Combining Static Analysis and Case-Based Search Space Partitioning for Reducing Peek Memory in Model Checking

Wenhui Zhang
Laboratory of Computer Science
Institute of Software, Chinese Academy of Sciences
P.O.Box 8718, Beijing 100080, China
zwh@ios.ac.cn

December 16, 2002

Abstract

Memory is one of the critical resources in model checking. This paper discusses a strategy for reducing peek memory in model checking by case-based partitioning of the search space. This strategy combines model checking for verification of different cases and static analysis or expert judgment for guaranteeing the completeness of the cases. Description of the static analysis is based on using PROMELA as the modeling language. The strategy is applicable to a subset of models including models for verification of certain aspects of protocols.

1 Introduction

A main concern of model checking is the state explosion problem. Much effort has been put into the research for reducing this problem. Related research topics can for instance be found in [4, 9, 25, 8, 5, 15, 17]. The topics include abstraction techniques for reducing models, compositional techniques for splitting verification tasks and techniques for compactly representing transition relations and system states.

In addition to general techniques which can be used to a wide range of models, it is also important to develop techniques for special types of models, in order to enhance the applicability of model checking to these types of models. The motivation of this paper is verification of models with non-deterministic choice and open-environment. The basic idea is to find a method for adequately representing different cases in such models, in order to reduce peek memory usage in model checking. We propose a strategy to partition a model checking task into several cases (the verification of which is done by model checking) and prove that the collection of these cases covers all possible cases. The latter can either be done by using expert judgment or by static analysis for models satisfying certain criteria. The static analysis can be implemented as an automatic tool which may be offered as an option to the user of the model checker. The contents of this paper are as follows:

- The principle of *case-based partitioning*.
- An example that demonstrates the advantage of *case-based partitioning*, where expert knowledge is used for ensuring the soundness of the strategy.
- Presentation of a type of models that is feasible for *case-based partitioning*.
- Conditions for the soundness of using *case-based partitioning*.

- A description of the static analysis for checking the conditions.
- A discussion on algorithm for searching variables to be used for partitioning and therefore also for automatically apply this strategy for models of the discussed type.

As mentioned at the beginning of this section, there are many techniques for alleviating the state-explosion problem. Some techniques are implemented as a standard part in model checking tools, for instance, on-the-fly model checking [12, 1] and partial order reduction [14, 20, 21] are implemented in the model checking tool SPIN [12, 13]. Other techniques may be used as pre-processing methods for reducing the complexity of models, for instance, we may use cone of influence reduction [2] or program slicing [19] for reducing the complexity of models. On the other hand, the representation of properties is also important, since a verification task consists of two parts: the model and the property. For instance, for the application of SPIN, one usual representation of the property is propositional linear temporal logic (PLTL) formulas [7] which are to be transformed into Büchi automata. Therefore, succinct representation of PLTL formulas in Büchi automata is a means for improving efficiency of model checking [10, 3, 23]. In addition, one may consider decomposition of properties, i.e. a given property may be decomposed into a set of sub-properties and verified (with abstraction of the model pertinent to each of the properties) separately [24]. The proposed strategy can be regarded as such a pre-processing method and can be used in combination with the aforementioned techniques. Since different techniques may have different advantages and limitations, for a given verification problem, we may need to consider many techniques and choose a suitable combination of such techniques in order to successfully solve the problem.

2 Partition Strategy

Let T be a system and $\vec{x}$ be the global variable array of T. The system is in the state $\vec{v}$, if the value of $\vec{x}$ at the current moment is $\vec{v}$. A trace of T is a sequence of states. The property of such a trace can be specified by PLTL formulas [7].

- φ is a PLTL formula, if φ is of the form $z = w$ where $z \in \vec{x}$ and w is a value.
- Logical connectives of PLTL include:

  $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction) and $\rightarrow$ (implication).

  If φ and ψ are PLTL formulas, then so are $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, and $\varphi \rightarrow \psi$.
- Temporal operators include:

  X (next-time), U (until), $\Diamond$ (future) and $\Box$ (always).

  If φ and ψ are PLTL formulas, then so are $X\,\varphi$, $\varphi\,U\,\psi$, $\Diamond\,\varphi$, and $\Box\,\varphi$.

Let t be a trace of T. Let HEAD(t) be the first element of t and TAIL$^i(t)$ be the trace constructed from t by removing the first i elements of t. For convenience, we write TAIL(t) for TAIL$^1(t)$. Let $t \models \varphi$ denote the relation "t satisfies φ".

Definition 2.1 *$t \models \varphi$ is defined as follows:*

$t \models x = v$	iff	the statement $x = v$ is *true* in HEAD(t).
$t \models \neg\varphi$	iff	$t \not\models \varphi$.
$t \models \varphi \wedge \psi$	iff	$t \models \varphi$ and $t \models \psi$.
$t \models \varphi \vee \psi$	iff	$t \models \varphi$ or $t \models \psi$.
$t \models \varphi \rightarrow \psi$	iff	$t \models \varphi$ implies $t \models \psi$.
$t \models X\,\varphi$	iff	TAIL$(t) \models \varphi$.
$t \models \varphi\,U\,\psi$	iff	$\exists k$ such that TAIL$^k(t) \models \psi$ and TAIL$^i(t) \models \varphi$ for $0 \leq i < k$.
$t \models \Diamond\varphi$	iff	$\exists k$ such that TAIL$^k(t) \models \varphi$.
$t \models \Box\varphi$	iff	$t \models \varphi$ and TAIL$(t) \models \Box\varphi$

Let $\mathcal{T}$ be a set of traces.

