Modular Verification of Web Services Using Efficient Symbolic Encoding and Summarization

Fang Yu
joint work with: Chao Wang*, Aarti Gupta* and Tevfik Bultan

University of California, Santa Barbara
and NEC Labs America, Princeton*

November 12, 2008
Outline

1 Introduction
   ■ Motivation
   ■ An Overview of Our Approach

2 Technical Details
   ■ Summarization
   ■ Assertion Checking

3 Experiments

4 Conclusion
Motivation

- Increasing interest in web-based business management involving inter-organizational interactions and critical transactions
Motivation

- Increasing interest in web-based business management involving inter-organizational interactions and critical transactions
- Web services provide mechanisms implementing such applications
Motivation

- Increasing interest in web-based business management involving inter-organizational interactions and critical transactions
- Web services provide mechanisms implementing such applications
- Need formal mechanisms to ensure that web services behave properly
Motivation

- Increasing interest in web-based business management involving inter-organizational interactions and critical transactions
- Web services provide mechanisms implementing such applications
- Need formal mechanisms to ensure that web services behave properly
- We propose an automatic verification tool featuring efficient symbolic encoding and modular verification using summarization
Web Services

- Interoperable Machine to Machine software
- Some Industry Standards: Business Process Execution Language (BPEL), Web Service Description Language (WSDL)
BPEL Web Services

A distributed system with both multi-threading (internal) and message-passing (external).

\[\Delta \triangledown \text{: fork/join structure (shared variables)}\]

\[\leftrightarrow \rightarrow \text{: synchronous/asynchronous communication (messages)}\]
BPEL Web Services

A distributed system with both multi-threading (internal) and message-passing (external).

- *flow activities* ⇒ *fork/join structure*

\[ \Delta \nabla : \text{fork/join structure (shared variables)} \]
\[ \leftrightarrow : \text{synchronous/asynchronous communication (messages)} \]
BPEL Web Services

A distributed system with both multi-threading (internal) and message-passing (external).

- **flow activities** ⇒ fork/join structure
- **invoke, receive, reply** activities ⇒ asynchronous/synchronous communications

\[\text{\textDelta\nabla} : \text{fork/join structure (shared variables)}\]
\[\text{\Rightarrow} : \text{synchronous/asynchronous communication (messages)}\]
Monolithic Analysis

- Consider all of them as one composite service by adding a outer fork/join structure
- Need to consider all interleavings among threads
- Suffer from state explosion problem
Modular Verification

From processes to summaries.

- Interference among processes is limited to the values of messages
- Summarize processes on messages
Modular Verification

- Modular Analysis: check one process within which interactions among other processes are patched by their summaries
Modular Verification

- Modular Analysis: check one process within which interactions among other processes are patched by their summaries
- From $P_1 \times \ldots \times P_n$ to $P_1 + \ldots + P_n$
Modular Verification of Web Services Using Efficient Symbolic Encoding and Summarization

Introduction

An Overview of Our Approach

Modular Verification

- Modular Analysis: check one process within which interactions among other processes are patched by their summaries
- From $P_1 \times \ldots \times P_n$ to $P_1 + \ldots + P_n$
- No precision loss with respect to assertion checking within processes

![Diagram showing modular verification process]
## Our Framework

<table>
<thead>
<tr>
<th>OASIS Standard</th>
<th>Intermediate Format</th>
<th>Symbolic Encoding</th>
<th>Reachability Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSDL</td>
<td>Concurrent Process Graph</td>
<td>Interleaving Semantics</td>
<td>Fixpoint Computation</td>
</tr>
<tr>
<td>BPEL</td>
<td></td>
<td>sel $= 1$ $\land b_0 = b_0$ $\land i_1 = i_1 + i_2$ ...</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>sel $= 2$ $\land b_0 = b_2$ $\land i_1 = i_1$ ...</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>sel $= 3$ $\land b_0 = b_2$ $\land i_2 = 3$ ...</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Summaries $S(m1,m2,...)$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Assertion Checking (Frontier)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Summarization (on Messages)</td>
</tr>
</tbody>
</table>
Summarization: A Simple Example

Consider the following two concurrent processes.

