Web Service Composition Verification of Safety Properties Based on Predicate Abstraction

Yuying Wang
Science School, Xi'an University of Architecture and Technology, Xi'an 710055, China
Email: yinyin632@sina.com

Ning Yang
The School of Automation, Northwestern Polytechnical University, Xi'an, Shaanxi, China
Email: ningyang@nwpu.edu.cn

***Abstract*—State space explosion is one of the biggest problem in model checking. Predicate abstraction technique is used to reduce the size of state space of colored Petri net models, and an algorithm was proposed to obtain the abstracted state space of a colored Petri net model without its original state space generated. A method to verify safety properties of Web service composition by abstracted state space and the Counterexample-Guided Abstraction Refinement was proposed. The problem of state space explosion is solved to some extend by this way. Finally, with an example, an application of this method is illustrated, which its efficiency shown.**

***Index Terms*— model checking, web service composition, safety property, predicate abstraction, colored Petri net**

I. Introduction

A safety property is in the sense of means that "nothing bad" can happen in a system, which assert that the system always stays within some allowed region.

Model checking[1-5] is an automatic formal verification technique which is widely used in a variety of fields since 1980's. Given a state transition system and a property, It explores the state space of the system exhaustively to determine whether the system satisfies the property. State space explosion remains a large of obstacle to use of model checking.

Abstraction[6] is a kind of efficient ways to overcome the state space explosion problem. During the verification process, irrelevant information is removed from the original system and a simplified model, called abstract model, is obtained. In general, the abstract model has less states than the original. So the verification is higher efficient while we do it under the abstract model. By this verification method, we can deal with lager-scale designs to a certain degree.

For web service composition, there are some research works employed varieties of methods, which includes Petri net, process algebra, abstract state machine, SPIN, and automaton methods, etc[6-8]. Some researches employed more than one method, so we can not classify them accurately.

Based on Colored Petri Net (CP-net for short) models of Web service composition, a new algorithm is presented to verify safety properties of the web service composition in this paper. Predicate abstraction technique is employed in it.

II. Counter-example-Guided Abstraction Refinement

Program analysis must be precise and scalable for verification. Precision is required so that the analysis does not be fooled by spurious errors and dose not overlook genuine errors. Scalability is necessary so that the method is suitable for large software systems which demands most accurate analysis. These two features are often mutually exclusive and need a trade-off between them. Flow-based analyses[10,11] achieve scalability. It fixes a small domain of dataflow facts which are tracked, and computes flow functions over the abstract semantics of the program on this fixed set. For complicated properties, the set of facts that are tracked is too small and will lead to a high rate of false positives, i.e., a large number of bugs reported which never arise during the program execution. Some model checking methods, precise or path-sensitive, often end up tracking too many facts and lead to state explosion in the way of scalability.

To avoid the pitfalls arising from using a fixed set of facts, much recent interest has focused on analyses that automatically tune the precision of the analysis using false positives. It is called Counterexample-Guided Abstraction Refinement(CEGAR for short). It is a loop[6,12-15] with 3 steps as follows.

Step 1 ("Abstraction") A finite set of predicates is chosen, and an abstract model is built automatically as a finite or push-down automaton for the given program. States of the automaton represent truth assignments of the chosen predicates.

Step 2 ("Verification") For the desired property, The abstract model is checked automatically. If error-free the abstract model is, then so is the original program (return "program correct"); otherwise, an abstract counter-example is produced automatically which demonstrates how the model violates the property.

Step 3 ("Counter-example-driven refinement") Whether the abstract counter-example corresponds to a concrete counter-example in the original program, it is checked automatically. If so, then a program error has been found (return "program incorrect"); otherwise, the chosen set of predicates does not contain enough information to prove program correctness and new predicates required to be added into the set. The selection of such predicates is automated, or at least guided, by the failure to concretize the abstract counter-example.

Goto Step 1.

III. Predicate Abstraction on CP-net models

Predicate abstraction was presented by S.Graf and H.Saidi firstly [17]. It is a kind of abstraction with keeping properties. It defines an equivalence relation on the initial model by predicates, and changes the concrete model which is large or contains infinite states, into an abstract model which has finite states and is easy to manipulate. Predicate abstraction can be seem as a special conservative abstraction.

