On Recognition of Replicated And-Or Series-Parallel Digraphs

Jichiang Tsai and De-Ron Liang

Institute of Information Science
Academia Sinica
Taipei, Taiwan

Address of Correspondent:
Jichiang Tsai
Institute of Information Science 20
Academia Sinica
Taipei 115, Taiwan

TEL: (886-2)27883799 ext. 2411
FAX: (886-2)27824814
Email: jctsai@iis.sinica.edu.tw

Abstract- The computation task of a distributed processing system usually can be partitioned into a set of modules and then modeled as a directed graph, called the *task digraph*. In the task digraph, vertices represent modules and arcs represent message passing links between two modules. Particularly, according to the logical structures and precedence relationships among modules, a large class of task digraphs can be modeled as *And-Or Series-Parallel (AOSP)* digraphs. For this type of digraph, we can calculate its task reliability and task response time in linear time; whereas these problems are known to be NP-hard for general digraphs. Hence, it becomes a useful work for evaluating computation tasks to examine if a task digraph is an AOSP digraph. A polynomial time algorithm of recognizing AOSP digraphs has been proposed in [1] earlier. In this paper, we consider the fault-tolerant variants of AOSP digraphs, named *Replicated And-Or Series-Parallel (RAOSP)* digraphs, which are obtained from AOSP digraphs by adding replications to each vertex and adding proper arcs between two vertices. For RAOSP digraphs, we can also calculate its task reliability and task response time in linear time with the similar methods of AOSP digraphs. So it is another important work to recognize RAOSP digraphs. The existing polynomial time recognition algorithm for AOSP digraphs does not apply here since it can be shown that RAOSP digraphs are not AOSP digraphs. To make up this deficiency, we will propose a polynomial time algorithm for recognizing RAOSP digraphs in the context. Moreover, it will be first described how to recognize a special subclass of RAOSP digraphs, *Replicated Edge Series-Parallel (RESP)* digraphs, in polynomial time as a preliminary step.

Keywords: Task digraphs, AOSP digraphs, graph recognition, fault tolerance, distributed processing systems.

1. Introduction

In the past decade, distributed processing systems have become increasingly popular because they provide cost-effective means for resource sharing and extensibility, and obtain potential increases in performance, reliability, fault tolerance and resource utilization [2] - [4]. Usually, the computation task of a distributed processing system can be partitioned into a set of software modules (or simply, *modules*) and then modeled as a directed graph, called the *task digraph*. In such a digraph, vertices represent modules and arcs represent message passing links between two modules.

Since job decomposition and mergence are two major operators in distributed programming, a large class of task digraphs can be expressed by the combination of three common types of subgraphs based on the logical structures and precedence relationships among modules [5, 6]: *sequential*, *And-Fork to And-Join (AFAJ)* and *Or-Fork to Or-Join (OFOJ)*, where AFAJ and OFOJ subgraphs may consist of several sequential subgraphs in a parallel structure. These three types of subgraphs are depicted in Figure 1. The sequential subgraph contains a sequence of modules executed in series. Each module except the last has a single successor. As for the AFAJ subgraph, it begins from a module which simultaneously enables several succeeding modules and ends at a module which is enabled only when all of its preceding modules have completed their executions. On the contrary, the beginning module of the OFOJ subgraph enables one of its succeeding modules, and the ending module can be enabled by any one of its preceding modules. In [7], this large class of task digraphs has been modeled as *And-Or Series-Parallel (AOSP)* digraphs. Such a graph model is acyclic. If a computation contains a loop, it can be unrolled and different instances of the loop body can be represented by different modules. The same technique can also apply to recursive structures by first determining the mean number of recursive calls. Using this technique together with others in [8], we can convert cyclic graphs to acyclic graphs.

There is a certain probability, called the *task reliability*, associated with the event that a task completes successfully. This measure accurately models the reliability of a task running in the system. The task reliability problem is known to be NP-hard for general digraphs. But for AOSP digraphs, task reliability can be found in linear time

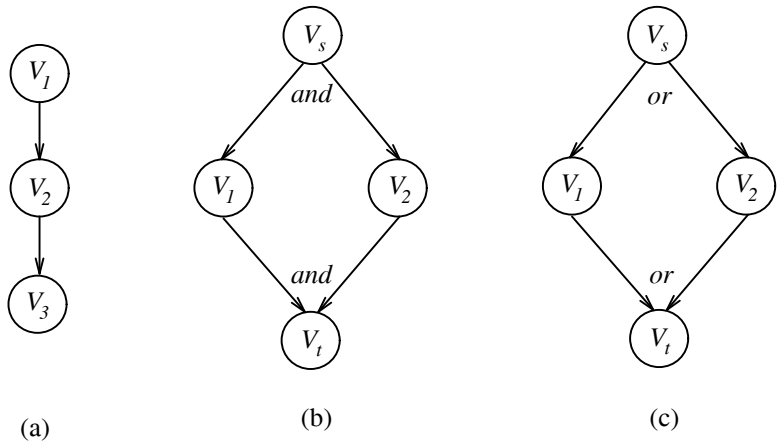


Figure 1: (a)Sequential subgraph; (b)And-Fork to And-Join subgraph; (c)Or-Fork to Or-Join subgraph.

using the technique proposed in [7]. Moreover, *task response time* is an important design criterion for real-time computer systems. It is the time from an invocation of the application task to the completion of its execution. A new analytic model developed in [5] is able to precisely estimate task response time of AOSP digraphs in linear time, instead of time-consuming simulation methods. Hence, it becomes a useful work for evaluating computation tasks to examine if a task digraph is an AOSP digraph. Previously, the recognition for *Edge Series-Parallel (ESP)* digraphs, which arise in the analysis of electrical networks [9] - [11], was proposed in [12]. The ESP digraph is a special case of the AOSP digrpah, and contains only two types of subgraphs: *sequential* and *Fork to Join*. Namely, ESP digraphs do not take the logic structures among modules into consideration. Obviously, ESP digraphs can not satisfy modern varieties of distributed computation tasks. To make up this deficiency, a polynomial time algorithm to recognize AOSP digraphs has been already proposed in [1].

Modules and communication links may fail due to two main factors: software failures and hardware failures. In order to increase the survival rate of the task, a straight forward method is to replicate the complete task several times, and execute them independently on distinct computers. Primary site approach [13] is one such example. The disadvantage of this approach is that the system cannot tolerate more than one fault in each replicated task. Recently, replication of software modules has been proposed and

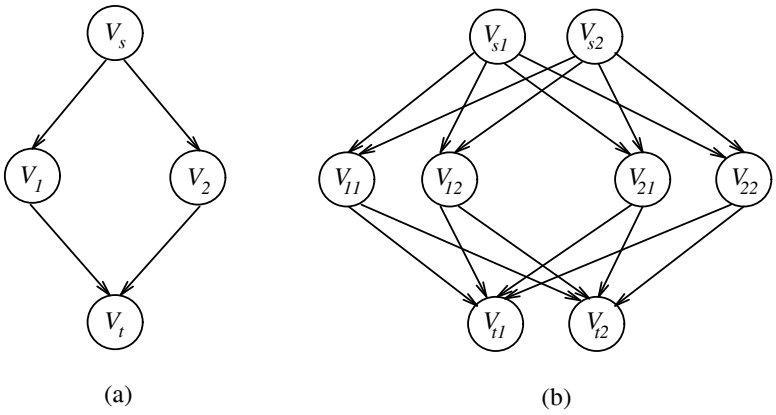


Figure 2: (a)An AFAJ task digraph; (b)Its replicated digraph.

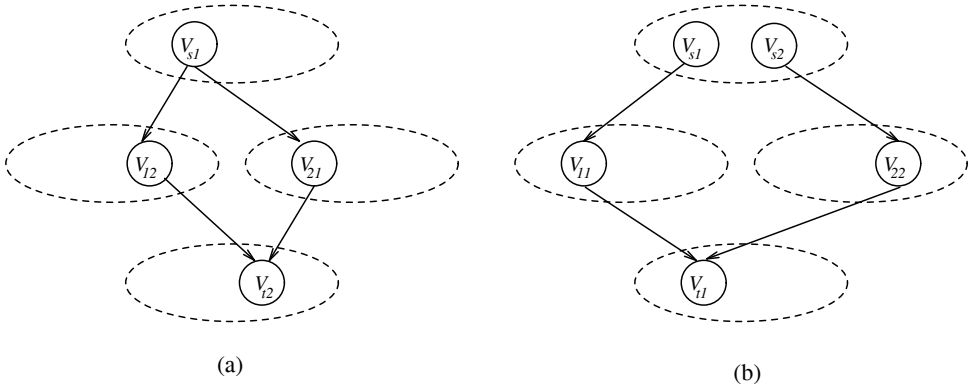


Figure 3: Operational examples of the replicated task digraph in Figure 2(b).

implemented, such as in *Maruti* [14] and *Delta-4* [15]. The idea behind this approach can be illustrated in the following example. Consider a simple application modeled by an AFAJ digraph as shown in Figure 2(a). Suppose the application is implemented with an extra replication. In this approach, each module receives messages not only from its predecessors in the original task digraph, but also from the corresponding replicas of its predecessors. Figure 2(b) shows one such implementation. Thus a task finishes successfully only if there is a set of modules which forms this application, and their associated communication links are operational. Obviously, this application may tolerate more than one fault in each task replication, depending on the fault patterns. Figure 3 shows a few examples where the task in Figure 2(b) is operational.

