Extended Goal Recognition Design with First-Order Computation Tree Logic

Tsz-Chiu Au

Supplementary Material

Abstract

This is a technical appendix for the paper entitled "Extended Goal Recognition Design with First-Order Computation Tree Logic" we submitted to AAAI 2022. This document contains the pseudocodes of two algorithms and the definition of goal query statements that are omitted in the paper. We also provide additional explanations to the examples in the paper, and add two more examples to illustrate the use of EP, choice vertices, and choice edges. Finally, we briefly explain why we should use **EF** instead of **AF** in Statement 3 in Section "Finding WCD by Model Checking".

The Pseudocode of the Goal Query Graph Translation Algorithm

Algorithm A1 is the pseudocode of the algorithm that we described in Section "Translation to FO-CTL sentences" in the paper. $\mathbf{TransV}(v_1)$ and $\mathbf{TransE}(v_1, v_2)$ are $\mathsf{tr}(v_1)$ and $\mathsf{tr}(v_1, v_2)$, respectively. The algorithm generates correct FO-CTL statements but not the most succint FO-CTL statements. In our C++ implementation of the algorithm, we have implemented some optimization techniques to shorten the FO-CTL statements generated by the algorithm. For example, our implementation replaces **AF** with **AG** at terminal vertices when certain conditions are satisfied.

The Pseudocode of a Model Checking Algorithm with Caches

The **EVAL** function in Algorithm 2 in the paper uses an external model checking algorithm **MC** to evaluate a FO-CTL. The algorithm assumes we can hijack the model checking code and intercept the recursive functions in order to implement the caching mechanism. Algorithm A2 is the pseudocode of a simple model checking algorithm that evaluates a FO-CTL sentence using the definition of entailment directly, without using any external code. The algorithm is a depth-first search that explores the tree structure of the sentence and the transition system in parallel. The meanings of the symbols in the pseudocode are:

- node_i^c is the child of node_i if node_i has one child only,
- node_i^l and node_i^r are the left and right children of node_i, respectively, if node_i has two children,
- $E(s)$ denotes the set of subsequent states of s, and
- $P^{leg}(M, s)$ is the set of legal paths in M that passes through s in $\mathcal{T}(M)$.

In the pseudocode, the cost function for computing the cost attached to the truth value are deferred to Line 46. The reason for deferring the call of the cost function to the end is to make the pseudocode looks shorter, and it does not offer any computational advantage. In our implementation, the evaluation function is invoked immediately whenever t is assigned a truth value.

Goal Query Statements

To facilitate communication, we devise a way to define a goal query graph without drawing the graph. We can formulate a goal query graph as a set of traces, each of them is an interleaving sequence of state conditions and edges with edge conditions. We call the set of traces a *goal query statement*. The goal query statements of the goal query graphs in Figures 2 and 3 are

$$[\mathsf{Nil}\ \text{AP}\ x_1^*\ \text{AP}\ x_2]$$

and

$$[\mathsf{Nil}\ \text{AP}\ (x \wedge \mathsf{XA}_x)\ \text{AP}_{\mathsf{XA}}\ \mathsf{End}],$$

respectively. These goal query statements consist of just one trace. We use an asterisk to specify a free variable as a weakly-matched variable. If the free variable occurs multiple times in a goal query statement, it is sufficient to add an asterisk to one occurance of the variable to specify that it is a weakly-matched variable.

The goal query graph in Figure 4 involves several branches of traces. We can partition a trace into smaller traces and denote them by *trace variables*. The choice vertices and the choice edges can be encoded in a trace by replacing the last state condition with a sequence of alternative traces as follows:

$$[x\ \text{AP}\ [g_1\ \text{AX}\ g_2\ \text{AP}\ T]\ [g_2\ \text{EX}\ g_1\ \text{EP}\ T]]; T = [g_3\ \text{AP}\ x],$$

where T is a trace variable. When a goal query statement has more than one trace, the traces are separated by semicolons. Since a goal query graph is a directed acyclic graph and has no cycle, the set of traces in a goal query statement is finite.

Our implementation of the goal query translation algorithm takes goal query statements as inputs. One advantage of the use of goal query statements is that we can easily specify additional constraints such as $x_1 \neq x_2$, which states that the variables x_1 and x_2 cannot match the same goal. Our implementation of the goal query graph translation algorithm can handle some additional constraints such as $x_1 \neq x_2$ and $x_1 \neq g$. However, the implementation of this feature is not specified in the pseudocode in Algorithm A1.