Definition 2.2 *$\mathcal{T} \models \varphi$ if and only if $\forall t \in \mathcal{T}.(t \models \varphi)$.*

Definition 2.3 *$\mathcal{T} \not\models \varphi$ if and only if $\forall t \in \mathcal{T}.(t \not\models \varphi)$.*

Let TOP$(\mathcal{T})$ be the set consisting of HEAD(t) for all $t \in \mathcal{T}$ and SUB$(\mathcal{T})$ be the set consisting of TAIL(t) for all $t \in \mathcal{T}$. From the above definitions, we derive the following:

$\mathcal{T} \models x = v$	iff	the statement $x = v$ is *true* in s for all $s \in$ TOP$(\mathcal{T})$.
$\mathcal{T} \models \neg\varphi$	iff	$\mathcal{T} \not\models \varphi$.
$\mathcal{T} \models \varphi \wedge \psi$	iff	$\mathcal{T} \models \varphi$ and $\mathcal{T} \models \psi$.
$\mathcal{T} \models \varphi \vee \psi$	iff	there are $\mathcal{T}'$ and $\mathcal{T}''$: $\mathcal{T} = \mathcal{T}' \cup \mathcal{T}''$ and $\mathcal{T}' \models \varphi$ and $\mathcal{T}'' \models \psi$.
$\mathcal{T} \models \varphi \rightarrow \psi$	iff	there are $\mathcal{T}'$ and $\mathcal{T}''$: $\mathcal{T} = \mathcal{T}' \cup \mathcal{T}''$ and $\mathcal{T}' \not\models \varphi$ and $\mathcal{T}'' \models \psi$.
$\mathcal{T} \models X\,\varphi$	iff	SUB$(\mathcal{T}) \models \varphi$.
$\mathcal{T} \models \varphi\,U\,\psi$	iff	there are $\mathcal{T}'$ and $\mathcal{T}''$: $\mathcal{T} = \mathcal{T}' \cup \mathcal{T}''$, $\mathcal{T}' \models \psi$, $\mathcal{T}'' \models \varphi$ and SUB$(\mathcal{T}'') \models \varphi\,U\,\psi$.
$\mathcal{T} \models \Diamond\varphi$	iff	there are $\mathcal{T}'$ and $\mathcal{T}''$: $\mathcal{T} = \mathcal{T}' \cup \mathcal{T}''$, $\mathcal{T}' \models \varphi$ and SUB$(\mathcal{T}'') \models \Diamond\varphi$.
$\mathcal{T} \models \Box\varphi$	iff	$\mathcal{T} \models \varphi$ and SUB$(\mathcal{T}) \models \Box\varphi$.

Let $\mathcal{T}$ be the set of the traces of T and φ be a propositional linear temporal logic formula. Let $T \models \varphi$ denote the relation "T satisfies φ".

Definition 2.4 *$T \models \varphi$ if and only if $\mathcal{T} \models \varphi$.*

Suppose that there is a model M and a formula φ, and our task is to check whether φ holds in M, i.e. we would like to prove:

$$M \models \varphi.$$

The principle of *case-based partitioning* is to partition the search space of M, so the formula φ can be proved within each portion of the search space. In order to do so, we have to characterize different portion of the search space. The technique for this characterization is to attach formulas to φ, so that in the verification of $M \models \varphi$, only paths relevant to the attached formulas are fully explored (or paths irrelevant to the attached formulas are discarded at an early phase of model checking).

Theorem 2.1 *Let $\psi_1, ..., \psi_n$ be formulas such that $M \models \psi_1 \vee \cdots \vee \psi_n$. $M \models \varphi$ if and only if $M \models \psi_i \rightarrow \varphi$ for all $i \in \{1, ..., n\}$.*

Proof: It is obvious that $M \models \varphi$ implies $M \models \psi_i \rightarrow \varphi$. We prove that $M \models \psi_i \rightarrow \varphi$ for all $i \in \{1, ..., n\}$ implies $M \models \varphi$ as follows.

