On Recognition of And-Or Series-Parallel Digraphs

Jichiang Tsai, De-Ron Liang and Hsin-Hung Chou

Institute of Information Science
Academia Sinica
Taipei 11541, Taiwan

Address of Correspondent:

Jichiang Tsai
Institute of Information Science
Academia Sinica
Taipei 11541, 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 expressed by the combination of three common types of subgraphs: *sequential*, *And-Fork to And-Join (AFAJ)* and *Or-Fork to Or-Join (OFOJ)*. This class of task digraphs has been modeled as *And-Or Series-Parallel (AOSP)* digraphs. 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. Moreover, we can also precisely estimate *task response time*, which is the time from the invocation of a task to the completion of its execution, in linear time for AOSP digraphs. Task response time is an important design criterion for real-time computer systems. Hence, to examine if a task digraph is an AOSP digraph becomes a useful work for evaluating computation tasks. In this paper, we propose a polynomial time algorithm to recognize AOSP digraphs. The logical structures among modules of an AOSP digraph will be formulated as Boolean formulas, and such formulas own the defined fully factorable property. The main part of our work is the factoring algorithm, which can fully factor a positive CNF.

Keywords: Task digraphs, Boolean formulas, graph recognition, distributed processing systems, reliability, response time.

1. Introduction

Distributed processing involves cooperation among several loosely coupled computers communicating over a subnetwork. 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 [1] - [3]. Several issues of such systems, namely, process management, load balancing, file management, access control, distributed algorithms, etc., are under widespread investigation [2] - [7].

The computation task of a distributed processing system can usually 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. Particularly, 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 [8, 9]: *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. This type of subgraph indicates a thread of the computation task. 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. This type of subgraph may correspond to the case in which the modules assigned to different computers require concurrent processing. 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. This type of subgraph facilitates the system to process one of several threads based on certain selection criteria. In [10], 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

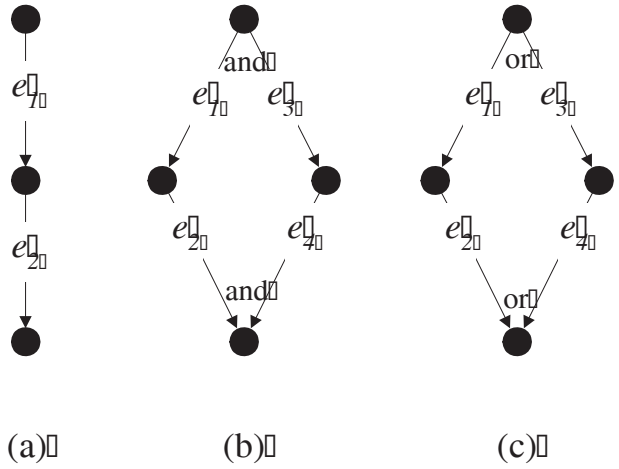

Figure 1: (a)Sequential subgraph; (b)And-Fork to And-Join subgraph; (c)Or-Fork to Or-Join subgraph.

represented by different modules. The same technique can also applied to recursive structures by first determining the mean number of recursive calls, and then represent different instances of the recursive routine by different modules. Using this technique together with others in [11], cyclic graphs can be converted to acyclic graphs [9].

Modules and communication links may fail due to two main factors: software failures and hardware failures. Software failures are caused by design faults or implementation faults. Hardware failures are caused by transient failures or permanent failures. So modules and communication links have a certain probability of being operational. Then 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 using the technique proposed in [10]. 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. Key parameters that affect task response time include interprocessor communications, processor loading, module precedence relationships and interconnection network delay. A new analytic model developed in [8] 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 (sometimes called two-terminal series-parallel digraphs), which arise in the analysis of electrical networks [12] - [14], was proposed in [15]. 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, we propose a polynomial algorithm to recognize AOSP digraphs in this paper. The logical structures among modules of an AOSP digraph will be formulated as Boolean formulas, and such formulas own the defined fully factorable property. Moreover, in order to achieve the goal of recognize AOSP digraphs, we will also introduce factoring algorithms to see if a Boolean formula is fully factorable.

The rest of the paper is organized as follows. In Section 2, some definitions and notations employed in the context are described. Then we will define factorable formulas and factorable trees, and also give the formal definition of AOSP digraphs in the next section. In Section 4, the recognition algorithm for AOSP digraphs, including factoring algorithms for Boolean formulas, will be proposed. Finally, we conclude the paper in Section 5.

2. Preliminaries

Our graph-theoretical terminology follows Bondy and Murty [16]. A graph $G = (V, E)$ consists of a finite set of vertices V and a finite set of edges E. Each edge is a pair (u, v) where u and v are distinct vertices. A *subgraph* of G is a graph having all of its vertices and edges in G. A graph is *connected* if there is a path joining each pair of vertices. A *connected component* of a graph is a maximal connected subgraph. If the edges of a graph G are unordered pairs, then G is an *undirected graph*; if the edges are ordered pairs, called *arcs*, then G is a *directed graph* (abbreviated *digraph*). For each arc (v, w) which *leaves* v and *enters* w, v is a *predecessor* of w and w is a *successor* of v. A vertex v in a digraph is a *source* if no arc enters v and a *sink* if no arc leaves v.

Next, we give the logical terminology according to [17]. 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** if and only if both x_1 and x_2 are **true**. Symmetrically, the *disjunction* of x_1 and x_2, $x_1 \vee x_2$, is **false** if and only if 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 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*. Symmetrically, a conjunction of literals in that no variable appears twice is called a *fundamental conjunctive formula*. Any disjunction of fundamental conjunctive formulas is called a *Disjunctive Normal Formula* (abbreviated *DNF*) or a formula in the *disjunctive normal form*. Since CNF and DNF are *dual*, when we discuss properties of normal formulas in the context, we only consider CNF for simplicity of presentation. However, it is easy to show that these properties can also apply to DNF.

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 $|L(F)|$. A clause C_1 is said to be an *induced clause* of another clause C_2 provided that $L(C_1)$ is a subset of $L(C_2)$, denoted by $C_1 \propto C_2$. Two clauses C_1 and C_2 are said to be *distinct* if and only if $C_1 \not\propto C_2$ and $C_2 \not\propto C_1$. If two clauses C_1 and C_2 with $L(C_1) = L(C_2)$, we say C_1 and C_2 are *isomorphic*, denoted by $C_1 \stackrel{iso}{=} C_2$. Furthermore, 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, denoted by $F_1 \stackrel{iso}{=} F_2$. 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*.

There are three laws in Boolean algebra, which are useful to our algorithms. We

describe them in the following.

Idempotent Law:

1. $P \vee P = P$;
2. $Q \wedge Q = Q$;

Absorption Law:

1. $P \wedge (P \vee Q) = P$;
2. $P \vee (P \wedge Q) = P$;

Distributive Law:

1. $P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)$;
2. $P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)$, where P, Q and R are Boolean formulas.

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.

3. AOSP Digraphs

Prior to the definition of AOSP digraphs, we will describe the definition of *fully factorable formulas* first. 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:

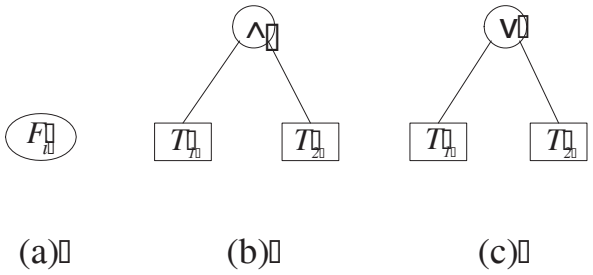

Figure 2: (a)A leaf node; (b)Conjunctive composition of two factoring trees; (c)Disjunctive composition of two factoring trees.