A. Predicate Abstraction

In what follows, logic predicates shall be used to represent sets. If a predicate P represents a set, then an element, x, is a member of the set if and only if $P(x)$ is true.

Elements of a concrete system (a system without abstraction) are described as follows.

1. The set of concrete sates, denoted by C;

2. The concrete transition relations which are described by an initial state predicate $I_C : C \to bool$, and a concrete transition relation predicate $R_C : C \times C \to bool$.

3. A state x is an initial state, iff $I_C(x)$ is true; state y is a concrete successor of x iff $R_C(x, y)$ is true.

Let $\varphi_1, \varphi_2, \cdots, \varphi_N$ be the abstraction predicates defined on the concrete system M, M is partitioned by these predicates equivalently, and each concrete state is mapped to an equivalent class which called abstract state. An abstract state can be represented by a vector of length N. The abstract set of states, A ,can be defined as :

$A = \{(b_1, b_2, \cdots, b_N) \mid b_i$ is a bool expression$\}$

The correspondence exists between a concrete state and an abstract state. A concrete state corresponds to only one abstract state, and an abstract state to a set of concrete states. This relationship can be described by abstraction and concretization functions. The abstraction function α maps each concrete state to an abstract state, and the concretization is the inverse of an abstraction. The concretization function γ maps each abstract state to the set of all concrete states it represents.

Definition 1. (Abstraction and concretization functions)[16]. Let $\varphi_1, \varphi_2, \cdots, \varphi_N$ be the abstraction predicates defined on the concrete system M . $B_1, B_2, \cdots, B_N$ are boolean variables where each B_i represents all concrete states satisfying the predicated φ_i . Abstract states are represented by a boolean expression $\exp^A(B_1, B_2, \cdots, B_N)$. That means that the set of concrete states represented by the abstract state can easily be computed by substituting each occurrence of each variable B_i by the concrete predicate φ_i which it represents:

$\gamma(\exp^A(B_1, B_2, \cdots, B_N)) = \exp^A[\overline{\varphi} / \overline{B}]$

whereas the implicitly defined abstraction function as bellow.

$\alpha(\varphi) = \wedge\{\exp^A(B_1, B_2, \cdots, B_N) \mid \varphi \Rightarrow \exp^A[\overline{\varphi} / \overline{B}]\}$

where φ represents a vector which consists of $\varphi_1, \varphi_2, \cdots, \varphi_N$, and B represents a vector which consists of $B_1, B_2, \cdots, B_N$, $[\overline{\varphi} / \overline{B}]$ represents substitution of φ_i for B_i, for each i.

For each concrete state φ, $\alpha(\varphi)$ is a conjunctive normal form of all formulas $\exp^A(B_1, B_2, \cdots, B_N)$ which satisfying $\varphi \Rightarrow \exp^A[\overline{\varphi} / \overline{B}]$. In general it is not easy to compute this conjunction, so an upper approximation of the function α, α' is used by Graf and Saidi[17] to substitute for it . α' is less expensive to compute and results in a monomial on $B_1, B_2, \cdots, B_N$.

$\alpha'(\varphi) = \wedge\{B_i \mid \varphi \Rightarrow \varphi_i,\ 1 \le i \le n\}$

(Notice: a monomial on $B_1, B_2, \cdots, B_N$ means a conjunction of B_i and $\neg B_i$, in which B_i occurs at most one time, and the predicate false is seemed as a monomial.)

A method of abstract state computation from a concrete state was given out by S.Graf and H.. An abstract state is a certain assignment of N boolean variables, and is presented by atomic formula false or a conjuctive normal form $c_1 \wedge c_2 \wedge \cdots \wedge c_N$ where $c_i (1 \le i \le N)$ takes on value B_i, $\neg B_i$ or true.

If the current concrete state satisfies the predicate φ_i, c_i takes on vlaue B_i. Otherwise it satisfies predicate $\neg\varphi_i$, c_i takes on vlaue $\neg B_i$. If neither φ_i nor $\neg\varphi_i$ it satisfies, c_i takes value true.