- Let $\mathcal{M}$ be the set of traces of M. Since $\mathcal{M} \models \psi_1 \vee \cdots \vee \psi_n$, there are $\mathcal{M}_1, ..., \mathcal{M}_n$ such that $\mathcal{M} = \mathcal{M}_1 \cup \cdots \cup \mathcal{M}_n$ and $\mathcal{M}_i \models \psi_i$ for all i.
- On the other hand, we have $\mathcal{M} \models \psi_i \rightarrow \varphi$, hence $\mathcal{M}_i \models \psi_i \rightarrow \varphi$ and $\mathcal{M}_i \models \varphi$ for all i. Therefore $\mathcal{M} \models \varphi$. $\square$

For a given model M, in order to be successful with this strategy, we have to be sure that the proof of $M \models \psi_i \rightarrow \varphi$ is simpler than the proof of $M \models \varphi$ for each i. Therefore $\mathcal{M}$ (the set of traces representing the behavior of M) should have the following properties: $\mathcal{M}$ can be partitioned into $\mathcal{M}'_i$ and $\mathcal{M}''_i$ such that:

- $\mathcal{M}'_i \not\models \psi_i$ and $\mathcal{M}''_i \models \varphi$;
- $\mathcal{M}'_i \not\models \psi_i$ can be checked with high efficiency;
- $\mathcal{M}''_i$ is significantly smaller than $\mathcal{M}$.

For the selection of ψ_i (which determines $\mathcal{M}'_i$ and $\mathcal{M}''_i$), it would be better to ensure $\mathcal{M}''_i$ (with $i = 1, ..., n$) be pair-wise disjoint, whenever this is possible. In addition, we shall discharge $M \models \psi_1 \vee \cdots \vee \psi_n$ by one of the following methods:

- Application of static analysis;
- Application of expert knowledge.

The reason for not verifying $M \models \psi_1 \vee \cdots \vee \psi_n$ with model checking is that the verification of this formula is not necessarily simpler than the verification of $M \models \varphi$ (cf the discussion in Section 2.2). In order to be able to discharge $M \models \psi_1 \vee \cdots \vee \psi_n$ easily by the proposed methods, we restrict the formula ψ_i to be of the form $\Box(x = v_0 \vee x = v_i)$ where x is a global variable and v_0, v_i are constants. We call the global variable x for the *case-basis* of a partitioning.

2.1 Example

We use PROMELA as our modeling language. PROMELA is a language loosely based on CSP (Communication Sequential Processes) concepts [11] and is the input language of the model checker SPIN [12, 13]. PROMELA programs consist of processes, message channels, and variables. Processes are global objects. Message channels and variables can be declared either globally or locally within a process. Processes specify behavior, channels and global variables define the environment in which the processes run. Statements for the transition of states are similar to that of normal programming languages. There are basic elements (i.e. guards, assignments and communication statements), sequential composition of statements, case statements (with deterministic or non-deterministic choice of execution paths) and loops. We illustrate the *case-based partitioning* strategy with an example from [26]. Let S_1 and S_2 be two processes specified in PROMELA as follows:

```
proctype S1()
{
    byte i,k,num;
    atomic{ c0?num; c0!num; }
    do
    :: k< num → i=i+1; k=k+i;
    :: k> num → c1!0;
    :: k == num → c1!1;
    od;
}
```

```
proctype S2()
{
    byte i,num;
    atomic{ c0?num; c0!num; }
    do
    :: i × i < num → i=i+1;
    :: i × i > num → c2!0;
    :: i × i == num → c2!1;
    od;
}
```

S_1 is a process that reads a value from the channel $c0$ and tests whether it equals $\sum_{i=1}^{k}(i)$ for some k. It reports through the channel $c1$, with 1 meaning that it has found a k such that $num = \sum_{i=1}^{k}(i)$. S_2 is a process that reads a value and tests whether it equals k^2 for some k. It reports through the channel $c2$ in a similar manner.

Suppose that we have a procedure which first puts a number in the channel $c0$ and then uses S_1 or S_2 for performing a test to determine the property of the number. Suppose further that the choice of using S_1 or S_2 is determined by an input from the environment modeled by the process E as follows.

```
proctype P()
{
   c0!n1; e0?a1;
   if
   :: a1==1; do :: c1?r1; od;
   :: a1==2; do :: c2?r1; od;
   fi;
}
proctype E()  { if :: e0!1; :: e0!2; fi; }
```

The process E puts randomly 1 or 2 into $e0$. The process P puts the number $n1$ into $c0$, reads from $e0$ and gets the reported values through channel $c1$ or $c2$ according to $e0$. Assume that we want to verify whether there is no k such that $n1 = \sum_{i=1}^{k}(i)$ or $n1 = k^2$ and we represent the property by $\varphi : \Box(r1 \neq 1)$, i.e. we want to verify:

$$P||E||S_1||S_2 \models \varphi.$$

We chose $a1$ as the case-basis with two cases $a1 \neq 0 \rightarrow a1 = 1$ and $a1 \neq 0 \rightarrow a1 = 2$, in which, 0 is the initial value of $a1$. Let $M = P||E||S_1||S_2$. Instead of verifying the above formula, we verify:

$$M \models \psi_1 \rightarrow \varphi \text{ and } M \models \psi_2 \rightarrow \varphi$$

where

$$\psi_1 = \Box(a1 \neq 0 \rightarrow a1 = 1)$$
$$\psi_2 = \Box(a1 \neq 0 \rightarrow a1 = 2)$$

with the assumption that the following holds (by using expert judgment):

$$M \models \psi_1 \vee \psi_2$$

The model checker SPIN (version 3.2.4) was used for verification. Table 1 presents the number of states, the number of transitions, and the peek memory in the verification of the model with $n1 = 2$. It shows that by using *case-based partitioning*, we have reduced a problem to two sub-problems with complexity around 75 percent of the original one. Because the example is too simple, there is no advantage with respect to the memory usage in this example.

