9-1990

A Colored Petri Net-Based Approach for Automated Deadlock Detection in Parallel Programs

N. Mansouri
Syracuse University, Department of Engineering and Computer Science, namansou@ecs.syr.edu

Amrit L. Goel
Syracuse University, algoel@syr.edu

SU-CIS-90-29

A Colored Petri Net-Based Approach for Automated Deadlock Detection in Parallel Programs

Nashat Mansour and Amrit L. Goel

September 1990

School of Computer and Information Science
Suite 4-116
Center for Science and Technology
Syracuse, New York 13244-4100

(315) 443-2368

A COLORED PETRI NET-BASED APPROACH FOR AUTOMATED DEADLOCK DETECTION IN PARALLEL PROGRAMS

by

Nashat Mansour*
Amrit L. Goel**

September 1990

* N. Mansour is with the School of Computer and Information Science, Syracuse University, Syracuse, NY 13244.

** Amrit L. Goel is with the Department of Electrical and Computer Engineering and the School of Computer and Information Science, Syracuse University, Sryacuse, NY 13244.

ABSTRACT

A static analysis approach is proposed for automated detection of deadlocks in a common class of parallel programs, referred to as Single Code Multiple Data (SCMD) programs. It is based on colored Petri net (CP-net) modeling and reachability analysis, where colors correspond to parallel processes. An SCMD program is first translated into a CP-net and a reachability tree is then derived and analyzed for deadlock information. CP-subnets representing basic programming language constructs are described. These subnets are employed as building blocks by an algorithm that translates synchronization-related statements of a process in an SCMD program and connects the resulting subnets. The connection technique makes use of the characteristics of SCMD programs to produce a unified and folded CP-net model. These charateristics are also used to introduce a notion, referred to as poset-covering, that leads to a reduced reachability tree for the CP-net. The usual algorithm for generating and analyzing reachability trees of CP-nets is modified by including poset-covering and excluding notions that are irrelevant to our application. The compactness of the CP-net model and the reachability tree makes the proposed approach appealing for practical implementation.

INDEX TERMS:

Colored Petri nets, deadlock detection, parallel programs, static analysis.

I. INTRODUCTION

Testing is a complex and expensive phase in the software development lifecycle. For parallel software, the complexity of testing is further increased due to concurrency, communication and synchronization among processes. Research in this field is still in its infancy, but as parallel computing proliferates, there is an increasing need for testing approaches that reduce the complexity and effort involved. Specifically, there is a need for practical approaches, which can be supported by automated tools.

In general, a parallel program may include synchronization errors as well as the usual errors found in sequential programs. The most important synchronization error is deadlock. In this paper and in most of the testing literature, deadlock refers to all kinds of infinite blockages. Current approaches for testing parallel programs can be divided into two categories; static and dynamic. In static analysis, the program code is transformed into a model which is then analyzed for detecting specific error states. Static analysis approaches have been based on flow-graph [18] or Petri net [16], [6] models of parallel processes. For both models a state space is derived, which is a concurrency state graph for the former and a reachability graph for the latter, and then searched for deadlock states and other information. In dynamic analysis, the program is executed with selected input data, and its behavior and output are examined. Most of the dynamic analysis work has been based on deterministic execution testing [17] aiming at solving the reproducibility problem.

These approaches result in a large state space whose generation and analysis is expensive in terms of time and space. Other static analysis approaches have been proposed to reduce the size of the state space. For example, place and transition invariants of a Petri net model have been employed to guide a selective generation of reachability graph paths [15]. Task interaction graphs, which divide a process according to interactions not control flow, have led to a smaller state space [13]. An algorithm has been suggested for analyzing shared memory multiprocessor programs in which parallelism is a result of executing multiple copies

of the same task in some sections of the program [14]. In this approach, a reduction in the search space is achieved by merging sets of states. In [10] and [11], colored Petri nets (CP-nets) are proposed as models for concurrent algorithms which offer a compact and folded representation. CP-nets are analyzed for deadlocks by place invariants or reachability analysis. Invariant analysis is difficult particularly for automation. Reachability analysis in [11] and [8] has used the notions of marking equivalence and w-covering to produce a reduced-size reachability tree. The class of concurrent algorithms for which this approach has been applied are system management algorithms, such as dining philosophers, readers-writers and database management. In the same spirit, CP-nets have been employed for modeling Lamport's mutual exclusion algorithm and net invariants have been used for the deadlock analysis.

The approach presented in this paper for automated testing of parallel programs is based on static analysis. A large class of parallel programs, referred to as single code multiple data (SCMD) programs, is considered. A program of this class is first translated into a folded CP-net model and a reduced-size reachability tree is then derived and employed as an analysis vehicle for deadlock detection. The class of SCMD parallel programs considered here consists of Multiple Instruction Multiple Data (MIMD) programs that are copies of a single program or process, where each process may have its own input data subdomain. The model of interprocess communication considered here is the message-passing rendezvous type. The testing approach will be concerned with the synchronization behaviour of these programs, specifically for deadlock detection. Therefore, only synchronization-related program statements are considered by the translation algorithm. In the sequel, it is assumed that testing for sequential errors has been carried out separately by appropriate techniques. The translation algorithm considers the code of a process and produces a unified and compact CP-net model that represents the synchronization behavior of all processes of an SCMD program. The algorithm uses CP-subnet models of basic language constructs as templates and devises a technique for connecting them appropriately. The total CP-net model is executed to produce a

reachability tree from which information about deadlocks can be inferred. A notion, referred to as poset-covering, based on the characteristics of SCMD programs is introduced and used in the algorithm that generates and analyzes the reachability tree. This results in a reduced-size tree. The modeling and analysis algorithms presented here are suitable for automation. Our approach shares common objectives with previous work in [11], [8], [1] and [14] in the sense that it leads to compact models and economic analysis. However, our contributions lie in considering the rendezvous model of interprocess communication in the SCMD class of parallel programs, the procedure for translating SCMD programs into a unified and compact CP-net model, and in the generation and analysis of reduced-size reachability tree based on poset-covering.

The paper is organized as follows. Section II provides a description of the SCMD class of programs and their characteristics. An overview of CP-nets is given in Section III. In Section IV, the translation of SCMD programs into CP-nets is described. The reachability analysis for deadlock detection is given in Section V. Illustrative examples are also included in Sections IV and V. Conclusions are presented in Section VI.

II. PROBLEM DESCRIPTION

As mentioned above, in this paper we are concerned with deadlock detection by static analysis of parallel programs for message-passing MIMD multiprocessors. The class of programs dealt with, i.e. Single Code Multiple Data (SCMD) programs, refers to programs that are formed of copies of a single process. For this class of programs, the input data domain is partitioned into subdomains and the same algorithm is applied to each one. That is, parallel processes, although asynchronous, behave the same way in terms of computation and communication on different data. Often, these processes repeat a sequence of computation and communication/synchronization statements until certain criteria are met. Inter-process communication is usually performed only with the nearest neighbors and the sequence of communication/synchronization statements is exactly the same in all processes except in the boundary processes. Depending upon the algorithm used, the communication structure of SCMD programs can be represented as either a one-dimensional chain, a two-dimensional four-nearest-neighbor array or an eight-nearest-neighbor array. This style of programming is applicable to a broad class of problems such as the solution of discretized partial differential equations and solution of linear equations by direct or iterative techniques [2], [5].

