Available online at sjuoz.uoz.edu.krd

Science Journal of University of Zakho

Vol. 9, No. 1, pp. 30– 37, March-2021

p-ISSN: 2663-628X
e-ISSN: 2663-6298

GRAPH INCLUSION AND MATCHING ALGORITHMS FOR PROGRAMS MANIPULATING SINGLY LINKED HEAPS

Muhsin H. Atto [a]

[a] Faculty of Science, University of Zakho, Kurdistan Region, Iraq – (muhsin.atto@uoz.edu.krd)

Received: Nov., 2020 / Accepted: Feb., 2021 / Published: Mar.,2021 ***https://doi.org/10.25271/sjuoz.2021.9.1.778***

ABSTRACT:
Programs that manipulate heaps such as singlylinked lists, doublylinked lists, skiplists, and treesare ubiquitous, and hence ensuring their correctness is of utmost importance. Analysing correctness properties for such programs is not trivial since they induce dynamic data structures, leading to unbounded state spaces with intricate patterns. One approach that has been adopted to tackle this problem is the use of symbolic searching techniques. The state space is encoded using graphs where the nodes represent memory cells, and the edges represent pointers between the cells. It is necessary to prune the search to avoid generating massive numbers of graphs, thus making the procedure unpractical. Pruning strategies are defined based on operations such as graph matching and inclusion. In this paper, a set of algorithms for performing these operations are presented. It is demonstrated that the proposed algorithms can handle typical graphs that arise in the verification of heap manipulating programs.

KEYWORDS: Trees; Heaps; Graph Inclusion; Graph Matching; Software Verification.

1. INTRODUCTION

The design of automatic methods to verify that the given programs are safe and well formed is challenging. The way that these programs are tested requires efficient algorithms and hence different methods are crucial to verify that given programs are safe and well formatted.
This includes merging two programs where the output must be well sorted and structured. Many approaches have been designed for addressing these problems for different kind of programs and using different type of algorithms. Each of these techniques uses special tools for investigating special types of verification properties (Abdulla et al., 2010), (Abdulla et al., 2008b).
Software verification used many techniques and algorithms to verify that merging two programs representing a singly data structure is also a safe program and is well formatted and sorted. These verification methods need algorithms to consider the graphs inclusion and matching, representing given programs (Abdulla et al., 2011), (Giamblanco and Anderson, 2019). Therefore, in this paper, algorithms to verify graphs inclusion and matching for programs representing heaps are designed and implemented.
In general, this paper aims to design some algorithms to test the inclusion and then find matching between two graphs representing a heap. More precisely,the proposed algorithms are designed to decide for two heaps say G 1 and G2 , whether G1 v G2, and then find all possible matching where G2 can be included from the G1.
In this work, heaps are considered as graphs to represent a linked lists with one next pointers. A heap is a graph which follows the following properties: (1) possibly be disconnected and (2) acyclic. Based on this, in this paper, it is assumed that a heap consists of a set of stars and trees. Therefore, in order to test the heap inclusion, new algorithms must be designed to deal with the trees and stars inclusion (Abdulla et al., 2008a), (Wimmer and Lammich, 2018).
The first algorithm to implement is described in (Valiente, 2005). This paper concerns the inclusion of one tree in another. However, this algorithm is improved where the whole target tree can be searched. This is because there might be the case that two small trees are covered by one big tree. Defining the exact nature of such a cover, and finding an algorithm for computing it, is a part of the work presented in this paper.
The rest of the paper is organized as follows: In Section 2, related work, research problem and motivations are given. In Sections 3 and 4, stars and trees matching and inclusion algorithms are described,respectively. Heaps inclusion is illustrated in Section 5.

2. RELATED WORK

Several works considering the verification of singly linked lists with data have been developed. These included new techniques based on algorithm verification for finite graphs. These algorithms were designed to verify the safety of the programs with different factors based on different methods. Graph connectivity, graphs inclusion and matching are the core idea for most of these techniques. However, they used different graph ordering which let to different algorithms and concepts (Abdulla et al., 2010), (Valiente, 2005), (Abdulla et al., 2009) and (Bouajjani, 2005).
The paper (Abdulla et al., 2010) proposed a new verification method for programs representing linked lists. This approach relied on a backward reachability analysis using a term named signatures in order to verify the safety of the given program. This method was another powerful tool and model for verifying designed for the aim to identifying bad heaps which have graphs representing bad lists, such as not sorted and not well organized lists.
The paper (Valiente, 2005) uses tree inclusion algorithm to check if the given pattern tree can be included in the target tree, by using some graph ordering algorithms. This algorithm used bipartite based graphs to verify graphs inclusion. This method stops searching when the first sub tree in the target tree is found. In this paper, this algorithm is improved so that all possible sub trees in the target tree can be found when the given patten tree can be included in each of them. This improvement is necessary when some factors need to be considered before the proper target tree is selected.

Another method to verify software manipulating linked lists is developed in (Abdulla et al., 2008b). This verification technique used a tool called symbolic (backward) reachability analysis based on graphs representing heaps. Different tools was used to find the set of minimal graph patterns corresponding to a set of bad conjurations (Berdine et al., 2007), (Gotsman et al., 2006).
The paper (Abdulla et al., 2011) implemented the use of monotonic abstraction and backward reachability analysis as means of performing programs with multiply pointed structures. This method uses signatures to predicates and define sets of bad configuration based on given heaps (Magill et al., 2007).
Another powerful tool and model for verifying programs based on a special technique is proposed in (Gallardo et al., 2008). This method was designed to check and verify programs written in C programming language with a dynamic memory allocation. This approach used the concept of heap structures to represent different data structures, such as vectors.
A new framework for program verification based on regular model checking is proposed and designed in (Bouajjani, 2005). This approach considered a non-recursive programs manipulating data structures with one next pointers, such as singly linked lists and circular lists. A new technique for calculating and refining the abstractions of the reachable graphs representing given programs was designed in this framework.
An automated approach is presented in (Bouajjani et al., 2006). This technique was aimed to ensure that the given program is safely terminated. The proposed method used a new counter to represent a method, which verifies that if the given program is safe and well structured.

