Innovations Syst Softw Eng (2011) 7:209–224
DOI 10.1007/s11334-011-0172-1

ORIGINAL PAPER

Algebraic approach to linking the semantics of web services

Huibiao Zhu · Jifeng He · Jing Li · Jonathan P. Bowen

Received: 25 May 2009 / Accepted: 5 September 2011 / Published online: 7 October 2011

Abstract Web services have become more and more important in these years, and BPEL4WS (BPEL) is a de facto standard for the web service composition and orchestration. It contains several distinct features, including the scope-based compensation and fault handling mechanism. We have considered the operational semantics and denotational semantics for BPEL, where a set of algebraic laws can be achieved via these two models, respectively. In this paper, we consider the inverse work, deriving the operational semantics and denotational semantics from algebraic semantics for BPEL. In our model, we introduce four types of typical programs, by which every program can be expressed as the summation of these four types. Based on the algebraic semantics, the strategy for deriving the operational semantics is provided and a transition system is derived by strict proof. This can be considered as the soundness exploration for the operational semantics based on the algebraic semantics. Further, the equivalence between the derivation strategy and the derived transition system is explored, which can be considered as the completeness of the operational semantics. Finally, the derivation of the denotational semantics from algebraic semantics is explored, which can support to reason about more program properties easily.

Keywords Web services · BPEL · Algebraic semantics · Operational semantics · Denotational semantics · Semantic linking

A short version of this paper appeared in SEFM 2007: 5th IEEE International Conference on Software Engineering and Formal Methods [53].

H. Zhu (✉) · J. He · J. Li
Shanghai Key Laboratory of Trustworthy Computing,
East China Normal University, 3663 Zhongshan Road (North),
Shanghai 200062, China
e-mail: hbzhu@sei.ecnu.edu.cn

J. He
e-mail: jifeng@sei.ecnu.edu.cn

J. Li
e-mail: jli@sei.ecnu.edu.cn

J. P. Bowen
Museophile Limited, Oak Barn, Sonning Eye,
Reading RG4 6TN, UK
e-mail: jpbowen@gmail.com
URL: http://www.jpbowen.com

1 Introduction

Web services and other web-based applications have been becoming more and more important in practice. In this blooming field, various web-based business process languages have been introduced, such as XLANG [46], WSFL [30], BPEL4WS (BPEL) [11] and StAC [7], which are designed for the description of services composed of a set of processes across the Internet. Their goal is to achieve the universal interoperability between applications using web standards, as well as to specify the technical infrastructure for carrying out business transactions.

The important feature of BPEL is that it supports the long-running interactions involving two or more parties. Therefore, it provides the ability to define fault and compensation handing in application-specific manner, resulting in a feature called *Long-Running* (*Business*) *Transactions* (*LRTs*). The concept of *compensation* is due to the use of Sagas [4,15] and open nested transactions [39]. The fault analysis has been considered in [41,43] for compensation processes. It addition, BPEL provides two kinds of synchronization techniques for parallel processes. In our model, shared-labels are introduced for the synchronization between a process and its partners within a single service, while channel communications are introduced for message transmission between different services.

As advocated in Hoare and He's Unifying Theories of Programming (*UTP*) [25], three different styles of mathematical representation are normally used for a programming language: operational, denotational and algebraic one [23,40,45]. Moreover, the linking theories [48] between different semantic models for a language are particularly interesting, which can provide the correct understanding for one semantics based on the viewpoint of another. For BPEL, we have already explored the denotational semantics [20] and operational semantics [54], where a set of algebraic laws has been explored based on the two formal semantics, respectively. This paper considers the inverse work, deriving the operational semantics and denotational semantics from algebraic semantics.

In order to support the linking from the algebraic semantics to the operational and denotational semantics for BPEL, we introduce four types of typical programs. Based on this, the summation of these typical programs is introduced, which is used to construct the head normal form for a program. The construction is based on the algebraic laws of BPEL. The derivation of operational semantics from algebraic semantics is based on the head normal form. The derivation strategy is defined and a set of transition rules is achieved based on the derivation strategy by strict proof. This can be regarded as the soundness consideration [49] of the operational semantics from the viewpoint of algebraic semantics. However, the derivation strategy may derive more transitions, compared with the strict derived transition system. The equivalence between the derivation strategy and the derived transition system is worthy of exploration. This is the completeness consideration [49] for the operational semantics from the viewpoint of algebraic laws. Further, the derivation of denotational semantics from algebraic viewpoint is also explored based on the head normal from, which can support to reason about program properties easily, especially for parallel programs.

The remainder of this paper is organized as follows. Section 2 introduces the language BPEL. We also explore the algebraic laws in this section. Four types of typical programs are introduced and the concept of head normal form is defined. Section 3 is devoted to the derivation of operational semantics from the algebraic semantics. The derivation strategy is defined and a set of transition rules is derived by strict proof. Further, we also explore the equivalence of the derivation strategy and the derived operational semantics in this section. Section 4 studies the derivation of denotational semantics from algebraic semantics. The approach is based on the head normal form as well. Section 5 discusses the related work about web services and semantic linking. Section 6 concludes the paper.

2 Algebraic semantics

2.1 The syntax of BPEL

BPEL is used to model business transactions for web services. Below is the BPEL model that we have proposed, which contains several interesting features, such as scope-based compensation and fault handling mechanism. Our language contains the following categories of syntactic elements:

$$\begin{aligned} BA &::= \texttt{skip} \mid x := e \mid \texttt{rec}\ a\ x \mid \texttt{rep}\ a\ x \mid \texttt{throw} \\ A &::= BA \mid @(g)\,A \mid b \rhd l \mid A;\ A \mid A \lhd b \rhd A \\ &\quad\ \mid b * A \mid A \parallel A \mid A \sqcap A \mid \texttt{undo} \\ &\quad\ \mid \{A\,?\,A,\ A\} \end{aligned}$$

where

- $x := e$ is the assignment, which assigns the value of e to local variable x. `skip` behaves the same as $x := x$. The activity `throw` indicates that the program encounters a fault immediately. In order to implement the communications between different services, two statements are introduced; i.e., $\texttt{rec}\ a\ x$ and $\texttt{rep}\ a\ x$. Command $\texttt{rec}\ a\ x$ represents the receiving of a value through channel a. The received value is stored in variable x. $\texttt{rep}\ a\ x$ represents the output of value of x via channel a.
- Several constructs are similar to those in traditional programming languages. $A; B$ stands for sequential composition. $A \lhd b \rhd B$ is the conditional construct and $b * A$ is the iteration construct. $A \sqcap B$ stands for the nondeterministic choice; i.e., either A or B could be executed.
- Shared-labels are introduced to implement the synchronization between a process and its partner. For $@(g)A$, if guard g is evaluated as true, control flow can pass to process A. Otherwise, it will wait for guard g to be set true. Here g is composed of a set of source links; i.e., a set of read only labels. These source links can be considered as the target links of other components; i.e., updatable labels. Besides, $b \rhd l$ assigns the value of b to label l.
- $\{A\,?\,C,\ F\}$ stands for the scope activity, where A, C and F stand for the primary activity, compensation program and fault handler correspondingly. If A terminates successfully, program C is installed in the compensation list for later compensating. On the other hand, if A encounters a fault during its execution, the fault handler F will be activated. Further, the compensation part C does not contain scope activity. For statement "`undo`", it activates the execution of the programs in the compensation list under the reverse order of their installed sequence.
- $A \parallel B$ stands for the parallel composition. For a shared label, it is owned by one parallel component; i.e., it can only be written by one parallel component. However, it can be read by all other parallel components. Further, the

communication mechanism via channels obeys the same rules as those in process algebra CSP [22], which is used to model message transmission between different services. Moreover, both of the two parallel components do not have `undo` statement.