The linear chain model is used here for illustrating the CP-net approach. This one-dimensional array structure covers a large number of important parallel algorithms, as emphasized in [12] and [9]. The two-dimensional array model is suggested in Section VI as an extension within the same approach. For the linear array model, the program can be viewed as a chain of N processes, each with unique identification number (ID). The sequence of ID's can be 1, 2, ..., N from left to right, as depicted in Figure 1. Each inner process communicates and synchronizes with its left and right neighboring processes whose ID's differ by 1. Boundary processes, with ID's 1 and N, communicate with only one neighbor.

The message-passing mechanism is assumed to be blocking. That is, a send statement blocks the sending process until a corresponding receive statement is executed in the receiving process. At this time, the message is passed and the sending and receiving

processes continue execution independently. Similarly, a receive statement blocks a process until a corresponding send statement is executed. In such a model, inter-process communication is associated with synchronization and the entire activity will be henceforth referred to as rendezvous. For example, Ada [3] and CSP [7] follow this model of synchronization. It is sometimes referred to as loose synchronization [5].

With the rendezvous model of message-passing, synchronization among processes in the linear chain occurs in a certain order. For example, if all the replicated processes execute a send-right statement, the rightmost process in the chain (with ID=N) is the first that can accept the rendezvous and proceed, then the second rightmost process (with ID=N-1), then process N-2, ..., etc. This natural sequence of rendezvous activity will be captured and exploited by the CP-net model and the analysis procedure.

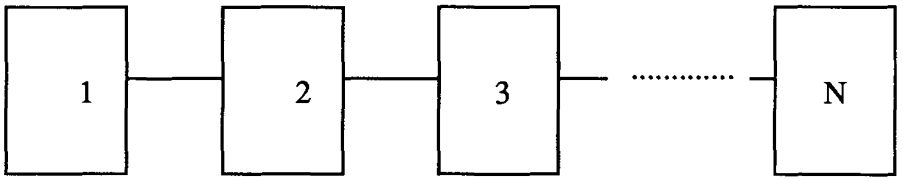


Fig. 1 A linear array model of an SCMD program with N processes.

III. OVERVIEW OF COLORED PETRI NETS

A colored Petri net (CP-net) is a 6-tuple CPN = (P, T, C, I_-, I_+, M_O) [11], where P is a set of places, T is a set of transitions, C is a color function that associates a set of possible token-colors with each place and a set of possible occurrence-colors with each transition, I_- and I_+ are the negative and positive incidence (linear) functions defined for all (p, t) ε PxT such that $I_-(p, t)$, $I_+(p, t)$ ε [BAG(C(t)) --> BAG(C(p))], and M_O is an initial marking function, defined on P such that $M_O(p)$ ε BAG (C(p)).

Graphically, a CP-net is represented as disjoint sets of places and transitions with directed arcs connecting them. A set of token-colors is associated with each place and a set of occurrence-colors is associated with each transition. Arcs are annotated with incidence functions. A function on an arc from a place to a transition determines what token-colors will be removed from the input place upon firing the transition. A function on an arc from a transition to a place determines what token-colors will be deposited into the output place upon firing the transition. An example of a CP-net is shown in Figure 2 [10].
Enabling Condition: Informally, a transition t with occurrence-color v_t ε C(t) is enabled at marking M if token-colors v_p ε C(p_i) are available at this marking in all input places p_i of t such that $INF_i(v_t) = v_{pi}$ for all p_i, where INF_i is the function on the arc from p_i to t.
Firing Rule: Informally, when a transition t with occurrence-color v_t is enabled, it removes token-colors $INF_i(v_t)$ from input places p_i and deposits token-colors $OUTF_i(v_t)$ into output places, where $OUTF_i$ is the function on the arc from t to an output place.

Both, the enabling condition and the firing rule, assume that the color sets attached to the output places of a transition include the token-color to be deposited in these places after firing the transition.

The state of a CP-net is defined by the marking of all places. A marking refers to the number of tokens and their colors. Figure 3 shows an example of the state of the CP-net of Figure 2 before and after firing a transition.

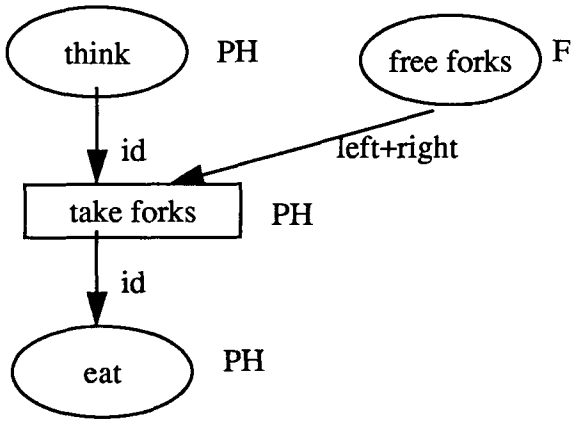


Fig. 2 Part of a CP-net for the dining philosophers system.

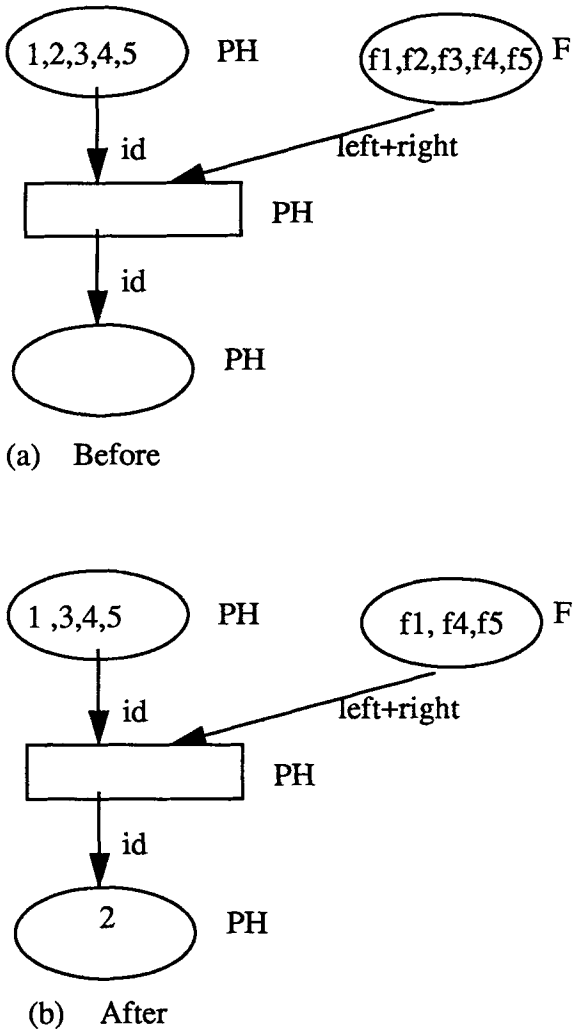


Fig. 3 A CP-net before and after firing a transition (with occurrence-color 2) .