- $P_A$ invokes $P_B$
- An assertion within $P_A$ at node 4

![Diagram of two concurrent processes $P_A$ and $P_B$](image)
Summarize Process Behavior

A relation among input and output messages
Summarize Process Behavior

A relation among input and output messages

- Encode each send activity \((ch_i!x)\) as an assignment to a message \((m'_i = x)\)
Summarize Process Behavior

A relation among input and output messages

- Encode each send activity \((ch_i!x)\) as an assignment to a message \((m'_i = x)\)
- Encode each receive activity \((ch_i?x)\) as an assignment to a variable \((x' = m_i)\)
Summarize Process Behavior

A relation among input and output messages

- Encode each send activity \((ch_i!x)\) as an assignment to a message \((m_i' = x)\)
- Encode each receive activity \((ch_i?x)\) as an assignment to a variable \((x' = m_i)\)
- Compute the forward fixpoint of reachable states
Summarize Process Behavior

A relation among input and output messages

- Encode each send activity \((ch_i!x)\) as an assignment to a message \((m'_i = x)\)
- Encode each receive activity \((ch_i?x)\) as an assignment to a variable \((x' = m_i)\)
- Compute the \textbf{forward fixpoint} of reachable states
- Project the fixpoint to \textbf{input and output} messages (using existential quantifier elimination)
Summarize Process Behavior: A Simple Example

The summary of $P_B$ is:

$$(m_1 > 0 \land m_2 = 1) \lor (m_1 = 0 \land m_2 = 0) \lor (m_1 < 0 \land m_2 = -1)$$
Compose Summaries: A Simple Example

Compose summaries by conjoining the summaries of other processes with the receive activities.
Compose Summaries: A Simple Example

- Compose summaries by conjoining the summaries of other processes with the receive activities
- One can prove $P_A$'s assertion modularly

```
 PA
  0
    x:=1
  1
    eh1!x  m1:=x
  2
    x:=x-1
  3
    eh2?y  y:=m2 and ((m1>0 and m2=1) or ...)
  4
    assert(y==1)
```
Modularity of Processes

An assertion can be proven via modular analysis if and only if it can be proven via monolithic analysis.

- \( T \): transition relation, \( I \): initial states, \( X \): variables
- \( \text{reach}(T, I) \) returns the fixpoint of reachable states
- The insight comes from the property:

\[
\text{reach}(T, I(C) \land I(X)) \equiv I(C) \land \text{reach}(T, I(X))
\]

if \( C \subseteq X \) are parameterized constants (not defined in \( T \)).
Modularity of Processes

- From the receiver’s perspective, a message is a parameterized constant
Modularity of Processes

- From the receiver’s perspective, a message is a parameterized constant.
- One can summarize the receiver’s behavior ($reach(T, I(X))$) without knowing the states of its input messages ($I(C)$).
Modularity of Processes

- From the receiver’s perspective, a message is a parameterized constant.

- One can summarize the receiver’s behavior ($reach(T, I(X))$) without knowing the states of its input messages ($I(C)$).

- One can compute the **precise** reachable states of the receiver’s output messages ($reach(T, I(C) \land I(X))$) by conjoining:
  - the states of the receiver’s input messages ($I(C)$) and
  - the receiver’s summary ($reach(T, I(X))$).
Modularity of Processes: A Simple Example

The state of $m_1$ is initialized upon sending and is imposed implicitly after sending.

```
P_A
    0
      x:=1
    1
      -eh4!x m1:=x
    2
      x:=x-1
    3
      -eh2>y y:=m2 and ((m1>0 and m2=1) or ...)
    4
      assert(y==1)
```
Modularity of Processes: A Simple Example

- The state of $m_1$ is initialized upon sending and is imposed implicitly after sending.
- The summary of $P_B$ (the relation among $m_1$ and $m_2$) is conjoined upon receiving.

```
PA
  0
  x:=1
  1
  ¬eh1!x m1:=x
  2
  x:=x-1
  3
  ¬eh2?y y:=m2 and ((m1>0 and m2=1) or ...)
  4
  assert(y==1)
```
Modularity of Processes: A Simple Example

- The state of $m_1$ is initialized upon sending and is imposed implicitly after sending.
- The summary of $P_B$ (the relation among $m_1$ and $m_2$) is conjoined upon receiving.
- $P_A$ gets the precise reachable states of $m_2$ ($m_2 = 1$).

```
P_A
0
  \downarrow x:=1
1
  \downarrow -eh4!x m1:=x
  \downarrow x:=x-1
2
  \downarrow -eh2?y y:=m2 and ((m1>0 and m2=1) or ...)
3
  \downarrow assert(y==1)
4
```
Restrictions