2.2 Discussion

The formula $\psi_1 \vee \psi_2$ can be verified by model checking as well. For comparison, the number of states, the number of transitions, and the peek memory for checking $M \models \psi_1 \vee \psi_2$ are respectively 614, 1038 and 1.493mb. This means that the numbers of states and

Table 1: Verification of the Model with $n1 = 2$

Verification Task	States	Transitions	Memory
φ	396	506	1.493mb
$\psi_1 \to \varphi$	299	369	1.493mb
$\psi_2 \to \varphi$	291	361	1.493mb

transitions for checking $M \models \psi_1 \vee \psi_2$ are larger than that for checking the original problem. Therefore there is no advantage with *case-based partitioning*, if we have to verify $M \models \psi_1 \vee \psi_2$ by model checking. Comparing the proposed strategy to that presented in [26], the performance of the latter is better with respect to this example. However, these strategies have different application areas, for instance, the latter is not applicable to the case study discussed in section 4. In addition, it is based on informal detail knowledge of the model, while the former is based on formal arguments which can be verified algorithmically and has therefore better potential for developing tools and for application in larger scale.

3 Static Analysis

In the previous example, in order to gain advantage with *case-based partitioning*, we have assumed that the correctness of $M \models \psi_1 \vee \psi_2$ is guaranteed by using the expert knowledge. In this section, we discuss how to guarantee the correctness of $M \models \psi_1 \vee \psi_2$ by static analysis for models that satisfy certain criteria, namely, models in which some variable is changed at most once during an execution of a model. Let v be a variable and *none* be the initial value of v.

Theorem 3.1 *Suppose that v is changed at most once during an execution of M. $M \models \varphi$ if and only if $M \models \Box(v \neq none \to v = i) \to \varphi$ for all $i \neq none$ in the range of variable v.*

Proof: Suppose that $\{none, v_1, ..., v_n\}$ is the range of v. Since v is changed at most once during an execution of M, the traces of M can be partitioned into n subsets $\mathcal{M}_1$,...,$\mathcal{M}_n$ such that $\mathcal{M}_i \models \Box(v = none \vee v = v_i)$ for $i \in \{1, ..., n\}$ and therefore $M \models \Box(v = none \vee v = v_1) \vee \cdots \vee \Box(v = none \vee v = v_n)$. The rest follows from Theorem 2.1. □

3.1 Models with Non-Deterministic Choice

We first consider models of a particular type with a situation where non-deterministic choice (from a set of given values) is used (as in the case-study in Section 4). Let the non-deterministic choice be represented by: $choice(x, \vec{v})$ where x is the global variable used to hold the chosen value and $\vec{v} = \{v_1, ..., v_n\}$ is the set of the possible values. We assume that $choice(x, \vec{v})$ is implemented as follows in PROMELA:

```
if
:: x = v1; ...
   ⋮
:: x = vn; ...
fi;
```

or

```
if
:: atomic{ x = v1; ... } ...
   ⋮
:: atomic{ x = vn; ... } ...
fi;
```

The set of traces of a model of this type has the potential (depending on the successful analysis explained in the subsequent subsections) to be partitioned into subsets such that each subset satisfies one of of the following formulas:

$$\Box(x \neq none \rightarrow x = v_1),\ \cdots,\ \Box(x \neq none \rightarrow x = v_n)$$

where *none* is the initial value of x, before the non-deterministic value has been assigned.

3.2 Conditions

The purpose of the static analysis is to show that the partition of the search space into these cases is complete, i.e. to show

$$M \models \Box(x \neq none \rightarrow x = v_1) \vee \cdots \vee \Box(x \neq none \rightarrow x = v_n).$$

Basically, we analyze the model in order to determine the set of cases and to ensure that x is changed at most once in an execution of a model (in accordance with Theorem 3.1), i.e. check the following conditions:

- The range of x is $\{none, v_1, ..., v_n\}$.
- The value of x is *none* before x is assigned a value, and after x is assigned a value it remains unchanged.

It is easy to determine the initial value of x (i.e. the actual value of *none*). In order to develop an algorithm to determine $v_1, ..., v_n$ and for checking the conditions, we further restrict the conditions. The details are as follows:

1. There is only one program segment that matches $choice(x, \vec{v})$ where x is given and $\vec{v}$ is free and may be matched to any set of values.
2. $choice(x, \vec{v})$ does not occur in a loop for the given pair $(x, \vec{v})$ where the value of $\vec{v}$ is obtained in the previous task.
3. The process declaration containing the program segment matching $choice(x, \vec{v})$ is only instantiated (in a manner similar to procedure calls) once during a run of the model.
4. The only statements that may update the value of x are those in $choice(x, \vec{v})$, i.e. neither statements of the form "$x = v$" nor "$c?x_1, ..., x_{i-1}, x, x_{i+1}, ..., x_k$" are allowed outside of the choice-statement.

