Pragmatic Requirements for Goal-Driven Adaptive Systems

Felipe Pontes Guimaraes
IME - USP
Rua do Matao, 1010
Sao Paulo, Brasil
felipepg@ime.usp.br

Raian Ali
Bournemouth University
Address pending
Bournemouth, England
rali@bournemouth.ac.uk

Genaina Nunes Rodrigues
Universidade de Brasilia
Address pending
Brasília, Brasil
genaina@cic.unb.br

Daniel Macedo Batista
IME - USP
Rua do Matao, 1010
Sao Paulo, Brasil
dbatista@ime.usp.br

ABSTRACT

This is the demonstration for the Linear complexity of the algorithm presented in the full paper.

Categories and Subject Descriptors

D.2.1 [**Software**]: Software EngineeringRequirements

General Terms

Contextual goal-model

Keywords

Contextual goal-model

1. ALGORITHM AND DESCRIPTION

Algorithm 1 is a recursive function that builds on the Tropos GM being tree-structured to decide on an execution plan - the set of all goals to be pursued and tasks to be executed - in order to achieve the desired quality constraints or to decide whether the available possibilities under the current context are insufficient to deliver the quality required. This algorithm has a linear complexity with respect to the number of nodes in the CGM (demonstration in Section 0.0.1).

It considers the root node of the CGM (line 1) and checks whether the root goal is itself applicable under the current context (line 2), returning NULL if it is not (line 3).

In the particular case when the variant's root node is a task (line 5) it can readily decide on the achievability. This is because, as it was presented in Figure **??**, the task nodes have the knowledge of the expected quality level it can deliver for each metric considered in the CGM variant. By comparing the delivered and required quality levels (line 7), the algorithm can decide whether it is capable or not of delivering such quality constraints at that branch. In the first case, it will return a plan consisting of this task (lines 8-10) while in the latter it will return `NULL` (line 12) and indicate its inability of achieving the goal.

Whenever the root goal is a Delegation or a Goal, it falls within one of the two cases previously mentioned: OR-Decompositions and AND-Decompositions. In both of these cases, the algorithm will firstly check whether, besides the `Quality Constraints` passed on as parameters, the root goal also has its own QR. It will then choose the more strict quality requirement (line 16) for the remainder of the execution.

For OR-decompositions (lines 17-25), it defines the applicable Refinements under the current context (line 18) and recursively invokes the `isAchievable` method for each of them (lines 19-20). Should any of its possible refinements be able to provide the desired quality levels (line 21), it adds such refinement's execution plan to its complete plan (line 22) and returns such plan immediately (line 23). If none of the Refinements are able to deliver the desired quality level, it will return the `NULL` value (line 25) indicating that, for the provided CGM variant and the required quality levels, this goal is not achievable.

For AND-decompositions (lines 26-36), the behavior is somewhat similar. The algorithm verifies the applicable goals for the current context (line 27) and, for each found Refinement (line 29), evaluates whether it can deliver the expected quality constraints (line 30). If it can, the execution plan is added to the `complete` plan (lines 31-32). Otherwise, the goal is unachievable and `NULL` is returned (lines 33-35). If all Refinements are able to deliver the expected quality constraints and the `foreach` loop ends, then the node includes itself in the `complete` plan and returns it (lines 38-38).

After the execution of this method, an execution plan is returned if the goal is achievable or `NULL` otherwise. If a plan is returned, its execution is likely to result in achieving the root goal with the expected quality constraints as long as the `Task`'s expected delivered quality values are respected.

2. ALGORITHM COMPLEXITY ANALYSIS

SEAMS 2013 Hyderabad, India

Algorithm 1: isAchievable(CGM cgm, Context current, QualityConstraint qualReq)