- We assume that each channel is associated with precisely one send activity and one receive activity.
- The examples we analyzed do not violate this condition.
- For the specifications which violate this condition:
  - Rename channels if multiple send/receive pairs use the same channel.
  - If there is a send or receive activity within a loop, unwind the loop a fixed number of times.
Efficient Assertion Checking

- Use *frontier*, the new states reachable from the previous iteration, to detect violation and convergency.
Efficient Assertion Checking

- Use \textit{frontier}, the new states reachable from the previous iteration, to detect violation and convergency
- No need to store whole reachable states
Efficient Assertion Checking

- Use *frontier*, the new states reachable from the previous iteration, to detect violation and convergency.
- No need to store whole reachable states.
- \( I \): initial states, \( T \): transition relation, \( Err \): risk states (violate assertions).
Efficient Assertion Checking

- Use *frontier*, the new states reachable from the previous iteration, to detect violation and convergency.
- No need to store whole reachable states.
- $F^0 = I$ and $F^i = post(T, F^{i-1}) \setminus F^{i-1}$
Efficient Assertion Checking

- Use *frontier*, the new states reachable from the previous iteration, to detect violation and convergency.
- No need to store whole reachable states.
- $F^0 = I$ and $F^i = post(T, F^{i-1}) \setminus F^{i-1}$
- Assertion violated at the $i^{th}$ iteration when $F^i \cap Err \neq \emptyset$
Termination Condition

- When the CPG is acyclic,
  - No back edges
  - Terminate when $F^i$ is empty
Termination Condition

- When the CPG is acyclic,
  - No back edges
  - Terminate when \( F^i \) is empty
- When the CPG is not acyclic,
  - Compute \( S_{back} \), the states associated with the source nodes of the back edges (much smaller than the universe)
  - At each iteration, compute \( R_{back} \), the set of reached states fall in \( S_{back} \)
  - Terminate when \( F^i \setminus R_{back} \) is empty
The Assertion Checking Algorithm

\[
\text{Reach}\_\text{frontier}(T, I, Err, S_{back})
\]

- \( F = I \);
- \( R_{back} = I \cap S_{back} \);

- \text{false} - assertion violated. \text{true} - assertion proven.
The Assertion Checking Algorithm

\[
\text{Reach\_frontier}(T,I,Err,S_{\text{back}})
\]

- \( F = I; \)
- \( R_{\text{back}} = I \cap S_{\text{back}}; \)
- WHILE \((F \neq \emptyset)\) {
- }

\text{false} - assertion violated. \text{true} - assertion proven.
The Assertion Checking Algorithm

\[
\text{Reach}_{\text{frontier}}(T, I, Err, S_{\text{back}})
\]

- \( F = I; \)
- \( R_{\text{back}} = I \cap S_{\text{back}}; \)
- WHILE (\( F \neq \emptyset \)) {
  - IF (\( (F \cap Err) \neq \emptyset \)) RETURN false;
}

false - assertion violated. true - assertion proven.
The Assertion Checking Algorithm

\[
\text{Reach}\_\text{frontier}(T,I,\text{Err},\text{S}_{\text{back}}) =
\]

\[
\begin{align*}
F &= I; \\
R_{\text{back}} &= I \cap \text{S}_{\text{back}}; \\
\text{WHILE } (F \neq \emptyset) \{
\begin{align*}
\text{IF } ((F \cap \text{Err}) \neq \emptyset) & \text{ RETURN } false; \\
F &= (\text{post}(T, F) \setminus F) \setminus R_{\text{back}};
\end{align*}
}\}
\]

\text{\emph{false} - assertion violated. \textit{true} - assertion proven.}
The Assertion Checking Algorithm

\[
\text{Reach\_frontier}(T, I, Err, S_{back})
\]

- \( F = I; \)
- \( R_{back} = I \cap S_{back}; \)
- \( \text{WHILE } (F \neq \emptyset) \{ \)
  - \( \text{IF } ((F \cap Err) \neq \emptyset) \text{ RETURN false;} \)
  - \( F = (post(T, F) \setminus F) \setminus R_{back}; \)
  - \( R_{back} = R_{back} \cup (F \cap S_{back}); \)
- \} \)

\( \text{false} \) - assertion violated. \( \text{true} \) - assertion proven.
The Assertion Checking Algorithm

\[
\text{Reach\_frontier}(T, I, \text{Err}, S_{\text{back}})
\]

