Linking the Semantics of BPEL using Maude

Peng Liu and Huibiao Zhu

East China Normal University
1. Introduction

2. BPEL and Rewriting Logic

3. Algebraic Semantics and Head Normal Form

4. Generation of Operational Semantics from Algebraic Semantics

5. Linking Theories of the Semantics

6. Conclusion and Future work
Outline

1 Introduction

2 BPEL and Rewriting Logic

3 Algebraic Semantics and Head Normal Form

4 Generation of Operational Semantics from Algebraic Semantics

5 Linking Theories of the Semantics

6 Conclusion and Future work
Motivation:
A BPEL-like Language to study the semantics and mechanisms of BPEL

Linking theories of semantics:
Operational semantics, Denotational semantics and Algebraic semantics, the three semantics should provide the same understanding of the language from different viewpoints and they should be consistent

Works in this paper
We study the operational semantics and its linking with algebraic semantics
Peng Liu and Huibiao Zhu

Introduction (cont.)

Linking the Semantics of BPEL using Maude

Diagram:

1. BPEL Program
2. Algebraic Laws
3. Head Normal Form
4. Operational Semantics

Flow:
- BPEL Program → Algebraic Laws → Head Normal Form
- Head Normal Form → Operational Semantics
Outline

1. Introduction

2. BPEL and Rewriting Logic

3. Algebraic Semantics and Head Normal Form

4. Generation of Operational Semantics from Algebraic Semantics

5. Linking Theories of the Semantics

6. Conclusion and Future work
The Syntax of BPEL

\[
BA ::= \text{skip} \mid x := e \mid \text{rec } a \times \mid \text{rep } a \times \mid \text{throw}
\]

\[
A ::= BA \mid g \circ A \mid b \triangleright l \mid A; A \mid A \triangleleft b \triangleright A \mid b \ast A
\]

\[
\mid A \parallel A \mid A \triangledown A \mid \text{undo} \mid \{A? A, A\}
\]
The Syntax of BPEL

\[ BA ::= \text{skip} | x := e | \text{rec } a x | \text{rep } a x | \text{throw} \]

\[ A ::= BA | g \circ A | b \triangleright l | A; A | A \triangleleft b \triangleright A | b * A \]

\[ | A \parallel A | A \sqcap A | \text{undo} | \{ A ? A, A \} \]

1 Local Variables
The Syntax of BPEL

\[ BA ::= \text{skip} \mid x := e \mid \text{rec} \ a \ x \mid \text{rep} \ a \ x \mid \text{throw} \]

\[ A ::= BA \mid g \circ A \mid b \triangleright l \mid A; A \mid A \triangleleft b \triangleright A \mid b \star A \]
\[ \mid A \parallel A \mid A \sqcap A \mid \text{undo} \mid \{ A ? A, A \} \]

1. Local Variables
2. Shared Labels
The Syntax of BPEL

\[ BA ::= \text{skip} \mid x := e \mid \text{rec } a \ x \mid \text{rep } a \ x \mid \text{throw} \]

\[ A ::= BA \mid g \circ A \mid b \triangleright i \mid A; A \mid A \triangleleft b \triangleright A \mid b \triangleright A \]
\[ \mid A \parallel A \mid A \sqcap A \mid \text{undo} \mid \{ A? A, A \} \]

1. Local Variables
2. Shared Labels
3. Scope-based compensation
The Syntax of BPEL

\[ BA ::= \text{skip} \mid x := e \mid \text{rec}\ a\ x \mid \text{rep}\ a\ x \mid \text{throw} \]

\[ A ::= BA \mid g \circ A \mid b \triangleright l \mid A; A \mid A \triangleleft b \triangleright A \mid b \ast A \]

\[ \mid A \parallel A \mid A \sqcap A \mid \text{undo} \mid \{ A ? A, A \} \]

1. Local Variables
2. Shared Labels
3. Scope-based compensation
4. Fault Handler
Guarded Choices

For exploring the parallel expansion laws, our language is enriched with the concept of guarded choice, expressed in the form:

\[ \{h_1 \rightarrow P_1\} \parallel \ldots \parallel \{h_n \rightarrow P_n\} \]

where:

1. \(h_i\) can be a skip guard, expressed as \(b_i \& \text{skip}\), where \(b_i\) is a Boolean expression and satisfies the condition “\(\lor_i b_i = \text{true}\)”.

2. \(h_i\) can also be a communication guard, expressed as \(\text{rec } a x\) and \(\text{rep } a x\).