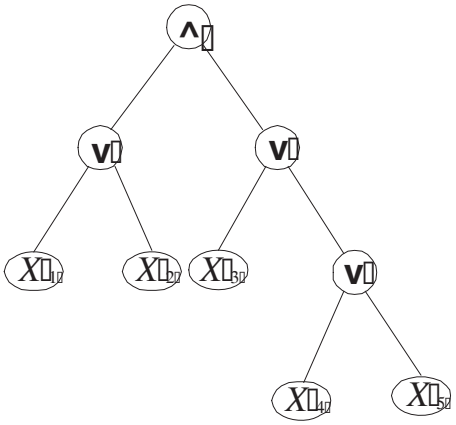

Figure 3: The factoring tree corresponding to the Boolean formula $(x_1 \vee x_2) \wedge (x_3 \vee x_4 \vee x_5)$.

(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 are the Boolean formulas equivalent to F.

To record the factoring process on a Boolean formula F, we construct a binary tree T_f using the rules in Figure 2. This binary tree is named the *factoring tree* corresponding to F. The internal node in T_f has a sort in $\{\wedge, \vee\}$ and the external leaf has a sort of Boolean formula F_i. 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 3. Obviously, every two distinct external leaves of a fully factoring tree contain distinct literals according to the definition of fully factorable formulas.

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 call that F is *expanded* from T_F. For example, the formula $F = (x_1 \vee x_2) \wedge (x_3 \vee x_4 \vee x_5)$ is expanded from the fully factoring tree shown in Figure 3.

Now, we begin to give a formal definition for AOSP 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 $G = (V(v), E, F(v))$, 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 [10].

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 G_1 and G_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 G_1 and G_2, with t_1 identified with s_2;
   (b) *Parallel-and composition* ($P_\wedge$): The digraph $H_{P_\wedge}$ is an AOSP digraph with terminals s and t, where $H_{P_\wedge}$ is the disjoint union of G_1 and G_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* ($P_\vee$): The digraph $H_{P_\vee}$ is an AOSP digraph with terminals s and t, where $H_{P_\vee}$ is the disjoint union of G_1 and G_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 $F_{t_1} \vee F_{t_2}$.

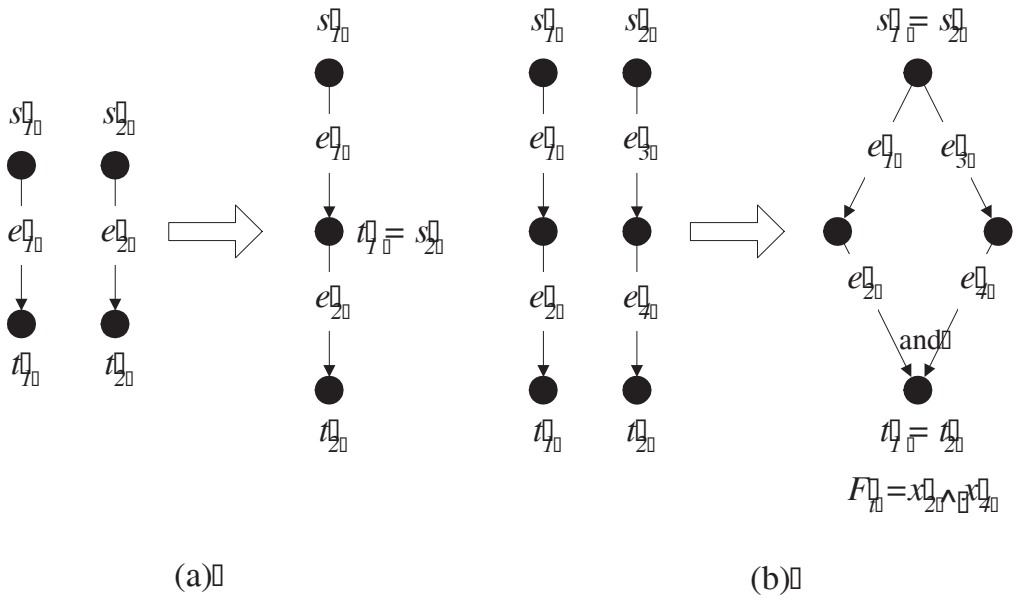

Figure 4: (a) Series composition; (b) Parallel-and composition; or Parallel-or composition if $F_t = x_2 \vee x_4$.

Figure 4(a) gives an example series composition, and Figure 4(b) depicts an example parallel-and composition. Moreover, if we change the formula F_t in Figure 4(b) to be $x_2 \vee x_4$, it will become a parallel-or composition.

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, $P_\wedge$ or $P_\vee$ to represent the series, parallel-and or parallel-or composition of the AOSP digraphs corresponding to the subtrees rooted at the children of the node. A parsing tree thus provides a concise description of the structure of an AOSP digraph and facilitates the calculation of task reliability [10] and task response time [8]. Figure 5 gives an example AOSP digraph and its corresponding parsing tree.

According to Definition 2, we know that a Boolean formula attached to a vertex must be able to be fully factored. Besides, if we ignore the Boolean formulas and just consider the topology of an AOSP digraph, it is exactly an ESP digraph. Therefore, two

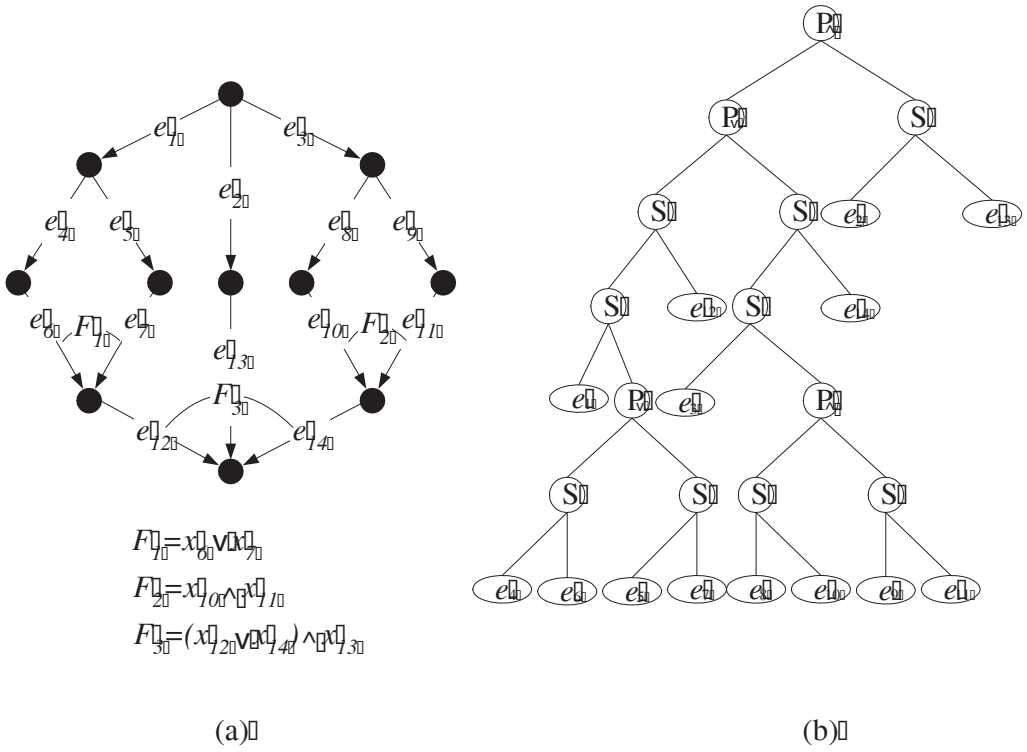

Figure 5: (a) An AOSP digraph; (b) Its corresponding parsing tree.

main works needed by the recognition algorithm for AOSP digraphs are to recognize whether this digraph is an ESP digraph and to examine whether attached formulas are fully factorable. We will illustrate this algorithm in the next section.