In this paper, we consider the fault-tolerant variants of AOSP digraphs, named *Repli-*

cated And-Or Series-Parallel (RAOSP) digraphs. RAOSP digraphs are obtained from AOSP digraphs by applying the approach used in *Maruti* and *Delta-4*, i.e. by adding replications to each vertex and adding proper arcs between two vertices. For RAOSP digraphs, it is interesting that we can also calculate its task reliability and task response time in linear time with the similar methods of AOSP digraphs. Therefore, it is another crucial work to recognize RAOSP digraphs. The existing polynomial time recognition algorithm for AOSP digraphs does not apply here since it can be shown that RAOSP digraphs are not AOSP digraphs. The main contribution of this paper is the design of a polynomial time algorithm for RAOSP digraph recognition .

The rest of the paper is organized as follows. In Section 2, some logical definitions and notations employed in the context are described, and we will also define fully factorable formulas and fully factorable trees which can characterize the logic structures among modules of AOSP digraphs. Then the formal definitions of AOSP digraphs and RAOSP digraphs will be given in the next section. In Section 4, we will describe how to recognize the fault-tolerant variants of ESP digraphs, RESP digraphs, as a basis for the recognition of RAOSP digraphs. In Section 5, the recognition algorithm for RAOSP digraphs will be proposed. Finally, we will conclude the paper in Section 6.

2. Preliminaries

Our logical terminology follows [16]. A *Boolean variable* is denoted by x_i to represent a Boolean value **true** or **false** but not both. Variables and negations of variables will be spoken of collectively as *literals*. The *conjunction* of x_1 and x_2, $x_1 \wedge x_2$, is **true** only when both x_1 and x_2 are **true**. Symmetrically, the *disjunction* of x_1 and x_2, $x_1 \vee x_2$, is **false** only when both x_1 and x_2 are **false**. A *Boolean formula* is made up of literals, conjunctions and disjunctions. A formula is said to be *trivial* if it is made up of only one single literal, opposite to a *nontrivial* formula. A *positive formula* is a formula without negative variables. Two formulas F_1 and F_2 are said to be *equivalent* provided that the formula F_1 is **true** (**false**) if and only if the formula F_2 is **true** (**false**). A disjunction of literals in that no variable appears twice is called a *fundamental disjunctive formula*. Any conjunction of fundamental disjunctive formulas is called a *Conjunctive*

Normal Formula (abbreviated *CNF*) or a formula in *the conjunctive normal form*. The fundamental disjunctive formulas in a CNF F are called the *clauses* of F. A conjunctive normal formula with minimum number of literals and minimum number of clauses is called *irreducible*.

For a Boolean formula F, the literal set $L(F) = \{l \mid l \text{ is a literal in } F\}$. The number of literals in F is thus denoted as $|L(F)|$. If two clauses C_1 and C_2 with $L(C_1) = L(C_2)$, we say C_1 and C_2 are *isomorphic*. Two CNFs F_1 and F_2 are isomorphic if and only if each clause in F_1 is isomorphic to some clause in F_2 and vice versa. Furthermore, two CNFs F_1 and F_2 are said to be *similar* if and only if there exists a one-to-one mapping between literals of F_1 and those of F_2 such that F_1 and F_2 are isomorphic. For example, $((x_1 \vee x_2) \wedge x_3)$ is similar to $((x_4 \vee x_5) \wedge x_6)$ with the mapping of $(x_1 \leftrightarrow x_4)$, $(x_2 \leftrightarrow x_5)$ and $(x_3 \leftrightarrow x_6)$. Finally, for any two Boolean formulas F_1 and F_2, if $L(F_1) \cap L(F_2) = \emptyset$, we say F_1 and F_2 are *disjoint*.

In order to formulate the logical structures among modules of a computation task, each arc of the corresponding task digraph is assigned with a distinct Boolean variable. So without loss of generality, all variables can be assumed to be positive. Moreover, for the sake of simplicity, every formula associated with the module is assumed to be given as a positive formula. Thus, we assume that all literals and formulas are positive throughout the remainder of the context.

Next, we will describe the definition of *fully factorable formulas*. If a Boolean formula F can be expressed as $F_1 \oplus F_2$ where F_1 and F_2 are two *disjoint* Boolean formulas, called the *subformulas* of F, and $\oplus$ is a *Boolean operation*, i.e. $\wedge$ or $\vee$, F is said to be *factorable*. The previous operation on F to find the conjunctive or disjunctive expression of subformulas is called *factoring* on F. If the Boolean operation is $\wedge$, the factoring is called *and-factoring*. On the other hand, if the Boolean operation is $\vee$, the factoring is called *or-factoring*. Formally, the class of fully factorable formulas is defined as below.

Definition 1 The class of *fully factorable formulas* includes

1. A literal is an elementary *fully factorable formula*;
2. If F_1 and F_2 are two disjoint *fully factorable formulas*, so are the formulas constructed by each of the following operations:

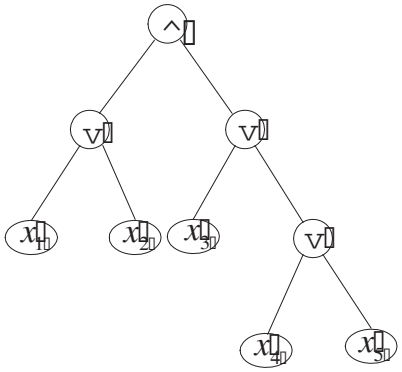


Figure 4: The factoring tree corresponding to the Boolean formula $(x_1 \vee x_2) \wedge (x_3 \vee x_4 \vee x_5)$.

(a) Conjunctive composition: $F_\wedge = F_1 \wedge F_2$;

(b) Disjunctive composition: $F_\vee = F_1 \vee F_2$;

3. If F is a *fully factorable formula*, so is the Boolean formula equivalent to F.

In order to record the factoring process on a Boolean formula F, we construct a binary tree T_f, named the *factoring tree* corresponding to F. The external leaf in T_f has a sort of Boolean formula F_i which cannot be factored further, and the internal node has a sort in $\{\wedge, \vee\}$ to represent the conjunction or disjunction of the subformulas corresponding to the subtrees rooted at the children of this node. If a factoring tree with all external leaves containing one single literal, it is named a *fully factoring tree*. One example fully factoring tree is depicted in Figure 4. Obviously, every two distinct external leaves of a fully factoring tree contain distinct literals according to the definition of factoring operations.

Remark that a factoring tree may not be unique since we may construct different factoring trees corresponding to a given Boolean formula by different factoring algorithms. Moreover, a formula F can be obtained by traversing a factoring tree T_f according to the inorder sequence. We then call that F is *expanded* from T_f.

3. AOSP Digraphs and RASOP Digraphs

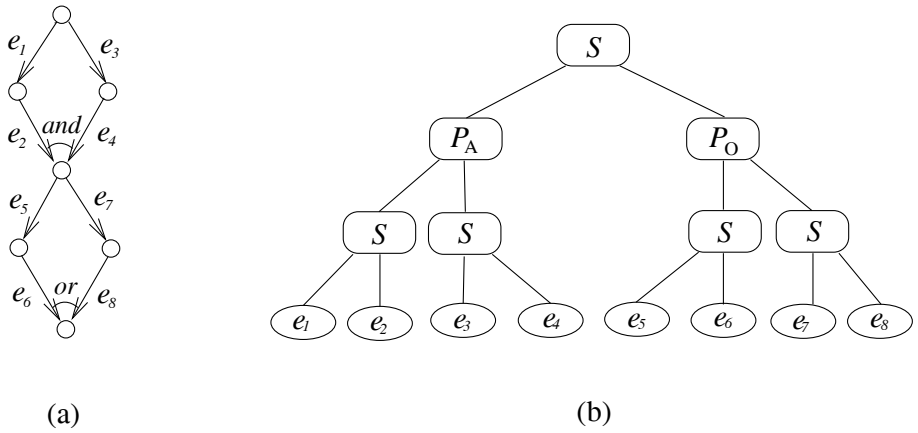
AOSP digraphs are the extensions of ESP digraphs. For an AOSP digraph, each vertex containing entering arcs is assigned with a formula to represent the logical structures

among modules. We denote an AOSP digraph as $H = (V(v), E, F(v))$, where $V(v)$ is a finite set of vertices, E is a finite set of arcs and $F(v)$ is a finite set of formulas attached to vertices. Specifically, the class of AOSP digraphs is defined recursively as below [7].