IV. MODELING SCMD PROGRAMS BY CP-NETS

The strategy for modeling an SCMD program by a CP-net is based upon the translation of relevant programming language constructs into CP-subnets and the use of appropriate techniques for connecting the subnets. The CP-subnets are the building blocks of the model. They correspond to communication/synchronization statements, referred to as rendezvous statements, and to control statements that may affect the synchronization behavior of the program. Both types of statements will be referred to as synchronization-related statements. The techniques used for connecting the subnets and, thus, for composing the program's CP-net model have a significant impact on the adequacy of the model and its subsequent analysis. These techniques depend upon the type and sequence of the program statements and are derived from the characteristics of SCMD programs. Since an SCMD program consists of multiple copies of processes, the modeling procedure considers the code of a typical process for translation into a CP-net. However, the CP-net model represents the synchronization behavior of all processes by its folded nature. Each color in the folded model represents a process and the techniques for connecting rendezvous subnets ensures that folding does not mask the requirements for synchronization between distinct processes. This results in a unified and compact representation of SCMD programs. In this section, CP-subnets corresponding to the synchronization-related statements are described and a technique for connecting these subnets to represent an SCMD program is discussed. A translation procedure is then given and illustrated by an example.

A. Model Building Blocks

The language constructs, considered as synchronization-related, are send message or rendezvous-request, receive message or rendezvous-accept, nondeterministic select (i.e. poll) and if-then-else. Since in SCMD programs rendezvous between processes occur in a certain order in the chain, as explained in Section II, the direction in which a rendezvous-request is made or from which such a request is received needs to be reflected in the model. This is

accomplished by including direction information in the CP-subnets of the rendezvous statements as depicted in Figure 4. The labels of the places reflect the direction of the rendezvous, that is whether it is made with the left or the right neighboring process in the chain. The direction also determines the incidence functions that appear as arc labels. The rationale and explanation of the CP-subnets in Figure 4 is given below.

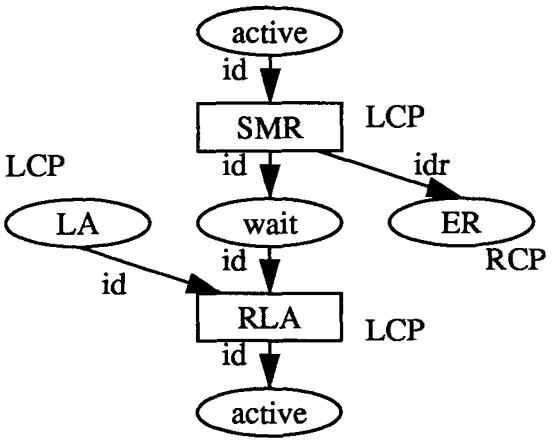
The labels of transitions in the CP-subnets correspond to the statements being modeled. They are Send-Message-Right (SMR) and Receive-Left-Acknowledgement (RLA) for sending a message to the right neighboring process, SML and RRA as the reciprocal of SMR and RLA respectively, Receive-Message-Left (RML) and Send-Left-Acknowledgement (SLA) for receiving a message from the left process, and RMR and SRA as the reciprocal of RML and SLA respectively. Places in the CP-subnets can be divided into rendezvous-places and sequential-places. Rendezvous-places refer to synchronization with neighboring processes. Place labels denote place semantics associated with tokens that may exist in the place. Place semantics will be made use of in the analysis of the CP-net model in Section V. As shown in Figure 4 for the four types of rendezvous statements, labels of rendezvous-places can be Entry-Right (ER) for sending a message to the right or receiving from the left, Left-Acknowledgement (LA) for rendezvous acknowledgement after sending to the right or receiving from the left, and EL and RA which are the reciprocal of ER and LA respectively. Labels of sequential-places can be active or waiting. 'Waiting' labels refer to output places of send-message transitions to express waiting for rendezvous acknowledgement.

The elements of the color sets associated with places and transitions in the CP-subnets are the ID numbers of the processes in the chain representing an SCMD program. To express the natural sequence of rendezvous in the chain model, which has been explained in Section II, a partial order relation "less than" is defined on the elements of token-color sets and occurrence-color sets. The partially (totally in this case) ordered sets, or simply posets, shown in Figure 4 are LCP = {1, 2, ..., N-1} and RCP = {2, 3, ..., N}, where N is the total number of processes in an SCMD program. Other posets, which will be used in the composed CP-net

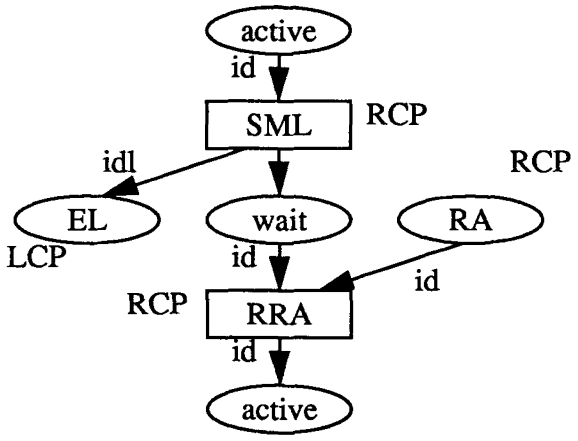
model of the program, are LMCP = {1} and RMCP = {N} corresponding to boundary processes, CP = {1, 2, ..., N} corresponding to the whole chain and INCP = {2, 3, ..., N-1} corresponding to inner processes. The definition of the partial order relation on color sets is central in the analysis of the model as will be explained in Section V.

Arc labels of the CP-subnets are incidence functions. They determine what token-colors are removed from the input places upon firing a transition with a specific occurrence-color, and what token-colors are deposited into the output places. As shown in Figure 4, the functions are idr, idl and id. idr is the label of arcs from SMR and SRA transitions to output rendezvous-places and is defined as idr: LCP --> RCP such that idr(ID) = ID + 1. That is, when a send-right transition fires with an occurrence-color corresponding to process ID, the token-color deposited in the rendezvous place is meant to contribute to the enabling condition of the receive transition of the next process in the chain, on the right-hand side, whose number is (ID + 1). Similarly, idl: RCP --> LCP such that idl(ID) = ID - 1. The function id labels other arcs and is given by id(ID) = ID.

Figure 5 shows CP-subnets for the control statement if-then-else and for a nondeterministic select command. These statements will be included in the program model only when they include rendezvous statements in their direct scope of control. All the places in these subnets are labeled as active, all arcs are annotated with function ID, and posets for the token-colors and the occurrence-colors are determined by those of the joining subnets, as described below in the translation algorithm. Loops do not affect the synchronization behavior of an SCMD program. In such programs, all processes traverse loops the same number of times and execute exactly the same loop body. Therefore, the number of rendezvous occurring within loops is consistent in neighboring processes. The consistency in type and sequence can be analyzed by traversing the loop body only once. As a result, looping can be ignored in the representation model.