4. Algorithms of Recognizing AOSP Digraphs

4.1 Overall Algorithms

Now we proceed to design algorithms needed for recognizing AOSP digraphs. For our overall algorithm, the input is a task digraph with a set of Boolean formulas attached to vertices. The flow chart of the overall recognition algorithm is depicted in Figure 6. This algorithm is generalized from the recognition algorithm for ESP digraphs proposed in [15]. The latter 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. We redesign the

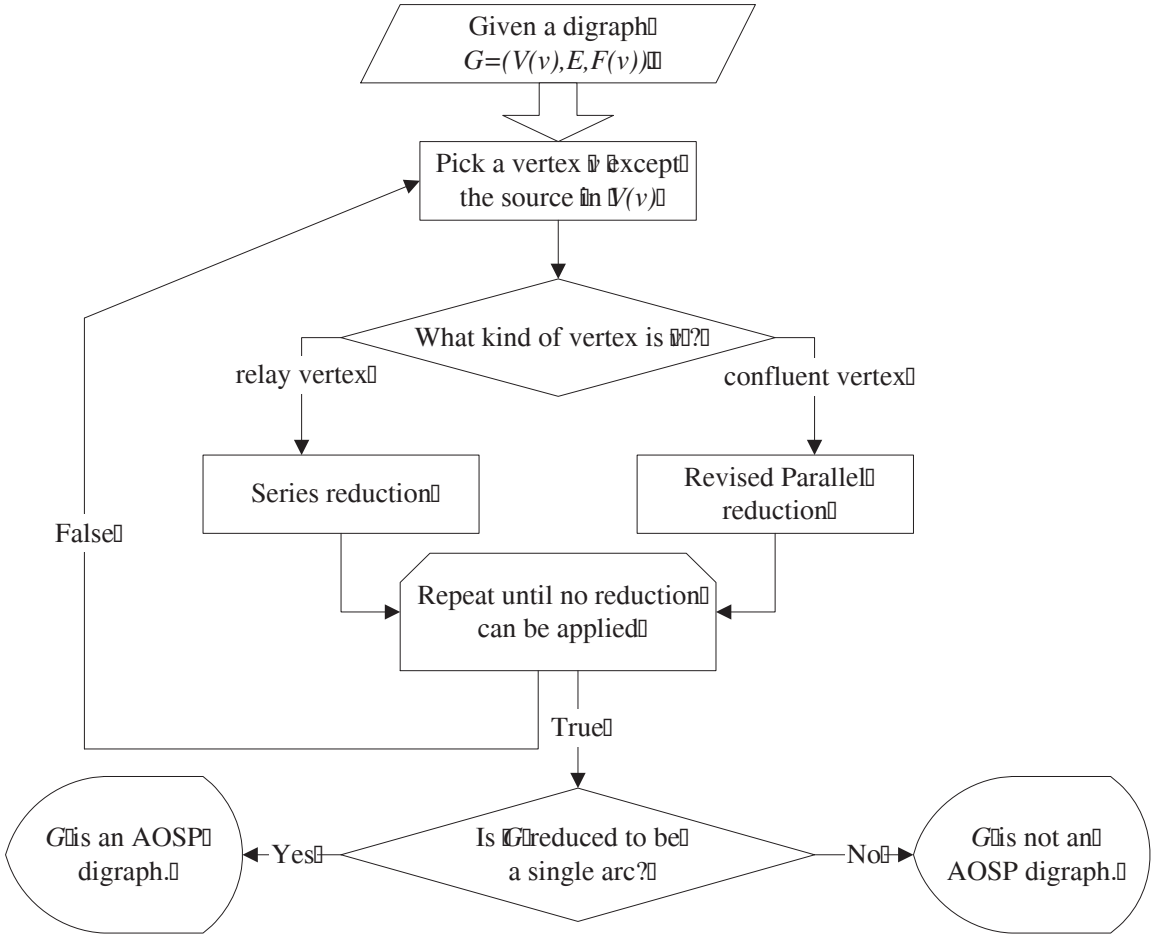

Figure 6: The flow chart of the overall recognition algorithm.

parallel reduction algorithm 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 7.

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, this type of vertex is called a *relay vertex*. For a vertex containing more than one entering arcs but only one predecessor, this type of vertex is called a *confluent vertex*. We maintain a list of vertices called *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 overall

algorithm **RECOGNITION** is described as below.

RECOGNITION$(G(V(v), E, F(v)))$

Begin

1 Add all vertices in $V(v)$ 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 revised parallel reduction. If the parallel reduction fails, reply "G is not an AOSP digraph" and stop; else if u is not the source and also not in UL, 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 reduced to be a single arc, reply "G is an AOSP digraph" and the parsing tree T_p; else reply "G is not an AOSP digraph".

End

We continue to introduce the revised parallel reduction mentioned above. The input for the corresponding algorithm **PARALLEL REDUCTION** 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, we assume that the input Boolean formula is in its conjunctive normal form.

PARALLEL REDUCTION(E', F)

Begin

1 Recognize whether F is a fully factorable formula by the factoring algorithm **FACTORING**. 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" and return.

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

End

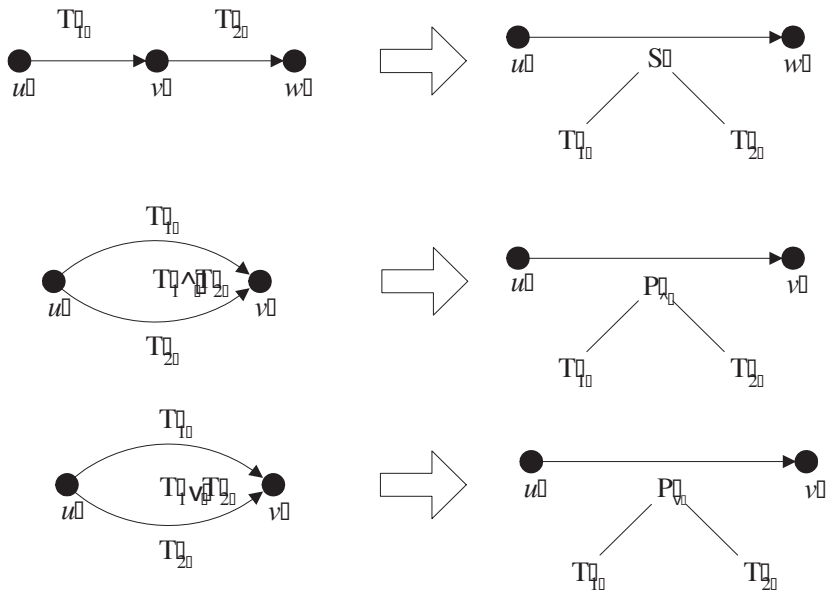

Figure 7: Rules to perform reductions and to construct the parsing tree.

In the next subsection, we will begin to design the factoring algorithm exploited in the previous algorithm. This algorithm can fully factor a CNF and construct the corresponding factoring tree.