In order to support parallel expansion laws, we enrich our language with the concept of guarded choice, expressed in the form:

$\{h_1 \to P_1\}[] \ldots\ldots [] \{h_n \to P_n\}$

where

(1) h_i can be a `skip` guard, expressed as b_i&`skip`, where b_i is a Boolean expression. We assume that the disjunction of all b_i in a guarded choice is always satisfied (i.e., $\vee_i b_i = true$).
(2) h_i can also be a communication guard, expressed as `rec` $a\ x$ and `rep` $a\ x$, where a is the channel name.
(3) h_i can also be expressed as $@(g_i)$; i.e, waiting for Boolean guard g_i to be set true.

2.2 Algebraic semantics of BPEL

In order to explore the head normal form, we introduce four typical forms. The definition of head normal form can be expressed as one of the four forms. The first form stands for the general guarded choice, as mentioned above. For each guarded component, the corresponding guard can be of `skip` guard, event guard or communication guard.

(form-1) $[]_{i\in I}\{b_i\&\mathtt{skip} \to P_i\}$
$[][]_{j\in J}\{@(g_j) \to Q_j\}$
$[][]_{l\in L}\{\mathtt{comm}\ \ a_l x_l \to R_l\}$

where, `comm` can be `rec` or `rep`.

The second form represents for the behaviour which contains two parts sequentially, where the first part can be of the form $assign(x, e)$, which stands for the assignment of local variables or the update of shared-labels. It can also be $compen(C)$, which stands for the behaviour of installing the compensation program C.

(form-2) $X \to P$

where X can be of the form $assign(x, e)$ or $compen(C)$.

The third and fourth form are special, which are simply represented by `throw` and `undo` commands.

(form-3) `throw`
(form-4) `undo`

In order to deal with the scope behaviour, we introduce a new operator $\circ$. The subsequent behaviour after $\circ$ stands for the fault handler. For $A \circ B$, if A terminates successfully, the whole program also terminates successfully. However, if A encounters fault during its execution, the fault hander B will be activated. Based on the understanding of $compen(C)$, a scope activity can be defined as below. The scope activity can be explained via the newly introduced command $compen(C)$ and the $\circ$ operator.

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

Now we consider the laws for sequential composition. Firstly we introduce a function **seq1**:

$$\mathbf{seq1}(P, Q) =_{df} \begin{cases} Q & \text{if } P = II \\ \mathtt{throw} & \text{if } P = \mathtt{throw} \\ P; Q & \text{otherwise} \end{cases}$$

where, II stands for the empty process.

Sequential composition satisfies the following algebraic laws. We study the laws for the cases that the first component of sequential composition is one of the four typical forms

(seq-1) $[]_{i\in I}\{h_i \to P_i\};\ R = []_{i\in I}\{h_i \to \mathbf{seq1}(P, R)\}$
(seq-2) $(X \to P);\ R = X \to \mathbf{seq1}(P, R)$
where, $X = compen(C) or\ assign(x, e)$
(seq-3) $X;\ R = X$
where, $X = \mathtt{throw}\ \ or\ \ X = \mathtt{undo}$.

The first two laws indicate that sequential composition distributes backward through the first and second type of typical forms. The third law (seq-3) indicates that `throw` (or `undo`) is the left zero of sequential composition.

Next, we consider the laws for the newly added operator $\circ$. Firstly, we introduce the function **seq2**:

$$\mathbf{seq2}(P, Q) =_{df} \begin{cases} II & \text{if } P = II \\ Q & \text{if } P = \mathtt{throw} \\ P \circ Q & \text{otherwise} \end{cases}$$

Operator $\circ$ satisfies a set of algebraic laws shown below. For the following laws, the first component of $\circ$ can also be one of the four typical forms.

(circ-1) $[]_{i\in I}\{h_i \to P_i\} \circ R = []_{i\in I}\{h_i \to \mathbf{seq2}(P, R)\}$
(circ-2) $(X \to P) \circ R = X \to \mathbf{seq2}(P, R)$
where, $X = compen(C) or assign(x, e)$
(circ-3) $\mathtt{throw} \circ \mathtt{R} = \mathtt{R}$
(circ-4) $\mathtt{undo} \circ \mathtt{R} = \mathtt{undo}$

Similarly, the first and second law indicates that operator $\circ$ distributes backwards through the first and second type of typical forms. The third law indicates that fault handler can be immediately fired when an error is encountered. This law also indicates that `throw` is the unit of operator $\circ$. The fourth law indicates that `undo` is the left zero of operator $\circ$.

Now we introduce a new construct called summation. It is denoted as as $\bigoplus\{P_1, \ldots, P_n\}$, where each P_i is initially deterministic. The selection among all P_i is nondeterministic. Further, the elements in a summation can be re-arranged, shown as the law below.

$$\left(\bigoplus -1\right) \bigoplus\{P_1, \ldots, P_n\} = \bigoplus\{P_{i_1}, \ldots, P_{i_n}\},$$
where $i_1, \ldots, i_n$ is a permutation of $\{1, \ldots, n\}$.

Now we consider the transformation of nondeterministic choice to the form of summation.

(nonde-1) If $P = \bigoplus\{P_1, \ldots, P_n\}$ and
$$Q = \bigoplus\{Q_1, \ldots, Q_m\}$$
then $P \sqcap Q = \bigoplus\{P_1, \ldots, P_n, Q_1, \ldots, Q_m\}$

The above two laws can assist the transformation of every program into a summation of a set of initially deterministic processes. Next we consider the laws for conditional and iteration:

(cond-1) $P \lhd b \rhd Q = [\!]\{b\&\mathtt{skip} \rightarrow P,\ \neg b\&\mathtt{skip} \rightarrow Q\}$

(iter-1) $b * P = [\!]\{b\&\mathtt{skip} \rightarrow (P; b * P),$
$\neg b\&\mathtt{skip} \rightarrow II\}$

The above two laws indicate that conditional and iteration can be expressed in the form of guarded choice via the `skip` guarded components.

Now we consider the algebraic laws for parallel composition. Let

$$\mathbf{par}(P, Q) =_{df} \begin{cases} Q & \text{if } P = II \\ P & \text{if } Q = II \\ P \parallel Q & \text{otherwise} \end{cases}$$

Parallel composition is symmetric; i.e., $P \parallel Q = Q \parallel P$. Our further exploration is based on the four typical forms.

For parallel composition, one case is that two parallel components are within one service. This indicates that there is no message communication between two parallel components; i.e., the two components do not share the same communication channels. Law (para-1) reflects this case.