3. \(h_i\) can also be a Boolean guard \(\@ (g_i)\); i.e., waiting for \(g_i\) to be fired.
Maude is a high-performance implementation of rewriting logic as well as the underlying MEL sublogic. Its syntax is so simple that it is almost identical with mathematical notations. And with rewriting logic as its fundamental logic, it is very expressive, specifying semantics of programming languages and modeling concurrent systems.
Basic Commands:

sorts Assignment UpdateOfLabel Channel ThrowCommand UndoCommand .
sort PrimitiveCommand .
subsort Assignment AssignmentOfLabel Channel < PrimitiveCommand .
subsort ThrowCommand UndoCommand < PrimitiveCommand .
op _ /> _ : BoolExp Qid -> UpdateOfLabel [ctor prec 1] .
op rec _ _ : Qid Qid -> Channel [ctor prec 1] .
op rep _ _ : Qid Qid -> Channel [ctor prec 1] .
op throw : -> ThrowCommand .
op undo : -> UndoCommand .
And other syntax constructions are defined as in most programming languages.

```plaintext
fmod PROGRAM is ...
subsort PrimitiveCommand < Program .
op nil special fault : -> Program [ctor] .
        ---- Nondeterministic Choice
op _ ; _ : Program Program -> Program [ctor gather (e E) prec 40] .
op if _ then _ else _ : BoolExp Program Program -> Program [ctor prec 20] .
endfm
```
Outline

1. Introduction
2. BPEL and Rewriting Logic
3. Algebraic Semantics and Head Normal Form
4. Generation of Operational Semantics from Algebraic Semantics
5. Linking Theories of the Semantics
6. Conclusion and Future work
The first form stands for the general guarded choice, as mentioned above.

sorts LabelGuard Guard .
subsort Channel LabelGuard < Guard .
op _&skip : BoolExp -> Guard [ctor].
op @( _) : ParterLink -> LabelGuard [ctor] .

subsort GuardedChoice < NormalForm .
op _ => _ : Guard Program -> GuaredChoice [ctor] .
op $ : -> GuardedChoice [ctor] .
op _ [[]]_ : GuardedChoice GuaredChoice -> GuardedChoice [ctor assoc
comm id: $]
(form-2) \( X \rightarrow P \)

where \( X \) can be of the form \( assign(x, e) \), \( updatelabel(l, b) \) or \( compen(C) \), which we introduce to represent actions of BPEL programs except for the communication actions.

\[
\begin{align*}
\text{op assign(\_, \_)} & : \text{Qid Exp} \rightarrow \text{Head [ctor]} . \\
\text{op updatelabel(\_, \_)} & : \text{Qid BoolExp} \rightarrow \text{Head [ctor]} . \\
\text{op compen(\_)} & : \text{Program} \rightarrow \text{Head [ctor]} . \\
\text{op \_ \_Rightarrow \_} & : \text{Head Program} \rightarrow \text{NormalForm [ctor]} .
\end{align*}
\]
(form-3) \textit{throw}
subsort \texttt{ThrowCommand} < \texttt{NormalForm} .

(form-4) \textit{undo}
subsort \texttt{UndoCommand} < \texttt{NormalForm} .
For sequential composition,

\[ e_1 \{ h \Rightarrow P \} ; Q = \{ h \Rightarrow (P ; Q) \} . \]

\[ e_2 (\{ h \Rightarrow P \} [] GC) ; Q = (\{ h \Rightarrow P \} ; Q) [] (GC ; Q) . \]

\[ e_3 (X \Rightarrow P) ; Q = X \Rightarrow (P ; Q) . \]

\[ e_4 \text{throw} ; P = \text{throw} . \]

\[ e_5 \text{undo} ; P = \text{undo} . \]

For conditional branching and iteration,

\[ e_6 \text{if b then P else Q} = \{ b &\text{skip} \Rightarrow P \} [] \{ !b &\text{skip} \Rightarrow Q \} . \]

\[ e_7 \text{while b do P} = \{ b &\text{skip} \Rightarrow (P ; \text{while b do P}) \} [] \{ !b &\text{skip} \Rightarrow \text{nil} \} . \]

\[^{1}G_1 \ G_2 \ \text{GC} \ \text{GC}_1 \ \text{GC}_2 \ \text{are all variables of sort GuardedChoice in this paper.}\]
Scope-based compensation and fault handling mechanism are the core features of BPEL, so we describe them here specially. First we introduce the action called $compen(C)$, installing the program $C$ into the compensation list.

$$(\text{scope-1}) \ {A \ ? \ C, F} = df \ (A; \ compen(C)) \circ F$$
For scope structure,

\[ \{ h \Rightarrow P \} \circ Q = \{ h \Rightarrow (P \circ Q) \} . \]
\[ (\{ h \Rightarrow P \} \; [\;] \; GC) \circ Q = (\{ h \Rightarrow P \} \circ Q) \; [\;] \; (GC \circ Q) . \]
\[ (X \Rightarrow P) \circ Q = X \Rightarrow (P \circ Q) . \]
\[ \text{throw} \circ P = P . \]
\[ \text{undo} \circ P = \text{undo} . \]
The four typical forms can describe most BPEL programs except for the ones containing nondeterministic choices. In order to include nondeterministic choices, we introduce a summation of normal forms. The basic idea is to distribute current possible choices to the front, summing up by the summation operator.
Here we define the Summation,

sort Summation .
subsort Program < Summation .