4.2 Boolean Formula Factoring Algorithms

Figure 8 illustrates the flow chart of the factoring algorithm. According to Definition 1, a formula is fully factorable if and only if its irreducible form is fully factorable. And we found that an irreducible formula is easier to be factored. Hence, our factoring algorithm reduces the input formula to be irreducible in advance. Reducing a general conjunctive normal formula to be irreducible is known to be an NP-complete problem, but it is not so hard to reduce a positive conjunctive normal formula. The following theorem characterizes the property of irreducible positive CNFs.

Theorem 1: Given a positive CNF F, F is irreducible *if and only if* every two clauses of F are distinct [18].

Thus, applying the previous theorem, the irreducible form of a positive CNF can be obtained by a polynomial time algorithm shown below.

REDUCE(F)

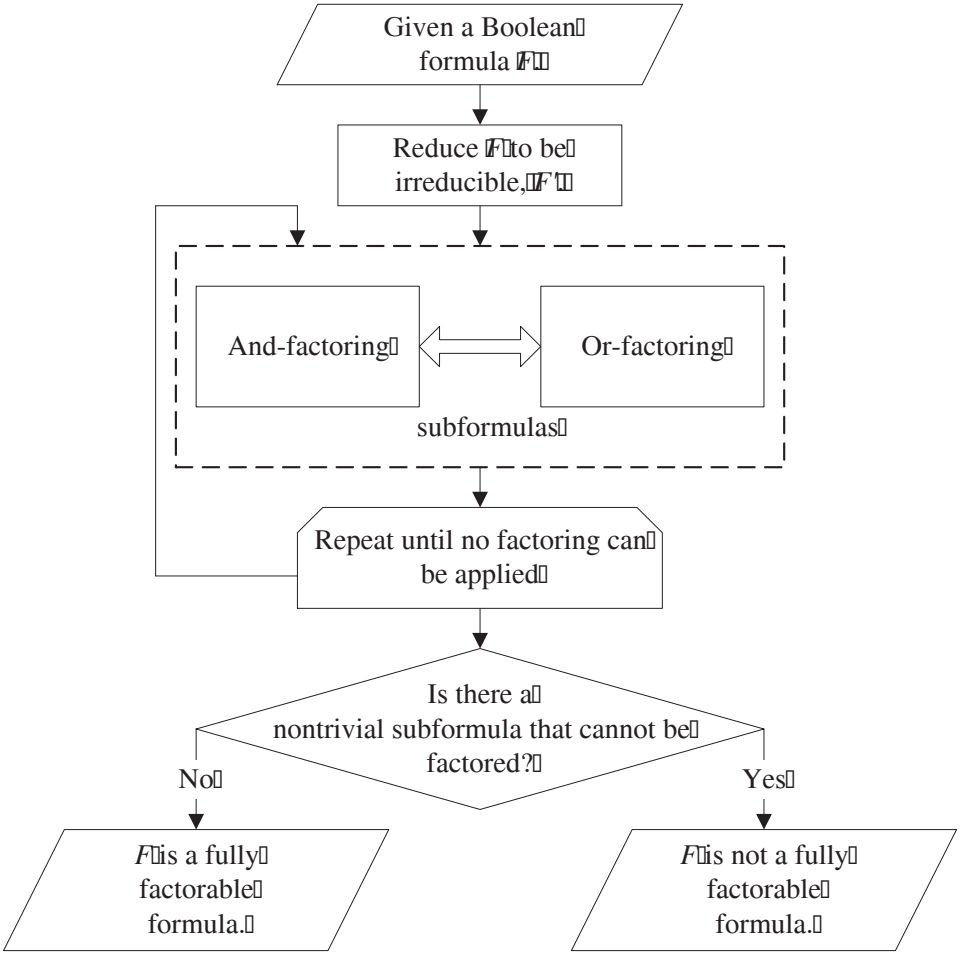

Figure 8: The flow chart of the factoring algorithm.

Begin

Eliminate redundant clauses by the idempotent law and the absorption law.

End

The next step is to fully factor the irreducible CNF. We 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. Particularly, the following theorem demonstrates that a formula can be only factored by one of the and-factoring and or-factoring algorithms, but not both.

Lemma 1: Given k CNFs $F_1, F_2, \ldots, F_k$ where $F_i = C_{i1} \wedge C_{i2} \wedge \ldots \wedge C_{in_i}$ and C_{ij} is

the clause of F_i, for $1 \leq i \leq k$ and $1 \leq j \leq n_i$. Let $F' = (C_{11} \wedge C_{12} \ldots \wedge C_{1n_1}) \wedge (C_{21} \wedge C_{22} \ldots \wedge C_{2n_2}) \ldots \wedge (C_{k1} \wedge C_{k2} \ldots \wedge C_{kn_k})$, i.e. $F_1 \wedge F_2 \wedge \ldots \wedge F_k$, and $F" = (C_{11} \vee C_{21} \vee \ldots \vee C_{k1}) \wedge (C_{11} \vee C_{21} \vee \ldots \vee C_{k2}) \wedge \ldots \wedge (C_{11} \vee C_{22} \vee \ldots \vee C_{k1}) \wedge (C_{11} \vee C_{22} \vee \ldots \vee C_{k2}) \wedge \ldots \wedge (C_{1n_1} \vee C_{2n_2} \vee \ldots \vee C_{kn_k})$, i.e. $F_1 \vee F_2 \vee \ldots \vee F_k$ expanded by the distributive law. If $F_1, F_2, \ldots, F_k$ are irreducible and pairwisely disjoint, both F' and $F"$ are irreducible CNFs.

Proof: First, we prove $F' = (C_{11} \wedge C_{12} \ldots \wedge C_{1n_1}) \wedge (C_{21} \wedge C_{22} \ldots \wedge C_{2n_2}) \ldots \wedge (C_{k1} \wedge C_{k2} \ldots \wedge C_{kn_k})$ is an irreducible CNF. Since F_i is irreducible, by Theorem 1, any two clauses of F_i are distinct, for $1 \leq i \leq k$. And F_i and F_j are disjoint, for $i \neq j$. Then we have that C_{ij} is a clause of F' for $1 \leq i \leq k$ and $1 \leq j \leq n_i$, and that any two clauses of F' are distinct. Therefore, from Theorem 1, F' is an irreducible CNF.

Next we begin to demonstrate $F" = (C_{11} \vee C_{21} \vee \ldots \vee C_{k1}) \wedge (C_{11} \vee C_{21} \vee \ldots \vee C_{k2}) \wedge \ldots \wedge (C_{11} \vee C_{22} \vee \ldots \vee C_{k1}) \wedge (C_{11} \vee C_{22} \vee \ldots \vee C_{k2}) \wedge \ldots \wedge (C_{1n_1} \vee C_{2n_2} \vee \ldots \vee C_{kn_k})$ is also an irreducible CNF. Similarly, since F_i is irreducible, by Theorem 1, any two clauses of F_i are distinct, for $1 \leq i \leq k$. Moreover, F_i and F_j are disjoint, for $i \neq j$. Then we have that $(C_{1i_1} \vee C_{2i_2} \vee \ldots \vee C_{ki_k})$ is a clause of $F"$ for $1 \leq i_j \leq n_j$ and $1 \leq j \leq k$, and that any two clauses of $F"$ are distinct. Therefore, from Theorem 1, $F"$ is also an irreducible CNF. □

Lemma 2: Given any two irreducible CNFs F_1 and F_2, $F_1 = F_2$ *if and only if* $F_1 \stackrel{iso}{=} F_2$.