3.3 Analyzing the Conditions

For simplicity, we assume that the model consists of global variable declarations and a set of process declarations where the process *init* is the only process that is active at the beginning and all other processes must be initiated by using the statement *run p(...)*.

Condition 1. It is easy to check whether there is only one program segment that matches $choice(x, \vec{v})$ in the model.

Condition 2. Loops are supposed to be of the following forms. It is not difficult to check whether $choice(x, \vec{v})$ is inside such loops. The details are omitted.

```
do          lab:
  ⋮    or     ⋮
od;         goto lab;
```

Condition 3. Let $insert(p, l)$ be a function that returns a list extending l with p. Let q be a process name and l a list of process names. Let the function for checking condition 3 be $once(q, l)$ defined as follows.

- if q is the process *init* or it is not instantiated anywhere, return $true$;
- if the instantiation of q appears in a loop or in more than one place,

  return $false$;
- if the instantiation of q appears in q' and q' is not in l,

  return $once(q', insert(q, l))$;
- if the instantiation of q appears in q' and q' is in l,

  i.e. an invalid loop of process instantiations is detected, return $true$;

Proposition 3.1 *Let p be the process containing $choice(x, \vec{v})$ and [] be the empty list. Condition 3 holds if $once(p, [])$ is true.*

Condition 4. Let p be a process name, x a variable name and l a list of process names. Let the function for checking condition 4 be $nochange(p, x, l)$ defined as follows.

- if (p in l), return $true$;
- if (x is the name of a formal parameter or a local variable of p),

  let $\{p_1, ..., p_n\}$ be the set of process instantiations inside of p,

  return $nochange(p_1, x, insert(p, l)) \wedge \cdots \wedge nochange(p_n, x, insert(p, l))$;
- if (x is not the name of any formal parameter or local variable of p),

  for each basic statement s of p

  - if s is of the form $c?x_1, ..., x_{i-1}, x, x_{i+1}, ..., x_k$, return $false$;
  - if s is of the form $x = w$ and $s \notin choice(x, \vec{v})$, return $false$;

  let $\{p_1, ..., p_n\}$ be the set of process instantiations inside of p,

  return $nochange(p_1, x, insert(p, l)) \wedge \cdots \wedge nochange(p_n, x, insert(p, l))$;

The list l and the first line in this definition is used to break recursion. Details may be added to the above definition in order to make the function $nochange(p, x, l)$ more accurate, for instance, we may add special treatment for assignment statements of the form $x = x$.

Proposition 3.2 *Let x be the case-basis. Condition 4 holds if $nochange(init, x, [])$ is true.*

3.4 Summary and Discussion

By summarizing the above discussions, we refine the steps of *case-based partitioning* as follows:

- Check the initial value of x in the global variable declaration and denote the value by *none*.

- Determine whether $choice(x, \vec{v})$ has a unique match to a program segment.
- Find the unique match for $choice(x, \vec{v})$ and denote the potential values (except *none*) of x by $v_1, ..., v_n$ and the process containing $choice(x, \vec{v})$ by q.
- Check whether $choice(x, \vec{v})$ is within the scope of a loop.
- Calculate $once(q, [])$ to determine whether q is called no more than once.
- Calculate $nochange(init, x, [])$ to determine that no other statements than $choice(x, \vec{v})$ may change the value of x.
- For each $i \in \{1, ..., n\}$, construct $\varphi_i = \Box(x \neq none \rightarrow x = v_i) \rightarrow \varphi$ as a subgoal for verification.
- Use the model checker SPIN to check whether $M \models \varphi_i$ holds for $i = 1, ..., n$.

Automated detection of case-bases. In the previous analysis, the user has to provide a variable as the case-basis for partitioning. For the models of the discussed type, the case-basis can be detected automatically. After detecting potential case-bases, we may apply this strategy automatically.

- For each global variable z in the model, determine whether there is a unique match for $choice(z, \vec{v})$.
- If several variables satisfy the condition, order these variables according to some heuristics.
- For each variable in the list created in the previous step, perform the static analysis, until success or the list is empty.
- If success, perform *case-based partitioning* and model checking.

The first task is not difficult for simple variables. However it is more complicated, if variables are composed (such as arrays). For developing an algorithm, we may only implement those strategies that are feasible for implementation and give advice about the limitation of the algorithm.

The heuristics could be a function estimating the placing of $choice(x, \vec{v})$ in the process definition. The few actions before $choice(x, \vec{v})$, the better should x be as the candidate of the bases for *case-based partitioning*. In addition, we may consider the number of potential cases for a variable as a parameter for such a heuristic function.

Parallel computation. It is easy to take advantage of parallel and networked computing power when the problem can be decomposed in independent sub-problems. One problem is how to fully exploit the available computing resources. It may not be possible (with the proposed strategy) to divide a problem in such a way that all sub-problems require approximately the same amount of time for model checking. It could be better with respect to the utilization of the available computing power, if there are more sub-problems than available computing units. In such cases, we may estimate the difficulty of the sub-problems and make a schedule for the sub-problems.

