Needs Analysis for Partial Ordering

Elly Winner and Manuela Veloso

April 19, 2001

1 Introduction

Our first step in finding a partial order is to do needs analysis on the total order. Needs analysis provides an answer to the question of why each step is necessary. We define a need to be a term which must be true on execution of a particular step for the rest of the plan to run correctly and result in the goal state. We calculate the need DAG N of the plan by propagating needs backwards from the goal state.

We first create a dummy goal step, g and define its needs to be the terms of the goal state. Then we go backwards one step, and ask, what terms need to be true before the final plan step executes in order for these things to be true afterwards? We continue this calculation all the way back to the initial state. The needs structure allows us to easily identify the necessary effects of a given step and to identify most of the dependencies in the plan (the only dependencies unidentified after needs analysis are threats which are not active in the total order).

We must draw a distinction between needs of a given step and needs of a given need, which are associated with a step, but are not needs of that step. A need $n1_s$ of a step s is a term which must be true for the step to be executable, in other words, a precondition of the step. A need n_m of another need m is a term which must be true for the need m to be satisfied. For example, a need of the step, **sprinkle front-lawn**, could be **on sprinkler**, since that term must be true for the step **sprinkle front-lawn** to be executable. To give an example of a need of another need, let us assume that we hope that, after executing **sprinkle front-lawn**, **wet shoe** will be true. This could be accomplished by ensuring that **at shoe front-lawn** was true before **sprinkle front-lawn** was executed, or by ensuring that **wet shoe** was already true before **sprinkle front-lawn** executed. These two terms would be needs not of the step **sprinkle front-lawn**, though they would be associated with it (since they need to be true before it executes in order to accomplish **wet shoe**). Instead, they are needs of the need **wet shoe**.

This last example illustrates a second distinction we need to make: we label needs as either "maintain" needs or as "add" needs. As mentioned above, if **wet shoe** needs to be true after the execution of the step **sprinkle front-lawn**, there are multiple ways to accomplish it. One way is that the shoe could have been wet previously, so we can say that one need of **wet shoe** is **wet shoe**. We call this a "maintain" need, since the step does not affect the fulfillment of this need, it simply maintains the need. But another way **wet shoe** could be accomplished by the step **sprinkle front-lawn** is by ensuring that **at shoe front-lawn** is true before **sprinkle front-lawn** executes. We call this an "add" need, since the need **at shoe front-lawn** is new.

Also note that the needs structure must necessarily include logical operators. The previous example shows that the needs of the need **wet shoe** would be **wet shoe OR at shoe front-lawn**. Only one of the two need be true to satisfy **wet shoe**. Similarly, we can imagine how an **AND** might be necessary: the preconditions of the step **paint cabinet** could be **sanded cabinet AND open paint-can**. And negations are also important; a precondition of the step **unload box** could be **NOT empty box**.

2 Algorithm

We will now present the needs analysis algorithm in painful detail and will then illustrate its behavior with an example.

```
and0 = new and
for cur-term = goal-terms(first) to goal-terms(last)
  and0.add_operand(cur-term)
dummy-goal-step.needs = and0
prev-step = dummy-goal-step
for cur-step = plan-steps(last) to plan-steps(first)
  and0 = new and
  for cur-prec = cur-step.preconditions(first) to cur-step.preconditions(last)
    and0.add_operand(cur-prec);
  cur-step.needs = and0;
  for cur-need = prev-step.needs(first) to prev-step.needs(last)
    and1 = new and
    or1 = new or
    cur-step.needs.add(MAINTAIN cur-need)
    or1.add_operand(MAINTAIN cur-need)
    and2 = new and
    for cur-eff = cur-step.cond-effs(first) to cur-step.cond-effs(last)
      if cur-eff adds cur-need
        and3 = new and
        if cur-eff.conditions = null
          inevitable = true
        for cur-cond = cur-eff.conditions(first) to cur-eff.conditions(last)
          cur-step.needs.add(ADD cur-cond)
          and3.add_operand(ADD cur-cond)
        or1.add_operand(and3)
      if cur-eff dels cur-need
```

```
      or2 = new or
      if cur-eff.conditions = null
        impossible = true
      for cur-cond = cur-eff.conditions(first) to cur-eff.conditions(last)
        cur-step.needs.add(ADD NOT cur-cond)
        or2.add_operand(ADD NOT cur-cond)
      and2.add_operand(or2)
    and1.add_operand(or1)
    and1.add_operand(and2)
    cur-need.needs = and1
    if inevitable and NOT impossible
      cur-need.accomplished = true
  prev-step = cur-step
```

2.1 Sprinkler Example

We will use our sprinkler example to illustrate the behavior of the needs-analysis algorithm.