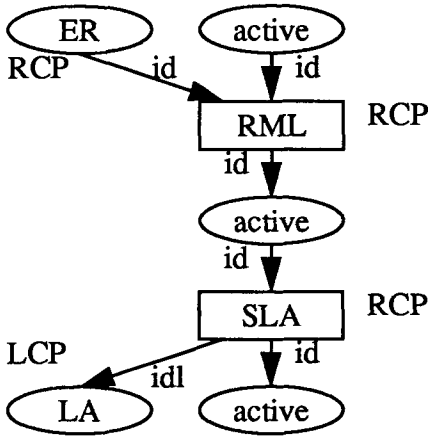


(a)Send-Message-Right : Process[i+1]!Message

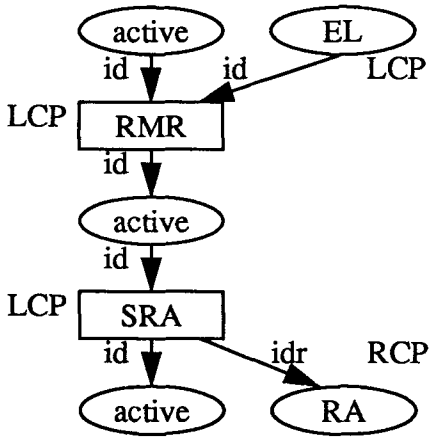


(b)Send-Message-Left : Process[i-1]!Message

Fig. 4

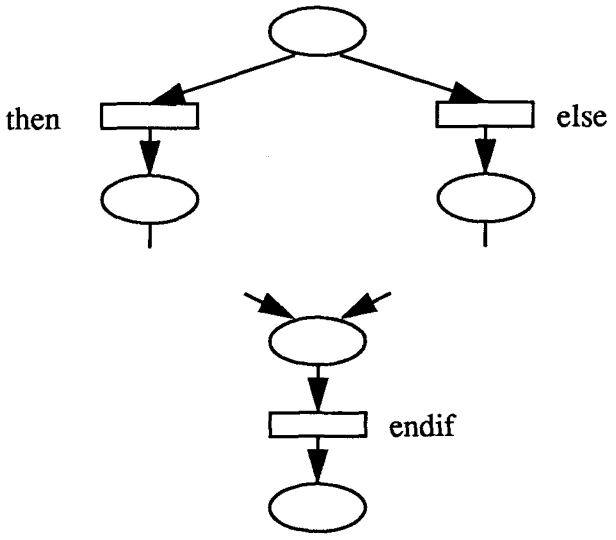


(c)Receive-Message-from Left : Process[i-1]?m

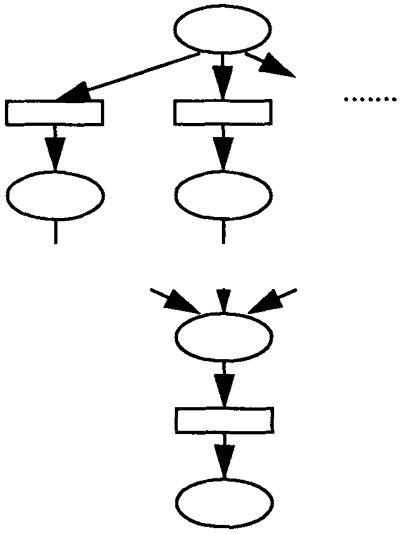


(d)Receive-Message-from Right : Process[i+1]?m

Fig. 4 CP-subnet models of rendezvous statements.



(a) if-then-else-endif



(b) select rendezvous 1
or rendezvous 2
or .....
endselect

Fig. 5 CP-subnets for decision and select statements.
(All arcs are annotated with function id)

B. Subnet Connection and the Translation Procedure

In translating an SCMD program into a CP-net, the CP-subnets depicted in Figures 4 and 5 are used as building blocks, or templates, and only the synchronization-related statements of the program are considered. All templates have places as terminal components, which are either sequential-places or rendezvous-places. Connecting these templates to compose the total CP-net model is an interesting part of the procedure that translates an SCMD program. Specifically, merging rendezvous-places of CP-subnets is a salient feature of the translation. The subnets may correspond to an arbitrary sequence of rendezvous of a process with neighboring processes. The composed model must reflect the synchronization behavior of individual processes as well as that of the whole chain, while retaining the advantage of offering a compact folded representation. Furthermore, the unified model must take into account the idiosyncrasy of the boundary processes which communicate only in one direction.

The direction of a rendezvous, which determines the participant processes, is accounted for by choosing appropriate occurrence-color posets and assigning them to transitions in the CP-subnets of rendezvous and relevant control statements. For example, LCP poset (i.e. {1, 2, ..., N-1}) is assigned to transitions in a send-right CP-subnet to express the restriction that the rightmost boundary process (ID = N) in the chain does not take part in such a rendezvous.

The representation of the synchronization behavior of the chain of processes in the folded model is achieved by merging compatible rendezvous-places. Merging rendezvous-places is based upon self-enabling, which is a condition derived from the characteristics of SCMD programs and is explained in the following. In SCMD programs, a rendezvous activity in one direction must match a rendezvous in the opposite direction. For example, a send-right matches a receive-left. Such a matching is henceforth referred to as reciprocal rendezvous activity and a pair of statements (subnets) representing reciprocal rendezvous activities is referred to as a rendezvous pair of statements (subnets). Also, in SCMD

programs, rendezvous usually occur only between adjacent processes whose IDs differ by a known distance, referred to as the adjacency distance, in the direction of communication. For example, the adjacency distance is 1 in the chain model of SCMD programs. It is positive in the right direction and negative in the left direction. These characteristics of SCMD programs allow merging of compatible rendezvous-places in rendezvous pairs of subnets to produce self-enabling conditions in the folded CP-net. Self-enabling refers to the case where a process enables a transition in a subnet that belongs to another process while both processes are being represented by the same collection of subnets in the folded model. Self-enabling is accomplished by the use of appropriate incidence functions on the arcs as shown in Figures 4 and 5. These functions ensure that appropriate token-colors are deposited in the rendezvous-places of the subnet representing the reciprocal rendezvous activity. They simply increment or decrement the integers representing token-colors by the adjacency distance. For example, a transition in a subnet representing a send-left communication has an output arc annotated with function idl that decrements token-colors by 1. This contributes to the enabling of a receive statement in the left neighboring process, although represented by the same folded model. As a result, self-enabling conditions can be realized by merging the compatible rendezvous-places of rendezvous pairs of subnets in the folded model. for example, the ER place of a Send-Right subnet should be merged with the ER place of the Receive-from Left subnet in the rendezvous pair. In the translation procedure, merging of compatible places is accomplished by using send and receive First-In-First-Out (FIFO) queues, so that places of a rendezvous subnet in the jth location in a queue are merged with compatible places of the reciprocal rendezvous subnet in the same location in the reciprocal queue. The procedure is outlined in Figure 6, in which the following acronyms are used: pid for process ID, TC for Token-Color, OC for Occurrence-Color, seq-place for sequential place, ren-place for rendezvous-place, SR for Send-Right, SL for Send-Left, RL for Receive-Left, RR for Receive-Right, ER for Entry-Right, LA for Left-Acknowledgement, EL for Entry-Left, RA for Right-Acknowledgement and Q for Queue.

Phase I: Scan the entire program to determine initial information and prepare for phase II.

1. Determine the total number of processes N.
2. Determine the numbering scheme for the configuration of interconnected processes, that is 1, 2, ..., N for a chain.
3. Determine the color posets CP, LCP, RCP, LMCP, RMCP and INCP.
4. Filter out irrelevant statements and retain only begin, end and synchronization-related statements in "SR.program".

