tree
• Implement proposed while loop stuff (hah!)
• Clean up template language
• Save template in form of c program
• Hang moon
• DEFEND!
• Parrr-tay
while (not (in_current_state (broken(v?1:package))) and
  in_current_state (at(v?1:package v?2:loc)) and
  in_goal_state (broken(v?1:package))) do
if (not (in_goal_state (blownup(?2:loc)))) or
  (in_goal_state (blownup(?2:loc))) then
  if (in_current_state (blownup(?2:loc))) and
    in_goal_state (at(v?2:package ?1:loc)) and
    in_current_state (at(v?1:package))
  then
  else if (in_goal_state (blownup(?3:loc)) and
    not (in_current_state (blownup(?3:loc)))) then
else if (in_current_state (blownup(?3:loc))) then
  move(?1:package ?2:loc)
else if (not (in_goal_state (blownup(?3:loc))))) then
while (in_goal_state (blownup(v?1:loc)))
  and
  not (in_current_state (blownup(v?1:loc)))) do
while (in_current_state (at(v?2:package ?1:loc)) and
  in_goal_state (not (broken(v?2:package)))) do
if (in_goal_state (at(?2:package ?3:loc)) and
  not (in_goal_state (blownup(?3:loc)))) or
  (in_current_state (blownup(?3:loc))) then
else if (in_current_state (blownup(?3:loc))) then
  move(?1:package ?2:loc)
else if (not (in_goal_state (blownup(?3:loc))))) then
blow(?1:loc)
while (in_current_state (at(v?1:package v?2:loc)) and
  in_goal_state (broken(v?1:package))) and
  not (in_current_state (broken(v?1:package)))) do
  while (in_current_state (at(v?3:package v?2:loc)) and
    in_goal_state (not (broken(v?3:package)))) then
    move(?1:package ?2:loc)
blow(?2:loc)
while (in_current_state (at(v?1:package v?2:loc)) and
  in_goal_state (at(v?1:package v?3:loc))) do

Table 3: Bomb domain template
generate intermediate states btw steps by propagating init state forward
identify CEs that do occur
for each step do
    for each precondition of it do
        find last provider of precondition
        add link between provider and this step
    if precondition added by active CE then
        add conditions of active CE to that step’s precs

Table 4: Poly-time algorithm for finding MAC POs
Input: Minimal annotated consistent partial order $P$, current template $T_i$.
Output: New template $T_{i+1}$, updated with $P$

procedure Distill $(P, T_i)$:

$A \leftarrow \text{Find Variable Assignment}(P, T_i.\text{variables}, \emptyset)$

until match or can’t match do

if $A = \emptyset$ then
  can’t match
else

  $N \leftarrow \text{Make New If Statement}(\text{Assign}(P, A))$
  match $\leftarrow \text{Is A Match}(N, T_i)$
  if not can’t match and not match then
    $A \leftarrow \text{Find Variable Assignment}(P, T_i.\text{variables}, A)$
  if can’t match then
    $A \leftarrow \text{Find Variable Assignment}(P, T_i.\text{variables}, \emptyset)$
    $N \leftarrow \text{Make New If Statement}(\text{Assign}(P, A))$
  $T_{i+1} \leftarrow \text{Add To Template}(N, T_i)$

procedure Make New If Statement($P_A$):

$N \leftarrow \text{empty if statement}$

for all terms $t_m$ in initial state of $P_A$ do

  if exists a step $s_n$ in plan body of $P_A$ such that $s_n$ needs $t_m$ or goal state of $P_A$ needs $t_m$ then
    Add To Conditions($N$, in current state ($t_m$))
  for all terms $t_m$ in goal state of $P_A$ do

    if exists a step $s_n$ in plan body of $P_A$ such that $t_m$ relies on $s_n$ then
      Add To Conditions($N$, in goal state ($t_m$))
  for all steps $s_n$ in plan body of $P_A$ do
    Add To Body($N, s_n$)
  return $N$

procedure Is A Match($N, T_i$):

for all if-statements $I_n$ in $T_i$ do

  if $N$ matches of $I_n$ then
    return true

procedure Add To Template($N, T_i$):

for all if-statements $I_n$ in $T_i$ do

  if $N$ matches $I_n$ then
    $I_n \leftarrow \text{Combine}(I_n, N)$
  return

if $N$ is unmatched then
  Add To End($N, T_i$)

Table 5: The current Distill algorithm: updating a template with a new observed plan.