2.1. Problem Statement and Motivation

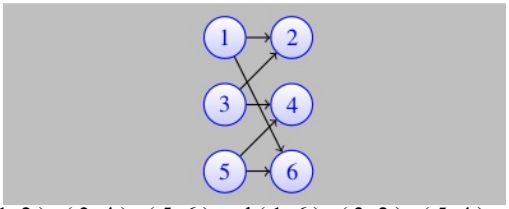
Based on the given related work, non of the given methods stated if the full matching for the graphs inclusion were used. Therefore, non of the given methods can be used when all the possible matching between the pattern and the target graphs are crucial to be considered. This is the case, when given graphs need to consider all possible matching in order to select the best possible match. An example of this case is when the proposed verification method needs to consider linked lists which share some special relations(such as equality and inequality relations).
To find all possible matching between given graphs, new efficient algorithms and methods need to be implemented. This is because given graphs can have many possible matching when the first graph can be included in the second graph. Therefore, the problem which motivated me to propose this work is designing new algorithms where all the possible matching between given graphs (heaps) can be calculated, based on different methods.
Based on this, the following are the novel motivations given in this paper:

1. Modify the algorithm given in (Valiente, 2005) so that is could be used to check the trees inclusion from their roots.
2. Design a new algorithm to check acyclic based graphs inclusion and matching.
3. Design new algorithms to check the heaps inclusion and matching.
4. Implement new methods to find all possible matching between given graphs using the proposed algorithms.

3. TREE MATCHING AND INCLUSION

3.1. Tree Inclusion

In this section, the algorithm given in (Valiente, 2005) is described. This algorithm is designed to solve the trees inclusion. However, this algorithm needs to be improved when different factors are considered. Details about this algorithm and the new improvement will be discussed in the rest of this section.
A tree can be denoted by a graph, where each node has only one path from the root of the the given tree and all nodes can be reached from the root. Each node has one parent and a set of children. The root node has no parent. This means that a tree is a connected and acyclic graph where each node has at most one successor (Wimmer, 2016).
In this section, for given two trees T_1 and T_2, algorithms to verify if T_2 can be included from the T_1 is described. The given algorithm checks a sub tree form the T_1 can be allocated where T_1 could be included and matched. This algorithm depends on the methods given in (Valiente,2005), which involves using bipartite matching problems to test inclusion for the given trees.
Given $p \in T1$ and $t \in T2$, where t has children $t_1, t_2, ..., t_n$ and p has children $p_1, p_2,..., p_m$. In order to determine whether $T1[p]$ can be included in $T2[t]$. It suffices to know if $T1[x] \subseteq T2[y]$, for all $x \in \{p_1, p_2,..., p_m\}$ and $y \in \{t_1,t_2,...,t_n\}$. Morepreciesely, given two vertices $p \in T1$ and $t \in T2$, to decide whether $T1[p]$ v $T2[t]$, construct a bipartite graph $G = (\{t_1,t_2,t_3,...,t_n\} U \{p_1, p_2, p_3,..., p_m\}, E)$ with $(t_j, p_i) \in E$. We say $T1[p]$ can be included from $T2[t]$ iff G has at least one matching. We write S(t) to denote the included sub tree at node $t \in T2$. We compute S(t) as follows: $S(t) = \{p \in V(T1) \mid T1[p]$ v $T2[t]\}$. We say that $T1$ can be included in $T2$ if $root(T1) \in S(t)$.
Bipartite Graph: Given a graph G =(V,E), G is a bipartite graph, if all vertices in G can be separated into two groups where no two vertices within the same group are adjacent (Abdulla et al., 2010). Bipartite graph is used in the proposed algorithms given in this paper. This is to find the proper match between given graphs in order to check the graph inclusion (Asratian et al., 1998). An example of bipartite graph matching is given in figure 1.
Matching in a Bipartite graph: Given a bipartite graph G = (V,E), a possible *matching* in G is a subset of edges $M \subseteq E$, where no edges has both the start and end vertices from the same set. *Hopcroft–Karp algorithm* (Katrenic and Semanisin,2011) is to used to find the matching. The following are two possible matching in the Figure 1.



(1, 2) , (3, 4) , (5, 6) and (1, 6) , (3, 2) , (5, 4) .
Figure 1: Matching in a Bipartite Graph

3.2. Trees Inclusion Algorithm

As mentioned before, a tree inclusion algorithm given in this section is based on the specification given in (Valiente, 2005). This algorithm determines the smallest sub tree in T where P can be included. In this case, the whole target tree is not used to find more sub trees where P can be included. In additional, it may be required to check if P can be included from T, through their roots. These two aspects were not implemented in the algorithm given in (Valiente, 2005).

Based on this, two new algorithms are designed. The first algorithm is modified when the whole target tree is considered to find all possible sub trees in $T1$ such that $T2$ can be included. The second algorithm is designed so that it checks if for given tree $T1$ and tree $T2$, $T1$ can be included from the root of the $T2$. This means that the root of the $T1$ must be matched only to the root of the $T2$, otherwise, $T1$ can not be included in $T2$.

The second algorithm is modified based on the first algorithm where the whole target tree is tested for the trees inclusion until a sub tree is found where the root of the T_1 can be included from the root of the T_2. This means that, this algorithm follows most of the steps given in the first algorithm, except that it returns true if T_1 can be included from the root in the $T2$, otherwise, returns false. The required problems and the design of the modified algorithms are given below.