Discussion of the Examples in the Paper

We generated a set of files illustrate different features of goal query graphs. The files can be found in the folder aaai22-example that we submitted along with this document. Most of them are based on the examples in the paper. We also added two new examples to illusrate how EP edges and choice vertices work.

The Example in Figure 1

In Introduction, we presented a scenario in Figure 1 in which we want to recognize the agent's first and second goals so that we deploy the security guard to the correct checkpoints to intercept the agent if the second goal is g_{hack}. The goal query statement for this scenario is:

$$[\ \mathsf{Nil}\ \mathsf{AP}\ x_1\ \mathsf{AP}\ x_2\ ] \tag{A1}$$

We want both x_1 and x_2 to be strongly match the goals so that both goals can be revealed. Before redesigning the environment, the WCD is 6 since the agent who aims for g_{hack} can reveal its first goal only at either D5 or F5. But it would be too late to notice that the first goal is g_B at F5. Our EGRD search algorithm found that if we put a barrier D3 and E3, as discussed in Introduction, the WCD can be reduced to 4 since the latest position we can determine an agent who aims for g_B and then g_{hack} is at E5.

We can do better by using the following query statement instead:

$$[\ \mathsf{Nil}\ \mathsf{AP}\ x_1\ \mathsf{AP}\ g_{hack}\ ] \tag{A2}$$

This statement ignores g_{exit} as the second goal and only focus on the substructures in which g_{hack} is the second goal. Thus, optimizing the WCD of this statement can get a better result: after redesigning the environment, the WCD is 3 which corresponds to the position E2 or F1. We can do better because the EGRD search algorithm ignores the situation in which the second goal is g_{exit}, which is okay for our scenario since we do not need to intercept the agent if the second goal is g_{exit}.

The Example in Figure 2

In Section "Goal Query Graph", we reused the example in Introduction to show when to use weakly-matched variables. Figure 2 is a goal query graph for "the observer in Figure 1 does not need to know whether the first goal is g_A or g_B as long as it recognizes g_{hack} as the second goal." Notice that this is a weaker requirement than the one discussed in Introduction. If we do not need to know the first goal, we can use a weakly matched variable for x_1. The corresonding goal query statement is

$$[\ \mathsf{Nil}\ \mathsf{AP}\ x_1^*\ \mathsf{AP}\ x_2\ ] \tag{A3}$$

Before redesigning the environment, the WCD is 2 because in the worst case, the agent has to visit E2 in order to reveal g_{hack} as its second goal, if we can ignore the first goal. Redesigning the environment cannot give a lower WCD. Clearly, when compared with the scenario in Introduction, the WCD is smaller because Statement A3 is weaker than Statement A1. This is what we want in some situations. For example, as stated in Introduction, "if security guards have enough resources to protect two locations simultaneously, it is sufficient to know two possible locations an intruder plans to visit." We can simply deploy security guards to A5 and F5 whenever the agent visits E2.

The Example in Figure 3

Figure 3 in Section "Goal Query Graph" illustrates the use of special predicates: XA and End. The goal query statement is:

$$[\ \mathsf{Nil}\ \mathsf{AP}\ (x \wedge \mathsf{XA}_x)\ \mathsf{AP}_{\mathsf{XA}}\ \mathsf{End}\ ] \tag{A4}$$

Algorithm A1 will translate this statement into

$$\exists x\{\mathbf{AF}\,(x \wedge (\forall x_1[(x_1 \neq x) \Rightarrow \neg x_1]) \\ \wedge(\mathbf{AX}\,\mathbf{A}\,[\forall x_2\,[\neg x_2]\,\mathbf{U}\,\mathsf{End})]\} \tag{A5}$$

Moreover, our program performs some optimizations to the translation and generates the following FO-CTL sentence instead:

$$\exists x\{\mathbf{AF}\,(x \wedge (\forall x_1 \notin \{x\}[\neg x_1]) \wedge (\mathsf{Last} \vee \mathbf{AX}\,\mathbf{AG}\,\forall x_2[\neg x_2])\} \tag{A6}$$

$\forall x_1 \notin \{x\}[\neg x_1]$ is equivalent to $\forall x_1[(x_1 \neq x) \Rightarrow \neg x_1]$ but there will be two less nodes in the FO-CTL sentence. We can replace $\mathbf{A}\,[\forall x_2\,[\neg x_2]\,\mathbf{U}\,\mathsf{End}$ with $\mathbf{AG}\,\forall x_2[\neg x_2]$ to shorten the sentence, but we need to add Last to check whether the current state is a terminal state. Last is like End except that Last matches a terminal state before End.