- \( F = I; \)
- \( R_{\text{back}} = I \cap S_{\text{back}}; \)
- \( \text{WHILE } (F \neq \emptyset) \{ \)
  - \( \text{IF } ((F \cap \text{Err}) \neq \emptyset) \text{ RETURN false;} \)
  - \( F = (\text{post}(T, F) \setminus F) \setminus R_{\text{back}}; \)
  - \( R_{\text{back}} = R_{\text{back}} \cup (F \cap S_{\text{back}}); \)
- \} \)
- \( \text{RETURN true;} \)

\text{false} - assertion violated. \text{true} - assertion proven.
### Experiment: Loan Approval

<table>
<thead>
<tr>
<th></th>
<th>Monolithic Verification</th>
<th>Modular Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All</td>
<td>Approval</td>
</tr>
<tr>
<td>Result</td>
<td>P</td>
<td>P</td>
</tr>
<tr>
<td>Time (s)</td>
<td>1227.2</td>
<td>124.5</td>
</tr>
<tr>
<td>Memory (MB)</td>
<td>810</td>
<td>490</td>
</tr>
<tr>
<td>ITRs</td>
<td>32</td>
<td>16</td>
</tr>
</tbody>
</table>

- Customer invokes Approval which invokes Assessor and Approver
- Result: NA-did not terminate, P-passed assertion checks, S-summarized
- ITRs: the number of iterations of the fixpoint computation
### Experiment: Travel Agency

<table>
<thead>
<tr>
<th></th>
<th>Monolithic Verification</th>
<th>Modular Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All</td>
<td>VTA</td>
</tr>
<tr>
<td>Result</td>
<td>NA</td>
<td>P</td>
</tr>
<tr>
<td>Time (s)</td>
<td>18947</td>
<td>814</td>
</tr>
<tr>
<td>Memory (MB)</td>
<td>1663</td>
<td>363</td>
</tr>
<tr>
<td>ITRs</td>
<td>57</td>
<td>55</td>
</tr>
</tbody>
</table>

- User invokes VTA which invokes Hotel and Flight
- Result: NA-did not terminate, P-passed assertion checks, S-summarized
- ITRs: the number of iterations of the fixpoint computation
Conclusion

- We propose an automatic symbolic model checker for concurrent systems having multi-threading and message-passing.
Conclusion

- We propose an automatic symbolic model checker for concurrent systems having multi-threading and message-passing.
- We propose modular verification for message-passing processes to achieve scalability.
Conclusion

- We propose an automatic symbolic model checker for concurrent systems having multi-threading and message-passing.
- We propose modular verification for message-passing processes to achieve scalability.
- We propose an efficient symbolic encoding and reachability analysis to facilitate our approach.
Conclusion

- We propose an automatic symbolic model checker for concurrent systems having multi-threading and message-passing.
- We propose modular verification for message-passing processes to achieve scalability.
- We propose an efficient symbolic encoding and reachability analysis to facilitate our approach.
- We have implemented a prototype tool that can automatically analyze web services specified in BPEL+WSDL.
Related Work

- **BPEL Verification:**
  - Safety property [Foster et al. ICWS04] [Lohmannnet et al. BPM06]
  - LTL property [Fu et al. WWW04] [Nakajima ENTCS06]
  - Timed CTL property [Qiu et al. ISFM05]
Related Work

- **BPEL Verification:**
  - Safety property [Foster et al. ICWS04] [Lohmann et al. BPM06]
  - LTL property [Fu et al. WWW04] [Nakajima ENTCS06]
  - Timed CTL property [Qiu et al. ISFM05]

- **Summarization:**
  - Sequential summarization for BPEL [Duan et al. ICWS04]
  - Transaction-based summarization [Qadeer et al. POPL04]
Related Work

- **BPEL Verification:**
  - Safety property [Foster et al. ICWS04] [Lohmannet et al. BPM06]
  - LTL property [Fu et al. WWW04] [Nakajima ENTCS06]
  - Timed CTL property [Qiu et al. ISFM05]

- **Summarization:**
  - Sequential summarization for BPEL [Duan et al. ICWS04]
  - Transaction-based summarization [Qadeer et al. POPL04]

- **Compositional Reasoning:**
  - LTSA [Cobleigh et al. TACAS03]
  - Magic/Comfort [Chaki et al. FMSD04] [Sharygina et al. CAV05]
Thank you. Any Questions?