Phase II: Translate synchronization-related statements and connect subnets.
While not end of "SR.program" repeat steps 5 and 6:

5. Read a statement, look up the table of CP-subnets (Figures 4.1 and 4.2) and fetch the corresponding subnet.
6. If a statement is any of the following types, perform the specified action:
   (a) Begin statement: Assign CP to input place, transition and output place.
   (b) End-type statement: Assign CP to transition and output place. Merged input place takes TCposet of preceding output place.
   (c) Control Statement whose condition does not involve pid:
   - Merge compatible input seq-place of current subnet with output seq-place of preceding subnet. Also assign TCposet of preceding output place to the merged place and to current output place.
   - Assign OCposet = TCposet (of merged place) to transitions.

   (d) Control statement whose condition involves pid:
   - Merge input and output places of current and preceding subnets respectively. Also assign TCposet of preceding output place to the merged place.
   - Assign a consistent pair of OCposets to transitions in alternative paths, i.e. (LMCP and RCP) if condition is (pid > 1) or (LCP and RMCP) if condition is (pid < N).
   - Assign TCposet = OCposet (of input transition) to output places, except for the last output place which is assigned TCposet of the first input place.

   (e) Request rendezvous statement:
   - Merge input seq-place with output seq-place of preceding subnet, and assign to the merged place the TCposet of preceding output place.
   - Assign OCposets to transitions and TCposets to ren-places, and annotate arcs by functions as specified in the subnet.
   - Assign TCposet = OCposet (of input transition) to the output seq-place.
   - Add ren-places to send FIFO queues, i.e. ER place (EL) to SRERQ (SLELQ) and LA Place (RA) to SRLAQ (SLRAQ).
   - If receive FIFO queues are not empty, merge places as follows (deleting places from queues after merging):
     Front(SRERQ) with front(RLERQ), front(SRLAQ) with front(RLLAQ), front(SLELQ) with front(RRELQ) and front(SLRAQ) with front(RRRAQ)

   (f) Receive rendezvous statement:
   - First 3 steps are the same as 6(e).
   - Add ren-places to receive FIFO queues, i.e.,
     ER place (EL) to RLERQ (RRELQ) and LA place (RA) to RLLAQ (RRRAQ).
   - If send FIFO queues are not empty then merge (then delete) places as in step 6(e).

end {phase II}

Fig. 6 Translation procedure.

Example: A simple SCMD pseudocode program is given in Figure 7. It uses Jacobi's iterative method to solve partial differential euqations. The program employs one-dimensional strip-partitioning of the data domain and allocates adjacent subdomains to processes whose indices are consecutive in the chain of processes. The processes repeat Jacobi computations until convergence is reached. In each iteration, every process communicates with its nearest-neighbors for updating boundary data of subdomains and for exchanging information about convergence status.

The CP-net model of a part of the program, obtained by using the translation procedure is shown in Figure 8. Missing function labels of arcs are considered as id and missing color posets are considered as CP.

C. Remarks on the translation procedure and the CP-net model

Clearly, the complexity of the translation procedure is of the order of the number of synchronization-related statements. A CP-net model of an SCMD program will be connected since all subnets have beginning and ending sequential places and such places of successive subnets are always merged by the algorithm. However, the CP-net model suffers from a lack of program semantics information just as do all models produced by static analysis approaches. Its correctness, up to symbolic execution, in terms of its representation of the synchronization behavior of the underlying SCMD program will be dealt with in Section V.

```
Declaration P[pid:1..10]  /*10 copies of process*/
Process P [pid] is
begin
        initialize solution:=0; convergence_flag:=false
        determine data subdomain according to pid
        while not(convergence_flag) do begin
                perform Jacobi computations in subdomain
                if pid < N then {send right}
                        P[pid + 1] ! boundary_data
                        P[pid + 1] ! convergence_flag
                endif
                if pid > 1 then {receive from left}
                        P[pid - 1] ? boundary_data
                        P[pid - 1] ? convergence_flag
                endif
                if pid > 1 then {send left}
                        P[pid - 1] ! boundary_data
                        P[pid - 1] ! convergence_flag
                endif
                if pid > N then {receive from right}
                        P[pid + 1] ? boundary_data
                        P[pid + 1] ? convergence_flag
                endif
        endwhile
        Output partial solution
end process
```

Fig. 7 An SCMD program.

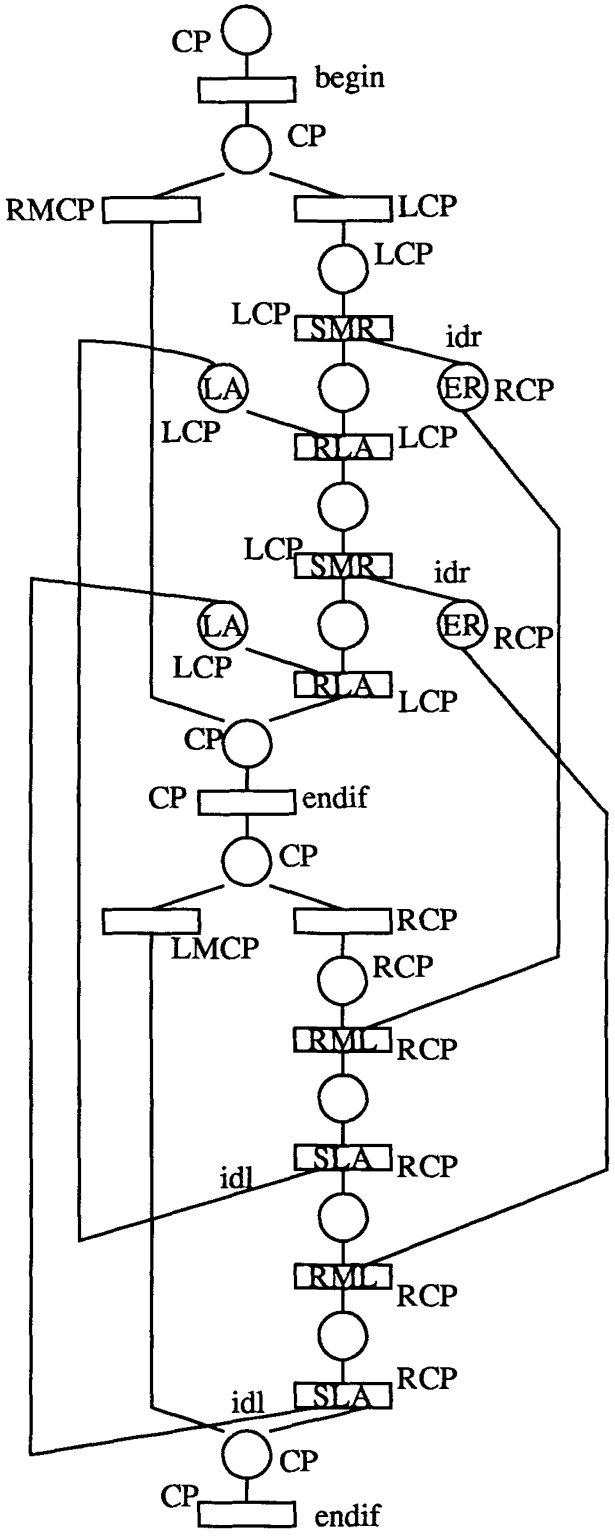


Fig. 8 CP-net model of a part of the program in Fig. 7.