eq (P (•) Q) ; R = P ; R (•) Q ; R .
eq (P (•) S1) ; R = (P ; R) (•) (S1 ; R) [owise] .
eq (P (•) Q) ° R = (P ° R) (•) (Q ° R) .
eq (P (•) S1) ° R= (P ° R) (•) (S1 ° R) [owise] .
With NormalForm and Summation, we are able to translate all BPEL programs to the two kinds of forms. And this is the essence of the head normal form and the algebraic semantics. We combine them together as sort HeadNormalForm. Operator HF is declared to compute head normal forms of programs.

```
sort HeadNormalForm .
subsort NormalForm Summation < HeadNormalForm .
op HF(_) : Program -> HeadNormalForm .
```
Take some simple examples,
\[\text{eq } \text{HF}(x := e) = \{t \& \text{skip} \Rightarrow \text{assign}(x,e)\} .\]
\[\text{eq } \text{HF}(\text{rec } a \ x) = \{\text{rec } a \ x \Rightarrow \text{nil}\} .\]
\[\text{eq } \text{HF}(\text{rep } a \ x) = \{\text{rep } a \ x \Rightarrow \text{nil}\} .\]
\[\text{eq } \text{HF}(\text{throw}) = \text{throw} .\]
\[\text{eq } \text{HF}(\text{undo}) = \text{undo} .\]
First, take this parallel program as an example,

\[
(\text{rec } \textit{a } \textit{x} ; \text{ if } \textit{x} < 0 \text{ then } \textit{x} := -\textit{x} \text{ else } \textit{x} := \textit{x} + 1 ; \ \text{rep } \textit{a } \textit{x}) \ || \\
(\text{rep } \textit{a } \textit{y} ; \ \text{rec } \textit{a } \textit{y}).
\]

Maude> red HF(
(rec 'a 'x ; if 'x < 0 then 'x := (- 'x) else 'x := ('x + 1); rep 'a 'x) || (rep 'a 'y ; rec 'a 'y)) .

result GuardedChoice:
{t &skip => 'x:='y ; (if 'x<0 then 'x:=(-'x) else 'x:=(’x+1) ; rep’a’x) || rec’a’y)}
Outline

1. Introduction
2. BPEL and Rewriting Logic
3. Algebraic Semantics and Head Normal Form
4. Generation of Operational Semantics from Algebraic Semantics
5. Linking Theories of the Semantics
6. Conclusion and Future work
Since every BPEL program can be represented by the Head Normal Form, we can define operational semantics based on the head normal forms. We call the definition Derivation Strategy.

crl [0]::< P,sigma,label,cpens > => {tau}::< Pi,sigma,label,cpens >
if Pi(*)S1 := HF(P) .
For Guarded Choice, the first kind of the four typical forms,
crl \[a1\] : \langle P, \sigma, \text{label}, \text{cpens} \rangle \Rightarrow \{c\} \langle \Pi, \sigma, \text{label}, \text{cpens} \rangle

\[
\text{if } \{b \& \text{skip } \Rightarrow \Pi\} [] G := \text{HF}(P) \land b[\sigma].
\]
crl \[a2\] : \langle P, \sigma, \text{label}, \text{cpens} \rangle \Rightarrow \{c\} \langle R, \sigma, \text{label}, \text{cpens} \rangle

\[
\text{if } \{@(g) \Rightarrow R\} [] G := \text{HF}(P) \land g(\text{label}).
\]
crl \[a3\] : \langle P, \sigma, \text{label}, \text{cpens} \rangle

\[
\Rightarrow \{x \& a ! (\sigma).x\} \langle R, \sigma, \text{label}, \text{cpens} \rangle
\]

\[
\text{if } \{\text{rep } a x \Rightarrow R\} [] G := \text{HF}(P).
\]
crl \[a4\] : \langle P, \sigma, \text{label}, \text{cpens} \rangle

\[
\Rightarrow \{x @ a ? i\} \langle Q, \sigma \leftarrow (x,i), \text{label}, \text{cpens} \rangle
\]

\[
\text{if } \{\text{rec } a x \Rightarrow Q\} [] G := \text{HF}(P).
\]
Derivation Strategy