Definition 2 The class of *AOSP* (*And-Or Series-Parallel*) digraphs includes

1. A single arc $e = (s, t)$ with the source s and the sink t, and the Boolean formula F_t attached to the sink, which equals to a single literal x, is an elementary AOSP digraph;
2. If H_1 and H_2 are AOSP digraphs with sources s_1 and s_2 and sinks t_1 and t_2, and the corresponding Boolean formulas attached to the sinks are F_{t_1} and F_{t_2} respectively, so are the digraphs constructed by each of the following operations:
   (a) *Series composition* (S): The digraph H_S is an AOSP digraph with terminals s and t, where H_S is the disjoint union of H_1 and H_2, with t_1 identified with s_2;
   (b) *Parallel-and composition* (P_A): The digraph H_{P_A} is an AOSP digraph with terminals s and t, where H_{P_A} is the disjoint union of H_1 and H_2, with s_1 identified with s_2 and t_1 identified with t_2 and the Boolean formula F_t attached to t is $F_{t_1} \wedge F_{t_2}$;
   (c) *Parallel-or composition* (P_O): The digraph H_{P_O} is an AOSP digraph with terminals s and t, where H_{P_O} is the disjoint union of H_1 and H_2, with s_1 identified with s_2 and t_1 identified with t_2 and the Boolean formula F_t attached to t is $F_{t_1} \vee F_{t_2}$.

Similar to a Boolean formula, an AOSP digraph can be represented in a natural way by a binary tree T_p, called the *parsing tree*. Each external leaf of T_p represents an elementary AOSP subdigraph in the AOSP digraph, denoted as its corresponding arc; whereas each internal node is labeled S, P_A or P_O to represent the series, parallel-and or parallel-or composition of the AOSP digraphs corresponding to the subtrees rooted at the children of this node. A parsing tree provides a concise description of the structure of an AOSP digraph and facilitates the calculation of task reliability and task response time. Moreover, according to Definition 2, we know that a Boolean formula attached to a vertex of AOSP digraphs must be able to be fully factored. Besides, if we ignore the



(a) (b)

Figure 5: (a) An example AOSP digraph; (b) Its corresponding parsing tree.

Boolean formulas and just consider the topology of an AOSP digraph, it is exactly an ESP digraph. Figure 5 gives an example AOSP digraph and its corresponding parsing tree.

Figure 6(b) shows a digraph H' which is constructed from an elementary AOSP digraph H (shown in Figure 6(a)) by replicating the source vertex m times and sink vertex n times. And the arcs in H' are established in such a way that each vertex is not only descendant to its original predecessors, but is also descendant to the corresponding replicas of its predecessors. This digraph is called an elementary $m \times n$ RAOSP digraph. Particularly, Figure 6(c) depicts an elementary 2×2 RAOSP digraph. Imagine that each arc in H represents a thread of execution, then a replicated edge in H' represents a replica thread of the corresponding thread in H. (Such as (s_i, t_j), for $1 \leq i \leq m$ and $1 \leq j \leq n$, are replicas of (s, t) in Figure 6(a).) Therefore, it is an or-relation among all entering arcs corresponding to t_j, for $1 \leq j \leq n$.

Now we are ready to formally define the $m \times n$ RAOSP digraph [7]. We denote an RAOSP digraph as $H^{m \times n} = (V(v), E, F(v))$, where $V(v)$ is a finite set of vertices, E is a finite set of arcs and $F(v)$ is a finite set of formulas attached to vertices.

Definition 3 The class of *RAOSP* (*Replicated And-Or Series-Parallel*) digraphs includes

1. A digraph with a set of m source terminals, $S = \{s_1, \ldots, s_m\}$, a set of n sink

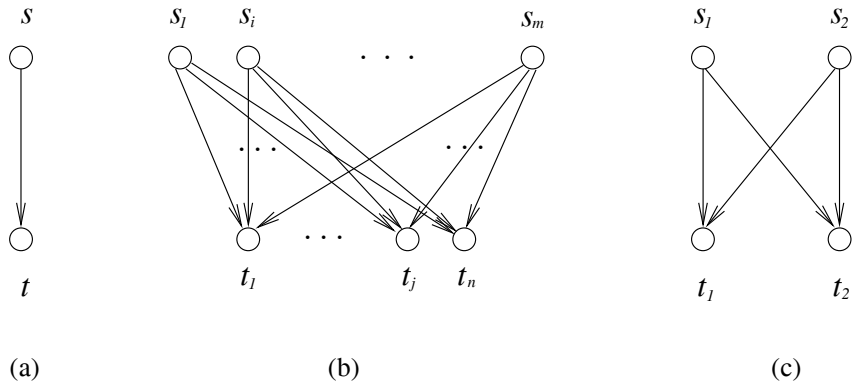
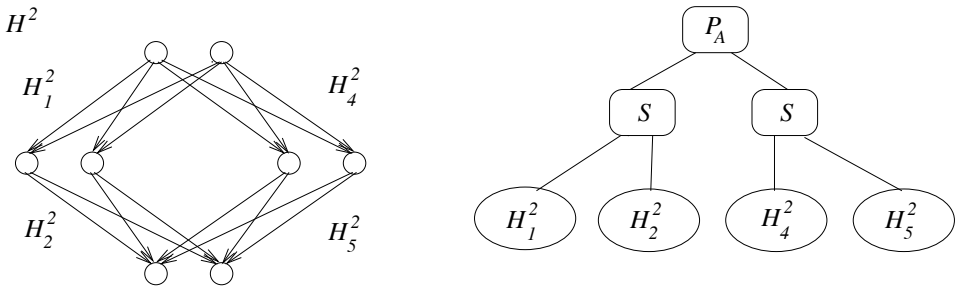


Figure 6: (a) An elementary AOSP digraph; (b) An elementary $m \times n$ RAOSP digraph; (c) An elementary 2×2 RAOSP digraph.

terminals, $T = \{t_1, \ldots, t_n\}$, $m \times n$ arcs, (s_i, t_j) $\forall i, j$, and $\forall j$ the Boolean formula F_{t_j} attached to the sink t_j, which equals to the disjunction of m distinct literals, $x_{1j} \vee x_{2j} \vee \cdots \vee x_{mj}$, is an elementary $m \times n$ RAOSP digraph;

2. If $H_1^{m\times n}$ and $H_2^{m\times n}$ are $m \times n$ RAOSP digraphs with sources S_1 and S_2 and sinks T_1 and T_2, and $\forall j$ the corresponding Boolean formulas attached to the sinks $t_{1j} \in T_1$ and $t_{2j} \in T_2$ are $F_{t_{1j}}$ and $F_{t_{2j}}$ respectively, so are the digraphs constructed by each of the following operations:
   (a) *Parallel-and composition* (P_A): The digraph $H_{P_A}^{m\times n}$ is an $m \times n$ RAOSP digraph with terminals S and T, where $H_{P_A}^{m\times n}$ is the disjoint union of $H_1^{m\times n}$ and $H_2^{m\times n}$, with S_1 identified with S_2 and T_1 identified with T_2 and $\forall j$ the Boolean formula F_{t_j} attached to $t_j \in T$ is $F_{t_{1j}} \wedge F_{t_{2j}}$;
   (b) *Parallel-or composition* (P_O): The digraph $H_{P_O}^{m\times n}$ is an $m \times n$ RAOSP digraph with terminals S and T, where $H_{P_O}^{m\times n}$ is the disjoint union of $H_1^{m\times n}$ and $H_2^{m\times n}$, with S_1 identified with S_2 and T_1 identified with T_2 and $\forall j$ the Boolean formula F_{t_j} attached to $t_j \in T$ is $F_{t_{1j}} \vee F_{t_{2j}}$;
   (c) *Series composition* (S): Suppose $H_1^{m\times k}$ and $H_2^{k\times n}$ are $m \times k$ and $k \times n$ RAOSP digraphs, respectively. The digraph $H_S^{m\times n}$ is an $m \times n$ RAOSP digraph with terminals S and T, where $H_S^{m\times n}$ is the disjoint union of $H_1^{m\times k}$ and $H_2^{k\times n}$, with T_1 identified with S_2.



(a) (b)

Figure 7: (a) An example 2-AOSP digraph; (b) Its corresponding parsing tree.

Likewise, we can also concisely represent the structure of RAOSP digraphs by parsing trees. Remark that in this case every external leaf of parsing trees contains an elementary RAOSP digraph instead. Moreover, if every vertex is replicated k times in an RAOSP digraph, we name such a digraph as *k-replicated AOSP (k-AOSP)* digraph, and denote it by H^k. Figure 7 shows the corresponding 2-AOSP digraph of the AFAJ digraph and its corresponding parsing tree.

4. Algorithms for Recognizing RESP Digraphs

In this section, we will describe how to recognize the fault-tolerant variants of ESP digraphs, named *Replicated ESP (RESP)* digraphs, as a basis for the recognition of RAOSP digraphs. RESP digraphs are constructed from ESP digraphs by exploiting the same replication approach of RAOSP digraphs, i.e. by adding replications to each vertex and adding proper arcs between two vertices. We denote an RESP digraph as $G^{m\times n}$. Since the ESP digraph is a special case of the AOSP digrpah, which does not take the logic structures among modules into consideration, the RESP digraph is thus the special case of the RAOSP digraph without considering the logic structures among modules similarly.