Algorithm 1: Trees Inclusion

input : A Pattern Tree T_1 and Target Tree T_2
output : $T_1 \sqsubseteq T_2$, Returns true iff T_1 can be included in T_2

for *for all* $t \in V(T_2)$ *in postorder* **do**
  $S(t) = \{ p \in V(T_1) \mid outdeg(p) = 0 and label(t) = label(p) \}$
  if $outdeg(t) \neq 0$ **then**
    Let $t_1, t_2, t_3, \ldots, t_n$ *be children of* t
    $S(t) = S(t_1) \cup S(t_2) \cup S(t_3) \cup, \ldots, \cup S(t_n)$
    for ***all*** *non leaf* $p \in V(T_1)$ *with* $outdeg(p) \leq outdeg(t)$ *in postorder* **do**
      if $p \notin S(t)$ **then**
        Let $p_1, p_2, p_3, \ldots, p_m$ *be children of* p
        Construct a bipartite graph $G = \{t_1, t_2, t_3, \ldots, t_n\} \cup \{p_1, p_2, p_3, \ldots, p_m\}, E)$ *with* $(t_j, p_i) \in E$, *if and only if* $p_i \in S(t_j)$
        if *G has matching with m edges* **then**
          $S(t) := S(t) \cup \{p\}$
          if $root(T_1) \in S(t)$ **then**
            return (*TRUE*)
          end
        end
      end
    end
  end

Algorithm 2: Trees Inclusion from Roots

input : A Pattern Tree T_1 and Target Tree T_2
output : $T_2 \sqsubseteq T$, Returns true iff T_1 can be included in T_2, from roots

for *for all* $t \in V(T_2)$ *in postorder* **do**
  $S(t) = \{ p \in V(T_1) \mid outdeg(p) = 0 and label(t) = label(p) \}$
  if $outdeg(t) \neq 0$ **then**
    Let $t_1, t_2, t_3, \ldots, t_n$ *be children of* t
    $S(t) = S(t_1) \cup S(t_2) \cup S(t_3) \cup, \ldots, \cup S(t_n)$
    for ***all*** *non leaf* $p \in V(T_1)$ *with* $outdeg(p) \leq outdeg(t)$ **do**
      if $p \notin S(t)$ **then**
        Let $p_1, p_2, p_3, \ldots, p_m$ *be children of* p *Construct a bipartite graph* $G = \{t_1, t_2, t_3, \ldots, t_n\} \cup \{p_1, p_2, p_3, \ldots, p_m\}, E)$ *with* $(t_j, p_i) \in E$, *if and only if* $p_i \in S(t_j)$
        if *G has matching with m edges* **then**
          $S(t) := S(t) \cup \{p\}$
          if $root(T_2) \in S(t)$ *and* $t = root(T_2)$ **then**
            return (*TRUE*)
          end
        end
      end
    end
  end
end
return (*FALSE*)

Problem Definition 1: Let T1 =(V,E) be a pattern tree and T2 =(V,E) be a target tree, this algorithm is designed to determine a smallest sub tree in T2 whether T1 can be included. For example, given a target tree T1 and a list of pattern trees P1, P2, P3, P4 and P5, shown below, check if any of the given pattern trees can be included from the given target tree, using the algorithm 1.

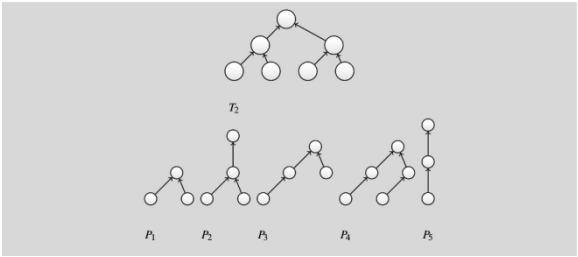


Figure 2: Examples of Trees Inclusion

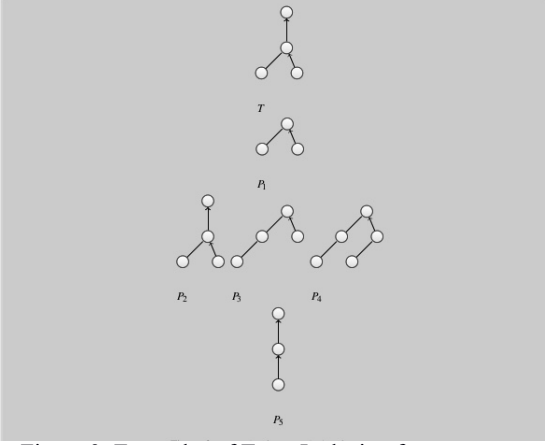


Figure 3: Examples of Trees Inclusion,from roots

Problem Definition 2: Given trees T , and a list of trees in P, shown below, decide whether any given P can be included in T, from roots, using the algorithm 2..

3.3. A complexity of the Modified Algorithm

The second algorithm 2 is required where all the matching between given trees are needed. In this case, this algorithm finds all possible matching between given trees from their roots. Therefore, the complexity of the modified algorithm is based on number of the sub trees from the target tree which pattern tree can be included. Let assume that complexity of the algorithm 1 is **C**, then the modified algorithm could test trees inclusion in a cost = **C** * a, where a is the number of matching between given trees, from roots. This led us to the conclusion that the modified algorithm does not increase the complexity of the algorithm 1.

3.4. Tree Matching

In the graph theory, matching is a set of edges without common vertices. Matching is required when all possible cases are considered to find the perfect matching between given graphs (Katrenic and Semanisin, 2011), (Duan et al., 2018). In this paper, all possible match between given graphs is considered. This is the novel contribution for this work over the related works.

All possible matching is computed based on two different concepts. These concepts are called *Children propagation* and *Matching Propagation*. Therefore, new algorithms to find all possible matching for two trees, if there is any, are given in this section. The required notations and terminologies of the designed algorithms are given, then the details about trees matching algorithms with some examples are described.

3.4.1. Notations and Terminologies

Matching Concept: If we have two trees, T 1 and T 2 , in order to find matching for T 1 from T 2 , we have to find all possible sub trees from the T 2 where T 1 can be included. Matching is assumed to be a set of edges where each vertex from the pattern tree has a corresponding vertex from the target tree.

Therefore, new algorithms to find all possible matching for two trees, if there is any, are given in this section. The required notations and terminologies of the designed algorithms are given, then the details about trees matching algorithm with some examples are described. These two concepts are described below.

As mentioned before, Children Propagation and Matching Propagation are used to find all possible matching for given trees. These two methods are given below.