For the other three typical forms,

\[ \text{crl} \text{[b]} : \langle P, \sigma, \text{label}, \text{cpens} \rangle \Rightarrow \{c\} \langle P', \sigma \leftarrow (x,e), \text{label}, \text{cpens} \rangle \]

\[ \text{if assign}(x,e) \Rightarrow P' := \text{HF}(P) . \]

\[ \text{crl} \text{[c]} : \langle P, \sigma, \text{label}, \text{cpens} \rangle \Rightarrow \{v\} \langle P', \sigma, \text{label} \leftarrow [l,b], \text{cpens} \rangle \]

\[ \text{if assignlabel}(l,b) \Rightarrow P' := \text{HF}(P) . \]

\[ \text{crl} \text{[d]} : \langle P, \sigma, \text{label}, \text{cpens} \rangle \Rightarrow \{c\} \langle P', \sigma, \text{label}, \text{cpens} \uparrow C \rangle \]

\[ \text{if compen}(C) \Rightarrow P' := \text{HF}(P) . \]

\[ \text{crl} \text{[e]} : \langle P, \sigma, \text{label}, \text{cpens} \rangle \Rightarrow \{c\} \langle \text{fault}, \sigma, \text{label}, \text{cpens} \rangle \]

\[ \text{if throw} := \text{HF}(P) . \]

\[ \text{crl} \text{[f1]} : \langle P, \sigma, \text{label}, Y \uparrow X \rangle \Rightarrow \{c\} \langle X, \text{undo}, \sigma, \text{label}, Y \rangle \]

\[ \text{if undo} := \text{HF}(P) . \]

\[ \text{crl} \text{[f2]} : \langle P, \sigma, \text{label}, \text{cpens} \rangle \Rightarrow \{c\} \langle \text{special}, \sigma, \text{label}, \text{empty} \rangle \]

\[ \text{if undo} := \text{HF}(P) . \]
Outline

1 Introduction

2 BPEL and Rewriting Logic

3 Algebraic Semantics and Head Normal Form

4 Generation of Operational Semantics from Algebraic Semantics

5 Linking Theories of the Semantics

6 Conclusion and Future work
For Sequential Composition,
crl [Sequential-1] :
\[
\langle P;Q,\sigma,\text{label},\text{cpens} \rangle \Rightarrow \{\text{act}\} \langle Q,\sigma',\text{label}',\text{cpens}' \rangle
\]
if \-------------------------------------------------------------
\[
\langle P,\sigma,\text{label},\text{cpens} \rangle \Rightarrow \{\text{act}\} \langle \text{nil},\sigma',\text{label}',\text{cpens}' \rangle.
\]
crl [Sequential-2] :
\[
\langle P;Q,\sigma,\text{label},\text{cpens} \rangle \Rightarrow \{\text{act}\} \langle P';Q,\sigma',\text{label}',\text{cpens}' \rangle
\]
if 
-------------------------------------------------------------------
\[
\langle P,\sigma,\text{label},\text{cpens} \rangle \Rightarrow \{\text{act}\} \langle P',\sigma',\text{label}',\text{cpens}' \rangle \land P' \neq \text{nil} \land P' \neq \text{special} \land P' \neq \text{fault}.
\]
For Parallel Processes with nondeterministic choices,

crl [Parallel-1-1] :
< P||Q,sigma,label,cpens >=>{tau}< par(P’,Q’),sigma,label,cpens >
if

< P,sigma,label,cpens >=> {tau}< P’,sigma,label,cpens > /
< Q,sigma,label,cpens >=> {tau}< Q’,sigma,label,cpens >.


crl [Parallel-3] :
< P||Q,sigma,label,cpens >=> {c}< x := y ;
par(P’,Q’),sigma,label,cpens >
if

< P,sigma,label,cpens >=> {x @ a ? m}< P’,sigma’,label,cpens > /
< Q,sigma,label,cpens >=> {y & a ! m}< Q’,sigma,label,cpens >.
Now we study the mechanical proof of the equivalence of the derivation strategy and the derived operation semantics. Theoretically we have two theorems to be proved.

(1) If transition \( aa \) exists in the transition system of the derived operational semantics, then it also exists in the derivation strategy.
(2) If transition \( bb \) exists in the derivation strategy, then it also exists in the transition system of the derived operation semantics.
We have proven the equivalence of the two forms of operational semantics by hand, but we want to do the proof automatically. So we are trying the ITP (interactive theorem prover) implemented in Maude.
1 Introduction

2 BPEL and Rewriting Logic

3 Algebraic Semantics and Head Normal Form

4 Generation of Operational Semantics from Algebraic Semantics

5 Linking Theories of the Semantics

6 Conclusion and Future work
There have been a lot of works done on the semantics of BPEL, in different formal models, such as the Petri Net, and the π-calculus. Our work applied a different perspective of the study of semantics of BPEL. But compared to the great works above, our work only handles a small subset of the full semantics of BPEL.
Future work

1. **Enhancement** Try to support more of the syntax and semantics of BPEL

2. **Mechanical Proof** Mechanical proof of the equivalence of the two forms of operational semantics
Thanks very much! It’s such a pleasure attending the WS-FM 2012 in Tallinn!