(para-1) Let $P = [\!]_{i\in I}\{h_i \rightarrow P_i\}$ and
$$Q = [\!]_{j\in J}\{g_j \rightarrow Q_j\}.$$

Assume that P and Q do not have the same communication channels. Then

$$\begin{aligned} &P \parallel Q \\ &= [\!]_{i\in I}\{h_i \rightarrow \mathbf{par}(P_i, Q)\}[\!][\!]_{j\in J}\{g_j \rightarrow \mathbf{par}(P, Q_j)\}\end{aligned}$$

Two parallel components can be from different services. The parallel composition of sending and receiving messages via the same channel can be transformed into a local variable assignment. Law (para-2) reflects this case. The message communication can be taken place via the common channels c_m $(m \in M)$.

(para-2) Let $P1 = [\!]_{i\in I}\{h_i \rightarrow P1_i\}$,
$$\begin{aligned} Q1 &= [\!]_{j\in J}\{g_j \rightarrow Q1_j\}, \\ P &= P1[\!][\!]_{m\in M}\{\mathtt{rec}\ c_m\ u_m \rightarrow R_m\}, \\ Q &= Q1[\!][\!]_{m\in M}\{\mathtt{rep}\ c_m\ v_m \rightarrow T_m\},\end{aligned}$$

Assume that $P1$ and $Q1$ do not share the same communication channels. Then

$$\begin{aligned} &P \parallel Q \\ &= [\!]_{i\in I}\{h_i \rightarrow (P1_i \parallel Q)\} \\ &\quad [\!][\!]_{j\in J}\{g_j \rightarrow (P \parallel Q1_j)\} \\ &\quad [\!][\!]_{m\in M}\{\mathbf{true}\&\mathtt{skip} \rightarrow (u_m := v_m \rightarrow (R_m \parallel T_m))\}\end{aligned}$$

Law (para-3) studies the case that one parallel component is in the first typical form and another component is in the second typical form.

(para-3) Let $P = [\!]_{i\in I}\{h_i \rightarrow P_i\}$ and $Q = X \rightarrow Q'$, where X can be of $compen(C)$ or $assign(x, e)$.
Then

$$\begin{aligned} &P \parallel Q \\ &= [\!]\{h_i \rightarrow \mathbf{par}(P_i, Q)\} \\ &\quad [\!][\!]\{\mathbf{true}\&\mathtt{skip} \rightarrow (X \rightarrow \mathbf{par}(P, Q'))\}\end{aligned}$$

For (para-3), any guard h_i $(i \in I)$ can be scheduled. On the other hand, $compen(C)$ (or $assign(x, e)$) can also have chance to be scheduled. As $compen(C)$ (or $assign(x, e)$) does not appear in the form of guarded choice, `skip` guard is applied.

(para-4) $P \parallel \mathtt{throw} = \mathtt{throw}$

Law (para-4) indicates that `throw` is the zero of parallel composition.

(para-5) Let $P = X \rightarrow P'$ and $Q = Y \rightarrow Q'$.

Then

$$\begin{aligned} &P \parallel Q \\ &= [\!]\{\ \mathbf{true}\&\mathtt{skip} \rightarrow (X \rightarrow P' \parallel Q), \\ &\qquad \mathbf{true}\&\mathtt{skip} \rightarrow (Y \rightarrow P \parallel Q')\}\end{aligned}$$

where, $X = assign(x, e)$ or $compen(C)$, and $Y = assign(y, f)$ or $compen(D)$.

Law (para-5) stands for the case that both of the two parallel components belong to the second type of typical forms.

2.3 Head normal form

Now we explore the head normal form for BPEL programs. Our later discussion about deriving operational semantics and denotational semantics is based on the head normal form. Head normal form is expressed in the form of one step forward expansion; i.e., its definition is based on the four typical forms.

Assignment $x := e$ can be expressed as a guarded choice comprising of a single `skip` guarded component. The subsequent behaviour after the `skip` guard is expressed as $assign(x, e)$, which purely assigns value e to variable x.

$$\mathcal{HF}(x := e) =_{df} [] \{\textbf{true}\&\texttt{skip} \rightarrow assign(x, e)\} \tag{1}$$

For communication guard, it can be expressed as a guarded choice comprising of a single communication guard component.

$$\begin{aligned}\mathcal{HF}(\texttt{rec}\ a\ x) &=_{df} [] \{\texttt{rec}\ a\ x \rightarrow II\}\\ \mathcal{HF}(\texttt{rep}\ a\ x) &=_{df} [] \{\texttt{rep}\ a\ x \rightarrow II\}\end{aligned} \tag{2}$$

For statement `throw` and `undo`, their head normal forms are themselves. The head normal forms for event guard and label update are similar to communicating statement and local variable assignment, respectively.

$$\mathcal{HF}(X) =_{df} X, \text{ where, } X = \texttt{throw} \text{ or } \quad X = \texttt{undo} \tag{3}$$

$$\mathcal{HF}(@(g)\,A) =_{df} [] \{@(g) \rightarrow A\} \tag{4}$$

$$\mathcal{HF}(b \triangleright l) =_{df} [] \{\textbf{true}\&\texttt{skip} \rightarrow assign(l, b)\} \tag{5}$$

Now we consider the head normal form for $A; B$ and $A \circ B$. The definition is based on the four typical forms for the head normal form of A. If the head normal form of A is expressed as a guarded choice, the head normal form of $A; B$ can also be expressed as a guarded choice comprising with the same set of guards. The subsequent behaviour after the guard can be expressed as the sequential combination of the corresponding subsequent behaviour of A and B via function **seq1**. Similarly, the corresponding behaviour after the guard for $A \circ B$ is via function **seq2**.

Meanwhile, if A can be expressed as the sequential behaviour whose initial behaviour is $assign(x, e)$ (or $compen(C)$), then $A; B$ (or $A \circ B$) can also be expressed as the sequential behaviour. The subsequent behaviour can be expressed using function **seq1** and **seq2**, respectively. Moreover, if the head normal form of A is `throw`, the head normal form of $A; B$ is also `throw`. In this case, the head normal form of $A \circ B$ is B. If the head normal form of A is `undo`, the head normal forms for $A; B$ and $A \circ B$ are also `undo`.

$$\begin{aligned}
&\text{If} \quad \mathcal{HF}(A) = []_{i \in I}\{h_i \rightarrow A_i\},\\
&\text{then} \quad \mathcal{HF}(A; B) = []_{i \in I}\{h_i \rightarrow \textbf{seq1}(A_i, B)\}\\
&\qquad\quad \mathcal{HF}(A \circ B) = []_{i \in I}\{h_i \rightarrow \textbf{seq2}(A_i, B)\}\\
&\text{If}\, \mathcal{HF}(A) = X \rightarrow A',\\
&\text{then} \quad \mathcal{HF}(A; B) = X \rightarrow \textbf{seq1}(A', B)\\
&\qquad \mathcal{HF}(A \circ B) = X \rightarrow \textbf{seq2}(A', B)\\
&\text{where, } X = assign(x, e) or X = compen(C)\\
&\text{If } \mathcal{HF}(A) = \texttt{throw}, \text{ then } \mathcal{HF}(A; B) = \texttt{throw}\\
&\qquad\qquad\qquad\qquad\qquad \mathcal{HF}(A \circ B) = B\\
&\text{If } \mathcal{HF}(A) = \texttt{undo}, \quad \text{then } \mathcal{HF}(A; B) = \texttt{undo}\\
&\qquad\qquad\qquad\qquad\qquad \mathcal{HF}(A \circ B) = \texttt{undo}
\end{aligned} \tag{6}$$