Proof: There are two parts to be proved.

(a) Suppose $F_1 = F_2$. Let $F_1 = C_{11} \wedge C_{12} \wedge \ldots \wedge C_{1n_1}$ and $F_2 = C_{21} \wedge C_{22} \wedge \ldots \wedge C_{2n_2}$ where C_{ij} is the clause of F_i, for $1 \leq i \leq 2$ and $1 \leq j \leq n_i$. Now suppose F_1 and F_2 are not isomorphic. There are two possible cases.

(a.1) There exists one clause in F_1 in the sense that it is not isomorphic to any clause in F_2. Without loss of generality, C_{11} is assumed to be such a clause. First suppose there is a clause C_{2i} in F_2 with the property that $C_{2i} \propto C_{11}$. It is obvious that C_{2i} is not isomorphic to C_{11}. Since F_1 is irreducible, any two clauses of F_1 are distinct by Theorem 1. Then $C_{1k} \not\propto C_{2i}$ due to $C_{1k} \not\propto C_{11}$, for $2 \leq k \leq n_1$. This means that any clause in F_1 contains a variable foreign to C_{2i}. Hence let every variable in C_{2i} be **false** but other variables be **true**. It will result in that $F_1 =$ **true** but $F_2 =$ **false**. Obviously,

it leads to a contradiction.

Therefore there does not exist a clause C_{2i} in F_2 with $C_{2i} \propto C_{11}$. This means that any clause in F_2 contains a variable foreign to C_{11}. Likewise, we can let every variable in C_{11} be **false** but other variables be **true**. It will result in that $F_1 =$ **false** but $F_2 =$ **true** and also lead to a contradiction.

(a.2) There exists one clause in F_2 in the sense that it is not isomorphic to any clause in F_1. With the similar proof to the former case, we can show that this case is also impossible.

So we can conclude $F_1 \stackrel{iso}{=} F_2$.

(b) Now suppose $F_1 \stackrel{iso}{=} F_2$. It is trivial that $F_1 = F_2$. □

Theorem 2: Given an irreducible CNF F, F can be only factored by one of the and-factoring and or-factoring algorithms, but not both.

Proof: Suppose that F can be factored as $F_1 \wedge F_2$ where F_1 and F_2 are two disjoint formulas and can also be factored as $F_3 \vee F_4$ where F_3 and F_4 are two disjoint formulas. Without loss of generality, we can assume that F_1, F_2, F_3 and F_4 are all irreducible CNFs since every Boolean formula can be transformed into its irreducible conjunctive normal form. Let $F_i = C_{i1} \wedge C_{i2} \wedge \ldots \wedge C_{in_i}$ where C_{ij} is the clause of F_i, for $1 \leq i \leq 4$ and $1 \leq j \leq n_i$. Now let $F' = (C_{11} \wedge C_{12} \wedge \ldots \wedge C_{1n_1}) \wedge (C_{21} \wedge C_{22} \wedge \ldots \wedge C_{2n_2})$, i.e. $F_1 \wedge F_2$. Since F_1 and F_2 are irreducible and disjoint, from Lemma 1, F' is an irreducible CNF. Moreover, let F" $= (C_{31} \vee C_{41}) \wedge (C_{31} \vee C_{42}) \wedge \ldots \wedge (C_{31} \vee C_{4n_4}) \wedge (C_{32} \vee C_{41}) \wedge \ldots \wedge (C_{3n_3} \vee C_{4n_4})$, i.e. $F_3 \vee F_4$ expanded by the distributive law. Likewise, from Lemma 1, F" is also an irreducible CNF. Because F' and F" are equivalent irreducible CNFs, by Lemma 2, F' and F" are isomorphic. However, it is obvious that we cannot divide clauses of F" into two disjoint sets. On the contrary, the clauses of F' can be divided into two disjoint sets since it can be and-factored as $F_1 \wedge F_2$. This leads to a contradiction. □

The previous theorem states that the selection of factoring operations on a given formula is unique. In addition, we want to remind readers that if a formula can be exactly and-factored (or-factored) as k subformulas, it is the most effective to factor it into exactly k subformulas. This is because a subformula which can be further and-factored (or-factored) can not be or-factored (and-factored) subsequently. In the following, we give a

formal definition for this scenario. Note that $F_1, F_2, \ldots, F_k$ are pairwisely disjoint from the definition of factoring operations.

Definition 3: A Boolean formula F is *thoroughly and-factored* (*thoroughly or-factored*) as $F_1, F_2, \ldots, F_k$ if and only if F can be and-factored (or-factored) as $F_1, F_2, \ldots, F_k$ and F_i cannot be and-factored (or-factored) further, for $1 \leq i \leq k$.

Therefore, the selection of factoring operations can be considered as and-factoring and or-factoring alternately. Next, we characterize one property of thoroughly and-factoring as below.

Theorem 3: If F can be thoroughly and-factored as $F_1, F_2, \ldots, F_k$ and as $F'_1, F'_2, \ldots, F'_{k'}$ where F_i and F'_j are irreducible CNFs, for $1 \leq i \leq k$ and $1 \leq j \leq k'$, we have $k = k'$ and for any F_i, there exists an F'_j such that $F_i \stackrel{iso}{=} F'_j$ and vice versa.

Proof: Suppose that $F_i = C_{i1} \wedge C_{i2} \wedge \ldots \wedge C_{in_i}$ where C_{il} is the clause of F_i, for $1 \leq i \leq k$ and $1 \leq l \leq n_i$, and $F'_j = C'_{j1} \wedge C'_{j2} \wedge \ldots \wedge C'_{jn'_j}$ where C'_{jl} is the clause of F'_j, for $1 \leq j \leq k'$ and $1 \leq l \leq n'_j$. Let $F' = (C_{11} \wedge C_{12} \ldots \wedge C_{1n_1}) \wedge (C_{21} \wedge C_{22} \ldots \wedge C_{2n_2}) \ldots \wedge (C_{k1} \wedge C_{k2} \ldots \wedge C_{kn_k})$, i.e. $F_1 \wedge F_2 \wedge \ldots \wedge F_k$. Since $F_1, F_2, \ldots, F_k$ are irreducible and pairwisely disjoint, by Lemma 1, F' is an irreducible CNF. Also, let $F" = (C'_{11} \wedge C'_{12} \ldots \wedge C'_{1n'_1}) \wedge (C'_{21} \wedge C'_{22} \ldots \wedge C'_{2n'_2}) \ldots \wedge (C'_{k'1} \wedge C'_{k'2} \ldots \wedge C'_{k'n'_{k'}})$, i.e. $F'_1 \wedge F'_2 \wedge \ldots \wedge F'_{k'}$. Similarly, by Lemma 1, we have that $F"$ is an irreducible CNF. Because F' and $F"$ are equivalent irreducible CNFs, by Lemma 2, we have $F' \stackrel{iso}{=} F"$. Hence, for any C_{ip}, $1 \leq i \leq k$ and $1 \leq p \leq n_i$, there exists a C'_{jq}, $1 \leq j \leq k'$ and $1 \leq q \leq n'_j$, such that $C_{ip} \stackrel{iso}{=} C'_{jq}$, and vice versa. Now suppose there exists some F_i with that there is no F'_j, $1 \leq j \leq k'$, which is isomorphic to F_i. There are two possible cases for every F'_j.