As we described previously, A is the set of states of the abstract system. Abstract initial states and abstract transition relations are complete to define an abstract system.

Definition 2. (Initial state). The abstract initial states $I_A : A \to bool$ is defined to be $\alpha(I_C)$.

It may be shown that the concrete and abstract initial states satisfy the inclusion relation $I_C \Rightarrow \gamma(I_A)$.

Definition 3. (Abstract transition relation). The abstract transition relation is represented by a predicate $R_A : A \times A \to bool$ with two states, $s, t \in A$, as argments.

The transition relation is defined as

$R_A(s,t) = \exists x, y \in C.\ \gamma(s)(x) \wedge \gamma(t)(y) \wedge R_C(x, y)$.

Definition 4.(Predicate transformer) Let R be a binary relation on a set Q, and $\varphi \in P(Q)$ represent a subset of Q ,then

$\widetilde{pre}[R](\varphi) = \forall q'.\ (R(q, q') \Rightarrow \varphi(q'))$

$post[R](\varphi) = \exists q'.\ R(q', q) \wedge \varphi(q')$

$\widetilde{pre}[R](\varphi)$ defines the largest set of states such that all its successors satisfy φ (the weakest precondition); $post[R](\varphi)$ defines the set of successors of φ by R (the strongest postcondition). Let R_A denote the set of transitions in an abstract system, and R_C denote the set of transitions in the corresponding concrete system, then R_A and R_C have such relevance: For each concrete transition τ_j, we use τ_j^A to denote a transition in the abstract system which comes from $\exp^A(\mathrm{B}_1,\mathrm{B}_2,\cdots,\mathrm{B}_\mathrm{N})$ and corresponds to τ_j, that is, τ_j^A is a transition from a set of abstract states $\exp^A(\mathrm{B}_1,\mathrm{B}_2,\cdots,\mathrm{B}_\mathrm{N})$ to another set of abstract states which represents all successors of the concrete states represented by $\exp^A(\mathrm{B}_1,\mathrm{B}_2,\cdots,\mathrm{B}_\mathrm{N})$. τ_j^A can be determined by

$$\tau_j^A(\exp^A(B_1,B_2,\cdots,B_N)) = \begin{cases} \mathit{false} & \text{if } \exp^A[\overline{\varphi}/\overline{B}] \Rightarrow \neg g_j \\ \bigwedge\limits_{i\in[1,N]} \left\{ \begin{array}{ll} B_i, & \text{if } \mathrm{post}[\tau_j](\exp^A[\overline{\varphi}/\overline{B}]) \Rightarrow \varphi_i \\ \neg B_i, & \text{if } \mathrm{post}[\tau_j](\exp^A[\overline{\varphi}/\overline{B}]) \Rightarrow \neg\varphi_i \\ \mathit{true} & \end{array} \right\} & \text{otherwise.} \end{cases}$$

In this formula, g_j is a boolean expression and is the condition of τ_j occurrence. A transition occurs only when the system current state meets g_j . while $\exp^A[\overline{\varphi}/\overline{B}] \Rightarrow \neg g_j$ is satisfied $\tau_j^A(\exp^A(B_1,B_2,\cdots,B_N)) = \mathit{false}$. It means the abstract state $\exp^A(B_1,B_2,\cdots,B_N)$ has no successor. $\exp^A[\overline{\varphi}/\overline{B}] \Rightarrow g_j$ means the transition condition satisfied, $\mathrm{post}[\tau_j](\exp^A[\overline{\varphi}/\overline{B}])$ determines the value of c_i takes.

In conclusion, given a concrete model with its set of initial states I_C and set of transitions R_C , an abstract model can be computed with the abstract initial states set I_A and transitions set R_A, using abstract function α' and abstract transition τ_j^A.

Theory 1. For any concrete state x, $\gamma(\alpha(\mathrm{x}))(\mathrm{x})$ holds.

Theory 2. For any abstract states x,y ,and an abstract state s, $\gamma(s)(x) \wedge R_C(x,y) \Rightarrow \exists t.\ R_A(s,t) \wedge \gamma(t)(y)$ holds.