V. DEADLOCK DETECTION

In our approach, deadlock is detected by reachability analysis. An algorithm is described in this section to produce a compact reachability tree by executing the CP-net model and to examine the nodes of the tree for detecting the presence of deadlocks in the underlying program. The tree generation and anlysis algorithm is similar to that in [8] and [11] but is modified here to suit the distinctive features of CP-net models of SCMD programs. Specifically, the considerable reduction in the size of the reachability tree is achieved by using a notion, referred to as p-covering, instead of the notion of equivalence of markings.

The reachability tree is generated by firing all possible enabled transitions in the CP-net in a certain marking. Firing transitions creates new markings which are then explored further. A marking is represented by a node in the tree and refers to a program state. Such a marking or a state is determined by the number of tokens and their associated colors. In this paper, for the application considered, the number of tokens with a specific color in a place is either 1 or 0. In the analysis, some transitions, such as SLA and SRA, and some places, such as output seq-place of SLA, play a distinctive role. Thus, their identities are assumed to be known as a result of the translation algorithm presented in Section 4. Also certain token-colors, namely the least and greatest of posets, have an important expressive role in the generation and analysis of the reachability tree.

A. Poset-Covering and Reachability Analysis

The movement of token-colors is expected to reflect the natural sequence, which is explained in Section 2, in which rendezvous events take place in the chain configuration of SCMD programs. For example, when a send-right communication is performed by processes, token-color N will be the first to reach the end place of a rendezvous pair of subnets followed by token-color N-1, N-2, ..., 1. More precisely, in the CP-net of the chain configuration of an SCMD program, transitions in rendezvous subnets with occurrence-colors corresponding to the boundary process will be enabled first in a rendezvous activity. That is, occurrence-colors N or

1, depending upon the direction of communication, will fire first then its immediate predecessor (or successor) in the poset of occurrence-colors, and so on. The occurrence-color of the immediate predecessor (or successor) of the rightmost (or the leftmost) boundary process is the greatest (or least) in the poset INCP of inner processes. Since boundary and inner processes in the chain differ in that boundary processes communicate only in one direction, the analysis should consider the synchronization behavior of both types of processes. However, all inner processes behave exactly the same way and their behavior can be analyzed in a unified way. Our approach makes use of this observation and employs the universal bounds of the color poset, INCP, in analyzing the folded CP-net model. The universal bound, greatest (or least), is considered representative of the whole poset in a communicate-right (or communicate-left) activity. That is, the completion of a rendezvous by the greatest (or least) of INCP covers the behavior of the whole poset. Rendezvous completion is represented by the token-color reaching an end-place of the second subnet of a rendezvous pair of subnets. The resulting state or marking is referred to as poset-covering, or simply p-covering. The use of the notion of p-covering in the generation algorithm leads to considerable reduction in the size of the reachability tree.

In the reachability algorithm, which generates and analyzes the reachability tree of the CP-net, a p-covering marking M is given by the presence of appropriate token-colors in the end-place of rendezvous subnets in M. Appropriate token-colors are those corresponding to a boundary process and a universal bound (greatest or least) of poset INCP. The end-place of rendezvous subnets is the output seq-place of the second subnet in a rendezvous pair of subnets. These places are assumed to be appropriately labeled by the translation algorithm. Upon detecting a p-covering marking in the tree, the reachability algorithm refrains from generating markings covered by the p-covering marking on the same path. Instead, it summarizes the final outcome of the path in a marking created by directly removing all token-colors from seq-places and ren-places and adding token-colors to the end-place of the rendezvous subnets so that this place will contain all elements of INCP in addition to a universal bound of CP, for example

greatest, that is already there. The only token-color which remains unmoved is that corresponding to the other universal bound of CP, for example least. But, this color is added to the output ren-place of the rendezvous subnets to allow the enabling of transitions with compatible occurrence-color. In this state, the CP-net can be executed by firing enabled transitions and the corresponding reachability tree nodes are generated. The separate treatment of boundary processes stems from their distinctive rendezvous activities. If the token-color of the second universal bound does not deadlock and, hence, reaches the end-place of the rendezvous subnets, this place will contain all elements of CP. This procedure is both motivated and required by the sequence in which rendezvous statements occur in SCMD programs. These statements either occur such that reciprocal statements of a rendezvous pair directly follow each other, such as send-right followed by receive-left, or as a sequence of overlapped identical rendezvous pairs, such as send-right; send-right; receive-left; receive-left. The first type of rendezvous will be referred to as simple and the second one as compound. Such a sequence of rendezvous implies that the end-place of a rendezvous pair of subnets is, in the simple case, the terminal seq-place of the receive subnet and, in the compound case, the terminal seq-place of the last receive subnet. This end-place may be the begin-place of a rendezvous in the opposite direction which requires a different sequence of occurrence-color transition firing. Therefore, before the second rendezvous can be considered, all token-colors must be available in its begin-place in order to enable transitions with compatible occurrence-colors. Moreover, this information can be used to reduce the frequency of examining a marking for p-covering by performing such examination only after firing a transition of SLA or SRA type (see condition in step 2(b) in the algorithm in Figure 9).

A reachability algorithm is presented in Figure 9 and two illustrative examples are given below. Some properties of this algorithm, the reachability tree and the CP-net are discussed in the next subsection.

1. Place all token-colors of CP in begin-place of the CP-net This yields an initial marking at the root of the reachability tree. Add this node to the set UNEXPLORED of frontier nodes. Also, initially, let the set EXPLORED = $\{\emptyset\}$.

While more nodes in UNEXPLORED do:

2. (a) Consider a node D (marking M) in UNEXPLORED.

   (b) If the transition leading to node D is SLA (or SRA) transition, check for p-covering and proceed as follows:
   - Examine token-colors in the output seq-place of the fired transition.
   - If these token colors are only a universal bound (e.g. greatest) of INCP and a universal bound (greatest) of CP then this marking is p-covering.
   - If it is a p-covering marking, create the next marking by removing all tokens from seq-places and ren-places except the other universal bound (least) of CP, and adding token-colors to those in the output seq-place of fired transition such that this place will contain all colors of INCP in addition to the universal bound of CP (greatest) that is already there. Also the other universal bound token-color of CP (least) is added to the output ren-place of the fired transition.

3. (a) Determine an enabled transition t with occurrence color c in M (according to enabling conditions in Section 3).

   (b) For (t, c) enabled in M do:
   - Fire (t, c) and apply rules in Section 3 for token-colors.
   - Add the new nodes (i.e. new markings) to UNEXPLORED.

   (c) If there is no enabled transition in M then:
   - If M indicates a valid termination state, END {algorithm}.
   - Otherwise M indicates deadlock. Save information (marking, last transition fired...) in analysis table.

4. Add D to EXPLORED and delete it from UNEXPLORED.

end while

Fig. 9 Algorithm for reachability tree generation and deadlock detection.

Example without deadlocks:

Consider a portion of an SCMD program, which is simply <<send-right; receive-left>> with 9 processes (N = 9). The CP-net and the reachability tree are shown in Figures 10 and 11 respectively. For clarity of drawing, the tree is simplified by showing simultaneous firing of enabled transitions with several occurrence-colors. Note that place P7 is labeled in Figure 10 as the end-place of the rendezvous pair.