Models with open-environment. The static analysis is also feasible for models with open environment (such as input from a sensor). Let c represent the channel for the purpose of communicating the value of the environment variable to the main process. We may model the open environment with a process that sends a value (one of $v_1, ..., v_n$) through the communication channel as follows:

```
if                     if
:: c!v1; ···           :: atomic{ c!v1; ··· } ···
   ⋮          or          ⋮
:: c!vn; ···           :: atomic{ c!vn; ··· } ···
fi;                    fi;
```

On the other side, we may model the receipt of the value in the main process as follows:

$$c?x.$$

Similar to the previous discussion on non-deterministic choice, we can calculate a set of formulas that represents the set of cases for a given variable. For proving that the set of cases are complete, we can provide similar conditions and check whether the model complies with these conditions by algorithmic means. The details are omitted.

4 A Case Study

The case study is verification of Needham-Schroeder-Lowe Protocol [16]. For analyzing protocols, models of these must be created; and for modeling protocols, different modeling languages can be used. For instance, CSP [11] was used in [16] and Petri Nets [22] was used in [6]. We shall use PROMELA for modeling and SPIN for verification. We first consider the protocol and then consider a simpler version (i.e. the Needham-Schroeder Protocol) of the protocol.

Needham-Schroeder-Lowe Protocol. The following is a description of this protocol.

$$\begin{array}{ll} A \rightarrow B: & \{n_a, A\}_{PK(B)} \\ B \rightarrow A: & \{n_a, n_b, B\}_{PK(A)} \\ A \rightarrow B: & \{n_b\}_{PK(B)} \end{array}$$

Here A is an initiator who seeks to establish a session with responder B. A selects a nonce n_a, and sends it along with its identity to B encrypted using B's public key. When B receives this message, it decrypts the message to obtain the nonce n_a. It then returns the nonce n_a along with a new nonce n_b and its identity to A, encrypted using A's public key. When A receives this message, he should be assured that he is talking to B, since only B should be able to decrypt the first message to obtain n_a. A then returns the nonce n_b to B, encrypted using B's public key. Then B should be assured that he is talking to A.

The property to be checked is as follows: whenever the *responder* think that he is talking to the *initiator*, the *initiator* must have requested to establish a session with the *responder* at beforehand. Formally the property φ is specified as follow:

$$\Box(talkto[responder] = initiator \rightarrow request[initiator] = responder)$$

Table 2: Verification of the Model with and without the Strategy

Verification Task	Time	States	Transitions	Memory
φ	28.9	879613	1.69402m	100.00mb
$\psi_1 \rightarrow \varphi$	25.0	748951	1.44152m	85.35mb
$\psi_2 \rightarrow \varphi$	4.0	136143	0.25813m	16.64mb

where the formula $request[initiator] = responder$ is *true* from the time the first message is sent by the initiator and $talkto[responder] = initiator$ is *true* from the time the last message is received by the responder.

In order to check this property, we have created a model M of a communication system using this protocol with different actors. The modeling follows the strategy specified in [16], however, we use PROMELA instead of CSP. There are three actors: an *initiator*, a *responder* and an *intruder*. The behaviors of these actors are as follows.

- The initiator starts with sending a message to the *responder* or the *intruder*, then acts according to the protocol. The choice of the *responder* or the *intruder* as the communication partner is non-deterministic.
- The *responder* waits for an incoming message and acts according to the protocol.
- The *intruder* may engage in any action which is possible (in principle) for him to perform at any time.

The goal of the verification is

$$M \models \varphi.$$

Let send_to be a variable that is assigned a value randomly chosen from *intruder* and *responder* at the beginning of the behavior of the model. Instead of proving

$$M \models \varphi$$

we prove

$$M \models \psi_1 \rightarrow \varphi \text{ and } M \models \psi_2 \rightarrow \varphi$$

where

$$\psi_1 = \Box(send_to \neq none \rightarrow send_to = intruder)$$
$$\psi_2 = \Box(send_to \neq none \rightarrow send_to = responder)$$

with the assumption that the following holds (by the proposed static analysis):

$$M \models \psi_1 \vee \psi_2$$

Table 2 presents the model checking time (in seconds), the number of states, the number of transitions, and the peek memory in the verification of respectively φ, $\psi_1 \rightarrow \varphi$ and $\psi_2 \rightarrow \varphi$. Comparing the first line with the last two lines of Table 2, the benefit does not seem to be very big. This is because that the second case is too small and does not take away much of the burden of the other case.

If we had a more complicated model, for instance, a model where the initiator may seek to establish a session with the intruder, the responder and another responder, there would be a better result with the strategy. Table 3 presents the model checking time, the

Table 3: Verification of the Extended Model

Verification Task	Time	States	Transitions	Memory
φ	34.3	997839	1.92084m	125.29mb
$\psi_1 \to \varphi$	25.8	750778	1.44335m	94.67mb
$\psi_2 \to \varphi$	4.3	137970	0.25996m	18.59mb
$\psi_3 \to \varphi$	3.9	123707	0.23244m	16.75mb

Table 4: Error Detection of the Simpler Protocol

Verification Task	Time	States	Transitions	Memory	Error
φ	20.9	634140	1.22466m	72.45mb	yes
$\psi_1 \to \varphi$	18.2	748951	1.44152m	63.13mb	yes
$\psi_2 \to \varphi$	3.9	136143	0.25813m	16.64mb	no

number of states, the number of transitions, and the peek memory in the verification of such an extended model.