Children Propagation: Given two trees T1 and T2 , Let t1 , t2 ,t3 , . . . , tn be children of a t ∈ T2 . It is assumed that **r** to be a two dimensional array that will contain matching between each vertex in T1 and corresponding vertex in T2. The children propagation is defined for a given vertex p ∈ T1 and vertex t ∈ T2 as a set of matching between t and children of p. More precisely, children propagation is computed as follows: **Children propagation**:= r[p,t1] ∪ r [p,t2] ∪, . . . , ∪r [p,tn] , as shown in the algorithm 3.

Algorithm 3: Children Propagation

input :Trees T_1 and T_2
output :Children propagation for T_1 from T_2

Let p is root of the T_1 and $\{t_1,t_2,...,t_n\}$ are all sub trees in T_2 where p can be included based on the algorithm 1

if *p and t are leaf nodes* **then**
  $ChildrenPropagation(p,t) = (p,t)$ *iff p can be included in t*
end
if *p is leaf and t is non leaf* **then**
  for $c \in children(t)$ **do**
    $ChildrenPropagation(p,t) = (p,t) \cup ChildrenPropagation(p,c)$ *iff p can be included in t*
  end
end
if *p and t are non leaves* **then**
  for $c \in children(t)$ *and* $pp \in children(p)$ **do**
    $ChildrenPropagation(p,t) = (p,t) \cup ChildrenPropagation(c,pp)$ *iff p can be included in t*
  end
end

Matching Propagation: Given two trees T1 and T2 and t∈T 1 , p ∈ T2 , let t 1 , t2 , t3 , . . . , t n be children of t, and p1, p2 , p 3 , . . . , pm be a children of p. Suppose that **r** is a two dimensional array will contain all possible matching for T1 from T2 . A bipartite graph G =(V,E), where V : = { p 1 , p 2 , p 3 , . . . , pm } ∪ {t1 ,t2 ,t3 , . . . ,tn } is constructed with (pi ,tj) ∈ E, if and only if r [pi ,tj] = 0, where 1 ≤ i ≤ m and 1 ≤ j ≤ n.

A two dimensional array **M** is defined to contain all possible matching for graph G. Matching propagation is defined for p and t as a set of matching between children of t and p. For each match ((p1 ,t1) , ..., (pm ,tn)) ∈ M, matching propagations is computed as follows:

Matching propagation := r [p 1 ,t 1] ⊕r [p 2 ,t 2] ⊕, . . . , ⊕r [pm ,tn] , where ⊕ defined as follows: Given two sets A and B, A ⊕ B = { a.b | a ∈ A and b ∈ B }. New algorithms for these methods are designed in this section, as shown in the algorithm 4.

Algorithm 4: Matching Propagation

input :A list of the children propagation generated uisng the algorithm 3
output : A list of matching for P from T, results will be in M[p][t]

Let $\{(v_1,w_1),(v_2,w_2),...,(v_n,w_m)\}$ be the result of children propagation from the children propagation given in algorithm 3
Let $G = (V_P \cup V_T, E)$ be a bipartite graph where $V_P = \{v_1,v_2,v_3,...,v_p\} \in P$
$V_T = \{w_1,w_2,w_3,...,w_t\} \in T$
$(v_i,w_j) \in E$, *iff* $ChildrenPropogation(v_i,w_j)$ *is not empty*
if *There is matching in G with p edges* **then**
  $M[v,w]$ = all edges in the matching in G
else
  $M[v,w] = \emptyset$
end

Poblem Definition: Let P = (V , E) be a pattern tree and T = (V , E) be a target tree, the problem is to find all possible matching for P from T, using the algorithm 3.

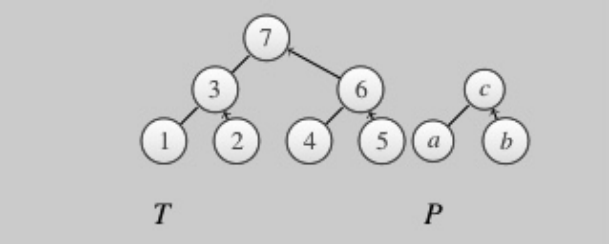


Figure 4: Trees Matching

The following are all possible matching between matching for given pattern tree from the given target the pattern and target trees , as shown in the Figure 4. As shown below, there are **24** possible matching:

{(a,1),(b,2),(c,3)},{(a,1),(b,4),(c,7)},(a,1),(b,5),(c,7)},{(a,1),(b,6),(c,7)}{(a,2),(b,4),(c,7)},{(a,2),(b,5),(c,7)},{(a,2),(b,6),(c,7)},{(a,2),(b,4),(c,7)},{(a,2),(b,1),(c,3)},{(a,3),(b,5),(c,7)},{(a,3),(b,6),(c,7)},{(a,3),(b,4),(c,7)},{(a,4),(b,6),(c,5)},{(a,4),(b,2),(c,7)},{(a,4),(b,1),(c,7)},{(a,4),(b,6),(c,7)},{(a,6), (b,4),(c,5)},{(a,5),(b,1),(c,7)},{(a,5),(b,2),(c,7)},{(a,5),(b,3),(c,5)},{(a,6),(b,4),(c,7)},{(a,6),(b,3),(c,7)},{(a,6),(b,1),(c,7)},{(a,6),(b,2),(c,7)}}

4. STARS MATCHING AND INCLUSION

In order to test the heaps inclusion, new algorithms to test star graphs representing the heaps are required. Therefore, in this section, a new algorithm to deal with stars inclusion problems is designed. This algorithm tests whether given pattern star is included in the target star. In this case, this algorithm must accept trees as input, rather than stars. All notations and concepts used in the proposed algorithm are defined, then the implementation of the stars inclusion and matching algorithms, with some definitions and examples, are described.

A star is a connected graph where all nodes or vertices in the given graph has maximum one parent. This means that graphs representing stars could have no root. Therefore, a star can be a set of trees where their roots are connected through a simple cycle.