Input: CGM, current context and desired quality constraints
Result: An execution plan or NULL if the goal is not achievable

```
1  Goal root ← cgm.getRoot();
2  if !root.isApplicable(current) then
3  |  return NULL;
4  end
5  if (root.myType() == task) then
6  |  begin
7  |  |  if
   |  |  root.compare(root.myProvidedQuality(qualReq.metric),
   |  |  qualReq.comparison, qualReq.threshold) then
8  |  |  |  Plan p ← new Plan();
9  |  |  |  p ← addRefinementToPlan(root, p);
10 |  |  |  return p;
11 |  |  else
12 |  |  |  return NULL;
13 |  |  end
14 |  end
15 end
16 QualityConstraint consideredQualReq;
17 consideredQualReq ←
   root.interp.stricterQualityConstraint(root.qualReq,
   qualReq);
18 if root.isOrDecomposition()) then
19 |  dependencies ←
   |  root.getApplicableRefinementsFromCGM(cgm,
   |  current);
20 |  foreach Refinement d do
21 |  |  Plan p ← d.isAchievable(cgm, context,
   |  |  consideredQualReq);
22 |  |  if (p != NULL then
23 |  |  |  complete ← addRefinementToPlan(d, p);
24 |  |  |  return p;
25 |  end
26 |  return NULL
27 else if (root.isAndDecomposition()) then
28 |  dependencies ←
   |  root.getApplicableRefinementsFromCGM(cgm,
   |  current);
29 |  Plan complete;
30 |  foreach Refinement d do
31 |  |  Plan p ← d.isAchievable(cgm, context,
   |  |  consideredQualReq);
32 |  |  if (p != NULL then
33 |  |  |  complete ← addPlanToPlan(p, complete);
34 |  |  else
35 |  |  |  return NULL;
36 |  |  end
37 |  end
38 |  complete ← addNodeToPlan(d, complete);
39 |  return complete
40 end
```

2.0.1 Algorithm complexity

This algorithm runs in time $O(n)$ where n is the number of nodes in the CGM. We will prove this using finite induction during the following paragraphs over the height h of the CGM tree.

Induction basis: Single task as CGM root (tree height $h = 0$).

The hypothesis is then well defined. We assume that the algorithm runs in linear time with regard to the number of nodes in the CGM. Our induction will be made on the height of the CGM tree. Our basis is the trivial case where the CGM has height 0. In this particular case, the root node is a task - due to the CGM restriction that states that any leaf node is a task. If we apply the algorithm in this scenario, the IF-clause at line 5 will be triggered and the algorithm will make a simple comparison of the quality constraints passed as input and the quality that the task can provide. If this task cannot fulfil the quality required then it returns NULL. This is done at constant time. If, otherwise, the task is able to deliver the quality level required, it then creates a plan p and includes itself in it. This is also done in constant time.

One particular case that must also be taken into consideration its when the root goal is not applicable. In this situation the algorithm simply returns `NULL` in constant time as well.

Induction step.

Assume that we have proven that:

Definition 1. For any tree of height h' where $h \geq h' \geq 0$ the algorithm runs in time $O(n)$ where n is the number of nodes in the CGM tree.

Let's now prove that the algorithm will also be executed in $O(n)$ for a tree of height $h + 1$.

Since the tree has a height greater than 1 ($h \geq 0$), the IF-clause at line 5 will not be triggered. So, the next step in the algorithm will be to define the `consideredQualReq`, which is simply to compare the quality requirement and choose the stricter one. This is done in constant time. Then the algorithm checks whether the decomposition is an OR-decomposition or an AND-decomposition. It extracts from the CGM the node's Refinements which is done in $O(m)$ time - where m is the number of Refinements of the root goal.

Finally, for each Refinement d in $refinements$ the algorithm recursively invokes itself for the CGM of the Refinement (sub goal, task or delegation). Since this invocation is performed on trees of height lower or equal to h we can use the fact that the root node has m refinements and Definition 1 to state that this is performed in $O(n_k)$ time, where n_k is the amount of nodes in the k sub tree.

The elapsed time for the whole `for each` can be calculated as:

$$Time_{\texttt{foreach}} = \sum_{k=1}^{m} O(n_k) \tag{1}$$

Since n is the amount of nodes in the whole tree (all subtrees plus the root goal node ($n = n_1 + n_2 + \ldots + n_k + 1$) and $O(n_1) + O(n_2) + \ldots + O(n_k) = O(n_1 + n_2 + \ldots + n_k)$, we can

conclude that the execution of the algorithm for a tree of height h, the algorithm is executed in time $O(n)$ where n is the number of nodes in the tree. Thus proving the hypothesis that for any tree of height $h \geq 0$, the algorithm runs in linear time with regard to the number of nodes in the CGM.