The Example in Figure 4

The goal query statement of the goal query graph in Figure 4 is:

$$[x\ \mathrm{AP}\ [g_1\ \mathrm{AX}\ g_2\ \mathrm{AP}\ T]\ [g_2\ \mathrm{EX}\ g_1\ \mathrm{EP}\ T]]; T = [g_3\ \mathrm{AP}\ x],$$

However, this example is a bit contrived, and it is hard to come up with a real life example that uses this goal query graph. Nonetheless, we created one environment to show how this goal query graph works. The environment is shown in Figure A1.

Before redesigning the environment, the WCD is 1 because the goal query graph matches the vertex 8. It does not match the vertex 7 because g_2 cannot be matched in all paths starting at vertex 12. However, after redesiging the environment, the edge $(12, 17)$ is removed, causing g_2 can be matched on the remaining paths. The WCD is reduced to 0 after redesigning the environment.

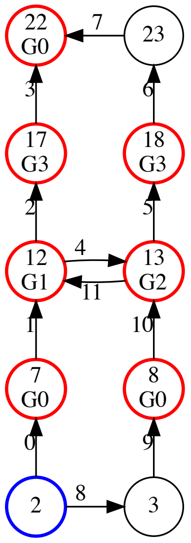

Figure A1: An environment for the goal query graph in Figure 4.

A New Example to Show How EP Edges Works

The goal query graph in Figure 4 may not be the best to illustrate how EP edges works and why we need EP edges. Therefore, we consider Figure 1 again and extend it to show why EP edges are needed in some situations. Figure A2 shows the extended environment, which is the same as Figure 1 except that there are three more legal paths that go through E3 but does not go to g_{hack}. In this environment, the Statement A1 and Statement A2 cannot match any substructure in the environment because *not* every legal path that goes through E3 will go to g_{hack}.

Although we cannot find a substructure in which an agent must reach g_{hack}, it would be helpful to find a structure that an agent will *probably* reach g_{hack}. The following goal query statement can do just that.

$$[\ \mathsf{Nil\ AP}\ x_1\ \mathsf{EP}\ g_{hack}\ ] \tag{A7}$$

When compared with Statement A2, this statement does not require the agent always reach g_{hack} after reaching a goal that matches x_1. This statement only requires the agent to have one path that reach g_{hack} after reaching a goal that matches x_1. Although this is a weaker statement, model checker can at least find some substructures that matches this statement. The WCD is 6 since we can confirm that x_1 is g_B at F5 only. After redesigning the environment, the WCD can be reduced to 2 by deleting the edge between D3 and E3. Although the security guards cannot be certain that the agent will hack the computer, the security guard can pay attention to the agent that could possibly go to the computer room and ignore the agent that will not go to the computer room.

A New Example to Show How Choice Vertices and Choice Edges Works

The example in Figure 4 in the paper may not be the best to illustrate how choice vertices and choice edges works.

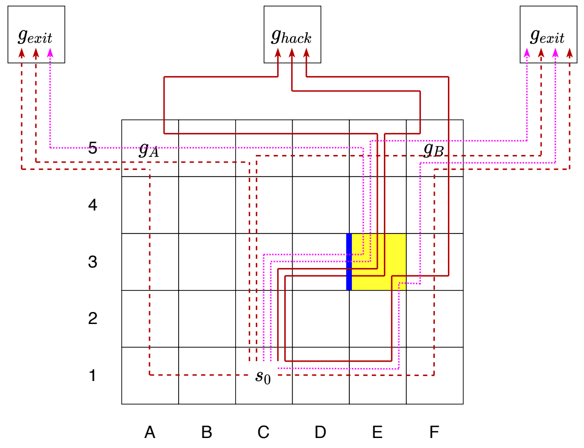

Figure A2: An extended version of the environment for in Figure 1 with addtional paths that go through E3.