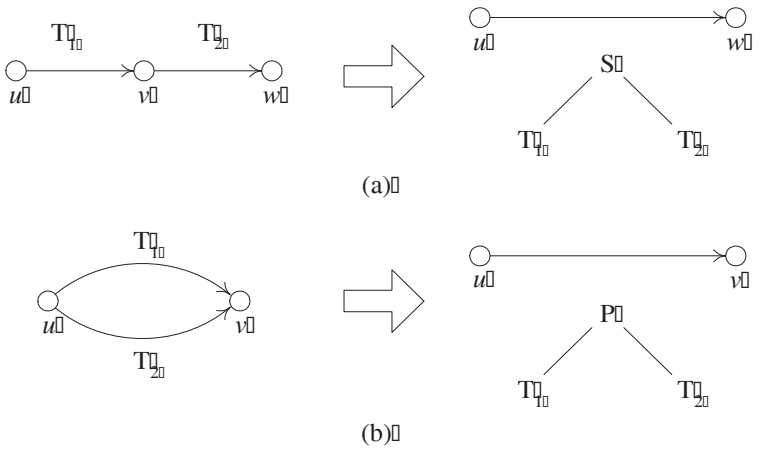


Figure 8: Rules to perform reductions and to construct the parsing tree: (a) Series reduction; (b) Parallel reduction.

4.1 ESP Digraph Recogniton Algorithms

Prior to designing algorithms needed for recognizing RESP digraphs, the ESP digraph recognition algorithm in [12] will be briefly described first. For more details, please refer to the paper mentioned previously. Note that we slightly adapt the original recognition algorithm to meet the needs for recognizing AOSP digraphs in the next section.

The input of the algorithm is a task digraph, and the algorithm comprises the series reduction and the parallel reduction. By applying series and parallel reductions until no more is applicable, an ESP digraph will be reduced to a digraph with only one single arc, but other digraphs will not. Moreover, during the reduction process, the corresponding parsing tree can be constructed by using the rules of Figure 8.

To perform series and parallel reductions on the digraph, it is necessary to identify two specific types of vertices: *relay vertices* and *confluent vertices*. For a vertex containing only one entering arc and only one leaving arc, like the vertex v shown in Figure 8(a), this type of vertex is called a *relay vertex* and is performed by the series reduction. For a vertex containing more than one entering arcs but only one predecessor, like the vertex v shown in Figure 8(b), this type of vertex is called a *confluent vertex* and is performed by the parallel reduction. We maintain a list of vertices called the *unsatisfied list*, represented as *UL*. *UL* contains the vertices on which reductions still

need to be tried. Hence UL initially contains all vertices except the source. The ESP recognition algorithm **ESP_RECOG** is described as below.

ESP_RECOG$(G(V, E))$

Begin

1 Add all vertices in V except the source into UL.

2 Remove some vertex v from UL and carry out the following steps until no vertex remains in UL.

2.1 If v is a confluent vertex, i.e. having more than one entering arcs but only one predecessor u, apply a parallel reduction. If u is not the source and not in UL either, add it to UL.

2.2 If v is (or becomes) a relay vertex, i.e. only one arc (u, v) entering v and only one arc leaving (v, w), apply a series reduction and replace (u, v) and (v, w) by a new arc (u, w). If w is not in UL, add it to UL.

3 If G is reduced to a single arc, reply "G is an ESP digraph" and the parsing tree T_p; else reply "G is not an ESP digraph".

End

The time complexity of the ESP recognition algorithm is $O(|V| + |E|)$.

4.2 *k*-ESP Digraph Recogniton Algorithms

As a preliminary step, we will propose the recognition algorithm for a subclass of RESP digraphs, *k-replicated ESP (k-ESP)* digraphs, in which every vertex is replicated k times. A k-ESP digraphs is denoted by G^k. Since a k-ESP digraph is constructed from an ESP digraph by adding k replications to each vertex and adding proper arcs between two vertices, the first step of recognizing such a digraph is to find all the k replicas of each vertex and then group them as a *super vertex*. Besides, all the k^2 arcs between two super vertices are also grouped as a *super arc*. This step is illustrated in Figure 9. After so, we will get a *super digraph* which should exactly have the same structure with an ESP digraph. Therefore, we can use the ESP recognition algorithm stated in the previous subsection to examine it subsequently. By these foregoing steps, we can recognize if a task digraph is a k-ESP digraph or not.

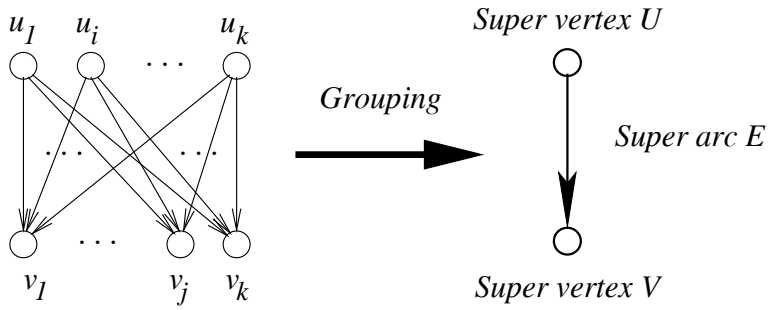


Figure 9: The scenario of grouping replicas of vertices and arcs in k-ESP digraphs.

Because all the k replicas of each vertex have the same predecessor and successor sets, we will group vertices with the same predecessor and successor sets in a k-ESP digraph together first. In the following, we give a formal definition for this relation between two vertices.

Definition 4 Let $P(v) = \{p \mid p \text{ is a predecessor of } v\}$, i.e. the predecessor set of v, and $S(v) = \{s \mid s \text{ is a successor of } v\}$, i.e. the successor set of v. Two vertices v_1 and v_2 are said to be *equivalent*, denoted as $v_1 \equiv v_2$, if and only if $P(v_1) = P(v_2)$ and $S(v_1) = S(v_2)$.

So the super vertex U in Figure 9 denotes the vertex set $\{u_1, u_2, \ldots, u_k\}$ with $u_1 \equiv u_2 \equiv \cdots \equiv u_k$, and the super vertex V denotes the vertex set $\{v_1, v_2, \ldots, v_k\}$ with $v_1 \equiv v_2 \equiv \cdots \equiv v_k$. The super arc E denotes the arc set $\{(u_i, v_j) \mid 1 \leq i, j \leq k\}$.

Note that if two vertices in an ESP digraph have the equivalent predecessor and successor sets, their replicas in the corresponding k-ESP digraph also have the equivalent predecessor and successor sets. Thus these $2k$ replicas will be grouped together. But we can divide this group of vertices arbitrarily into two super vertices due to the symmetry of the two original vertices in the ESP digraph. Moreover, since every vertex in a super vertex has the same predecessors and successors, we can easily have the following property.

Property 1 For two super vertices U and V, there is an arc between $u_i \in U$ and $v_j \in V$ if and only if there is a super arc between U and V.

According to the previous discussions, the grouping procedure for k-ESP digraphs is designed as follows. Here the parameter k is assumed to be given for the sake of

neatness, but in fact we can get its value from the number of sources or sinks.

***k*-ESP_GROUP**$(G^k(V,E))$

Begin

1 If $|V| \bmod k \neq 0$, reply "G^k is not a k-ESP digraph" and stop.

2 For every vertex v in V, get its corresponding $P(v)$ and $S(v)$, and if $|P(v)| \bmod k \neq 0$ or $|S(v)| \bmod k \neq 0$, reply "G^k is not a k-ESP digraph" and stop.

3 Add all vertices in V to a list VL.

4 Remove a vertex v from VL, and remove every vertex v' with $v' \equiv v$ in VL. Then make them as a group Q. If $|Q| \bmod k \neq 0$, reply "G^k is not a k-ESP digraph" and stop; else suppose $|Q| = nk$, divide Q into n super vertices with k vertices each and add these n super vertices to the super vertex set V_S. Repeat this step until no vertex remains in VL.

5 For every super vertex V_i in V_S, find all super arcs connected to it and add these super arcs to the super arc set E_S.

6 Return the super digraph $G_S(V_S, E_S)$.

End

Subsequently, we will use the the ESP recognition algorithm to examine if the super digraph G_S is an ESP digraph. If so, then G^k is a k-ESP digraph. The following argument shows that this algorithm for recognizing k-ESP digraphs is correct. If G^k is a k-ESP digraph, after the grouping procedure, the resultant super digraph G_S has the same structure with an ESP digraph by definition. Hence our algorithm can recognize it. On the other hand, if G^k is not a k-ESP digraph, it is either the digraph constructed from a non-ESP digraph with k replications or a digraph which is not constructed from any digraph with k replications. The former will be found by the ESP recognition algorithm, and the latter will be found during the grouping procedure.

The time complexity of the proposed algorithm for recognizing k-ESP digraphs is evaluated as follows. The dominant step of the grouping procedure is Step 4, which has the time complexity $O(|V|^3)$, because we have to compare every two vertices of the $|V|$ vertices with at most $(|V| - 1)$ predecessors and successors each. In addition, the time complexity of the ESP digraph recognition algorithm is $O(|V| + |E|)$ as stated

previously, and can be considered as $O(|V|^2)$. Since k is not large and can be considered as a constant, we can conclude that the time complexity of this algorithm is $O(|V|^3)$.