(a) F'_j does not contain all the corresponding isomorphic clauses of F_i in $F"$. This means that some corresponding isomorphic clauses of F_i belong to some other F'_l where $j \neq l$. Therefore, the clauses of F_i can be divided into disjoint sets. This leads to a contradiction since F_i cannot be and-factored further.

(b) F'_j contains not only all the corresponding isomorphic clauses of F_i in $F"$ but also some other clauses. This means that some corresponding isomorphic clauses of F'_j in F'

belong to some other F_l where $i \neq l$. Therefore, the clauses of F'_j can be divided into disjoint sets. This leads to a contradiction since F'_j cannot be and-factored further.

So we can conclude that for any F_i, there exists an F'_j such that $F_i \stackrel{iso}{=} F'_j$. And since F'_p and F'_q are disjoint for $p \neq q$, there exists only one F'_j such that $F_i \stackrel{iso}{=} F'_j$. Likewise, we can get that for any F'_j, there exists a unique F_i such that $F'_j \stackrel{iso}{=} F_i$. Consequently, we also have $k = k'$. □

Since the conjunctive normal form and the disjunctive normal form are dual, we can easily get the following corollary on the property of thoroughly or-factoring, applying the similar proof with the previous theorem.

Corollary 1: If F can be thoroughly or-factored as $F_1, F_2, \ldots, F_k$ and as $F'_1, F'_2, \ldots, F'_{k'}$ where F_i and F'_j are irreducible DNFs, for $1 \leq i \leq k$ and $1 \leq j \leq k'$, we have $k = k'$ and for any F_i, there exists an F'_j such that $F_i \stackrel{iso}{=} F'_j$ and vice versa.

It is obvious that if two positive formulas are isomorphic in their irreducible disjunctive forms, they are also isomorphic in their irreducible conjunctive forms. Therefore, we get another corollary.

Corollary 2: If F can be thoroughly or-factored as $F_1, F_2, \ldots, F_k$ and as $F'_1, F'_2, \ldots, F'_{k'}$ where F_i and F'_j are irreducible CNFs, for $1 \leq i \leq k$ and $1 \leq j \leq k'$, we have $k = k'$ and for any F_i, there exists an F'_j such that $F_i \stackrel{iso}{=} F'_j$ and vice versa.

Because every subformula can be transformed into its irreducible conjunctive normal form, we can consider the result of thoroughly and-factoring (thoroughly or-factoring) an irreducible formula as unique according to the previous results. For the sake of neat presentation, we assume that the considered Boolean formula is an irreducible CNF F in the rest of this section. And $F = C_1 \wedge C_2 \wedge \ldots \wedge C_n$ where C_i is the clause of F, for $1 \leq i \leq n$. Moreover, we assume that there are k irreducible and pairwisely disjoint CNFs $F_1, F_2, \ldots, F_k$ and $F_i = C_{i1} \wedge C_{i2} \wedge \ldots \wedge C_{in_i}$ where C_{ij} is the clause of F_i, for $1 \leq i \leq k$ and $1 \leq j \leq n_i$.

Prior to designing the and-factoring algorithm, one important property of the and-factoring operation is demonstrated.

Theorem 4: F can be thoroughly and-factored as $F_1, F_2, \ldots, F_k$ *if and only if* $(C_1 \wedge$

$C_2 \wedge \ldots \wedge C_n) \stackrel{iso}{=} (C_{11} \wedge C_{12} \wedge \ldots \wedge C_{1n_1} \wedge C_{21} \wedge C_{22} \wedge \ldots \wedge C_{2n_2} \wedge \ldots \wedge C_{k1} \wedge C_{k2} \wedge \ldots \wedge C_{kn_k})$, i.e. $F_1 \wedge F_2 \wedge \ldots \wedge F_k$.

Proof: There are two parts to be proved.

(a) Suppose F can be thoroughly and-factored as $F_1 \wedge F_2 \wedge \ldots \wedge F_k$. Here let $F' = (C_{11} \wedge C_{12} \ldots \wedge C_{1n_1}) \wedge (C_{21} \wedge C_{22} \ldots \wedge C_{2n_2}) \ldots \wedge (C_{k1} \wedge C_{k2} \ldots \wedge C_{kn_k})$, i.e. $F_1 \wedge F_2 \wedge \ldots \wedge F_k$. Since $F_1, F_2, \ldots, F_k$ are irreducible and pairwisely disjoint, from Lemma 1, F' is an irreducible CNF. Because F and F' are equivalent irreducible CNFs, by Lemma 2, we have $F \stackrel{iso}{=} F'$. Hence, $(C_1 \wedge C_2 \wedge \ldots \wedge C_n) \stackrel{iso}{=} (C_{11} \wedge C_{12} \wedge \ldots \wedge C_{1n_1} \wedge C_{21} \wedge C_{22} \wedge \ldots \wedge C_{2n_2} \wedge \ldots \wedge C_{k1} \wedge C_{k2} \wedge \ldots \wedge C_{kn_k})$.

(b) Suppose $(C_1 \wedge C_2 \wedge \ldots \wedge C_n) \stackrel{iso}{=} (C_{11} \wedge C_{12} \wedge \ldots \wedge C_{1n_1} \wedge C_{21} \wedge C_{22} \wedge \ldots \wedge C_{2n_2} \wedge \ldots \wedge C_{k1} \wedge C_{k2} \wedge \ldots \wedge C_{kn_k})$. It is trivial that F can be thoroughly and-factored as $F_1 \wedge F_2 \wedge \ldots \wedge F_k$. □

Now we begin to implement the and-factoring algorithm. According to previous theorem, we know that performing an and-factoring operation on an irreducible CNF can directly divide its clauses into disjoint sets. Moreover, this CNF can be thoroughly and-factored into k subformulas if and only if its clauses can be divided into exactly k disjoint sets. So if this formula cannot be and-factored, its clauses cannot be divided into disjoint sets. In order to divide clauses into disjoint sets, we introduce the *clause-joint graph* to exhibit joint relation among clauses, which is defined as below.

Definition 4: The graph $G = (V, E)$ is a *clause-joint graph* corresponding to F if and only if each vertex $v_i \in V$ associates with a clause C_i and each edge $(v_i, v_j) \in E$ represents that $L(C_i) \cap L(C_j) \neq \emptyset$, for $1 \leq i, j \leq n$ and $i \neq j$.

Figure 9 depicts an example clause-joint graph.

Since a connected component of the clause-joint graph associates with a set of clauses which are not disjoint, a property of the clause-joint graph utilized to design the algorithm is given in the following, according to Theorem 4.

Property 1: F can be thoroughly and-factored into k subformulas if and only if the clause-joint graph constructed from F is a graph with exactly k connected components.

According to the property of the clause-joint graph described above, the algorithm **AND-factoring** is consequently implemented as the following procedures. Note that if

Figure 9: The clause-joint graph corresponding to the formula $F = (x_1 \vee x_3) \wedge (x_2 \vee x_3) \wedge (x_4 \vee x_6) \wedge (x_5 \vee x_6)$ where the clauses $C_1 = x_1 \vee x_3$, $C_2 = x_2 \vee x_3$, $C_3 = x_4 \vee x_6$ and $C_4 = x_5 \vee x_6$.

F cannot be and-factored, F will be returned intact.

AND-factoring(F)

Begin

1 If F is a single literal, reply F and return.

2 Construct the clause-joint graph G_C from F.

3 Suppose there are k connected components in G_C. Find the connected components G_C^i of G_C for $1 \leq i \leq k$, by the *Breadth-First Search (BFS)* algorithm.