In Table 3, ψ_3 is $\Box(\mathit{send_to} \neq \mathit{none} \to \mathit{send_to} = \mathit{responder_new})$ where $\mathit{responder_new}$ is the identifier of the new responder. This table clearly shows the advantage of *case-based partitioning* with respect to the peek memory. It is surprising that the total model checking time is also better (although not significant) with this strategy. One explanation is that in the case of checking $M \models \varphi$, the model checker is less efficient with respect to "time per transition", because of the bigger memory usage.

A Simpler Protocol. The following is a description of this protocol. This protocol is not necessarily incorrect, however, due to the absence of certain assumptions on the usage and the environment, some problem may occur.

$$\begin{array}{ll} A \to B: & \{n_a, A\}_{PK(B)} \\ B \to A: & \{n_a, n_b\}_{PK(A)} \\ A \to B: & \{n_b\}_{PK(B)} \end{array}$$

The difference between this and the previous one is that B is missing in the second message. We have created a similar model and tried to verify the same properties.

Table 4 presents the model checking time, the number of states, the number of transitions, the peek memory for model checking, and whether errors were detected in the verification attempt. The first line of Table 4 is the verification without using *case-based partitioning*. When using the strategy, the error was detected in line 2 which is the case where the initiator tries to establish a session with the intruder.

5 Discussion

The proposed strategy looks similar to a strategy proposed in [18]. A theorem in [18] is as follows:

If, for all i in the range of variable v, $\models \Box(v = i \to \varphi)$, then $\models \Box\varphi$.

Table 5: Error Detection using a Related Strategy

Verification Task	Time	States	Transitions	Memory	Error
φ_1	32.5	985613	1.89852m	111.88mb	no
φ_2	20.9	634140	1.22466m	72.45mb	yes
φ_3	32.5	985613	1.89852m	111.88mb	no

The basic idea here is to break the proof of temporal property $\Box\varphi$ into cases based on the value of a given variable v. Suppose that the value of v refers to the location in some large data structure (or array), to prove any given case $v = i$, it is only necessary to refer to element i of the array and the other elements of the array can be eliminated from the model by replacing them with an "unknown "value [18].

This theorem is much simpler than Theorem 3.1 when restricted to formulas of the form $\Box\varphi$. However, when the strategy is used alone, it does not work well with the example in Section 2 and the case study in Section 4 in the experimental study using the model checker SPIN (version 3.2.4), The experiment with the case study is reported as follows.

Let φ' be

$$(talkto[responder] = initiator \rightarrow request[initiator] = responder).$$

Recall that the property we wanted to prove in the case study was $\Box\varphi'$. Instead of verifying this property, we may try to verify the following three properties (according to the above strategy):

$$\varphi_1\text{: } \Box(send_to = none \rightarrow \varphi')$$
$$\varphi_2\text{: } \Box(send_to = intruder \rightarrow \varphi')$$
$$\varphi_3\text{: } \Box(send_to = responder \rightarrow \varphi')$$

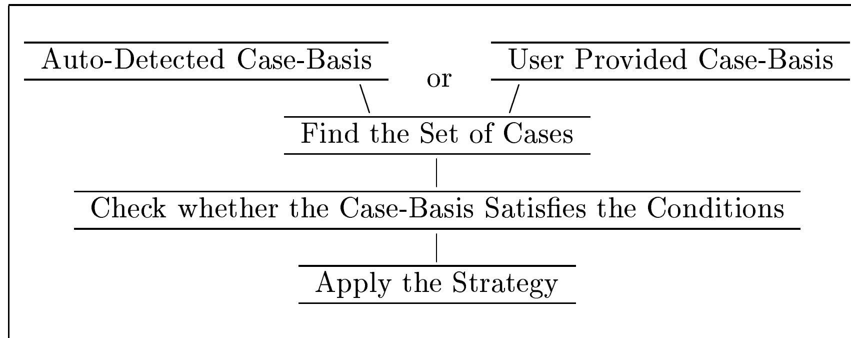
However, the model checking of each of the above 4 formulas (i.e. $\Box\varphi'$, φ_1, φ_2, φ_3) takes exactly the same amount of resource. This means that no advantage can be gained by verifying the three properties instead of the original property in this case study.

The same strategy can be applied to the simper version of the protocol. Table 5 presents the data for the verification of the three properties with the above strategy. Line 2 of Table 5 is the same as the first line of Table 4 except an insignificant difference on the model checking time. This also means that no advantage can be gained by using the above strategy for error detection in this case.

6 Concluding Remarks

A strategy for reducing the peek memory for model checking was proposed. The strategy uses case-based partitioning of search space in model checking. Completeness of the cases may be established by using expert judgment or by static analysis. The latter can be done automatically for certain types of models. A framework for developing a tool for static analysis has been described.