For conditional and iteration, their head normal form can be expressed as a guarded choice comprising of `skip` guarded components.

$$\begin{aligned}
\mathcal{HF}(P \triangleleft b \triangleright Q) &=_{df} [] \{b\&\texttt{skip} \rightarrow P,\ \neg b\&\texttt{skip} \rightarrow Q\}\\
\mathcal{HF}(b * P) &=_{df} [] \{b\&\texttt{skip} \rightarrow (P; b * P),\\
&\qquad \neg b\&\texttt{skip} \rightarrow II\}
\end{aligned} \tag{7}$$

The head normal form of guarded choice is itself.

$$\text{If } P \text{ is a guarded choice, then } \mathcal{HF}(P) =_{df} P \tag{8}$$

If the head normal form of P and Q both contain a set of initially deterministic processes, The head normal form of their nondeterminism consists of all the typical forms of P and Q.

$$\begin{aligned}
&\text{If}\, \mathcal{HF}(P) = \bigoplus_{i \in I} P_i \text{ and}\, \mathcal{HF}(Q) = \bigoplus_{j \in J} Q_j,\\
&\quad \text{then}\, \mathcal{HF}(P \sqcap Q) = \bigoplus_{i \in I} P_i \bigoplus \bigoplus_{j \in J} Q_j
\end{aligned} \tag{9}$$

If the head normal form of P and Q both contain a set of initially deterministic processes, the head normal form of $P \parallel Q$ consists of the initially deterministic processes generated from the parallel composition of the typical forms from P and Q.

$$\begin{aligned}
&\text{If}\, \mathcal{HF}(P) = \bigoplus_{i \in I} P_i \quad \text{and} \quad \mathcal{HF}(Q) = \bigoplus_{j \in J} Q_j,\\
&\text{then}\, \mathcal{HF}(P \parallel Q) = \bigoplus_{i \in I, j \in J} P_i \parallel Q_j\\
&\text{where, } \mathcal{HF}(P_i \parallel Q_j) \text{is defined by applying algebraic}\\
&\text{laws (para-1) to (para-5).}
\end{aligned} \tag{10}$$

The head normal form of $\mathcal{HF}(\{A\,?\,C,\ F\})$ can be expressed as the head normal form of $\mathcal{HF}((A; compen(C)) \circ F)$.

$$\mathcal{HF}(\{A\,?\,C,\ F\}) =_{df} \mathcal{HF}((A; compen(C)) \circ F) \tag{11}$$

Moreover, we also define $\mathcal{HF}(X) =_{df} X \rightarrow II$, where X can be of the form $assign(x, e)$, $compen(C)$.

In this section, we have studied the head normal form for each statement of BPEL. For program P, if it has initially non-deterministic behaviour, its head normal form can be expressed as the summation of a set of initially deterministic processes. On the other hand, for each process P, if it is initially deterministic, its head normal form can be expressed as one of the four typical forms.

3 Deriving operational semantics from algebraic semantics for BPEL

In contrast to the standard style of defining operational semantics, this section is to derive an operational semantics for BPEL from its algebraic semantics. This approach aims to show the equivalence and consistency between the operational and algebraic semantics for BPEL.

3.1 Derivation strategy

For the operational semantics of BPEL, its transitions are written in a special notation Structural Operational Semantics (SOS) [40], which are of the four types:

$$C \xrightarrow{\tau} C' \text{ or } C \longrightarrow C' \text{ or}$$
$$C \xrightarrow{v} C' \text{ or } C \xrightarrow{a.m} C'$$

where C and C' are the configurations describing the states of an execution mechanism before and after a step, respectively. The first type models the case that a process does the nondeterministic selection. The second type is mainly used to model the assignment of a local variable, whereas the third type models the update of a shared-label. The fourth type is used to model the message communication between different services through channel a, where m stands for the message for communication.

The configuration can be expressed as $\langle P, \sigma, L, Cpens \rangle$, where

(1) The first component P is a program that remains to be executed.
(2) The second element σ is the state for all the local variables.
(3) The third element L stands for the state for all labels.
(4) The fourth element $Cpens$ stands for a compensation list; i.e., a sequence of programs to be executed as compensation.

Regarding the program P in configuration $\langle P, \sigma, L, Cpens \rangle$, it can either be a normal program. Further, it can also be one of the following special forms:

II : A program completes all its execution and terminates successfully. II is used to represent the empty program.
$\boxtimes$: A program may encounter a fault and stops at the fault state. $\boxtimes$ is used to represent the fault state.
$\boxminus$: The installed compensation programs for the current process can be activated for execution. After the termination of the compensating programs, the control will not be passed to the subsequent activity of the current process. This is the difference between the termination of the compensating programs and the termination of the current process. We use $\boxminus$ to represent the *undo* state; i.e., the terminating state for the execution of programs in the compensation list of the current process.

Now we consider the derivation strategy for deriving operational semantics from algebraic semantics.

Definition 3.1 (*Derivation Strategy*)
Let $\mathcal{HF}(P) = \bigoplus_{i \in I} P_i$.

(1) If $|I| > 1$,
then $\langle P, \sigma, L, Cpens \rangle \xrightarrow{\tau} \langle P_i, \sigma, L, Cpens \rangle$

(2) Otherwise,

(a) If $\mathcal{HF}(P)$
$= [\!]_{i \in I}\{b_i \& \texttt{skip} \rightarrow P_i\} [\!][\!]_{j \in J}\{\texttt{rec}\, a_j\, x_j \rightarrow Q_j\}$
$[\!][\!]_{k \in K}\{\texttt{repc}_k x_k \rightarrow R_k\} [\!][\!]_{l \in L}\{@(g_l) \rightarrow T_l\}$
Then
$\langle P, \sigma, L, Cpens \rangle \longrightarrow \langle P_i, \sigma, L, Cpens \rangle$, if $b_i(\sigma)$
$\langle P, \sigma, L, Cpens \rangle \xrightarrow{c_k.\sigma(x_k)} \langle R_k, \sigma, L, Cpens \rangle$
$\langle P, \sigma, L, Cpens \rangle \xrightarrow{a_j.m} \langle Q_j, \sigma[m/x_j], L, Cpens \rangle$
$\langle P, \sigma, L, Cpens \rangle \longrightarrow \langle T_l, \sigma, L, Cpens \rangle$, if $hold(g_l)$
where, $hold(g)$ is used to represent the satisfactory of the Boolean guard g.
$hold(g(l_1, l_2, \ldots, l_n))$
$=_{df}\ g(L(l_1), L(l_2), \ldots, L(l_n))$

(b) If $\mathcal{HF}(P) = assign(x, e) \rightarrow P'$,
then $\langle P, \sigma, L, Cpens \rangle \longrightarrow \langle P', \sigma[e/x], L, Cpens \rangle$