Therefore, we create a new environment in which an agent may aim for one or two goals. The environment is shown in Figure A3. This environment extends the example in (Keren, Gal, and Karpas 2014) with two more legal paths. Each new paths have two goals instead of one. Then Statement A4 cannot work correctly because Statement A4 assumes each legal path has one goal only.

The following goal query statement extends Statement A4 to allow two alternative traces. The first alternative trace is like Statement A4 that only matches one goal. The second alternative trace matches two goals. Figure A4 shows how the choice vertex and the choice edge merge the two traces together in a goal query graph.

$$\begin{gathered}[\ \mathsf{Nil\ AP_{XA}}\ [\ (x \wedge \mathsf{XA}_x)\ \mathsf{AP_{XA}\ End}\ ] \\ [\ (x_1 \wedge \mathsf{XA}_{x_1})\ \mathsf{AP_{XA}}\ (x_2 \wedge \mathsf{XA}_{x_2})\ \mathsf{AP_{XA}\ End}\ ]\ ]\end{gathered} \tag{A8}$$

With the new goal query statement, the WCD is 4 before redesiging the environment and 1 after redesiging the environment. This is the same as the WCDs for the goal query graph in Figure 3. In this example, we can see that choice vertices and choice edges provide some feasibility on the number of goals that are matched according to the goal query graph.

Using **AF**ϕ Instead of **EF**ϕ in Finding WCD by Model Checking

In Section "Finding WCD by Model Checking", we propose to use **EF** ϕ to compute the WCD of ϕ. We can replace **EF**ϕ with **AF**ϕ and get the correct result *only if* we assume that every path starting from the initial state s_0 contains a state that satisfies ϕ. Most existing GRD work assume that all legal paths will end with a goal (e.g., (Keren, Gal, and Karpas 2014)) . However, our work does not rely on this assumption. Even if every legal path ends with a goal, ϕ may not be true on every legal path.

Typically, the evaluation of **AF** uses short-circuit evaluation to skip some of the paths during evaluation—whenever

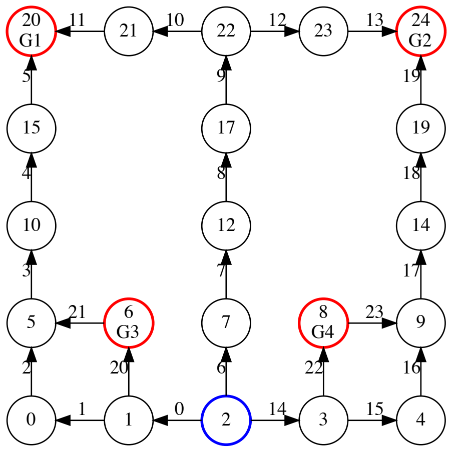

Figure A3: An extended version of the example in (Keren, Gal, and Karpas 2014). There are two more legal paths on which there are two new goals: $G3$ and $G4$

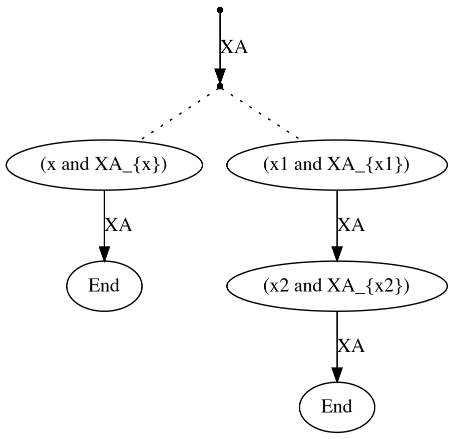

Figure A4: The goal query graph of Statement A8

there is a path that do not satisfies ϕ at all states on the path, the remaining paths will be skipped. Therefore, if we use **AF**ϕ to find WCD, the short-circuit evaluation can skip some paths that contain the state with the maximum cost. In contrast, **EF** will not skip these paths. If we disable the short-circuit evaluation of the **AF** operator, the evaluation of **AF**ϕ will return the maximum cost, but the truth value of **AF** can be false even if there exists a substructure that match ϕ.

On the other hand, **EF** also uses short-circuit evaluation to skip some paths during evaluation—whenever there is a path that satisfies ϕ, the remaining paths will be skipped. The computation of WCD requires the evaluation of every path such that the evaluation function of **EF** can identify the maximum cost among all paths. Thus, the model checking algorithm such as Algorithm A2 has to disable the short-circuit evaluation of the **EF** operator. Unlike **AF**, the truth value of **EF** can tell whether there exists a substructure that match ϕ. For this reason, **EF**ϕ has an advantage over **AF**ϕ when we disable short-circult evaluation.