For the use of *case-based partitioning* with automated static analysis, the types of problems have to be restricted. Two types of problems were described in the previous sections. One is models with non-deterministic choice and the other is models with open-environment (only the first type was discussed in detail). The process of applying *case-based partitioning* is as follows:



The case study has illustrated the usefulness of the strategy. For instance, if we had a computer with only 96mb memory, we would not be able to finish the verification of $M \models \varphi$ without using *case-based partitioning* (cf Table 2), unless techniques for compression of memory or other techniques are used. For the first, the use of compression techniques increases model checking time, and for the second, there is always cases where memory is critical, even with the use of compression techniques.

The strategy can also be used as the basis for utilizing parallel and networked computing power for model checking system models of the discussed types, although the the complexity of the sub-tasks with respect to model checking may be very different.

Acknowledgments: The author thanks Dongdai Lin for helpful discussions on protocol verification. The author also thanks anonymous referees for their constructive comments and critics.

References

[1] H. R. Andersen. Model checking and boolean graphs. Theoretical Computer Science 126(1): 3-13. 1994.

[2] S. Berezin and S. Campos and E. M. Clarke. Compositional Reasoning in Model Checking. Lecture Notes in Computer Science 1536:81-102. COMPOS 1997.

[3] Roderick Bloem, Kavita Ravi and Fabio Somenzi. Efficient Decision Procedures for Model Checking of Linear Time Logic Properties. Lecture Notes in Computer Science 1633: 222-235. CAV 1999, Trento, Italy.

[4] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. IEEE Symposium on Logic in Computer Science 5:428-439, 1990.

[5] E. M. Clarke, O. Grumberg and D. E. Long. Model Checking and Abstraction. ACM Transactions on Programming Languages and Systems 16(5):1512-1542, 1994.

[6] F. Crazzolara and G. Winskel. Petri Nets in Cryptographic Protocols. Proceedings of the 15th International Parallel and Distributed Processing Symposium. IEEE-IPDPS-01:149-157. 2001.

[7] E. A. Emerson. Temporal and Modal Logic. Handbook of Theoretical Computer Science (B):997-1072. 1990.

[8] E. A. Emerson and A. P. Sistla. Symmetry and Model Checking. Lecture Notes in Computer Science 697:463-478. CAV 1993.

[9] R. Enders, T. Filkorn and D. Taubner. Generating BDDs for Symbolic Model Checking in CCS. Lecture Notes in Computer Science 575:203-213. CAV 1991.

[10] Rob Gerth, Doron Peled, Moshe Vardi and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. 15th IFIP WG6.1 International Symposium on Protocol Specification, Testing and Verification: 3-18. June 1995, Warsaw, Poland.

[11] C. A. R. Hoare. Communicating Sequential Processes. Prentice Hall. 1985.

[12] G. J. Holzmann. Design and Validation of Computer Protocols. Prentice Hall, New Jersey, 1991.

[13] G. J. Holzmann. The model checker Spin. IEEE Transactions on Software Engineering 23(5):279-295. May 1997.

[14] S. Katz and D. Peled. Verification of distributed programs using representative interleaving sequences. Distributed Computing 6: 107-120, 1992.

[15] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. Journal of Formal methods in System Design 6:1-35. 1995.

[16] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. Proceedings of the 2nd International Workshop on Tools and Algorithms for the construction and Analysis of Systems.

[17] G. S. Manku, R. Hojati and R. K. Brayton. Structural Symmetry and Model Checking. Lecture Notes in Computer Science 1427:159-171. CAV 1998, Vancouver, Canada.

[18] K. L. McMillan. Verification of Infinite State Systems by Compositional Model Checking. Lecture Notes in Computer Science 1703:219-234. CHARME 1999.

[19] L. I. Millett, T. Teitelbaum. Issues in Slicing PROMELA and Its Applications to Model Checking, Protocol Understanding, and Simulation. STTT 2(4):343-349. 2000.

[20] D. Peled. All from one, one for all, on model-checking using representatives. Lecture Notes in Computer Science 697: 409-423. CAV 1993.

[21] D. Peled. Ten Years of Partial Order Reduction. Lecture Notes in Computer Science 1427:17-28. CAV 1998, Vancouver, Canada.

[22] W. Reisig. Petri Nets - An Introduction. Springer Verlag 1985.

[23] Fabio Somenzi and Roderick Bloem. Efficient Büchi Automata from LTL Formulae. Lecture Notes in Computer Science 1855: 248-263. CAV 2000, Chicago, USA.

[24] K. Stahl, K. Baukus, Y. Lakhnech and M. Steffen. Divide, abstract, and model-check. Lecture Notes in Computer Science 1680:57-76. Proceedings of the 5th International SPIN Workshop. July 1999, Trento, Italy.

[25] A. Valmari. A stubborn attack on state explosion. Formal Methods in System Design 1(4):297-322, December 1992.

[26] W. Zhang. A Strategy for Improving the Efficiency of Procedure Verification. Lecture Notes in Computer Science 2434:113-126. SAFECOMP 2002, Catania, Italy.