Example with deadlocks:

If "receive-left" is missing from the previous example then clearly node 3 in the tree will be a deadlock state, with color 9 stuck in place P4.

B. Properties of the Algorithm, the Reachability Tree and the CP-net

Property 1: There is no loss of deadlock information due to p-covering. That is, greatest (or least) is representative of the color poset. To support this claim, the following cases are considered:

(a) If greatest (or least) encounters a deadlock, it will not produce self-enabling conditions for the predecessors (successors). Hence, they will all obviously deadlock.

(b) If greatest (least) reaches the end-place of a rendezvous pair of subnets of the simple or compound type, all predecessors (successors) will also successively reach this place. That is, they will not deadlock as shown in the following argument. If greatest (least) reaches the end-place of the rendezvous subnets (i.e. terminates) after firing SLA (SRA), this would mean that its immediate predecessor (successor) is in LA (RA) place of the second subnet of the rendezvous pair. This enables RLA (RRA) transition in the first subnet of the rendezvous pair because, if this were not true, then RLA (RRA) would not have fired with the occurrence-color corresponding to greatest (least) and greatest (least) would not have been able to terminate. A similar argument applies to all other predecessors (successors) in

the color poset.

(c) There is no loss information for a sequence of rendezvous pairs, each can be in any direction. This claim is based upon the following:

- The property holds true for each individual pair in the sequence by (a) and (b), and
- For the composite synchronization behavior, the property also holds since when greatest reaches the end-place of a rendezvous pair, p-covering is detected and the new marking is created. The new marking, which includes all token-colers in the poset, at the interface of two rendezvous pairs implies that the behavior of the next rendezvous pair, is a problem of the same form as for individual rendezvous analyzed in (a) and (b) above. This is true for any two successive rendezvous pairs and hence for the whole CP-net.

Property 2: Large numbers of tokens (known as omega) are not generated by the algorithm in the reachability tree corresponding to the CP-net model of SCMD programs. This is because the algorithm starts with single token-colors, one for each process, in the begin-place of the CP-net, as in step 1, and there is no looping in the net to cause a build up of tokens of the same color. Therefore, the CP-net model is bounded. In fact, since the weight on all arcs is implicitly defined as one, the net is safe for each color, or simply color-safe.

Property 3: Since the CP-net is color-safe and involves no looping and since the net is finite, the token-colors will traverse the net in a finite number of steps corresponding to the finite number of transitions that fire. They end up either in the terminal end-place of the net or will deadlock in some intermediate place. In both cases, the generated reachability tree will be finite and the algorithm will terminate.

Property 4: The size of the reachability tree generated by executing the folded net of an SCMD program is reduced considerably due to the use of p-covering. The number of nodes is somewhat greater than three times that corresponding to a Petri net of a single process, but considerably smaller than that obtained by executing the CP-net for token-colors corresponding to all processes in the SCMD chain. The tripled size is caused by accounting for boundary as well as inner processes in the chain. The significant reduction in tree size is consistent with the

results for the applications discussed in [11] and [8].

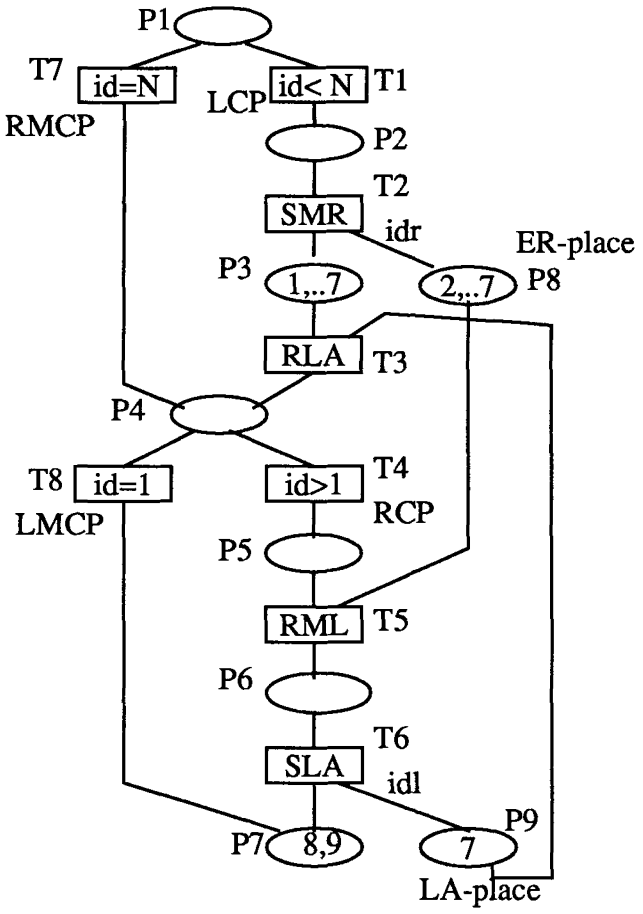


Fig. 10 An example of a CP-net model in an intermediate marking .