Above theories have been proved in reference [16].

The abstract model given in this section is conservative. That is, if a property holds in an abstract system, it also holds in the corresponding concrete system.

Theory 3.(Conservation of abstraction) if a CTL^* universal property holds in an abstract model, it also holds in the corresponding concrete model.

This theory was presented by CLARKE, et.al. [18]. A universal property means no path quantifier "$\exists$" existing in the negative normal form it translated.

B. CP-net Models

CP-net extends the Petri net in definition of data types and manipulation of data values. For a same system, its CP-net model should be simpler and more compact than its Petri net model in general. CP-net has been used to describe some properties, for example, security property[19].

All CP-net models appear in this paper are generated by CPN tools[20] developed by Danish Aarhus university. The color set "STATE" has bool type. Symbol "t" represents a bool constant "true", which denotes resources movement by representing a state of an activity. "MSG" is a color set of string type, "msg" is a variable of MSG.

C. Predicate Abstraction on CP-net Models

In essence, predicate abstraction is a method of partitions of equivalent classes. All applications of predicate abstract technique authors found in references, work on state diagram of systems. The biggest problem for a state diagram is states explosion.

To avoid analysis of big scale sate diagram, a new algorithm of the predicate abstraction is given in this paper. In this algorithm, the generation of a state diagram is combined with the abstraction of the same states diagram, the abstract states diagram was built during the state diagram generation. This new algorithm is based on CP-net algorithms of states diagram generation [21] and abstract states diagram generation[15] from concrete states diagram. Compared with algorithms available, this new one need not deal with concrete states diagram, so states explosion of the concrete states diagram is avoided.

Formulas of relations between transitions in a concrete system model and that in the abstract system model are employed in this algorithm. A system abstract states diagram will be obtained directly without the generation of the concrete states diagram, if we move some steps out this algorithm. So the explosion problem of the concrete state space is solved to a certain degree.

Algorithm 1. (Generations of CP-net states diagram and abstract states diagram)

```
symbols:
cpn:the CP-net model of a given system S
$M_0$:a initial mark of cpn;
$\varphi = \{\varphi_1,\varphi_2,\cdots,\varphi_n\}$ : set of predicates;
Input:cpn
OutPut:an abstract states diagram of cpn based on the set of predicates $\varphi$ , which composes of a set of nodes, $Nodes$ ,and a set of arcs, $Arcs$ ;
Steps:
 1. initialize the state diagram:
 $Nodes = \{M_0\}$; $Arcs = \phi$;
 2. initialize the data: $Unprocessed = \{M_0\}$ ;
```

```
3. compute the abstract function $\alpha'$ by the set of predicates $\varphi : \alpha'(\varphi) = \wedge\{B_i \mid \varphi \Rightarrow \varphi_i,\ 1 \le i \le n\}$
4. initialize the abstract state diagram:
   compute the initial state of the abstract state digram:
      $M_0^A = \alpha'(M_0)$ , $A_{tou} = \phi$ , $Nodes^A = \{M_0^A\}$ ;
5. while( $Unprocessed \ne \phi$ )
   { select a state $M_1 \in Unprocessed$
     $Unprocessed = Unprocessed \setminus \{M_1\}$
     caculate the abstract state with which $M_1$ correspond, $M_1^A = \alpha'(M_1)$ ;
     if ( $M_1^A \notin Nodes^A$ ) { $Nodes^A = Nodes^A \cup \{M_1^A\}$ ;}
     //treat bindings of $M_1$
     for ( each binding $(b, M_2)$ which satisfies
                                   $M_1[b > M_2$ )
     { if ( $M_2 \notin Nodes$ )
       { $Nodes = Nodes \cup \{M_2\}$ .
         $Unprocessed = Unprocessed \cup \{M_2\}$
         caculate the abstract state with which
                  $M_1$ correspond, $M_2^A = \alpha'(M_2)$ ;
         if ( $M_2^A \notin Nodes^A$ )
         { $Nodes^A = Nodes^A \cup \{M_2^A\}$ ;}}
         $Arcs = Arcs \cup (M_1, b, M_2)$
          seem $(M_1, b, M_2)$ as a transition $\tau_j$ of the concrete system, calculate the abstract transition with which $\tau_j$ correspond:
            $\tau_j^A(M_1^A) = \begin{cases} B_i\ , & \text{if}\,(M_2^A) \Rightarrow \varphi_i \\ \neg B_i, & \text{if}\,(M_2^A) \Rightarrow \neg\varphi_i \end{cases}$ .
            $A_{tou} = A_{tou} \cup \{\tau_j^A(M_1^A)\}$        } }
6. put out the concrete state diagram, that is, put out the set $Nodes$ and the set $Arcs$ ;
7. put out the abstract state diagram, that is, put out its initial state $M_0^A$, set of transitions $A_{tou}$ , and its reachable states set $Nodes^A$ .
```