(c) If $\mathcal{HF}(P) = assign(l, b) \rightarrow P'$,
then $\langle P, \sigma, L, Cpens \rangle \xrightarrow{v} \langle P', \sigma, L[b/l], Cpens \rangle$

(d) If $\mathcal{HF}(P) = compen(C) \rightarrow P'$,
then $\langle P, \sigma, L, Cpens \rangle \longrightarrow \langle P', \sigma, L, Cpens^\frown \langle C \rangle \rangle$

(e) If $\mathcal{HF}(P) = \texttt{throw}$,
then $\langle P, \sigma, L, Cpens \rangle \longrightarrow \langle \boxtimes, \sigma, L, Cpens \rangle$

(f) If $\mathcal{HF}(P) = \texttt{undo}$,
then $\langle P, \sigma, L, Cpens \rangle \longrightarrow \langle X; \texttt{undo}, \sigma, L, Y \rangle$,
$\langle P, \sigma, L, \varepsilon \rangle \longrightarrow \langle \boxminus, \sigma, L, \varepsilon \rangle$
where, $X = \textbf{final}(Cpens)$ and
$Y = \textbf{front}(Cpens)$.

Here, **final**(s) stands for the last element of sequence s and **front**(s) stands for all but the final element of sequence s. ε stands for the empty sequence. □

If a process is expressed as a summation of at least two processes, then the process performs the nondeterministic selection and reaches to any of these processes as shown in (1). Otherwise, the process is purely expressed as one of those four typical forms.

If a process is expressed as the form of guarded choice, it can perform various transitions shown as (2a). On the other hand, if a process is expressed as the form whose initial behaviour is the local variable assignment or shared-variable update, the process can perform the corresponding transition and reach to the subsequent process. Moreover, if the initial behaviour is $compen(C)$, the process can perform the transition dealing with the installation of compensation program C.

Further, if a process is purely expressed as “throw”, it immediately reaches the fault state. On the other hand, if a process is expressed as “undo”, it firstly performs the last process in the compensation list and will "`undo`" again with respect to the compensation list except the last element.

3.2 Deriving operational semantics for BPEL by strict proof

In this section, we will derive the operational semantics for BPEL statements according to the derivation strategy. This shows the soundness of our operational semantics. The derived operational semantics is expressed as theorems to be proved (Theorems 3.1–Theorem 3.5). Theorem 3.1 considers the operational semantics for $x := e, b \rhd l, g \circ P$, communication, `throw`, `undo`, conditional and iteration. Theorem 3.2 studies the operational semantics for sequential composition. Theorem 3.3 focuses on the operational semantics for nondeterminism. The operational semantics for parallel composition is studied in Theorem 3.4. Finally, Theorem 3.5 explores the operational semantics for scope activity.

From the head normal form and the derivation strategy, we can directly achieve the transitions below, shown in Theorem 3.1.

Theorem 3.1

(1) $\langle x := e,\ \sigma,\ L,\ Cpens\rangle \longrightarrow \langle assign(x, e),\ \sigma,\ L,\ Cpens\rangle$
$\langle assign(x, e),\ \sigma,\ L,\ Cpens\rangle \longrightarrow \langle II,\ \sigma[e/x],\ L,\ Cpens\rangle$
$\langle b \rhd l,\ \sigma,\ L,\ Cpens\rangle \longrightarrow \langle assign(l, b),\ \sigma,\ L, Cpens\rangle$
$\langle assign(l, b),\ \sigma,\ L,\ Cpens\rangle \xrightarrow{v} \langle II,\ \sigma,\ L[b/l],\ Cpens\rangle$
(2) $\langle g \circ P,\ \sigma,\ L,\ Cpens\rangle \longrightarrow \langle P,\ \sigma,\ L,\ Cpens\rangle,$ *if hold*(g)
(3) $\langle \texttt{rec a x},\ \sigma,\ L,\ Cpens\rangle \xrightarrow{a.m} \langle II,\ \sigma[m/x],\ L,\ Cpens\rangle$
(4) $\langle \texttt{rep a x},\ \sigma,\ L,\ Cpens\rangle \xrightarrow{a.\sigma(x)} \langle II,\ \sigma,\ L,\ Cpens\rangle$
(5) $\langle \texttt{throw},\ \sigma,\ L,\ Cpens\rangle \longrightarrow \langle \boxtimes,\ \sigma,\ L,\ Cpens\rangle$
(6) $\langle \texttt{undo},\ \sigma,\ L,\ Cpens\rangle \longrightarrow \langle X; \texttt{undo},\ \sigma,\ L,\ Y\rangle$
$\langle \texttt{undo},\ \sigma,\ L,\ \varepsilon\rangle \longrightarrow \langle \boxminus,\ \sigma,\ L,\ \varepsilon\ \rangle$
where , $X = \mathbf{final}(Cpens)$ *and* $Y = \mathbf{front}(Cpens)$.
(7) $\langle P \lhd b \rhd Q,\ \sigma,\ L,\ Cpens\rangle \longrightarrow \langle P,\ \sigma,\ L,\ Cpens\rangle,$ *if* $b(\sigma)$
$\langle P \lhd b \rhd Q,\ \sigma,\ L,\ Cpens\rangle \longrightarrow \langle Q,\ \sigma,\ L,\ Cpens\rangle,$ *if* $\neg b(\sigma)$
(8) $\langle b * P,\ \sigma,\ L,\ Cpens\rangle \longrightarrow \langle P; b * P,\ \sigma,\ L,\ Cpens\rangle,$ *if* $b(\sigma)$
$\langle b * P,\ \sigma,\ L,\ Cpens\rangle \longrightarrow \langle II,\ \sigma,\ L,\ Cpens\rangle,$ *if* $\neg b(\sigma)$

The transition rules for $x := e$ and $b \lhd l$ can be divided into two steps. This consideration is for the aim of linking operational semantics with denotational semantics. For event guard $@(g)$, it can be fired if the update of shared-labels makes the event guard g satisfied. For `rec a x`, the received message via channel is stored in variable x. For `rep a x`, the value $\sigma(x)$ is sent out via channel a.

`throw` makes the program immediately enter into the fault state while leaving all states unchanged. `undo` executes all the stored compensation programs. The execution is in the reverse order of the previous installed sequence. The transition rules for conditional and iteration are similar to those in traditional programming language.