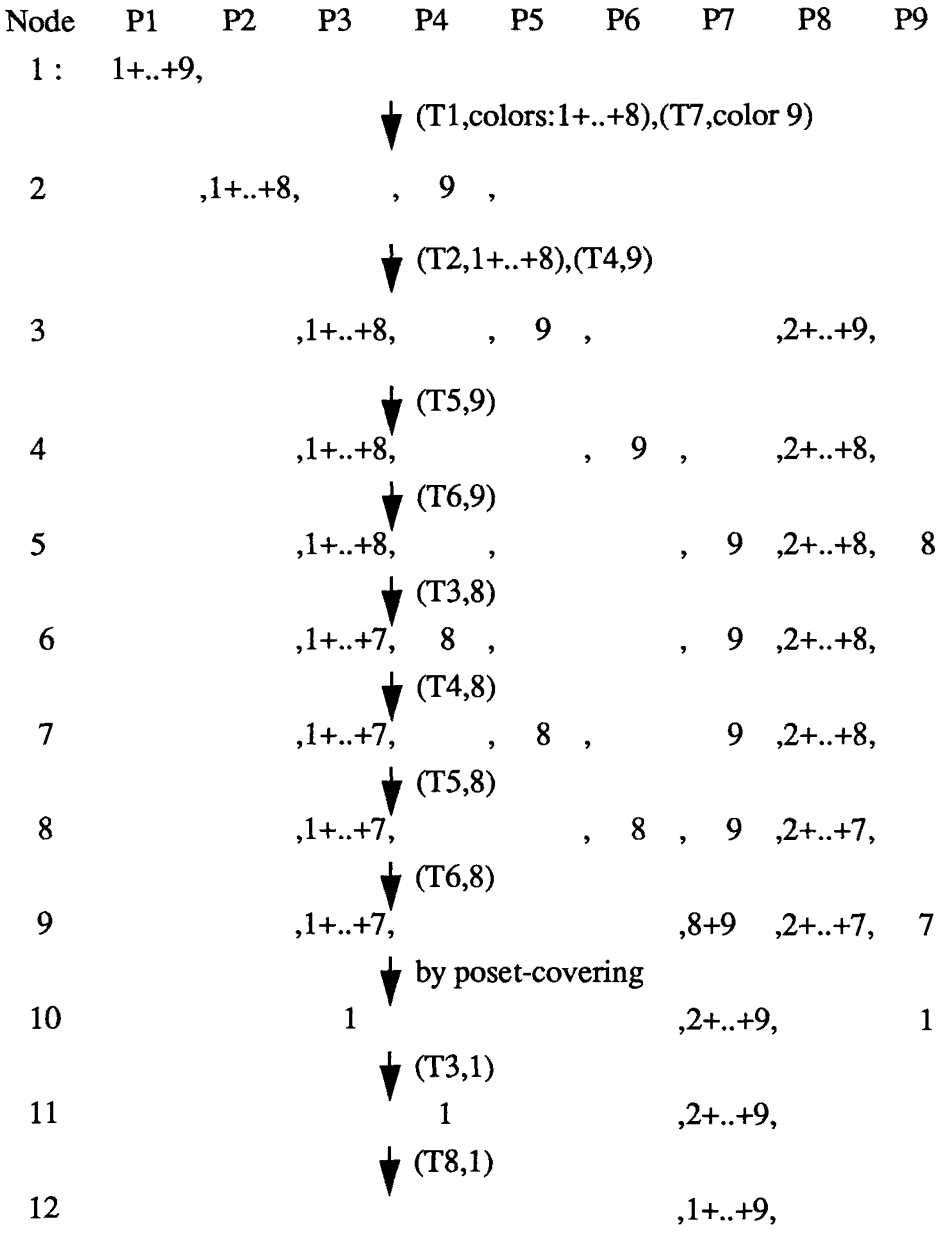


Fig. 11 Reachability tree for CP-net in Fig. 10.

VI. CONCLUSIONS AND FURTHER WORK

An approach has been presented for deadlock detection in SCMD programs. It is based on translating such programs into a folded CP-net model that, despite folding, represents the synchronization behavior of the unified SCMD program. Deadlocks in the underlying program can be detected by deriving and analyzing a reachability tree for the CP-net. The identical synchronization behavior of processes has been employed to considerably reduce the size of the reachability tree. This is accomplished by pruning paths in the reachability tree when a marking, referred to as poset-covering, is reached.

The proposed CP-net-based approach can also be applied to SCMD programs that can be represented as 2-dimensional arrays of asynchronous parallel processes. The same concepts used in the N-process linear chain case can be extended to an M by N array model. The CP-net will involve MN colors for tokens in places and occurrences in transitions. The numbering scheme of processes in step 2 of the translation algorithm may proceed, for example, form left to right and from top to bottom for M rows and N columns, i.e. 1, 2, ..., MN. Since rendezvous can take place in four directions (left, right, up and down) the adjacency distances are 1 in the horizontal direction and N in the vertical direction. Twice the number of color posets with "less than" as a relation is needed to represent the behavior of the inner processes in the 2-dimensional model. The difference between immediate seccessors in these posets is equal to the relevant adjacency distance. The CP-subnets for horizontal rendezvous statements remain the same as for the chain model, whereas those for vertical rendezvous subnets and are defined as idu (ID) = ID - N and idd (ID) = ID + N for upward and downward communication, respectively. The translation procedure retains the same steps with all the requirements of the vertical direction added. In particular, FIFO queues for merging compatible rendezvous places in vertical rendezvous subnets are required. The reachability algorithm also retains the same steps. But the notion of p-covering must be extended to incorporate the inner posets and boundary colors that arise in the second dimension.

The compact folded CP-net model and the reduced-size reachability tree makes the proposed approach suitable for practical implementation. The implementation may follow a procedure similar to that as in [16] or [6] for the translation of an SCMD program into a CP-net. The implementation of the reachability tree generation and analysis may also be done in a way similar to that for other reachability tools [4]. The implementation of the additional features is also straightfordward. In particular, the colors associated with tokens and transitions, the labeling of arcs with functions and the identities of transitions and places can be encoded in a predetermined way. The code can be added to the data structures representing transitions and places. This adds a reasonable amount of complexity to the data structures in return for the advantages of the CP-net-based approach. Work is in progress for implementing this approach for Ada concurrent programs.

REFERENCES

[1] G. Balbo, G. Chiola, S.C. Bruell and P. Chen, "An example of validation and evaluation of a concurrent program: Lamport's Fast Mutual Exclusion Algorithm," Performance Evaluation Review, Vol. 17, No. 1 (May 1989).

[2] D.P. Bertsekas and J.N. Tsitsiklis. Parallel and Distributed Computations. Englewood Cliffs, NJ, Prentice-Hall (1989).

[3] Department of Defense. The Programming Language Ada: Reference Manual: Proposed Standard Document, U.S. DoD. Berlin, Springer-Verlag (1981).

[4] F. Feldbrugge and K. Jensen, "Petri Net tool overview 1986," in Proc. Advanced Course on Petri Nets, Springer-Verlag, Bad Honnef, W. Germany (September 1986).

[5] G.C. Fox et al., Solving Problems on Concurrent Processors. Englewood Cliffs, NJ, Prentice-Hall (1988).

[6] A.L. Goel, N. Mansour, S. Zhang and C. Kan, "A concurrent program testing tool," Technical Report, Center for Computer Applications and Software Engineering, Syracuse University (1990).

[7] C.A.R. Hoare, "Communicating Sequential Processes," Comm. of the ACM, 21 (8), pp. 666-677 (August 1978).

[8] P. Huber, A.M. Jensen, L.O. Jepsen and K. Jensen, "Reachability trees for high-level Petri nets," Theoretical Computer Science 45, pp. 261-292 (1986).

[9] M.A. Iqbal, J.H. Saltz and S.H. Bokhari, "A comparative analysis of static and dynamic load balancing strategies," Proc. Int. Conf. Parallel Processing, pp. 1040-1047 (1986).

[10] K. Jensen, "Colored Petri nets and the invariant-method," Theoretical Computer Science, 14, pp. 317-336 (1981).

[11] K. Jensen, "Colored Petri nets," in Proc. Advanced Course on Petri Nets, Springer-Verlag, Bad Honnef, W. Germany (September 1986).

[12] H.T. Kung, "The structure of parallel algorithms," in Advances in Computers, ed. M.C. Yovits, Academic Press, NY, 19, pp. 65-112 (1980).

[13] D.L. Long and L.A. Clarke, "Task interaction graphs for concurrency analysis," in Proc. Int. Conf. Software Engineering, pp. 44-52 (May 1989).

[14] C.E. McDowell, "A practical algorithm for static analysis of parallel programs," J. of Parallel and Distributed Computing 6, pp. 515-536 (1989).

[15] T. Murata, B. Shenker and S. Shatz, "Detection of Ada static deadlocks using Petri net invariants," IEEE Trans. Software Engineering, Vol. 15, No. 3, pp. 314-325 (March 1989).

[16] S.M. Shatz, K. Mai, D. Moorthi and J. Woodward, "A toolkit for automated support of Ada tasking analysis," in Proc. Int. Conf. Distributed Comptuing Systems, pp. 595-402 (1989).

[17] K.C. Tai and R.H. Carver, "Testing and debugging of concurrent software by deterministic execution," in Proc. 7th Pacific Northwest Software Quality Conf. (1989).

[18] R.N. Taylor, "A general-purpose algorithm for analyzing concurrent programs," Comm. of the ACM, 26(5), pp. 362-376 (May 1983).