IV. CP-NET MODEL OF WEB SERVICE COMPOSITION

Web service composition is an error prone task in which service candidates interact complexly. The Business Process Execution Language for Web Services (BPEL4WS or BPEL for short) was proposed by BEA, IBM and Microsoft. It often is used to describe Web service compositions. BPEL represents a convergence of two languages: the Web Services Flow Language (WSFL) of IBM and XLANG of Microsoft. Like most languages, (the semantics of) BPEL is defined in English prose. Such descriptions, although often masterpieces of apparent clarity, usually suffer from inconsistency, ambiguity and incompleteness.

Due to the presence of concurrency and intricate features like compensation handling, correlation and death-path-elimination, BPEL processes are also error prone.

Based on CP-net, processes of Web service composition described by BPEL are translated into timed CP-net models in this paper, which have given in our other papers[22, 23]. Atomic activities of BPEL are seem as atomic operation in transitions, their execution successfully or not is the only factor under considered, which means the factor results in errors in not under considered, because we focus verification on web service compositions.

V. WEB SERVICE COMPOSITION VERIFICATION OF SAFETY PROPERTIES BY PREDICATE ABSTRACTION

In this sector, the predicate abstraction technique and CEGAR are employed to verify safety properties of web service compositions. The algorithm 1 is used to generate the states diagram from the system's CP-net model, the abstract state diagram or the concrete state diagram.

A. System Description

The system which was discussed in the BPEL specification[24] is used to demonstrate our work in this section. It is a Shopping Service, which presents a BPEL Abstract Process for a rudimentary shipping service.

This service handles the shipment of orders, and orders are composed of a number of items. The shipping service offers two options, one for partial shipments where the items are shipped in groups until the order is fulfilled, and another for shipments where the items are shipped all together.

B. System Model

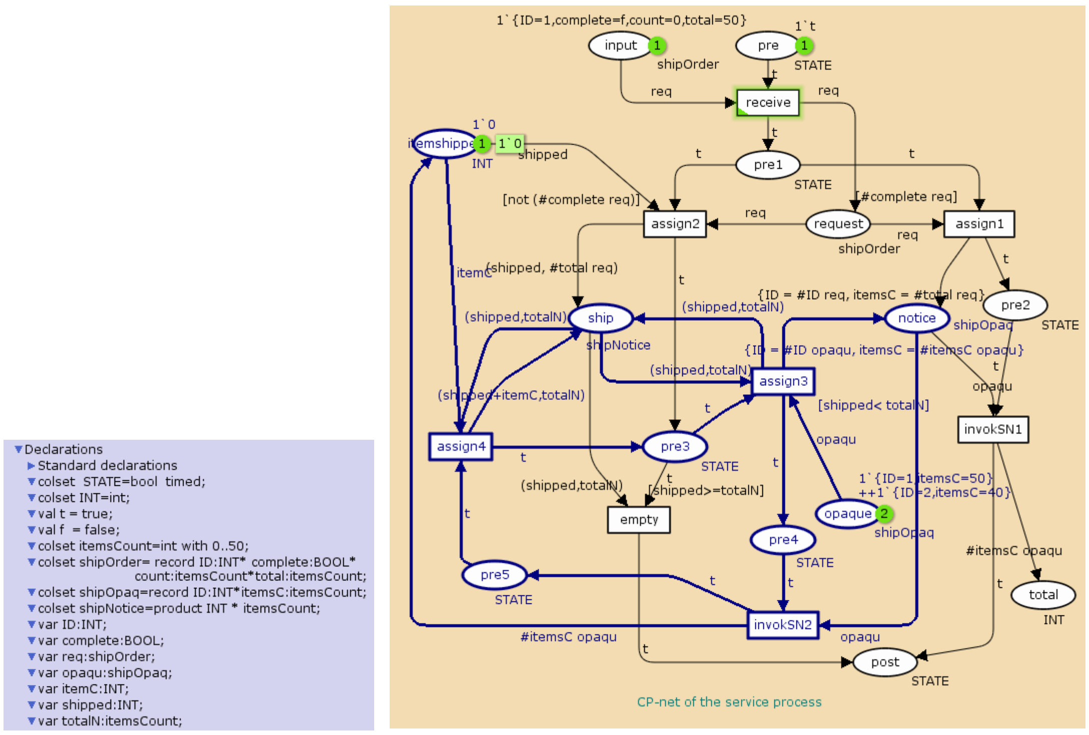
The CP-net model derived from this Abstract Process's instantiation is shown in fig.1. It presents the Shopping Service, a web service composition.