Theorem 3.2

(1) *if* $\langle P,\ \sigma,\ L,\ Cpens\rangle \xrightarrow{\beta} \langle P',\ \sigma',\ L',\ Cpens'\rangle$ *and* $P' \neq II, \boxtimes, \boxminus$, *then*
$\langle P; Q,\ \sigma,\ L,\ Cpens\rangle \xrightarrow{\beta} \langle P'; Q,\ \sigma',\ L',\ Cpens'\rangle$
(2) *if* $\langle P,\ \sigma,\ L,\ Cpens\rangle \xrightarrow{\beta} \langle II,\ \sigma',\ L',\ Cpens'\rangle$ *then*
$\langle P; Q,\ \sigma,\ L,\ Cpens\rangle \xrightarrow{\beta} \langle Q,\ \sigma',\ L',\ Cpens'\rangle$
(3) *if* $\langle P,\ \sigma,\ L,\ Cpens\rangle \xrightarrow{\beta} \langle x,\ \sigma',\ L',\ Cpens'\rangle$ *and* $x = \boxtimes$ *or* $\boxminus$
then $\langle P; Q,\ \sigma,\ L,\ Cpens\rangle \xrightarrow{\beta} \langle x,\ \sigma',\ L',\ Cpens'\rangle$ □

The above theorem illustrates the transition rules for sequential composition. The first two rules are similar to those in traditional programming languages. The third rule indicates that if P performs a transition reaching to fault state or undo state, the whole process $P; Q$ also performs the same transition reaching to fault state or undo state. The proof can be proceeded directly by the derivation strategy and head normal form.

Further, if a process is in the form of guarded choice, its transition rules are just those in the derivation strategy. Now we define the function:

$$stable(\langle P,\ \sigma,\ L,\ Cpens\rangle$$
$$=_{df} \neg(\langle P, \sigma,\ L,\ Cpens\rangle \xrightarrow{\tau})$$

The transition of nondeterministic choice can be derived as a theorem shown below.

Theorem 3.3

(1) *If* $\langle P,\ \sigma,\ L,\ Cpens\rangle \xrightarrow{\tau} \langle P',\ \sigma',\ L',\ Cpens'\rangle,$ *then*
$\langle P \sqcap Q, \sigma, L, Cpens\rangle \xrightarrow{\tau} \langle P' \sqcap Q, \sigma', L', Cpens'\rangle,$
$\langle Q \sqcap P,\ \sigma,\ L,\ Cpens\rangle \xrightarrow{\tau} \langle Q \sqcap P',\ \sigma',\ L',\ Cpens'\rangle$
(2) *If* $stable(\langle P,\ \sigma,\ L,\ Cpens\rangle),$
then $\langle P \sqcap Q,\ \sigma,\ L,\ Cpens\rangle \xrightarrow{\tau} \langle P,\ \sigma',\ L',\ Cpens'\rangle$
$\langle Q \sqcap P,\ \sigma,\ L,\ Cpens\rangle \xrightarrow{\tau} \langle P,\ \sigma',\ L',\ Cpens'\rangle$

Proof Here we only give the proof for (1).

$$\begin{aligned}
&\langle P,\ \sigma,\ L,\ Cpens\rangle \xrightarrow{\tau} \langle P',\ \sigma',\ L',\ Cpens'\rangle\\
\Rightarrow\ &\{\text{Derivation Strategy}\}\\
&P' \in \mathcal{HF}(P)\\
\Rightarrow\ &\{\text{Set Calculus}\}\\
&P' \in \mathcal{HF}(P) \cup \mathcal{HF}(Q)\\
\Rightarrow\ &\{\text{Derivation Strategy}\}\\
&\langle P,\ \sigma,\ L,\ Cpens\rangle \xrightarrow{\tau} \langle P',\ \sigma',\ L',\ Cpens'\rangle \qquad \square
\end{aligned}$$

Now we define the function:

$$\begin{aligned}
&notfault(\langle P,\ \sigma,\ L,\ Cpens\rangle\\
&\quad =_{df} \neg(\langle P, \sigma,\ L,\ Cpens\rangle \xrightarrow{\beta} \langle\boxtimes,\ \sigma,\ L,\ Cpens\rangle)
\end{aligned}$$

For parallel composition, the corresponding transition can be derived as shown below. Note that **channel**(P) stands for the channels that process P owns.

Theorem 3.4

(1) *if* $\langle P,\ \sigma_1,\ L,\ Cpens\rangle \xrightarrow{\tau} \langle P',\ \sigma_1',\ L',\ Cpens'\rangle$ *and* $stable(\langle Q,\ \sigma_2,\ L,\ Cpens\rangle)$*, then*
$\langle P \parallel Q, \sigma, L, Cpens\rangle \xrightarrow{\tau} \langle \mathbf{par}(P',\ Q), \sigma', L', Cpens'\rangle$
$\langle Q \parallel P, \sigma, L, Cpens\rangle \xrightarrow{\tau} \langle \mathbf{par}(Q,\ P'), \sigma', L', Cpens'\rangle$
where, $\sigma = \sigma_1 \cup \sigma_2$ *and* $\sigma' = \sigma_1' \cup \sigma_2$.
if $\langle P,\ \sigma_1,\ L,\ Cpens\rangle \xrightarrow{\tau} \langle P',\ \sigma_1',\ L',\ Cpens'\rangle$ *and* $\langle Q,\ \sigma_2,\ L,\ Cpens\rangle \xrightarrow{\tau} \langle Q',\ \sigma_2',\ L',\ Cpens'\rangle$,
then $\langle P \parallel Q, \sigma, L, Cpens\rangle \xrightarrow{\tau} \langle \mathbf{par}(P',\ Q'), \sigma', L', Cpens'\rangle$
where, $\sigma = \sigma_1 \cup \sigma_2$ *and* $\sigma' = \sigma_1' \cup \sigma_2'$.

(2) *if* $\langle P,\ \sigma_1,\ L,\ Cpens\rangle \xrightarrow{x} \langle P',\ \sigma_1',\ L',\ Cpens'\rangle$ $(P' \neq \boxtimes)$ *and* $stable(\langle Q,\ \sigma_2,\ L,\ Cpens\rangle)$ *and* $notfault(\langle Q,\ \sigma_2,\ L,\ Cpens\rangle)$*, then*
$\langle P \parallel Q, \sigma, L, Cpens\rangle \xrightarrow{x} \langle \mathbf{par}(P',\ Q), \sigma', L', Cpens'\rangle$
$\langle Q \parallel P, \sigma, L, Cpens\rangle \xrightarrow{x} \langle \mathbf{par}(Q,\ P'), \sigma', L', Cpens'\rangle$
where, $\sigma = \sigma_1 \cup \sigma_2$ *and* $\sigma' = \sigma_1' \cup \sigma_2$.
Here, $\xrightarrow{x} \in \{\longrightarrow,\ \xrightarrow{v}\}$.

(3) *if* $\langle P,\ \sigma_1,\ L,\ Cpens\rangle \xrightarrow{a.m} \langle P',\ \sigma_1',\ L',\ Cpens'\rangle$ *and* $a \in \mathbf{channel}(P) - \mathbf{channel}(Q)$ *and* $notfault(\langle Q,\ \sigma_2,\ L,\ Cpens\rangle)$*, then*
$\langle P \parallel Q, \sigma, L, Cpens\rangle \xrightarrow{a.m} \langle \mathbf{par}(P',\ Q), \sigma', L', Cpens'\rangle$
$\langle Q \parallel P, \sigma, L, Cpens\rangle \xrightarrow{a.m} \langle \mathbf{par}(Q,\ P'), \sigma', L', Cpens'\rangle$
where, $\sigma = \sigma_1 \cup \sigma_2$ *and* $\sigma' = \sigma_1' \cup \sigma_2$.
if $\langle P, \sigma_1, L, Cpens\rangle \xrightarrow{a.m} \langle P', \sigma_1', L', Cpens\rangle$ *and* $\langle Q, \sigma_2, L, Cpens\rangle \xrightarrow{a.m} \langle Q', \sigma_2, L', Cpens\rangle$ *and* $a \in \mathbf{channel}(P) \cap \mathbf{channel}(Q)$ *and* $m = \sigma_1(v)$ *and* $\sigma_1' = \sigma_1[m/u]$*, then*
$\langle P \parallel Q,\ \sigma,\ L,\ Cpens\rangle \longrightarrow$
$\quad\langle u := v\ ;\ \mathbf{par}(P',\ Q'),\ \sigma,\ L,\ Cpens\rangle$
$\langle Q \parallel P,\ \sigma,\ L,\ Cpens\rangle \longrightarrow$
$\quad\langle u := v\ ;\ \mathbf{par}(P',\ Q'),\ \sigma,\ L,\ Cpens\rangle$
where, $\sigma = \sigma_1 \cup \sigma_2$ *and* $\sigma' = \sigma_1' \cup \sigma_2$.

