Loop Notes

```
**Input**: Minimal annotated consistent partial order $\mathcal{P}$
**Output**: Template $T$ representing $\mathcal{P}$

**procedure** Convert_To_Template($\mathcal{P}$):
    $T$.conditions ← Find_Relevant_Current($\mathcal{P}$) + Find_Relevant_Goal()
    $T$.body ← $\mathcal{P}$
    Identify_Loops($T$)

**procedure** Identify_Loops($T$):
    change ← **true**
    **while** (change) **do**
        change ← **false**
        ∀ fan 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 algs 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
        move(?4:rocket ?5:loc ?2:loc)
    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 v?7:loc)) or
                in_goal_state (inside(v?6:obj ?4:rocket)))) do
        load(?6:obj ?4:rocket ?2:loc)
    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
        unload(?6:obj ?4:rocket ?2:loc)
while (in_current_state (inside(v?1:obj v?2:rocket)) and
        in_current_state (at(v?2:rocket v?3:loc)) and
        in_goal_state (at(v?1:obj v?4:loc))) do
    move(?2:rocket ?3:loc ?4:loc)
    while (in_current_state (inside(v?5:obj ?2:rocket)) and
            in_goal_state (at(v?5:obj ?4:loc))) do
        unload(?5:obj ?2:rocket ?4:loc)
while (in_current_state (at(v?1:rocket v?2:loc)) and
        in_goal_state (at(v?1:rocket v?3:loc))) do
    move(?1:rocket ?2:loc ?3:loc)
```

Table 2: Rocket domain template

- How to handle whiles with not-quite-matching bodies (whiles with nested ifs)? —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)) and
                in_current_state (blownup(?2:loc)))) then
        if (in_goal_state (at(?1:package ?3:loc)) and
                in_goal_state (blownup(?3:loc)) and
                not (in_current_state (blownup(?3:loc)))) then
            move(?1:package ?2:loc ?3:loc)
        else if (in_goal_state (blownup(?3:loc)) and
                not (in_current_state (blownup(?3:loc)))) then
            move(?1:package ?2:loc ?3:loc)
        else if (in_goal_state (not (blownup(?2:loc))) and (not (in_current_state (blownup(?3:loc))))) then
            move(?1:package ?2:loc ?3:loc)
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(?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
            move(?2:package ?1:loc ?3:loc)
        else if (in_current_state (blownup(?3:loc))) then
            move(?2:package ?1:loc ?3:loc)
        else if (not (in_goal_state (blownup(?3:loc))))
            move(?2:package ?1:loc ?3:loc)
        move(?2:package ?1:loc ?3:loc)
    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)))) then
    while (in_current_state (at(v?3:package v?2:loc)) and
            in_goal_state (not (broken(v?3:package)))) then
        move(?3:package ?2:loc ?4: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
    move(?1:package ?2:loc ?3:loc)
```

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 $\mathcal{P}$,
       current template $T_i$.
Output: New template $T_{i+1}$, updated with $\mathcal{P}$

procedure DISTILL ($\mathcal{P}$, $T_i$):
    $\mathcal{A} \leftarrow$ Find_Variable_Assignment($\mathcal{P}$, $T_i.variables$, $\emptyset$)
    until match or can't match do
        if $\mathcal{A} = \emptyset$ then
            can't match
        else
            $\mathcal{N} \leftarrow$ Make_New_If_Statement(Assign($\mathcal{P}$, $\mathcal{A}$))
            match $\leftarrow$ Is_A_Match($\mathcal{N}$, $T_i$)
        if not can't match and not match then
            $\mathcal{A} \leftarrow$ Find_Variable_Assignment($\mathcal{P}$, $T_i.variables$, $\mathcal{A}$)
    if can't match then
        $\mathcal{A} \leftarrow$ Find_Variable_Assignment($\mathcal{P}$, $T_i.variables$, $\emptyset$)
        $\mathcal{N} \leftarrow$ Make_New_If_Statement(Assign($\mathcal{P}$, $\mathcal{A}$))
    $T_{i+1} \leftarrow$ Add_To_Template($\mathcal{N}$, $T_i$)

procedure Make_New_If_Statement($\mathcal{P}_\mathcal{A}$):
    $N \leftarrow$ empty if statement
    for all terms $t_m$ in initial state of $\mathcal{P}_\mathcal{A}$ do
        if exists a step $s_n$ in plan body of $\mathcal{P}_\mathcal{A}$ such that
                $s_n$ needs $t_m$ or goal state of $\mathcal{P}_\mathcal{A}$ needs $t_m$ then
            Add_To_Conditions($N$, in_current_state ($t_m$))
    for all terms $t_m$ in goal state of $\mathcal{P}_\mathcal{A}$ do
        if exists a step $s_n$ in plan body of $\mathcal{P}_\mathcal{A}$ 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 $\mathcal{P}_\mathcal{A}$ do
        Add_To_Body($N$, $s_n$)
    return $N$

procedure Is_A_Match($\mathcal{N}$, $T_i$):
    for all if-statements $I_n$ in $T_i$ do
        if $\mathcal{N}$ matches of $I_n$ then
            return true

procedure Add_To_Template($\mathcal{N}$, $T_i$):
    for all if-statements $I_n$ in $T_i$ do
        if $\mathcal{N}$ matches $I_n$ then
            $I_n \leftarrow$ Combine($I_n$, $\mathcal{N}$)
            return
    if $\mathcal{N}$ is unmatched then
        Add_To_End($\mathcal{N}$, $T_i$)
```

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