In declarations of this model, the color set "shipOrder" is of record type and presents order type, its component "complete" is boolean and presents service options, whole shipment or partial shipments. The color set "shipOpaqp" presents a order during partial shipment.

In this CP-net model. The place "input" is of "shipOrder" type and presents order's information transmitted into the system. With initial token 0, the place "itemshipped" is of integer type and presents the mount of goods shipped during partial shipments. The place "opaque" is of "shipOpaq" type and presents all orders of partial shipments, while "notice" has same type and presents a order in each shipment. The place "ship" is of "shipNotice" type, its one component presents the current amount of goods shipped in a partial shipment and another presents the total amount of goods need to ship.

In the CP-net model, the transition "receive" presents the acceptance of orders from the outside system, "invokSN1" presents the service invocation of whole shipment and "invokSN2" the service of partial shipments.

Block line parts in fig.1 mean main activities of services of partial shipments.



```
▼Declarations
  ▶Standard declarations
  ▼colset  STATE=bool  timed;
  ▼colset INT=int;
  ▼val t = true;
  ▼val f  = false;
  ▼colset itemsCount=int with 0..50;
  ▼colset shipOrder= record ID:INT* complete:BOOL*
                 count:itemsCount*total:itemsCount;
  ▼colset shipOpaq=record ID:INT*itemsC:itemsCount;
  ▼colset shipNotice=product INT * itemsCount;
  ▼var ID:INT;
  ▼var complete:BOOL;
  ▼var req:shipOrder;
  ▼var opaqu:shipOpaq;
  ▼var itemC:INT;
  ▼var shipped:INT;
  ▼var totalN:itemsCount;
```

Figure 1. CP-net Model of the Service Process

C. System States

Resource movements in the system are presented by changes of places in CP-net models. At any time, the mark which consists of states of all tokens presents the system state. In order to describe the system state, states of tokens of places need to depict by predicates. The number of token of a place may be empty or not. If it is not empty, maybe a further description needed.

With containing information of tokens in CP-nets, predicates do not only have values "true" or "false" simply, because sometimes computing its value is impossible. In this case, we use symbol "-" to denote this situation. In other words, a predicate will have one of 3 values: "true", "false" or "-".

D. System Properties Verification

In this section, we demonstrate the method to verify a safe property of the system by predicate abstractions.

We assume the system holds a property which both whole shipment and partial shipment will be invoked in same time. That is, in the CP-net model transitions "invokSN1" and "invokSN2" will never be triggered at same time.

Applying the algorithm 1 on the CP-net model, we obtain the abstract state diagram shown in Fig.2(a), where M_0^A is the initial abstract state, the binary number in a circle presents a system state under the given predicates set $\{\varphi_1, \varphi_2\}$,what in a square presents a transition which causes system states change.