4.3 RESP Digraph Recogniton Algorithms

Now we proceed to introduce the recognition algorithm for RESP digraphs. The same with the recognition of k-ESP digraphs, the first step is also to find all the replicas of each vertex and then group them as a super vertex. Moreover, all the arcs between two super vertices are also grouped as a super arc. This step is shown in Figure 10. The super vertex U in Figure 10 denotes the vertex set $\{u_1, u_2, \ldots, u_m\}$ with $u_1 \equiv u_2 \equiv \cdots \equiv u_m$, and the super vertex V denotes the vertex set $\{v_1, v_2, \ldots, v_n\}$ with $v_1 \equiv v_2 \equiv \cdots \equiv v_n$. The super arc E denotes the arc set $\{(u_i, v_j) \mid 1 \leq i \leq m, 1 \leq j \leq n\}$. By doing so, we will also get a super digraph which should exactly have the same structure with an ESP digraph, and then we can use the ESP recognition algorithm to examine it in the next step. Specifically, the grouping procedure for RESP digraphs is designed as follows.

RESP_GROUP$(G^{m\times n}(V, E))$

Begin

1 For every vertex v in V, get its corresponding $P(v)$ and $S(v)$.

2 Add all vertices in V to a list VL.

3 Remove a vertex v from VL, and remove every vertex v' with $v' \equiv v$ in VL. Then group them as a super vertex U, and add U to the super vertex set V_S. Repeat this step until no vertex remains in VL.

4 For every super vertex V_i in V_S, find all super arcs connected to it and add these super arcs to the super arc set E_S.

5 Return the super digraph $G_S(V_S, E_S)$.

End

Likewise, we will then use the the ESP recognition algorithm to examine if the super digraph G_S is an ESP digraph. If so, $G^{m\times n}$ is a RESP digraph. With the similar argument for the correctness of the k-ESP digraph recognition algorithm, we can also prove the correctness of the proposed algorithm for recognizing RESP digraphs. Besides, it can be easily shown that the time complexity of this algorithm is also $O(|V|^3)$.

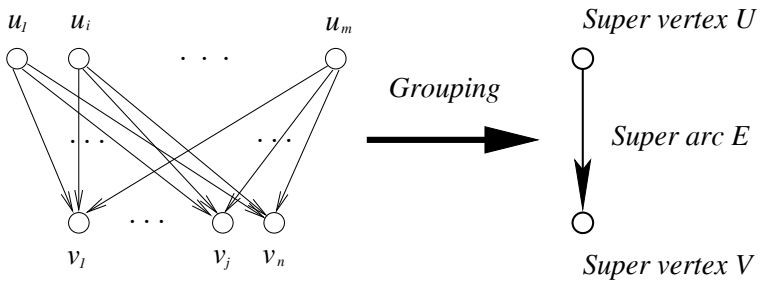


Figure 10: The scenario of grouping replicas of vertices and arcs in RESP digraphs.

Remark that a super vertex with m vertices can also be viewed as n super vertices with that the total number of vertices in these n super vertices is m, where $n \geq 1$. However, due to the following characteristic of ESP digraphs, we can conclude that any two vertices in ESP digraphs have at most one same predecessor and at most one same successor. So it is better to consider any super vertex as a whole one except that there is other information showing that it has to be divided.

Theorem 1 No ESP digraph contains the subgraph of a 2×2 complete bipartite digraph (such as the digraph shown in Figure 6(c)).

Proof: Suppose there exists an ESP digraph G containing the subgraph of a 2×2 complete bipartite digraph. By a sequence of series and parallel reductions, we can reduce G as the digraph depicted in Figure 11. (The arcs with dashed lines in Figure 11 are optional.) An ESP digraph can be reduced to the one-arc ESP digraph by a sequence of series and parallel reductions [12]. However, it can be easily verified that the reduced digraph of G cannot be series or parallel reduced further. Thus G is not an ESP digraph. This leads to a contradiction. □

5. Algorithms for Recognizing RAOSP Digraphs

We will propose algorithms for the recognition of RAOSP digraphs in this section. First, we will briefly describe the AOSP digraph recognition algorithm proposed in [1]. For more details, please refer to the previous paper.

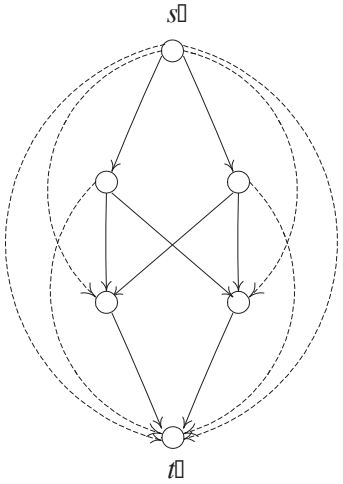


Figure 11: The resultant digraph reduced from G.

5.1 AOSP Digraph Recogniton Algorithms

For the AOSP digraph recogniton algorithm, the input is a task digraph with a set of Boolean formulas attached to vertices. This algorithm is similar with the recognition algorithm for ESP digraphs, **ESP_RECOG**, described in Subsection 4.1 except that the parallel reduction algorithm was revised such that it can also recognize fully factorable formulas. During the reduction process, the corresponding parsing tree can be constructed by using the rules of Figure 12.

We continue to introduce the revised parallel reduction mentioned above. The input for the corresponding algorithm **REV_PAR_REDUC** is a Boolean formula and a bunch of arcs. If the input formula is fully factorable, its corresponding factoring tree will be constructed. For the sake of neatness, the input Boolean formula is assumed to be in its conjunctive normal form.

REV_PAR_REDUC(E', F)

Begin

1 Recognize whether F is a fully factorable formula by the factoring algorithm **FACTOR**. If not, reply "Parallel reduction fails" and return.

2 Examine if the number of external leaves in the corresponding factoring tree T_f equals to the number of arcs in E'. If not, reply "Parallel reduction fails"

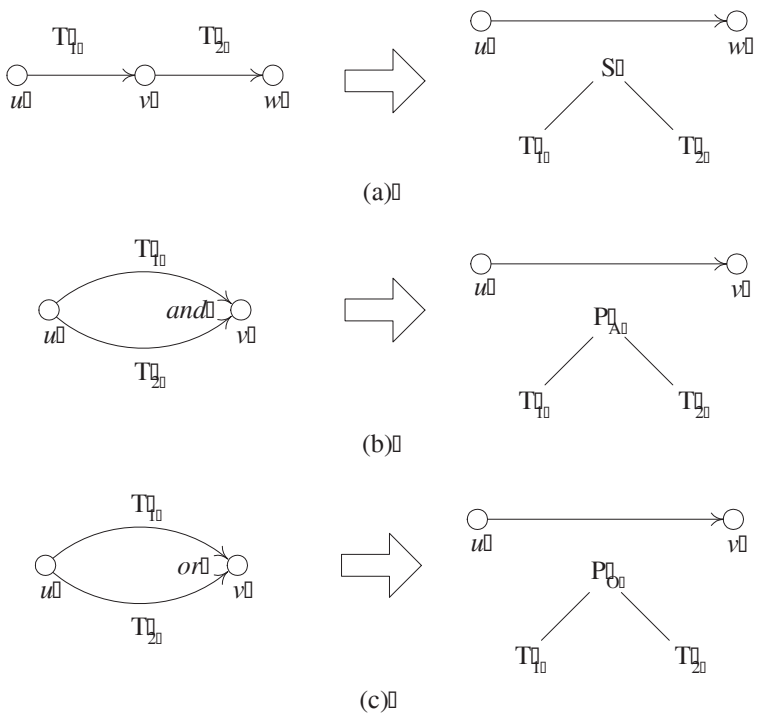


Figure 12: Rules to perform reductions and to construct the parsing tree: (a) Series reduction; (b) Parallel reduction for the and-factoring case; (c) Parallel reduction for the or-factoring case.

and return.

3 Keep only one arc in E' and delete all other arcs. Moreover, return T_f.

End

Subsequently, we will begin to describe the factoring algorithm exploited in the previous algorithm. This algorithm can fully factor a CNF and construct the corresponding factoring tree. The first step is to reduce the input formula to be irreducible. The irreducible form of a positive CNF can be obtained by a polynomial time algorithm which applies the idempotent law and the absorption law of Boolean algebra to eliminate redundant literals and clauses. The next step is to fully factor the irreducible CNF. We will factor the input formula into a set of subformulas by the and-factoring or or-factoring algorithm and repeatedly apply the same process to each subformula until no subformula can be factored any more. If one subformula is not trivial, such a formula is not fully

factorable.

The factoring algorithm is shown in the following. The formula reduction process of the first step is implemented as the algorithm, **REDUCE**, and the corresponding and-factoring algorithm (or-factoing algorithm), **AND-FACTOR** (**OR-FACTOR**), can and-factor (or-factor) a formula as many subformulas as possible. Besides, this factoring algorithm maintains two lists of Boolean formulas called the *AND list* and *OR list*, represented as AL and OL respectively. AL contains the Boolean formulas needed to be applied by the algorithm **AND-FACTOR**; while OL contains the Boolean formulas needed to be applied by the algorithm **OR-FACTOR**. Each formula F is thus attached with two Boolean tags, A-tag and O-tag, to indicate if F can be factored by **AND-FACTOR** or **OR-FACTOR**. Initially A-tag and O-tag are set to be **true**. If F cannot be factored by **AND-FACTOR** (**OR-FACTOR**), A-tag (O-tage) is consequently set to be **false**. The input of this algorithm is a positive CNF. Note that during the factoring process, all subformulas will be irreducible CNFs.