4 For $1 \leq i \leq k$, generate a CNF F_i containing the clauses corresponding to the nodes in G_C^i.

5 Reply $F_1, F_2, \ldots, F_k$.

End

The following argument shows that this algorithm is correct. If F can be and-factored, its corresponding clause-joint graph consists of two or more connected components according to Property 1. And the nodes of each connected components associate with the clauses of a subformula. Consequently, we can obtain subformulas by Step 4. On the other hand, if F cannot be and-factored, its corresponding clause-joint graph is a connected graph. Then F will be returned intact.

As for the or-factoring algorithm, we also characterize one property about or-factoring operations first.

Theorem 5: F can be thoroughly or-factored as $F_1, F_2, \ldots, F_k$ *if and only if* $(C_1 \wedge C_2 \wedge \ldots \wedge C_n) \stackrel{iso}{=} ((C_{11} \vee C_{21} \vee \ldots \vee C_{k1}) \wedge (C_{11} \vee C_{21} \vee \ldots \vee C_{k2}) \wedge \ldots \wedge (C_{11} \vee C_{22} \vee \ldots \vee C_{k1}) \wedge (C_{11} \vee C_{22} \vee \ldots \vee C_{k2}) \wedge \ldots \wedge (C_{1n_1} \vee C_{2n_2} \vee \ldots \vee C_{kn_k}))$, i.e. $F_1 \vee F_2 \vee \ldots \vee F_k$

expanded by the distributive law.

Proof: There are two parts to be proved.

(a) Suppose F can be thoroughly or-factored as $F_1 \vee F_2 \vee \ldots \vee F_k$. Let $F' = ((C_{11} \vee C_{21} \vee \ldots \vee C_{k1}) \wedge (C_{11} \vee C_{21} \vee \ldots \vee C_{k2}) \wedge \ldots \wedge (C_{11} \vee C_{22} \vee \ldots \vee C_{k1}) \wedge (C_{11} \vee C_{22} \vee \ldots \vee C_{k2}) \wedge \ldots \wedge (C_{1n_1} \vee C_{2n_2} \vee \ldots \vee C_{kn_k}))$, i.e. $F_1 \vee F_2 \vee \ldots \vee F_k$ expanded by the distributive law. Since $F_1, F_2, \ldots, F_k$ are irreducible and pairwisely disjoint, by Lemma 1, F' is an irreducible CNF. Because F and F' are equivalent irreducible CNFs, according to Lemma 2, we have $F \stackrel{iso}{=} F'$. Hence, $(C_1 \wedge C_2 \wedge \ldots \wedge C_n) \stackrel{iso}{=} ((C_{11} \vee C_{21} \vee \ldots \vee C_{k1}) \wedge (C_{11} \vee C_{21} \vee \ldots \vee C_{k2}) \wedge \ldots \wedge (C_{11} \vee C_{22} \vee \ldots \vee C_{k1}) \wedge (C_{11} \vee C_{22} \vee \ldots \vee C_{k2}) \wedge \ldots \wedge (C_{1n_1} \vee C_{2n_2} \vee \ldots \vee C_{kn_k}))$.

(b) Suppose $(C_1 \wedge C_2 \wedge \ldots \wedge C_n) \stackrel{iso}{=} ((C_{11} \vee C_{21} \vee \ldots \vee C_{k1}) \wedge (C_{11} \vee C_{21} \vee \ldots \vee C_{k2}) \wedge \ldots \wedge (C_{11} \vee C_{22} \vee \ldots \vee C_{k1}) \wedge (C_{11} \vee C_{22} \vee \ldots \vee C_{k2}) \wedge \ldots \wedge (C_{1n_1} \vee C_{2n_2} \vee \ldots \vee C_{kn_k}))$. It is trivial that F can be thoroughly or-factored as $F_1, F_2, \ldots, F_k$. □

Intuitively, performing an or-factoring operation on an irreducible CNF can directly extract clauses of subformulas from its clauses according to the previous theorem. Moreover, if this formula cannot be or-factored, it is not isomorphic to any formula of disjoint irreducible CNFs expanded by the distributive law. In the following, we will discuss how to design the extracting procedure.

Since we need to or-factor a formula as many subformulas as possible, a subformula with only one single clause is thus further or-factored into several subformulas with only one literal. Therefore, there is no subformula containing only one single clause. We can consequently conclude the following property.

Property 2: If F can be thoroughly or-factored as $F_1, F_2, \ldots, F_k$, F_i must be a single literal or a formula with two or more clauses, for $1 \leq i \leq k$.

From the distributive law, every two distinct clauses C_{ip} and C_{iq} of F_i are distributed to different clauses of F; whereas for any two clauses C_{ip} and C_{jq} of two distinct subformulas F_i and F_j, there exists a clause of F containing these two clauses. Thus, we have the following another two properties, in which F is assumed to be thoroughly or-factored as $F_1, F_2, \ldots, F_k$.

Property 3: For every two literals $x_a \in C_{ip}$ and $x_b \in C_{iq}$ of F_i where C_{ip} and C_{iq} belong to different disjoint sets of clauses, there is *no* clauses C_l of F such that both x_a and

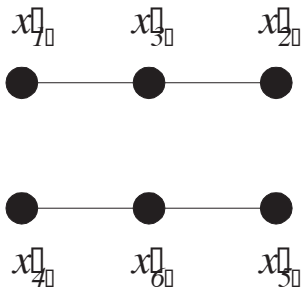

Figure 10: The literal-disjoint graph corresponding to the formula $(x_1 \vee x_2 \vee x_4 \vee x_5) \wedge (x_1 \vee x_2 \vee x_6) \wedge (x_3 \vee x_4 \vee x_5) \wedge (x_3 \vee x_6)$.

$x_b \in C_l$.

Remark that if C_{ip} and C_{iq} are in a set of clauses which are not disjoint, perhaps both x_a and x_b are in the same clause of F_i. Thus there trivially exists a clauses C_l such that both x_a and $x_b \in C_l$.

Property 4: For every two literal $x_a \in C_{ip}$ of F_i and $x_b \in C_{jq}$ of F_j where $i \neq j$, there exists a clauses C_l of F such that both x_a and $x_b \in C_l$.

Hence, we construct the *literal-disjoint graph* to exhibit disjoint relation among literals, which are defined as follows.

Definition 5: The graph $G = (V, E)$ is a *literal-disjoint graph* corresponding to F if and only if each vertex $v_i \in V$ associates with a literal x_i for $1 \leq i \leq m$ where $m = |L(F)|$, and each edge $(v_i, v_j) \in E$ represents that there is *no* clause C_l in F such that both x_i and $x_j \in C_l$.

Figure 10 gives an example literal-disjoint graph.

For a subformula with only one single literal, the corresponding node of its literal in the literal-disjoint graph of F is disconnected from other nodes according to Property 4. For a subformula containing two or more clauses which can be divided into disjoint sets, namely, it can be and-factored in the next step by Theorem 4, the corresponding nodes of its literals form a connected component and are also disconnected from other nodes applying Property 3 and 4. From the foregoing discussion, we obtain the following property.

Property 5: If F can be thoroughly or-factored into k subformulas and every sub-

formula with two or more clauses can be and-factored, its corresponding literal-disjoint graph has exactly k connected components.

The algorithm **OR-factoring** is thus designed as below. Also, if F cannot be or-factored, F will be returned intact.

OR-factoring(F)

Begin

1 If F is a single literal, reply F and return.

2 Construct the literal-disjoint graph G_L from F.

3 Suppose there are k connected components in G_L. Find the connected components G_L^i of G_L for $1 \leq i \leq k$, by the *Breadth-First Search (BFS)* algorithm.