4.1. Stars Inclusion

Notations and terminologies: The two stars given in the proposed stars inclusion and matching algorithms are called by pattern star and target star. Pattern star is represented by P and the target star is represented by T. Given a star S(V,E),*where V is a set of vertices and E is a set of edges, V is given by V(S) and E is given by E(S), respectively. A list of nodes in the cycle for the given S is denoted by Cycle(S). The operations to get a successor of a node t ∈ S is denoted by S.succ(t). The sub-tree rooted at a node t is denoted by S[t].*

Preprocessing: To test the proposed star inclusion algorithm, the following operations are required, before the proposed algorithm is applied:

1. **Alg(T1,T2):** is considered to be the algorithm to check trees inclusion from the roots. This algorithm returns true if given trees can be included from their roots.
2. **Extract Trees from a Star:** Before stars inclusion algorithm can be applied to the given stars, both the target star and the pattern star need to be extracted and converted to a list of trees. A new algorithm is required to extract trees from the given stars,

where these trees are connected through a cycle. Based on this, a new algorithm to do this is designed in this paper and its implementation is given below.

Algorithm 5: Extract Trees From Stars Algorithm

```
input  : A Star S
output : A list of trees which are connected from their roots in a simple cycle for the star S. Assume
         that List T will contain these trees
if Cycle(S) is empty then
    S is a tree and then returns S
    return (S)
else
    Assume that Cycle(S) = {t1,t2,t3...,tn} are a set of vertices belong to the cycle in the S
    for each v ∈ Cycle(T) do
        Let Children(v) = {c1,c2,c3,...,cn} are a set of children of v.  Let S.deleteEdge(v1,v2)
        is a method to delete an edge starting from the vertex v1 to vertex v2 in S
        for each c ∈ Children(v) do
            if c ∈ Cycle(S) then
                T.add(S.deleteEdge(v,c))
            end
        end
    end
end
```

4.1.1. Stars Inclusion Algorithm

A new algorithm to test whether a star is included in another star is implemented in this paper. The proposed algorithm depends on the list of the extracted trees from both stars in order to decide if given stars can be included. This means that given stars can be included if only a list of the trees from the first star can be included from the trees in the second star, using tree inclusion algorithms, given in the previous sections.

Let assume that L_1 is a set of trees for the star and L_2 is a set of trees for the target star. In order to test the inclusion of L1 from L2 , it is important to check that for each tree p ∈ L1 there is a tree t ∈ L2 , such that p can be included in t, from their roots. If it is found that all trees in L1 can be included in L2 , then the given star algorithm returns true, otherwise, it returns false.

This algorithm accepts trees as input. This means that if P is a tree and T is star, this algorithm converts T into a set of trees based on the extract trees algorithm given above. It returns true if and only if given tree P can be included in one of the extracted trees from T, sing the tree inclusion algorithms, denoted by **AlgRoot (P, T)** .

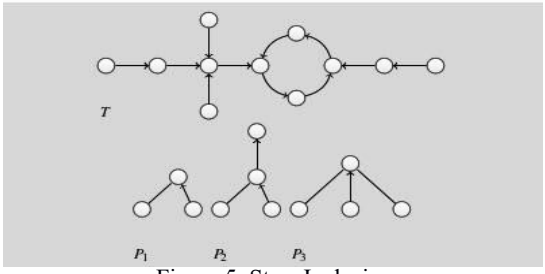


Figure 5: Stars Inclusion

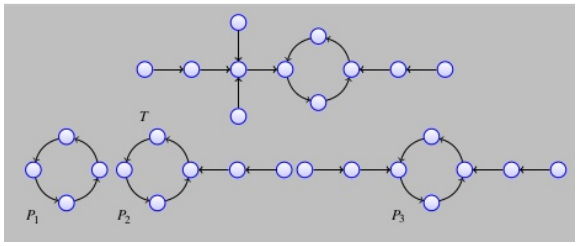


Figure 6: A Tree to Star Inclusion

Problem Definition 1: Let P =(V,E) be a pattern star and T = (V,E) be a target star, the algorithm given in this section decides whether P can be included in T. An example of the star to star inclusion is given in the Figure 5.

Problem Definition 2: Given a tree P and a star T, as shown in the figure 6, the problem defined by the algorithm defined in this section is to decide whether P can be included in T.

2. Stars Matching

The star Matching algorithm is designed to find all matching between stars P and T where P can be included in T. In this section, given two stars S_1 and S_2, the algorithms to find all possible matching between S_2 that S_1 are described.

A list of possible matching for given stars can be obtained based on the trees matching algorithm. This algorithm finds a set of matching for the extracted trees from stars P and T. A bipartie graph based on P and T is constructed and then all possible matching are generated when both the extracted trees can be matched between P and T. More details about this algorithm is given below.

Given two stars, S_1, S_2, to find all possible matching for S_1 from S_2, the following operations are required:

- **TreeToTree(T_1,T_2)** is used to find possible matching for given stars P and T, based on trees given in L_1 from L_2.
- To find all possible matching for S_1 from S_2, all possible combinations between L_1 and L_2 are computed, based on the trees inclusion. As set of possible matching for the given pattern star from the given target star are shown below: $\{\{(a,3),(b,4)\},\{(a,4),(b,3)\}\}$.

A list of trees from both S_1 and S_2, let say L_1 and L_2 respectively, using tress extract algorithm, are extracted.

Problem Definition 3: Given two stars P and T, shown in the Figure 7, Stars Matching algorithm finds all possible matching for P from T, only where P can be included from the T.