FACTOR(F)

Begin

1 F'= **REDUCE**(F).

2 If F' is a single literal, reply "F is fully factorable" and return the corresponding factoring tree T_f of a single leaf.

3 Set $F' \rightarrow$ A-tag and $F' \rightarrow$ O-tag to be **true** and add F' into AL.

4 Repeat removing a formula P from AL to perform the following procedures until AL is empty.

4.1 Call **AND-FACTOR**(P) to factor P into a set of subformulas S, and construct the corresponding subtree for the factoring tree T_f.

4.2 If $|S| = 1$ and P is not a single literal

4.2.1 Set $P \rightarrow$ A-tag to be **false**.

4.2.2 If $P \rightarrow$ A-tag and $P \rightarrow$ O-tag are both **false**, reply "F is not fully factorable" and return.

4.2.3 If $P \rightarrow$ O-tag is **true**, add the only one element in S to OL.

4.3 If $|S| > 1$, for each subformula P_i in S

4.3.1 Set $P_i \rightarrow$ O-tag to be **true** and $P_i \rightarrow$ A-tag to be **false**.

4.3.2 Add P_i to OL.

5 Repeat removing a formula Q from OL to perform the following procedures until OL is empty.

5.1 Call **OR-FACTOR**(Q) to factor Q into a set of subformulas T, and construct the corresponding subtree for the factoring tree T_f.

5.2 If $|T| = 1$ and Q is not a single literal

5.2.1 Set $Q \rightarrow$ O-tag to be **false**.

5.2.2 If $Q \rightarrow$ A-tag and $Q \rightarrow$ O-tag are both **false**, reply "F is not fully factorable" and return.

5.2.3 If $Q \rightarrow$ A-tag is **true**, add the only one element in T to AL.

5.3 If $|T| > 1$, for each subformula Q_i in T

5.3.1 Set $Q_i \rightarrow$ A-tag to be **true** and $Q_i \rightarrow$ O-tag to be **false**.

5.3.2 Add Q_i to AL.

6 Repeat Step 5 and 6 until AL and OL are empty.

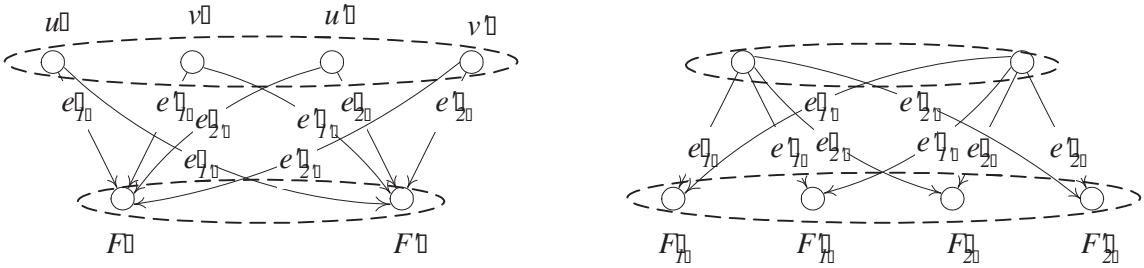
7 Reply "F is factorable" and return T_f.

End

If the input formula $F = C_1 \wedge C_2 \wedge \cdots \wedge C_n$ where C_i is the clause of F $\forall i$, the time complexity of the algorithm **FACTOR** is $O(l^4)$ where $l = \Sigma_{i=1}^{n} |L(C_i)|$, i.e. the number of total literals in F. Since the time complexity of the ESP recognition algorithm is $O(|V| + |E|)$ and the AOSP recognition algortihm needs at most $(|V| - 1)$ fully factoring operation, the time complexity of the latter is then $O(|V|L^4 + |E|)$, where L is the maximum value of l among all attached Boolean formulas.

5.2 *k*-AOSP Digraph Recognition Algorithms

Prior to designing the algorithm for recognizing RAOSP digraphs, we will introduce how to recognize k-AOSP digraphs as a preliminary step. First, similar with the recognition algorithm for k-ESP digraphs stated in the previous section, we will group vertices with the same predecessor and successor sets for the input digraph. Secondly, we have to examine the similarity of the Boolean formulas attached to vertices in a group. According



(a) (b)

Figure 13: Two scenarios of literal mapping, where (a) $F = (x_1 \vee x_{2'}) \wedge (x'_1 \vee x'_{2'})$ and $F' = (x_{1'} \vee x_2) \wedge (x'_{1'} \vee x'_2)$; (b) $F_1 = x_1 \vee x_{1'}$, $F'_1 = x'_1 \vee x'_{1'}$, $F_2 = x_{2'} \vee x_2$ and $F'_2 = x'_{2'} \vee x'_2$.

to the definition of k-AOSP digraphs, we know that the irreducible forms of the formulas attached to the k replicas of a vertex must be similar pairwise. So if all the vertices in a group are the replicas of the same vertex, the formulas attached to them have to be similar pairwise. In addition, the k literals corresponding to the arcs from a vertex v of a preceding group to the k vertices of this group should be mapped pairwise because they all play the role that the module corresponding to v enables one replica of its succeeding module to execute. One example of this scenario is shown in Figure 13(a). Figure 13(a) depicts a part of a 2-AOSP digraph, and vertices in an ellipse with dashed lines belong to the same group. The two attached formulas F and F' must be similar with the mapping of $(x_1 \leftrightarrow x_{1'})$, $(x'_1 \leftrightarrow x'_{1'})$, $(x_2 \leftrightarrow x_{2'})$ and $(x'_2 \leftrightarrow x'_{2'})$.

Next we consider the case that a group contains the replicas of two or more vertices. From Theorem 1, we can also have that every two vertices in an AOSP digraph have at most one same predecessor and at most one same successor. Hence if two vertices in an AOSP digraph are equivalent, they have only one predecessor. The formulas attached to them contain only one single literal. This leads to that if a group contains the replicas of two or more vertices, the irreducible forms of the formulas attached to these vertices are of the disjunction of distinct k literals. Moreover, if the literals corresponding to the arcs from a vertex of a preceding group to all the vertices of this group are mapped pairwise, the irreducible forms of the attached formulas of these vertices are also similar pairwise. Figure 13(b) illustrates an example of this scenario. Also, a part of a 2-AOSP

digraph is shown in Figure 13(b). The four formulas F_1, F_1', F_2 and F_2' are similar with the mapping of $(x_1 \leftrightarrow x_1' \leftrightarrow x_{2'} \leftrightarrow x_{2'}')$ and $(x_{1'} \leftrightarrow x_{1'}' \leftrightarrow x_2 \leftrightarrow x_2')$. Therefore, we can conclude that the formulas attached to vertices in a group have to be similar pairwise with the mapping that the literals corresponding to the arcs from a vertex of a preceding group to all the vertices of this group are mapped pairwise.

For k-ESP digraphs, we can arbitrarily divide a group with nk vertices into n groups with k vertices each. But in the case of k-AOSP digraphs, we have to divide such a group according to the logic structures attached to vertices of the succeeding group. Note that for a group with nk vertices where $n > 1$, its succeeding group is a group only containing k replicas of the same vertex since any two equivalent vertices in AOSP digraphs can only have one successor by Theorem 1. Moreover, the irreducible forms of the formulas attached to vertices in the succeeding group must be similar pairwise, so we can use the factoring result of any attached formula of the succeeding group to divide this group into n groups with k vertices each.

From the discussions on factoring Boolean formulas in [1], we can have that the factoring process of a positive irreducible CNF can be considered as unique. Namely, the selection of factoring operation is and-factoring and or-factoring alternatively, and a formula is factored as many subformulas as possible in each factoring operation. Hence, the factoring tree of a positive irreducible CNF can also be considered as unique. By the definition of k-AOSP digraphs, the irreducible forms of their attached formulas can be expanded from factoring trees with leaves of the disjunction of distinct k literals. So we can consider that the corresponding factoring trees of k-AOSP digraphs have leaves of the disjunction of distinct k literals. For example, the factoring tree corresponding to the formula F in Figure 13(a) is shown in Figure 14. It can be seen that the corresponding literals x_1 and $x_{2'}$ of arc e_1 and $e_{2'}$ are in the same leaf. Thus we know that vertex u and u' are the replicas of each other since e_1 and $e_{2'}$ eject from these two vertices respectively. Similarly, we can also have that vertex v and v' are the replicas of each other.