There is a path in the system of Shopping Service, $M_0^A \to M_3^A \to M_0^A \to M_1^A \to M_2^A \to M_0^A$, as shown in Fig.2(a). In its sub-path of $M_3^A \to M_0^A$, the transition "invokSN1" is triggered, while in another sub-path of $M_2^A \to M_0^A$, the transition "invokSN2" triggered also. In other words, both services of the whole and the partial shipments are invoked during the system execution. We guess this conclusion is due to the rough of models and in predicates no enough information to prove the correction of the system. So new predicates need to add.

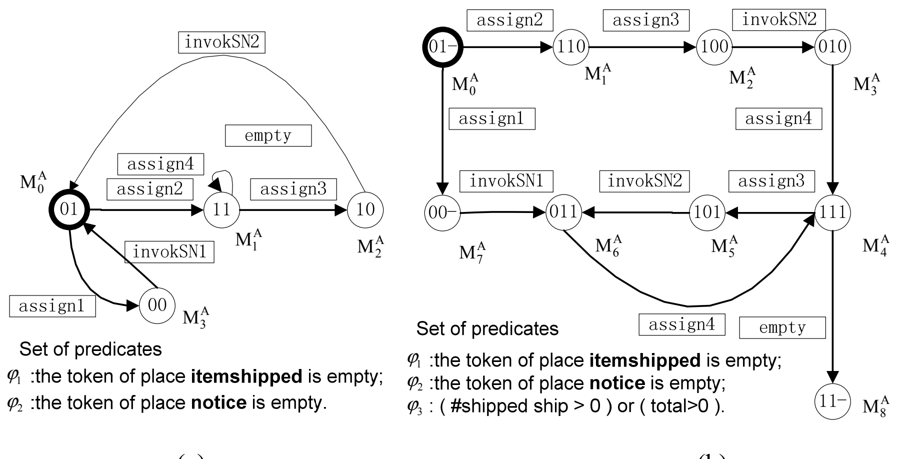
E. Abstraction and Concretization

In this section, $\varphi_3 : (\#shipped\ ship\ >\ 0)\ or\ (total > 0)$, a new predicate added to the set of predicates. The current set of predicates is $\{\varphi_1, \varphi_2, \varphi_3\}$, in which φ_1 and φ_2 are the same as before. The predicate φ_3 means at lest one of "invokSN2" and "invokSN1" be triggered.

For this new predicates set, we obtain the abstract state diagram shown in Fig.2(b).

Fig.2(b) shows none of path through both of M_2^A (or M_5^A) and M_7^A , which means transitions "invokSN2" and "invokSN1" can be both triggered during once execution of the system. In other words, it is impossible to invoke the whole shipment service and the partial shipment services successively.

From above analysis we know the counter-example: $M_0^A \to M_3^A \to M_0^A \to M_1^A \to M_2^A \to M_0^A$ is a pseudo counter-example.



(a) (b)

Figure 2. (a) abstract state space of Shipping service;
(b) abstract state space of Shipping Service after new predicate added.

VI. Conclusion

Web service composition is an error prone task. One of the biggest problems is state space exploration during model checking. The predicate abstraction technique is one of efficient ways to solve the problem of states space explosion to a certain extent.

To avoid the analysis of a big scale sate diagram, a new algorithm of the predicate abstraction on CP-net models is given in this paper. In the algorithm, the generation of a state diagram is combined with the abstraction of the same state diagram of CP-net models, the abstract state diagram was built during the generation.

An example also is given to illustrate the efficient of the new algorithm.

Acknowledgements

This work was financially supported by the Department of Education of Shaanxi Foundation (2013JK1190).

References

[1] E. M. Clarke, O. Grumberg, and D A. Peled. Model checking. MIT Press, Cambridge, MA, USA, 1999.

[2] N. M. Lin, W. H. Zhang. Model Checking: Theory, Mothod and Application．Chinese Journal of Electrinics, 2002,12(3):1906-912.