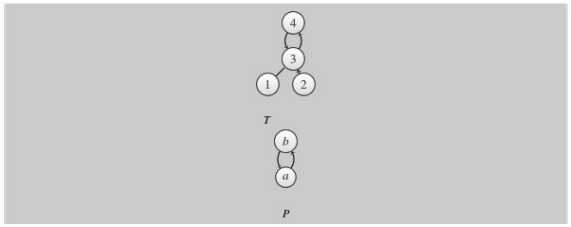


Figure 7: Stars Matching

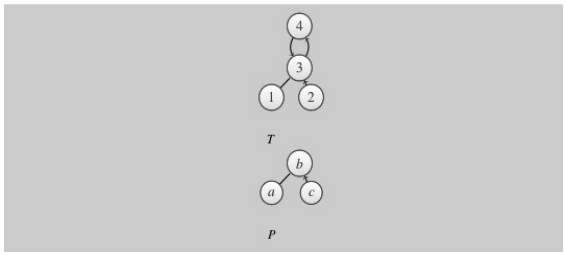


Figure 8: A Tree to Star Matching

Problem Definition 4: Given a tree P and a star T, shown in the Figure 8, the problem defined by this algorithm is to find all possible matching for P from S. The following are a set of matching for the given pattern tree from the given target star.

$\{(a,1),(b,2),(c,3)\},\{(a,1),(b,2),(c,4)\},\{(a,1),(b,4),(c,3)\},\{(a,2),(b,1),(c,3)\},\{(a,4),(b,1),(c,3)\},\{(a,2),(b,4),(c,3)\}$ $\{(a,4),(b,2),(c,3)\}$

5. HEAPS INCLUSION ALGORITHMS

In this section, a new algorithm to test whether a heap is included in another heap, is implemented. First, heaps and their representations are given, then the ordering relations between cells in heaps are described.

All notations and terminologies need to be used in heap inclusion algorithm, are outlined. This paper aims to design new algorithm to test the heaps inclusion, based on the stars and trees inclusion and matching algorithms.

A heap is represented as a graph, where this graph is a set of trees and stars. In summary, a graph H is said to be a heap if the following properties are satisfied:

1. Is a set of trees.
2. Is a set of stars.
3. Combination of trees and stars.

An example of a graph representing the heap is given in the Figure 9.

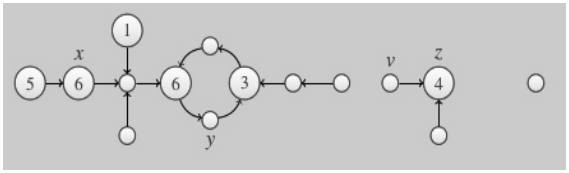


Figure 9: An example of Graph representing Heap

5.1. Heaps Ordering and Representation

Ordering relations between cells in a heap are represented by the following graphs:

1. **Equality Graph:** The equality ordering relations between vertices in the graph that represents a heap is defined by Equality graph. A set of edges in this graph represent the equality relations (if any) in the graph represents a corresponding heap.
2. **Inequality Graph:** The inequality ordering relations between vertices in the graph, that represents a heap, is denoted by an Inequality relation , where edges in this graph represent the Inequality relations in the graph that represents a corresponding heap.
3. **Structural Graph:** This graph represents the structure the heap corresponding to the given graph.

The inclusion algorithm for two components (trees or stars), in which defined in the previews sections, is denoted by **StarToStar (P, T)** , where P is a pattern star and T is a target star. A **TreeToTree (P, T)** , where P is a pattern tree and T is a target tree. It is assumed that a heap is combination of trees and stars. Given a heap H, the method to find a set of components for H is denoted by **getComponent (H)** .

5.2. Heap Inclusion Algorithm

As mentioned before, a new algorithm to test whether a heap is included in another heap, is implemented. Given two heaps H1, H2, a problem given by the proposed heap inclusion algorithm is to check if given H1 can be included from the H2, using the stars and trees inclusion and matching algorithms.

More precisely, in order to test the inclusion H1 in H2, there must be a set of vertices in H2, such that all vertices and edges from all graphs that represent ordering relations in H 1 are included. All components from H1 and H2 are required to be matched. The solution is to find the matching between all the components given in both heaps. Due to this, new algorithms to find all possible matching between two stars and trees were designed in this paper.

In order to find all possible matching for H1 from H2, a new bipartite graph G = (S 1 ∪ S 2 , E), where S1 and S2 are the set of components from H1 and H2, respectively, using **getComponenets** operation described above, are constructed. (s i , s j) ∈ E if and only if the matching for si from sj is not empty, where si ∈ S1 and sj ∈ S2.

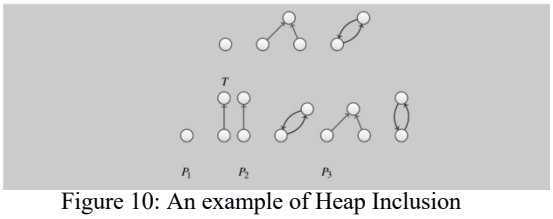


Figure 10: An example of Heap Inclusion

Algorithm 7: Heaps Inclusion Algorithm

input : A Pattern Heap P and Target Heap T
output : $P \sqsubseteq T$, Returns true iff P can be included in T

Assume that $Star(P) = \{p_1, p_2, p_3, \ldots, p_n\}$
and $Star(T) = \{t_1, t_2, t_3, \ldots t_m\}$
are a set of components from heaps P and T
Let $G = (V, E)$ *is a bipartite graph,*
where $V = Star(P) \cup Star(T)$ *and* $(p_i, t_j) \in E$,
if and only if star p_i *can be included in the star* t_j
using the algorithm 6
if *there is a matching in G with n edges* **then**
  return (*TRUE*)
end
return (*FALSE*)

Figure 10: Heaps Inclusion

6. RESULTS FROM THE PROPOSED ALGORITHMS

In this section, the proposed algorithms were tested based on the given heaps. The first heap is called pattern heap and denoted by P. The second heap is called the target heap and it is denoted by T . The heaps P and T consist of a set of trees and stars.

Therefore, when given heaps were tested, both stars and trees inclusion and matching algorithms were involved. When the given heap algorithm finds that P can be included from the T, then all possible matching between the heap P and T were calculated, based on the stars and trees algorithms.

Before results from the given heaps inclusion are given, each component from the heap P with the corresponding component from the heap T, are shown.