During the factoring process of a formula, we may encounter subformulas with the form of the disjunction of distinct mk literals where $m > 1$. For this scenario, we embed these mk literals into a leaf temporarily. After the factoring process, we will begin to

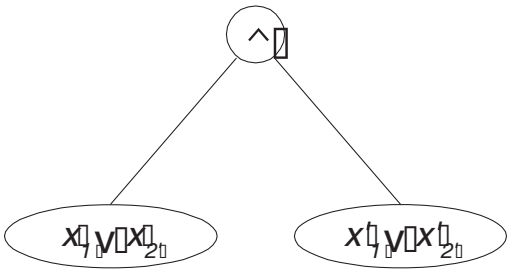


Figure 14: The factoring tree corresponding to $F = (x_1 \vee x_{2'}) \wedge (x'_1 \vee x'_{2'})$.

divide the preceding group with nk vertices where $n > 1$. If there exist k vertices of the preceding group with that the corresponding literals of the arcs ejecting from them are in the same leaf, these k vertices will be grouped as a new group. If there exist lk vertices with that their corresponding literals are contained in the same leaf where $l > 1$, we will arbitrarily divide these lk vertices into l new groups due to the symmetry of theses vertices both in the graph structure and in the logic structure. After this step, every group has exactly k vertices. Then a leaf with mk literals is divided into m leaves, where a leaf has k literals whose corresponding vertices are in the same group. The factoring tree is also performed with the corresponding revision, as an example depicted in Figure 15. In addition, we also have to check if all the corresponding literals are in the same leaf for a group that is not divided in the step of dividing groups. Subsequently, we will make a group of k vertices as a super vertex. And the k^2 arcs between two super vertices will be grouped as a super arc. Their corresponding k^2 literals will also be grouped as a *super literal*. The resultant super digraph should have the same structure with an AOSP digraph. Hence, we can use the AOSP recognition algorithm to examine it. Specifically, the grouping procedure for k-AOSP digraphs is designed as follows. The parameter k is also assumed to be given for the sake of neatness, but in fact we can get its value from the number of sources or sinks.

***k*-AOSP_GROUP**$(H^k(V(v), E, F(v)))$

Begin

1 If $|V| \bmod k \neq 0$, reply "H^k is not a k-AOSP digraph" and stop.

2 For every vertex v in V, get its corresponding $P(v)$ and $S(v)$, and if $|P(v)|$ mod

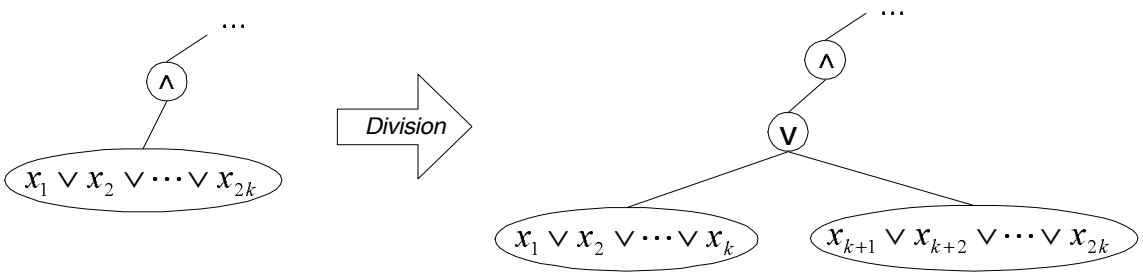


Figure 15: A leaf with $2k$ literals is divided into two leaves with k literals each.

$k \neq 0$ or $|S(v)| \bmod k \neq 0$, reply "H^k is not a k-AOSP digraph" and stop.

3 Add all vertices in V to a list VL.

4 Remove a vertex v from VL, and remove every vertex v' with $v' \equiv v$ in VL. Then make them as a group Q. If $|Q| \bmod k \neq 0$, reply "H^k is not a k-AOSP digraph" and stop; else add Q to a group set VG. Repeat this step until no vertex remains in VL.

5 For each group Q in VG,

5.1 For every formula F attached to the vertices in Q, **REDUCE**(F).

5.2 Check if all these irreducible formulas are similar pairwise with the mapping that the literals corresponding to the arcs from a vertex of a preceding group in the digraph to all the vertices of Q are mapped pairwise.

5.3 Get the irreducible formula F' attached to a vertex in Q and **FACTOR_k**(F'). If F' is not fully factorable, reply "H^k is not a k-AOSP digraph" and stop; else associate its factoring tree T_f with Q.

6 For each group Q in VG,

6.1 If $|Q| = k$,

6.1.1 Get all the corresponding factoring trees of its succeeding groups in the digraph.

6.1.2 Check if the k corresponding literals of vertices in Q are in the same leaf for every factoring tree. If not, reply "H^k is not a k-AOSP digraph" and stop.

6.1.3 Add Q to the super vertex set V_S.

6.1.4 For every factoring tree T_f, if the corresponding literals are con-

tained in a leaf of T_f with more than k literals, remove these corresponding literals from this leaf to make them as a new leaf, and then do the corresponding revision for T_f.

6.2 If $|Q| = nk$ where $n > 1$,

6.2.1 If the number of its succeeding groups in the digraph is more than one, reply "H^k is not a k-AOSP digraph" and stop; else get the corresponding factoring tree T_f of its only one succeeding group.

6.2.2 Divide Q into subgroups, where a subgroup contains vertices whose corresponding literals are in the same leaf of T_f. If there exists a vertex in Q whose corresponding literal is not in any leaf of T_f, reply "H^k is not a k-AOSP digraph" and stop.

6.2.3 Pick a subgroup P. If $|P| \bmod k \neq 0$, reply "H^k is not a k-AOSP digraph" and stop.

6.2.4 Suppose $|P| = lk$, arbitrarily divide P into l super vertices with k vertices each and add these super vertices to V_S. Moreover, associate the factoring tree corresponding to Q with these super vertices.

6.2.5 If the corresponding literals are in a leaf of T_f with exactly lk literals, divide this leaf into l new leaves, where a leaf has k literals whose corresponding vertices are in the same super vertex, and then do the corresponding revision for T_f.

6.2.6 If the corresponding literals are contained in a leaf of T_f with more than lk literals, remove these corresponding literals from this leaf to make them as l new leaves, where a leaf has k literals whose corresponding vertices are in the same super vertex, and then do the corresponding revision for T_f.

6.2.7 Repeat Step 6.2.3 to Step 6.2.6 for other subgroups.

7 For each super vertex V_i in V_S,

7.1 Get its corresponding factoring tree T_f, and check if the number of literals in every leaf of T_f equals to k. If not, reply "H^k is not a k-AOSP digraph" and stop; else replace the k literals of every leaf with a super literal.

7.2 Find all super arcs connected to it and add these super arcs to the super arc set E_S.

8 Return the super digraph $H_S(V_S(V_i), E_S, F_S(V_i))$

End

Note that the algorithm **FACTOR_k** exploited above is generalized from the factoring algorithm **FACTOR** such that a formula of the disjunction of distinct nk literals, where $n \geq 1$, will not be or-factored further into nk subformulas containing only one single literal each. It can be easily verified that we only need to slightly revise the algorithm **FACTOR** and the two corresponding algorithms, **AND-FACTOR** and **OR-FACTOR**, to achieve this goal. After the previous grouping procedure, we will use the AOSP digraph recognition algorithm to examine if the super digraph H_S is an AOSP digraph except that we do not need the first step in the algorithm **REV_PAR_REDUCE** since all attached formulas have been factored in the grouping procedure.

The following argument shows that this algorithm for recognizing k-AOSP digraphs is correct. If H^k is a k-AOSP digraph, after the grouping procedure, the resultant super digraph H_S has the same structure with an AOSP digraph by definition. So our algorithm can recognize it. On the contrary, if H^k is not a k-AOSP digraph, there are two cases. The first case is that the digraph is constructed from a non-AOSP digraph with k replications, either of incorrect graph structures or of incorrect logic structures. The other case is that the digraph is a digraph which is not constructed from any digraph with k replications. The former will be found by the grouping procedure or the AOSP recognition algorithm; whereas the latter will be found during the grouping procedure.

Here we begin to evaluate the time complexity of the algorithm ***k*-AOSP_GROUP**. The same with the grouping procedure for k-ESP digraph recognition, from Step 1 to Step 4 has the time complexity $O(|V|^3)$. As for Step 5, the time complexity of reducing and factoring formulas attached to $(|V| - 1)$ vertices is $O(|V|L^4)$, and the similarity checking step needs the time complexity $O(|E| + |V|L^3)$ since there are $|E|$ literals to be mapped and the time complexity for checking the isomorphism of two formulas is $O(L^3)$. So the time complexity of Step 5 is $O(|E| + |V|L^4)$. Another dominant step, Step 6, has the time complexity $O(|V||E|)$ since the corresponding literal of every vertex

needs to compare with at most $|E|$ literals to see which leaves it is in. Hence, the total time complexity for the grouping procedure is $O(|V|^3 + |V|L^4)$ because $|E|$ is bounded by $O(|V|^2)$. Moreover, the time complexity of the series and parallel reductions is the same as the ESP recognition algorithm, and thus is $O(|V| + |E|)$. So we can conclude that the k-AOSP recognition algortihm needs the time complexity $O(|V|^3 + |V|L^4)$.