(4) *if* $\langle P,\ \sigma_1,\ L,\ Cpens\rangle \longrightarrow \langle\boxtimes,\ \sigma_1',\ L',\ Cpens'\rangle$ *and* $stable(\langle Q,\ \sigma_2,\ L,\ Cpens\rangle)$
then $\langle P \parallel Q,\ \sigma,\ L,\ Cpens\rangle \longrightarrow \langle\boxtimes,\ \sigma',\ L',\ Cpens'\rangle$
$\langle Q \parallel P,\ \sigma,\ L,\ Cpens\rangle \longrightarrow \langle\boxtimes,\ \sigma',\ L',\ Cpens'\rangle$
where, $\sigma = \sigma_1 \cup \sigma_2$ *and* $\sigma' = \sigma_1' \cup \sigma_2$.

Proof Here we only give the proof for the channel communication transition (i.e., item (3) of this theorem). For others, their proofs are similar. Firstly we consider the proof for the first case of channel communication. From the assumption, we know that P can perform communication transition. Therefore, P can only be of the form below.

$$[]_{i\in I}\{b_i\&\texttt{skip} \rightarrow P_i\}[][]_{j\in J}\{@(g_j) \rightarrow U_j\}$$
$$[][]_{l\in L}\{\texttt{comm}\ a_l\ x_l \rightarrow R_l\}$$

On the other hand, Q can be one of the following forms below:

- $[]_{kinK}\{c_k\&\texttt{skip} \rightarrow V_k\}[][]_{m\in M}\{@(h_m) \rightarrow Q_m\}$ $[][]_{n\in N}\{\texttt{comm}\ d_n\ y_n \rightarrow T_n\}$
- $assign(x, e) \rightarrow E$
- $compen(C) \rightarrow F$

By using the parallel algebraic laws, $\mathcal{HF}(P \parallel Q)$ can also be of the forms below

- $[]_{i\in I}\{b_i\&\texttt{skip} \rightarrow P_i\}[][]\{c_k\&\texttt{skip} \rightarrow V_k\}$ $[][]_{j\in J}\{@(g_j) \rightarrow U_j\}[][]_{m\in M}\{@(h_m) \rightarrow Q_m\}$ $[][]_{l\in L}\{\texttt{comm}\ a_l\ x_l \rightarrow R_l\}[][]_{n\in n}\{\texttt{comm}\ d_n\ y_n \rightarrow T_n\}$

- $[]_{i\in I}\{b_i\&\texttt{skip} \rightarrow P_i\}[][]_{j\in J}\{@(g_j) \rightarrow U_j\}$ $[][]_{l\in L}\{\texttt{comm}\ a_l\ x_l \rightarrow R_l\}$ $[][]\{\mathbf{true}\&\texttt{skip} \rightarrow (assign(x, e) \rightarrow P \parallel E)\}$

- $[]_{i\in I}\{b_i\&\texttt{skip} \rightarrow P_i\}[][]_{j\in J}\{@(g_j) \rightarrow U_j\}$ $[][]_{l\in L}\{\texttt{comm}\ a_l\ x_l \rightarrow R_l\}$ $[][]\{\mathbf{true}\&\texttt{skip} \rightarrow (compen(C) \rightarrow P \parallel F)\}$

From the derivation strategy, we know that $P \parallel Q$ can do the first type of communication transition.

Next we consider the proof for the second case of channel communication. From the assumption, for simplicity, P can only have the following form:

$$[]\{b_i\&\texttt{skip} \rightarrow P_i\}[][]_{j\in J}\{@(g_j) \rightarrow U_j\}$$
$$[][]_{l\in L}\{\texttt{comm}\ a_l\ x_l \rightarrow R_l\}[]\{\texttt{rec}\ a\ u \rightarrow P'\}$$

Similarly, Q can also only have the following form.

$$[]\{c_k\&\texttt{skip} \rightarrow V_k\}[][]_{m\in M}\{@(k_m) \rightarrow Q_m\}$$
$$[][]_{n\in N}\{\texttt{comm}\ a_n\ x_n \rightarrow T_n\}[]\{\texttt{rep}\ a\ v \rightarrow Q'\}$$

Using the parallel expansion laws, $P \parallel Q$ can only have the form below:

$$\begin{aligned}&[]\{b_i\&\mathtt{skip} \rightarrow P_i\}[][]\{c_k\&\mathtt{skip} \rightarrow V_k\}\\&[][]_{j\in J}\{@(g_j) \rightarrow U_j\}[][]_{m\in M}\{@(k_m) \rightarrow Q_m\}\\&[][]_{l\in L}\{\mathtt{comm}\, a_l\, x_l \rightarrow R_l\}[][]_{n\in n}\{\mathtt{comm}\, a_n\, x_n \rightarrow T_n\}\\&[]\{\mathbf{true}\&\mathtt{skip} \rightarrow (u := v;\, \mathbf{par}(P', Q'))\}\end{aligned}$$

This indicates that $P \parallel Q$ can perform the corresponding communication transition. □

For Theorem 3.4, the first item considers the $\xrightarrow{\tau}$ transition for a parallel process. It can be divided into two cases. The first one models the case that only one parallel component can perform $\xrightarrow{\tau}$ transition. The second one models the case that both of the two parallel branches can perform $\xrightarrow{\tau}$ transitions.

If one parallel branch can perform $\longrightarrow$ (or $\xrightarrow{\tau}$), the whole process can also perform the same type of transition, where the subsequent process is still the parallel composition. In this case, we assume another parallel branch is stable and cannot perform transition leading to fault state. The second item models this case for parallel composition.

The third item models the communicating transition for a parallel process. If one parallel branch perform communication with outside (not another parallel branch), the whole parallel process can also perform transition with outside. On the other hand, two parallel branches may perform communication. It can be regarded as an assignment behaviour.

If a parallel branch can perform action reaching to the fault state. The whole process can also reach to the fault state. The fourth item of parallel process transitions reflects this fact.