References

Keren, S.; Gal, A.; and Karpas, E. 2014. Goal Recognition Design. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 154–162.

Algorithm A1: The goal query graph translation algorithm.

```
 1: procedure Translate($Graph$)
 2:     Let $v_0$ be the start vertex of $Graph$
 3:     $\phi = \mathbf{TransV}(v_0)$
 4:     For every weakly-matched variable $x$ in $\phi$
 5:         Let $P$ be the set of all paths to all $x$ in $\phi$
 6:         Let prefix be the common prefix of all paths in $P$
 7:         Insert $\exists x$ before the last node in prefix
 8:     For every strongly-matched variable $x$ in $\phi$
 9:         Insert $\exists x$ before $\phi$
10:     Return $\phi$
11: procedure TransV($v_1$)
12:     If $v_1$ is a terminal vertex, return $\mathsf{cond}(v_1)$
13:     If $v_1$ is a state vertex and $(v_1, v_2)$ is an edge,
14:         return $\mathsf{cond}(v_1) \wedge \mathbf{TransE}(v_1, v_2)$
15:     If $v_1$ is a nil vertex and $(v_1, v_2)$ is an edge,
16:         return $\mathbf{TransE}(v_1, v_2)$
17:     If $v_1$ is a choice vertex.
18:         Let $(v_1, v_2), \ldots, (v_1, v_m)$ be the choice edges.
19:         return $\mathbf{TransV}(v_2) \vee \ldots \vee \mathbf{TransV}(v_m)$
20: procedure TransE($v_1, v_2$)
21:     If $(v_1, v_2)$ is an AP edge
22:         If $v_1$ is a state vertex
23:             If $\mathsf{cond}(v_1) = \mathsf{True}$,
24:                 return $\mathbf{AX}\,\mathbf{AF}\,\mathbf{TransV}(v_2)$
25:             Else
26:                 return $\mathbf{AX}\,\mathbf{A}\,[\mathsf{cond}(v_1, v_2)\;\mathbf{U}\;\mathbf{TransV}(v_2)]$
27:         Else  // $v_1$ is a nil vertex
28:             If $\mathsf{cond}(v_1) = \mathsf{True}$,
29:                 return $\mathbf{AF}\,\mathbf{TransV}(v_2)$
30:             Else
31:                 return $\mathbf{A}\,[\mathsf{cond}(v_1, v_2)\;\mathbf{U}\;\mathbf{TransV}(v_2)]$
32:     Else If $(v_1, v_2)$ is an EP edge
33:         If $v_1$ is a state vertex
34:             If $\mathsf{cond}(v_1) = \mathsf{True}$,
35:                 return $\mathbf{EX}\,\mathbf{EF}\,\mathbf{TransV}(v_2)$
36:             Else
37:                 return $\mathbf{EX}\,\mathbf{E}\,[\mathsf{cond}(v_1, v_2)\;\mathbf{U}\;\mathbf{TransV}(v_2)]$
38:         Else  // $v_1$ is a nil vertex
39:             If $\mathsf{cond}(v_1) = \mathsf{True}$,
40:                 return $\mathbf{EF}\,\mathbf{TransV}(v_2)$
41:             Else
42:                 return $\mathbf{E}\,[\mathsf{cond}(v_1, v_2)\;\mathbf{U}\;\mathbf{TransV}(v_2)]$
43:     Else If $(v_1, v_2)$ is an AX edge,
44:         return $\mathbf{AX}\,\mathbf{TransV}(v_2)$
45:     Else     // $(v_1, v_2)$ is an EX edge
46:         return $\mathbf{EX}\,\mathbf{TransV}(v_2)$
```