5.3 RAOSP Digraph Recogniton Algorithms

Now the recognition algorithm for RAOSP digraphs will be proposed. The same with the recognition of k-AOSP digraphs, the first step is to group vertices with the same predecessor and successor sets for the input digraph. Secondly, we have to examine the similarity of the Boolean formulas attached to vertices in a group. Likewise, the formulas attached to vertices in a group have to be similar pairwise with the mapping rule that the literals corresponding to the arcs from a vertex of a preceding group to all the vertices of this group are mapped pairwise. In the next step, we have to divide a group according to the logic structures attached to the vertices of the succeeding group, by exploiting the factoring result of any attached formula of the succeeding group. By the definition of RAOSP digraphs, the irreducible forms of their attached formulas can be expanded from factoring trees with leaves of the disjunction of distinct literals. So we can consider that the corresponding factoring trees of RAOSP digraphs have leaves of the disjunction of distinct literals. If there exist l vertices in a group with that their corresponding literals are in a leaf of the factoring tree corresponding to its succeeding group, these l vertices will be grouped as a new group. After dividing groups, a leaf is divided into several leaves, where a leaf contains literals whose corresponding vertices are in the same group. In addition, we also have to check if the corresponding literals are in the same leaf for the vertices in a group that is not divided in the step of dividing groups. Subsequently, we will make a group of vertices as a super vertex. And the arcs between two super vertices will be grouped as a super arc. Their corresponding literals will also be grouped as a *super literal*. The resultant super digraph should have the same structure with an AOSP digraph. Therefore, we can use the AOSP recognition algorithm to examine it. Specifically, the grouping procedure for RAOSP digraphs is

designed as follows.

RAOSP_GROUP($H^{m\times n}(V(v), E, F(v))$)

Begin

1 For every vertex v in V, get its corresponding $P(v)$ and $S(v)$.

3 Add all vertices in V to a list VL.

4 Remove a vertex v from VL, and remove every vertex v' with $v' \equiv v$ in VL. Then make them as a group Q. Add Q to a group set VG. Repeat this step until no vertex remains in VL.

5 For each group Q in VG,

5.1 For every formula F attached to the vertices in Q, **REDUCE**(F).

5.2 Check if all these irreducible formulas are similar pairwise with the mapping that the literals corresponding to the arcs from a vertex of a preceding group in the digraph to all the vertices of Q are mapped pairwise.

5.3 Get the irreducible formula F' attached to a vertex in Q and **FACTOR_REV**(F'). If F' is not fully factorable, reply "$H^{m\times n}$ is not an RAOSP digraph" and stop; else associate its factoring tree T_f with Q.

6 For each group Q in VG,

6.1 If the number of its succeeding groups in the digraph is more than one,

6.1.1 Get all the corresponding factoring trees of its succeeding groups.

6.1.2 Check if the corresponding literals of the vertices in Q are in the same leaf for every factoring tree. If not, reply "$H^{m\times n}$ is not an RAOSP digraph" and stop.

6.1.3 Add Q to the super vertex set V_S.

6.1.4 For every factoring tree T_f, if the corresponding literals are contained in a leaf of T_f with more than $|Q|$ literals, remove these corresponding literals from this leaf to make them as a new leaf, and then do the corresponding revision for T_f.

6.2 If the number of its succeeding groups in the digraph is only one,

6.2.1 Get the corresponding factoring tree T_f of its only one succeeding group.

6.2.2 Divide Q into subgroups, where a subgroup contains vertices whose corresponding literals are in the same leaf of T_f. If there exists a vertex in Q whose corresponding literal is not in any leaf of T_f, reply "$H^{m\times n}$ is not an RAOSP digraph" and stop.

6.2.3 Pick a subgroup P. Add P to the super vertex set V_S, and associate the factoring tree corresponding to Q with P.

6.2.4 If the corresponding literals are contained in a leaf of T_f with more than $|P|$ literals, remove these corresponding literals from this leaf to make them as a new leaf, and then do the corresponding revision for T_f.

6.2.5 Repeat Step 6.2.3 and Step 6.2.4 for other subgroups.

7 For each super vertex V_i in V_S,

7.1 Get its corresponding factoring tree T_f, and replace the literals of every leaf with a super literal.

7.2 Find all super arcs connected to it and add these super arcs to the super arc set E_S.

8 Return the super digraph $H_S(V_S(V_i), E_S, F_S(V_i))$

End

Similarly, the algorithm **FACTOR_REV** exploited above is generalized from the factoring algorithm **FACTORING** such that a formula of the disjunction of distinct literals, will not be or-factored further into subformulas containing only one single literal each. Finally, we will also use the AOSP recognition algorithm to examine if the super digraph G_S is an AOSP digraph. If so, then $H^{m\times n}$ is an RAOSP digraph. With the similar argument for the correctness of the k-AOSP digraph recognition algorithm, we can also prove the correctness of this algorithm for recognizing RAOSP digraphs. Besides, the time complexity of the propose algortihm is also $O(|V|^3 + |V|L^4)$.

Remark that the same with RESP digraphs, a super vertex with m vertices can also be viewed as n super vertices with that the total number of vertices in these n super vertices is m, where $n \geq 1$. However, due to Theorem 1, it is better to consider any super vertex as a whole one except that there is other information showing that it has

to be divided.

6. Conclusions

Many modern varieties of task digraphs belong to the class of AOSP digraphs. For this type of digraph, we can calculate the task reliability in linear time; whereas this problem is known to be NP-hard for general digraphs [7]. In addition, the task response time of AOSP digraphs can also be precisely estimated in linear time by a new analytic model developed in [5], instead of time-consuming simulation methods. Therefore, it is crucial to recognize AOSP digraphs for evaluating computation tasks. A polynomial time algorithm has been proposed for recognizing AOSP digraphs in [1]. In this paper, we considered the fault-tolerant variants of AOSP digraphs, RAOSP digraphs, which are obtained from AOSP digraphs by adding replications to each vertex and adding proper arcs between two vertices. For RAOSP digraphs, we can also calculate its task reliability and task response time in linear time with the similar methods of AOSP digraphs, so it is another important work to recognize RAOSP digraphs. We have proposed a polynomial time algorithm for recognizing RAOSP digraphs in the context. Besides, as a basis for the recognition of RAOSP digraphs, it is first described how to recognize the fault-tolerant variants of ESP digraphs, RESP digraphs, in polynomial time. RESP digraphs are constructed from ESP digraphs by the same replication approach of RAOSP digraphs and thus are a special case of RAOSP digraphs without considering the logic structures among modules.

References

[1] J. Tsai, D. R. Liang and H. H. Chou, "On Recognition of And-Or Series-Parallel Digraphs," *Technical Report, TR-IIS-00-005*, Institute of Information Science, Academia Sinica, Apr. 2000. Also available at http://www.iis.sinica.edu.tw/LIB/threebone0005.html.

[2] J. A. Stankovic, "A perspective on distributed computer systems," *IEEE Trans. Comput.*, vol. C-33, pp. 1102–1115, Dec. 1984.

[3] T. C. K. Chou and J. A. Abraham, "Load redistribution under failure in distributed systems," *IEEE Trans. Comput.*, vol. C-32, pp. 799–808, Sep. 1983.

[4] J. Garcia-Molina, "Reliability issues for fully replicated distributed databases," *IEEE Computer*, vol. 16, pp. 34–42, Sep. 1982.

[5] W. W. Chu and K. K. Leung, "Module replication and assignment for real-time distributed processing systems," *Proceed. IEEE*, vol. 75, pp. 547–562, May 1987.

[6] V. W. Mak and S. F. Lundstrom, "Predicting performance of parallel computations," *IEEE Trans. Parallel and Distributed Systems*, no. 1, pp. 257–270, July 1990.

[7] D. R. Liang, R. H. Jan and S. K. Tripathi, "Reliability analysis of replicated and-or graphs," *Networks*, vol. 29, pp. 195–203, 1997.

[8] J. L. Baer, "A survey of some theoretical aspects of multiprocessing," *ACM Comput. Surveys*, vol. 5, pp. 31–80, Mar. 1983.

[9] J. Riordan and C. E. Shannon, "The number of two terminal series parallel networks," *J. Math. Physics*, vol. 21, pp. 83–93, 1942.

[10] A. Adam, "On graphs in which two vertices are distinguished," *Acta Math.*, vol. 12, pp. 377–397, 1961.

[11] R. J. Duffin, "Topology of series-parallel networks," *J. Math. Anal. Appl.*, vol. 10, pp. 303–318, 1965.

[12] J. Valdes, R. E. Tarjan and E. L. Lawler, "The recognition of series parallel digraphs," *Siam J. Comput.*, vol. 11, pp. 298–313, May 1982.

[13] P. A. Alsberg and J. D. Day, "A principle for resilient sharing of distributed resources," *In Proc. 2nd Intl. Conf. on Software Engineering*, pp. 562–570, Oct. 1976.

[14] S. T. Levi, S. Tripathi, S. Carson and A. Agrawala, "The Maruti hard real-time operating system," *ACM Operating System Review*, vol. 23, pp. 90–105, 1989.

[15] D. Powell, *Delta-4: Overall system specification*, The Delta-4 Project Consortium, 1988.

[16] E. Mendelson, *Boolean algebra and switching circuits*. McGraw-Hill, 1970.