1. Initial State
   - at shoe back-yard
   - on sprinkler
2. move shoe back-yard lake
3. sprinkle front-yard
4. Goal State
   - wet shoe
   - wet front-yard

First, the algorithm will create a dummy goal step that has, as its needs, **ADD wet shoe AND ADD wet front-yard**. These are "add" needs because they are new. Then it will go to the last plan step to see what terms need to be true before it in order to correctly produce the goal needs. In this case, the last plan step is **sprinkle front-yard**. This step has the need **ADD on sprinkler**, which is also an "add" need because it is new with this step. We now need to work on the individual needs: **ADD wet shoe** and **ADD wet front-yard**. As we discussed before, there are two ways to satisfy **wet shoe**: either **wet shoe** could be true before this step executes, or **at shoe front-yard** must be true before this step executes. So the needs of **ADD wet shoe** are **MAINTAIN wet shoe OR ADD at shoe front-yard**. As for **ADD wet front-yard**, we find that it is inevitable (whenever the step **sprinkle front-yard** executes, **wet front-yard** will result) and not impossible (there is no way to prevent it from occurring as long as **sprinkle front-yard** executes), and therefore is accomplished by the step **sprinkle front-yard**, so it has no needs.

Now we move to the next previous step, **move shoe back-yard lake**. This step has one need, **ADD at shoe back-yard**. We must consider how to accomplish the needs of the previous step: **ADD on sprinkler**. This can be accomplished if this step maintains the term or creates it. But since there is no way for the step **move shoe lake** to create the term **on sprinkler**, the need of **ADD on sprinkler** is simply **MAINTAIN on sprinkler**. Next we look at other needs associated with this step, or **MAINTAIN wet shoe OR ADD at shoe front-yard**. First let us look at **MAINTAIN wet shoe**. We find that it is inevitable (whenever the step **move shoe lake** executes, **wet shoe** will result) and not impossible (there is no way to prevent it from occurring as long as **move shoe lake** executes), and therefore is accomplished by this step and has no needs. Now let us look at **ADD at shoe front-yard**. There is no way for the step **move shoe lake** to create **at shoe front-yard** so the only way to create that need is to maintain it, so the need is **MAINTAIN at shoe front-yard**. Since **MAINTAIN wet shoe** had no needs, the needs of **MAINTAIN wet shoe OR ADD at shoe front-yard** are simply **MAINTAIN at shoe front-yard**.

Finally, we have arrived at the initial state. We will now see which needs will be satisfied and which will not. The remaining needs are **ADD at shoe back-yard**, **MAINTAIN on sprinkler** and **MAINTAIN at shoe front-yard**. Two of the needs, **at shoe back-yard** and **on sprinkler**, are true in the initial state, so they are satisfied, but **at shoe front-yard** is not true in the initial state, so **MAINTAIN at shoe front-yard** is unsatisfied.

The final needs structure created by the algorithm can be represented as follows. In our notation, we show that one need is needed by another by indenting the needed term below the needing term. We express that an operator needs a term by noting that the term is "needed by" the operator, and express that a term is needed associated with a given operator but not needed by it by noting that the term is "at" the operator.

```
ADD needed by operator GOAL
  ADD wet shoe at operator GOAL
    OR at operator sprinkle front-yard
      MAINTAIN wet shoe at operator sprinkle front-yard
        accomplished by operator move shoe back-yard lake
      ADD at shoe front-yard at operator sprinkle front-yard
        MAINTAIN at shoe front-yard at operator move shoe back-yard lake
          MAINTAIN at shoe front-yard at operator INITIAL
  ADD wet front-yard needed by operator GOAL
    accomplished by operator sprinkle front-yard
ADD on sprinkler needed by operator sprinkle front-yard
  MAINTAIN on sprinkler at operator move shoe back-yard lake
    accomplished by operator INITIAL
ADD at shoe back-yard needed by operator move shoe back-yard lake
  accomplished by operator INITIAL
```

Note that there is no "accomplished" notation after **MAINTAIN at shoe front-yard at operator INITIAL**, but the need-chain ends. This shows that this need is never accomplished in this plan.