[3] M. Huth, M. Ryan. Logic in Computer Science. Cambridge University Press: 2005

[4] H. Shi, W. Ma, M. Yang, etc.. A Case Study of Model Checking Retail Banking System with SPIN. Journal of Computers. 2012, Vol 7(10):2503-2510

[5] C. Zhou, B. Sun. Abstraction In Model Checking Real-Time Temporal Logic of Knowledge. Journal of Computers. 2012,Vol 7(2): 362-370

[6] R. Jhala. Program Verification by Lazy Abstraction. Ph.D. Thesis. Computer Science, University of Calfornia at Berkeley. Fall, 2004

[7] M. Khaxar, S. Jalili and N. Khakpour. Monitoring Safety Properties of Composite Web Services at Runtime Using CSP. proceeding of: Workshops Proceedings of the 12th IEEE International Enterprise Distributed Object Computing Conference, EDOCw 2009, September 2009, Auckland, New Zealand

[8] F. van Breugel, M. Koshkina. Models and Verification of BPEL. http://www.cse.yorku.ca/~franck/research/drafts/tutorial.pdf. September 2006.

[9] G. Q. Zhang, H. J. Shi. Model Checking for Asynchronous Web Service Composition Based on XYZ/ADL. Web Information Systems and Mining, Lecture Notes in Computer Science, 2011, 6988:428-435

[10] J. S. Foster, T. Terauchi, and Aiken A. Flow-sensitive type qualifiers. In PLDI02: Programming Language Design and Implementation, ACM,2002:1-12.

[11] M. Das, S. Lerner, and M. Seigle. ESP: Path-sensitive program verification in polynomial time. In PLDI 02: Programming Language Design and Implementation, ACM, 2002:57-68.

[12] R. Alur, A. Itai, B. P. Kurshan, et al. Timing verification by successive approximation. Information and Computation, 1995, 118(1):142-157.

[13] T. Ball, S. K. Rajamani. Automatically validating temporal safety properties of interfaces.In SPIN 2001: SPIN Workshop, LNCS 2057, Springer-Verlag, 2001:103-122.

[14] H. Saidi. Model checking guided abstraction and analysis. In SAS 00: Static-Analysis Symposium,. LNCS 1824, Springer-Verlag, 2000:377-396.

[15] E. M. Clarke, O. Grumberg, S. Jha, et al. Counterexample-guided abstraction refinement. In CAV 00: Computer Aided Verification, LNCS 1855, Springer-Verlag, 2000:154-169.

[16] S. Das. Predicate Abstraction. Ph.D. Thesis.Department of Electrical Engineering, Stanford University, December 2003.

[17] S. Graf, H. Saidi. Construction of abstract state graphs with PVS. Proceedings of the 9th Conference on Computer-Aided Verification (CAV'97), Haifa, Israel, June 199:72−83.

[18] E. M. Clarke, O. Grumberg, D. E. Long. Model checking and abstraction. ACM Trans. Program, 1994, 16(5):1512－1542.

[19] X. Yang, X Xie. Modeling and Analysis of Security Protocols Using Colored Petri Nets. Journal of Computers. 2011, Vol 6(1):19-27

[20] http://wiki.daimi.au.dk/cpntools/cpntools.wiki. 2009

[21] L. M. Kristensen. State Space Methods for Coloured Petri Nets. Department of Computer Science,University of Aarhus, Denmark, 2000.
[22] Y. Wang, P. Chen. Models of BPEL's flow activity based on color Petri nets. Application Research of Computers,2011,28(2):631-634.
[23] Y. Wang, P. Chen. Models of Web Services Composition Based on Timed Color Petri Nets. Computer Science, 2010,37(10):151-155.
[24] OASIS. Web Services Business Process Execution Language Version 2.0. https://www.oasis-open.org/committees/wsbpel/

Yuying Wang was born in Chifeng, Inner Mongolia, China in 1964. She awarded a BS degree in mathematics from Beijing Normal University in 1987, China. She received the MS and Ph.D degrees in software engineering from Xidian University, Xi'an, China in 2003 and 2012. She is an associate professor of math and software engineering, Xian University of Architecture and Technology, China. Her research interests include model checking, data mine and optimization.