Algorithm A2: Evaluation of a FO-CTL formula ϕ in $M = (S, E, \mathcal{G}, s_0)$ with a cache $\mathbf{C}$.

```
 1: procedure EVAL($\mathsf{node}_i$, $s$, $\theta$)
 2:     /* $\phi$, $M$, $\mathbf{C}$, and $P^{leg}$ are given by Algorithm 1 */
 3:     $t := \mathbf{C}((\mathsf{node}_i, s, \theta), P^{leg}(M, s))$
 4:     If $t$ exists, Return $t$
 5:     If $\mathsf{node}_i = \top$, Then $t := \mathsf{True}$
 6:     If $\mathsf{node}_i = \bot$, Then $t := \mathsf{False}$
 7:     If $\mathsf{node}_i = g$,
 8:         If $g \in L(s)$, Then $t := \mathsf{True}$ Else $t := \mathsf{False}$
 9:     If $\mathsf{node}_i = x$,
10:         If $(x\theta) \in L(s)$, Then $t := \mathsf{True}$ Else $t := \mathsf{False}$
11:     If $\mathsf{node}_i = \neg$,
12:         If EVAL($\mathsf{node}_i^c, s$) = True,
13:             $t := \mathsf{False}$ Else $t := \mathsf{True}$
14:     If $\mathsf{node}_i = op$ where $op$ is either $\wedge, \vee, \Rightarrow$, or $\Leftrightarrow$,
15:         $t :=$ EVAL($\mathsf{node}_i^l, s, \theta$) $op$ EVAL($\mathsf{node}_i^r, s, \theta$)
16:     If $\mathsf{node}_i = \forall x$,
17:         If EVAL($\mathsf{node}_i^c, s, \theta \cup \{x/g\}$) = True for all $g \in \mathbb{G}$,
18:             $t := \mathsf{True}$ Else $t := \mathsf{False}$
19:     If $\mathsf{node}_i = \exists x$,
20:         If EVAL($\mathsf{node}_i^c, s, \theta \cup \{x/g\}$) = True for some $g \in \mathbb{G}$,
21:             $t := \mathsf{True}$ Else $t := \mathsf{False}$
22:     If $\mathsf{node}_i = \mathbf{AF}$ or $\mathsf{node}_i = \mathbf{AG}$,
23:         If EVAL($\mathsf{node}_i, s, \theta$) = True, Then $t = \mathsf{True}$
24:         Else If EVAL($\mathsf{node}_i^c, s', \theta$) = True for all $s' \in E(s)$,
25:             $t := \mathsf{True}$ Else $t := \mathsf{False}$
26:     If $\mathsf{node}_i = \mathbf{EF}$ or $\mathsf{node}_i = \mathbf{EG}$,
27:         If EVAL($\mathsf{node}_i, s, \theta$) = True, Then $t = \mathsf{True}$
28:         Else If EVAL($\mathsf{node}_i^c, s', \theta$) = True for some $s' \in E(s)$
29:             $t := \mathsf{True}$ Else $t := \mathsf{False}$
30:     If $\mathsf{node}_i = \mathbf{AX}$,
31:         If EVAL($\mathsf{node}_i^c, s', \theta$) = True for all $s' \in E(s)$,
32:             $t := \mathsf{True}$ Else $t := \mathsf{False}$
33:     If $\mathsf{node}_i = \mathbf{EX}$,
34:         If EVAL($\mathsf{node}_i^c, s', \theta$) = True for some $s' \in E(s)$,
35:             $t := \mathsf{True}$ Else $t := \mathsf{False}$
36:     If $\mathsf{node}_i = \mathbf{AU}$,
37:         If EVAL($\mathsf{node}_i^r, s, \theta$) = True, Then $t = \mathsf{True}$
38:         Else If EVAL($\mathsf{node}_i^l, s, \theta$) = False, Then $t = \mathsf{False}$
39:         Else If EVAL($\mathsf{node}_i, s', \theta$) = True for all $s' \in E(s)$,
40:             $t := \mathsf{True}$ Else $t := \mathsf{False}$
41:     If $\mathsf{node}_i = \mathbf{EU}$,
42:         If EVAL($\mathsf{node}_i^r, s, \theta$) = True, Then $t = \mathsf{True}$
43:         Else If EVAL($\mathsf{node}_i^l, s, \theta$) = False, Then $t = \mathsf{False}$
44:         Else If EVAL($\mathsf{node}_i, s', \theta$) = True for some $s' \in E(s)$
45:             $t := \mathsf{True}$ Else $t := \mathsf{False}$
46:     Call the cost function of $\mathsf{node}_i$ to compute cost(t)
47:     $\mathbf{C}((\mathsf{node}_i, s, \theta), P^{leg}(M, s)) := t$ with cost(t)
48:     Return $t$ with cost(t).
```