6.1. Result from the Heap to Heap inclusion and Matching

The heaps (Pattern and Target) given in figures 11 and 12 were tested based on the algorithms proposed in this paper. A heap inclusion uses all of the stars and trees inclusion algorithms where given heaps were tested. This is due to the fact that heap is a graph represented by a set of components, as mentioned before. Each of these component is either a star or a tree.

Therefore, in order to test the heaps inclusion algorithm, the trees inclusion and stars inclusion algorithms must be used. This means that, the results given in this section includes all the algorithms proposed in this paper.

The graphs given in the Figures 11 and 12 are the pattern and target heaps which were given to the heaps inclusion. Each of these heaps consists of a set of stars and trees. Therefore, given stars in the pattern heap must be matched in the corresponding stars in the target stars. Trees must be matched from the pattern heap to the target heap.

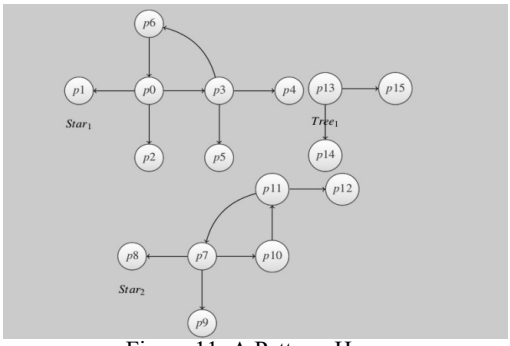


Figure 11: A Pattern Heap

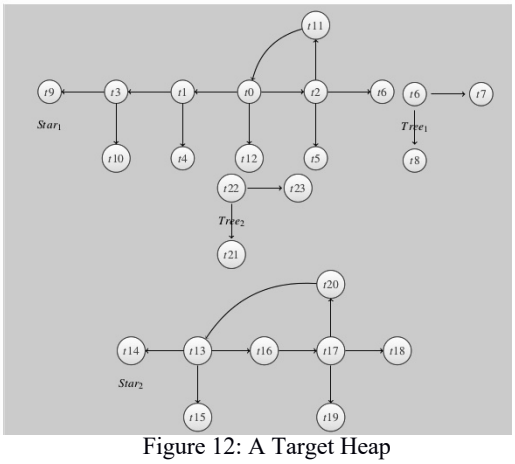


Figure 12: A Target Heap

The following are result of the proposed algorithms given in this paper:

A. Result of Matching Between the star1 in P and the star1 in T

- Based on the graphs given above, the following are result of matching of the tree p6 in the heap P from the sub-tree t11 in heap T. **{ (p6,t11) }**

- Results of matching for the tree p3 in P from the tree t0 in T, are the following:

{ (p5,t4)(p4,t10)(p3,t0) } , { (p4,t10)(p5,t12)(p3,t0) } { (p4,t10)(p5,t9)(p3,t0) } , { (p4,t10)(p3,t0)(p5,t3) } { (p4,t10)(p3,t0)(p5,t1) } , { (p5,t4)(p4,t1)(p3,t0) } { (p5,t10)(p4,t1)(p3,t0) } , { (p4,t1)(p5,t12)(p3,t0) } { (p4,t1)(p5,t9)(p3,t0) } , { (p4,t1)(p3,t0)(p5,t3) } { (p5,t10)(p4,t4)(p3,t0) } , { (p5,t12)(p4,t4)(p3,t0) } { (p4,t4)(p5,t9)(p3,t0) } , { (p4,t4)(p3,t0)(p5,t3) } { (p4,t4)(p3,t0)(p5,t1) } , { (p5,t4)(p3,t0)(p4,t9) } { (p5,t10)(p3,t0)(p4,t9) } , { (p5,t12)(p3,t0)(p4,t9) } { (p3,t0)(p5,t3)(p4,t9) }

- Result of matching between the tree p0 in P from the tree t2 in T, are the following:

{ (p1,t7)(p2,t6)(p0,t2) } , { (p1,t7)(p2,t8)(p0,t2) }
{ (p1,t7)(p0,t2)(p2,t5) } , { (p1,t5)(p2,t6)(p0,t2) }
{ (p1,t5)(p2,t7)(p0,t2) } , { (p1,t5)(p2,t8)(p0,t2) }
{ (p2,t6)(p0,t2)(p1,t8) } , { (p2,t7)(p0,t2)(p1,t8) }
{ (p0,t2)(p1,t8)(p2,t5) } , { (p2,t7)(p0,t2)(p1,t6) }
{ (p2,t8)(p0,t2)(p1,t6) } , { (p0,t2)(p2,t5)(p1,t6) }

B. Result of Matching between the tree1 in P and the tree2 in T

Result of the matching between p13 in the P from t22 in the T, are the following:

{ (p14,t21)(p13,t22)(p15,t23) } , { (p15,t21)(p14,t23)(p13,t22) }

C. Result of Matching Between the tree1 in P and the star1 in T.

- Result of matching between the tree p13 in P and the tree t2 in T, are the following:

{ (p13,t2)(p14,t5)(p15,t8)},{(p13,t2)(p14,t5)(p15,t7) }{ (p13,t2)(p14,t5)(p15,t6)},{(p13,t2)(p15,t5)(p14,t7) }{ (p13,t2)(p14,t7)(p15,t8) }, {(p13,t2)(p14,t7)(p15,t6) }{ (p13,t2)(p15,t5)(p14,t6)} ,{(p13,t2)(p14,t6)(p15,t8)}{(p13,t2)(p14,t6)(p15,t7)}, {(p13,t2)(p15,t5)(p14,t8)}{(p13,t2)(p14,t8)(p15,t7)},{(p13,t2)(p14,t8)(p15,t6) }

D. Result of Matching Between the tree1 in P and the star2 in T.

Result of matching between the tree p13 in P from the tree t17 in T, are the following:

{(p14,t18)(p15,t19)(p13,t17)},{(p13,t17)(p15,t18)(p14,t19) }

E. Result of Matching Between the star2 in P and the star1 in T.

- Results of matching between the tree p7 in P from the t0 in T, are the following:

{ (p8,t9)(p9,t3) , (p7,t0) } , { (p8,t9)(p7,t0)(p9,t10) }{ (p8,t9)(p7,t0)(p9,t4) } , { (p8,t9)(p7,t0)(p9,t1) } { (p8,t9)(p7,t0)(p9,t12) } , { (p8,t12)(p9,t9)(p7,t0) } { (p8,t12)(p9,t3)(p7,t0) } , { (p8,t12)(p7,t0)(p9,t10) } { (p8,t12)(p7,t0)(p9,t4) } , { (p8,t12)(p7,t0)(p9,t1) } { (p8,t10) , p9,t9)(p7,t0) } , { (p8,t10)(p9,t3)(p7,t0) } { (p8,t10)(p7,t0)(p9,t4) } , { (p8,t10)(p7,t0)(p9,t1) } { (p8,t10)(p7,t0)(p9,t12) }

- Results of matching between the tree p10 in P from the t11 in T, are the following: { (p10,t11) }

- Results of matching between the treep11 in P from the t2 in T, are the following:

{ (p11,t2)(p12,t5)}, {(p11,t2)(p12,t8)}
{ (p11,t2)(p12,t6) } , {(p11,t2)(p12,t7) }

F. Result of Matching Between the star 2 in P and the star 2 in T.

- Results of matching between the tree p7 in P from the t13 in T, are the following:

{ (p8,t14)(p9,t15)(p7,t13)},{ (p7,t13)(p8,t15)(p9,t14) }

- Results of matching fro the treep11 in P from the tree t17 in T, are the following:

{ (p11,t17)(p12,t18) }, { (p11,t17)(p12,t19) }

- Results of matching between the tree p10 in P and the t20 in T, are the following: { (p10,t20) }

6.1.2. Result Discussion

As shown above, given algorithms are able to compute all possible matching between the given pattern and target heaps. This concludes that the proposed algorithms can meet their purposes, as required. This means that the problems given in this paper are solved, based on the proposed algorithms.

7. CONCLUSIONS AND FUTURE WORK

A number of algorithms to verify the inclusion of graphs representing program that manipulate dynamic data structure are implemented in this paper. It is assumed that the given program has a simple data structure with a single next pointer, such as single linked lists and circular linked lists. The problems that have verified and described in the proposed algorithms were graph inclusion and matching. Based on this, some algorithms were designed in this paper to deal with inclusion and matching for different graphs, representing heaps, were implemented in this paper.

The results from given algorithms summarized and showed in this paper. Given results showed that the given algorithms are able to test the inclusion and matching for the given heaps. One future plan for this work is to extend the proposed algorithms on a program that manipulate more complex graphs so that different

data structures can be considered. For instance doubly linked list and tree like structures.

REFERENCES

Abdulla, P., Atto, M., Cederberg, J., and Ji, R. (2010). Automatic verification of dynamic data-dependent programs. LNCS, 5799:1–25.

Abdulla, P., Bouajjani, A., Cederberg, J., Haziza, F., Ji, R., and Rezine, A. (2008a). "shape analysis via monotonic abstraction". Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

Abdulla, P., Bouajjani, A., Cederberg, J., Haziza, F., and Rezine, A. (2008b). "monotonic abstraction for programs with dynamic memory heaps",lncs. volume 5123, pages 341–354.

Abdulla, P., Cederberg, J., and Vojnar, T. (2011). "monotonic abstraction for programs with multiply-linked structures",lncs. volume 6945, pages 125–138.

Abdulla, P., Delzanno, G., Henda, N., and Rezine, A. (2009). Monotonic abstraction: on efficient verification of parameterized systems. Int. J. Found. Comput. Sci., 20:779–801.

Asratian, A. S., Denley, T. M. J., and Häggkvist, R. (1998). Bipartite Graphs and Their Applications. USA.

Berdine, J., Calcagno, C., Cook, B., Distefano, D., O'Hearn, P. W., Wies, T., and Yang, H. (2007). Shape analysis forcomposite data structures. In Damm, W. and Hermanns, H., editors, Computer Aided Verification, pages 178–192, Berlin,Heidelberg. Springer Berlin Heidelberg.

Bouajjani, A. (2005). "regular model checking for programs with dynamic memory". 1:17–22.

Bouajjani, A., Bozga, M., Habermehl, P., Iosif, R., Moro, P., and Vojnar, T. (2006). "programs with lists are counter automata". In Ball, T. and Jones, R. B., editors, "Computer Aided Verification", pages 517–531, Berlin, Heidelberg. Springer Berlin Heidelberg.

Duan, R., Pettie, S., and Su, H.-H. (2018). Scaling algorithms for weighted matching in general graphs. ACM Trans. Algorithms, 14(1).

Gallardo, M.-d.-M., Merino, P., and Sanán, D. (2008). "model checking c programs with dynamic memory allocation". pages 219 – 226.

Giamblanco, N. and Anderson, J. (2019). "asap: Automatic sizing and partitioning for dynamic memory heaps in high-level synthesis", icfpt. pages 275–278.

Gotsman, A., Berdine, J., and Cook, B. (2006). Interprocedural shape analysis with separated heap abstractions. In Yi, K., editor, Static Analysis, pages 240–260, Berlin, Heidelberg.

Katrenic, J. and Semanisin, G. (2011). A generalization of hopcroft-karp algorithm for semi-matchings and covers in bipartite graphs. CoRR, abs/1103.1091.

Magill, S., Berdine, J., Clarke, E., and Cook, B. (2007). Arithmetic strengthening for shape analysis. In Nielson, H. R. and Filé, G., editors, Static Analysis, pages 419–436, Berlin, Heidelberg.

Valiente, G. (2005). Constrained tree inclusion. Journal of Discrete Algorithms, 3(2):431 – 447.

Wimmer, S. (2016). Timed automata. Archive of Formal Proofs.

Wimmer, S. and Lammich, P. (2018). Verified model checking of timed automata. In Beyer, D. and Huisman, M., editors, Tools and Algorithms for the Construction and Analysis of Systems, pages 61–78, Cham. Springer International Publishing.