4 For $1 \leq i \leq k$, generate a set S_i containing the literals corresponding to the nodes in G_L^i.

5 Generate a fundamental disjunctive formula D_{ij} with $L(D_{ij}) = S_i \cap L(C_j)$, for $1 \leq i \leq k$ and $1 \leq j \leq n$.

6 Generate a CNF $H_i = D_{i1} \wedge D_{i2} \wedge \ldots \wedge D_{in}$, for $1 \leq i \leq k$.

7 $F_i = \mathbf{REDUCE}(H_i)$, for $1 \leq i \leq k$.

8 If $n \neq n_1 \times n_2 \times \ldots \times n_k$, reply F.

9 If F is isomorphic to the formula of $F_1 \vee F_2 \vee \ldots \vee F_k$ expanded by the distributive law, reply $F_1, F_2, \ldots, F_k$; else reply F.

End

The following argument shows that this algorithm is correct. If F can be or-factored and every subformula with two or more clauses can be and-factored, its corresponding literal-disjoint graph consists of two or more connected components from Property 5. And the nodes of each connected components associate with the literals of a subformula. Consequently, we can obtain subformulas through Step 5 to 7 according to Theorem 5. On the other hand, if F cannot be or-factored and its corresponding literal-disjoint graph is a connected component, F will be returned intact. If the literal-disjoint graph is not a connected component, F will not pass Step 8 or 9 and then will also be returned intact since F is not isomorphic to any formula of disjoint irreducible CNFs expanded by the distributive law by Theorem 5. Remark that for F, which can be or-factored but one of

its subformulas with two or more clauses cannot be and-factored, if it can pass Step 9, obviously it can be thoroughly or-factored as $F_1 \vee F_2 \vee \ldots \vee F_k$; whereas if it can not pass Step 8 or 9, it does not violate the correctness of our algorithm because F is not fully factorable and we do not need to factor it any more.

Last but not least, the overall factoring algorithm is proposed. We maintain 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-factoring**; while OL contains the Boolean formulas needed to be applied by the algorithm **OR-factoring**. Each formula F is thus attached with two Boolean tags, A-tag and O-tag, to indicate if F can be factored by **AND-factoring** or **OR-factoring**. Initially A-tag and O-tag are set to be **true**. If F cannot be factored by **AND-factoring** (**OR-factoring**), A-tag (O-tage) is consequently set to be **false**. The algorithm is shown in the following.

FACTORING(F)

Begin

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

2 If F' is a single literal, reply "F is fully factorable" and return.

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 $AL = \emptyset$.

4.1 Call **AND-factoring**(P) to factor P into a set of subformulas S.

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

$OL = \emptyset$.

5.1 Call **OR-factoring**(Q) to factor Q into a set of subformulas T.

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 $AL = OL = \emptyset$.

7 Reply "F is factorable".

End

Here we begin to evaluate the time complexity of the algorithm **FACTORING**. Let m be the number of literals and n be the number of clauses in F. It is easy to see that the time complexity of the algorithm **REDUCE** is $O(mn^2)$ since we have to compare every two clause of the n clauses with at most m literals. As for the algorithm **AND-factoring**, the time complexity of Step 2 is also $O(mn^2)$, with the similar reason to the algorithm **REDUCE**. Moreover, the time complexity of Step 3 is $O(n^2)$ for the reason that there are n nodes in the clause-joint graph. Since the previous two steps are the dominant steps, the time complexity of the algorithm **AND-factoring** is thus $O(mn^2)$. Symmetrically, the time complexity of Step 2 and 3 in the algorithm **OR-factoring** is $O(nm^2)$ and $O(m^2)$, respectively. Besides, Step 7 is another dominant step and its time complexity is $O(mn^2)$ since the number of clauses in H_i is n and the total number of literals of $H_1, H_2, \ldots H_k$ is m. As for Step 9, because there are n clauses with at most m clauses in both two formulas, its time complexity is also $O(mn^2)$. So we can conclude that the time complexity of the algorithm **OR-factoring** is $O(mn^2+m^2n)$. Let $T(m,n)$ be the time complexity of the algorithm **FACTORING**. We can get $T(m,n)$

in the following recursive equations.

$$
\begin{array}{lll}
T(m,n) \leq & O(mn^2) + T(x,y) + T(m-x, n-y) & \text{if } F \text{ can be and-factored;} \\
T(m,n) \leq & O(mn^2 + m^2 n) + T(x,y) + T(m-x, n/y) & \text{if } F \text{ can be or-factored,} \\
 & \text{where } 1 \leq x \leq m \text{ and } 1 \leq y \leq n; & \\
T(1,1) = 1. & &
\end{array}
$$

Let $l = \Sigma_{i=1}^{n} |L(C_i)|$, i.e. the number of total literals in F. It is obvious that l is the upper bound of m and n. Then we can get the time complexity $T(m,n) = O(l^4)$. Since the time complexity of the ESP recognition algorithm is $O(|V| + |E|)$ [15] and our AOSP recognition algortihm needs at most $(|V| - 1)$ fully factoring operation, the time complexity of the algorithm **RECOGNITION** is then $O(|V|L^4 + |E|)$, where L is the maximum value of l among all attached Boolean formulas.

5. Conclusions

The computation task of a distributed processing system can be usually modeled as a task digraph, and 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 [10]. In addition, the task response time of AOSP digraphs can also be precisely estimated in linear time by a new analytic model developed in [8], instead of time-consuming simulation methods. Therefore, it is crucial to recognize AOSP digraphs for evaluating computation tasks. In this paper, we have proposed a polynomial time algorithm for recognizing AOSP digraphs. Our results extend the previous work on the recognition of ESP digraphs, which are a special case of AOSP digraphs. Moreover, the main part of our work is the factoring algorithm. This algorithm can fully factor a positive CNF, and thus is not only necessary for our problem but also useful for other problems, which need to factor positive Boolean formulas.

Acknowledgements

The authors wish to express their sincere thanks to Hsueh-I Lu (Sinica), Tze-Heng Ma (Sinica) and Da-Wei Wang (Sinica) whose comments greatly helped improve the

presentation of the paper. We would also like to thank Guan Shieng Huang (Sinica) and Kuen-Pin Wu (NTU) for their valuable discussions on the Boolean formula factoring algorithms, and the referees for their valuable comments.

References

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

[2] 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.

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

[4] J. Dion, "The cambridge file server," *ACM Oper. Syst. Rev.*, Oct. 1980.

[5] J. P. Ignizio, D. F. Palmer and C. M. Murphy, "A multicriteria approach to supersystem architecture definition," *IEEE Trans. Comput.*, vol. C-31, pp. 410–418, May 1982.

[6] K. B. Irani and N. G. Khabbaz, "A methodology for the design of communication networks and the distribution of data in distributed supercomputer systems," *IEEE Trans. Comput.*, vol. C-31, pp. 420–434, May 1982.

[7] W. H. Kohler, "A survey of techniques for synchronization and recovery in decentralized computer systems," *ACM Comput. Surveys*, vol. 13, pp. 149–183, June 1981.

[8] 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.

[9] 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.

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

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

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

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

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

[15] 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.

[16] J. A. Bondy and U. S. R. Murty, *Graph theory with applications*. Macmillan, 1976.

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

[18] W. V. Quine, "The problem of simplifying truth functions," *American Mathematical Monthly*, vol. 59, pp. 521–531, Oct. 1952.