Theorem 3.5

(1) *if* $\langle A,\ \sigma,\ L,\ Cpens\rangle \xrightarrow{\beta} \langle II,\ \sigma',\ L',\ Cpens'\rangle$, *then*
$\langle\{A?C,\ F\},\ \sigma,\ L,\ Cpens\rangle \xrightarrow{\beta} \langle X,\ \sigma',\ L',\ Cpens'\rangle$
$\langle X,\ \sigma',\ L',\ Cpens'\rangle \longrightarrow \langle II,\ \sigma',\ L',\ Cpens'^\frown\langle C\rangle\rangle$
where, $X = compen(C)$.

(2) *if* $\langle A,\ \sigma,\ L,\ Cpens\rangle \xrightarrow{\beta} \langle\boxtimes,\ \sigma',\ L',\ Cpens'\rangle$,
$\langle F,\ \sigma',\ L',\ Cpens'\rangle \xrightarrow{r} \langle F',\ \sigma'',\ L'',\ Cpens''\rangle$
then
$\langle\{A?C,\ F\}, \sigma, L, Cpens\rangle \xrightarrow{r} \langle F', \sigma'', L'', Cpens''\rangle$

(3) *if* $\langle A,\ \sigma,\ L,\ Cpens\rangle \xrightarrow{\beta} \langle\boxminus,\ \sigma',\ L',\ Cpens'\rangle$ *and then*
$\langle\{A?C,\ F\},\ \sigma,\ L,\ Cpens\rangle \xrightarrow{\beta} \langle\boxminus,\ \sigma',\ L',\ Cpens'\rangle$

(4) *if* $\langle A,\ \sigma,\ L,\ Cpens\rangle \xrightarrow{\beta} \langle A',\ \sigma',\ L',\ Cpens'\rangle$ *and* $A' \neq II, \boxtimes, \boxminus$
then
$\langle\{A?C,\ F\}, \sigma, L, Cpens\rangle \xrightarrow{\beta}$
$\quad\langle\{A'?C,\ F\}, \sigma', L', Cpens'\rangle$

Proof (1) For the first rule, we can consider the proof by analyzing the transition type $\xrightarrow{\beta}$. Here, we only give the proof for the transition type $\xrightarrow{a.m}$. Then, we can assume that $\mathcal{HF}(A)$ has the form below:

$$[]_{i\in I}\{h_i \rightarrow A_i\}[]\{\mathbf{comm}\ a\ x \rightarrow II\}$$

Then we can have

$$\begin{aligned}&\mathcal{HF}(\{A\,?\,C,\ F\})\\&\quad= \mathcal{HF}((\mathcal{HF}(A);\, compen(C)) \circ F)\\&\quad= []_{i\in I}\{h_i \rightarrow \mathbf{seq2}(\mathbf{seq1}(A_i, compen(C)), F)\}[]\\&\qquad[]\{\mathbf{comm}\ a\ x \rightarrow compen(C)\}\end{aligned}$$

Therefore, from the derivation strategy and the communication component in $\mathcal{HF}(\{A\,?\,C,\ F\})$, we know that this scope can also perform the corresponding transitions.

(2) From the assumption, we know that $\mathcal{HF}(P) = \mathtt{throw}$. Then we have

$$\begin{aligned}&\mathcal{HF}(\{A\,?\,C,\ F\})\\&\quad= \mathcal{HF}((\mathtt{throw};\, compen(C)) \circ F)\\&\quad= \mathcal{HF}(\mathtt{throw} \circ F)\\&\quad= F\end{aligned}$$

This indicates that $\mathcal{HF}(\{A\,?\,C,\ F\})$ can also perform the corresponding transition. The proofs of other transition rules for scope are similar. □

In Theorem 3.5 (also in Theorem 3.2), "$\xrightarrow{\beta}$" stands for the transition type, i.e., it can be one of the four transition types. For Theorem 3.5 about the transitions of scope, the analysis can be divided into four cases. The first case models that the primary activity performs action and reaches to the terminating state. Then the compensation program should be recorded in the compensation list. The second case models that the primary activity performs actions and reaches to the fault state. Then the fault handler will be activated to be performed. The third case models that the primary activity performs actions and reaches to the undo state. Then the whole scope also reaches to the undo state. The fourth case models the case that the primary activity performs action and reaches to the normal state. Then the scope can also perform action and reach to the normal state.

3.3 Equivalence of derivation strategy and transition system

The last subsection derived a set of transition rules according to a derivation strategy. The set of transition rules can be considered as a transition system (i.e., operational semantics) for BPEL. The soundness of this derived operational semantics is based on the derivation strategy. There remains an issue: *The* *derivation strategy may derive more transitions, compared with our transition system* (Theorems 3.1–3.5). We want to show that the set of transitions derived from the derivation strategy is same as the set of transitions generated from our transition system. If so, we can say the derivation strategy is equivalent with the transition system.

In order to study this equivalence problem, we need to prove the following items for every process.

(1) If transition $\langle P, \sigma, L, Cpens\rangle \xrightarrow{\beta} \langle P', \sigma', L', Cpens'\rangle$ exists in the *transition system*, then it also exists in the *derivation strategy*.
(2) If transition $\langle P, \sigma, L, Cpens\rangle \xrightarrow{\beta} \langle P', \sigma', L', Cpens'\rangle$ exists in the *derivation strategy*, then it also exists in the *transition system*.

The item (1) is correct because our transition system is derived from the derivation strategy. Now we consider (2) as a theorem to be proved.

Theorem 3.6 *If transition* $\langle P, \sigma, L, Cpens\rangle \xrightarrow{\beta} \langle P', \sigma', L', Cpens'\rangle$ *exists in the derivation strategy, then it also exists in the transition system.*

Proof Here we give the proof for scope activity (i.e., $P = \{A?C, F\}$), others are similar. For the detailed proof, we can proceed via the several cases for activity A. Here we only give the proof for case below.

$$\begin{aligned}&\mathcal{HF}(A)\\ &\quad = [\,]\{b_i \& \mathtt{skip} \rightarrow P_i\}[\,][\,]_{j\in J}\{@(g_j) \rightarrow U_j\}\\ &\qquad [\,][\,]_{l\in L}\{\mathtt{comm}a_l x_l \rightarrow R_l\}\end{aligned}$$

From the derivation strategy, A can perform the following transitions.

(p-1) $\langle A, \sigma, L, Cpens\rangle \longrightarrow \langle P_i, \sigma, L, Cpens\rangle$, if $b_i(\sigma)$
(p-2) $\langle A, \sigma, L, Cpens\rangle \longrightarrow \langle U_j, \sigma, L, Cpens\rangle$, if $hold(g_j)$
(p-3) $\langle A, \sigma, L, Cpens\rangle \xrightarrow{a_k.m_k} \langle R_k, \sigma_k, L, Cpens\rangle$

By the induction, we know that these transitions also exist in our transition system. Further, we can have:

$$\begin{aligned}&\mathcal{HF}(P)\\ &\quad = [\,]\{b_i \& \mathtt{skip} \rightarrow \mathbf{seq2}(\mathbf{seq1}(P_i, compen(C)), F)\}\\ &\qquad [\,][\,]\{@g_j \rightarrow \mathbf{seq2}(\mathbf{seq1}(Q_j, compen(C)), F)\}\\ &\qquad [\,][\,]\{\mathbf{comm}\ a_k\ x_k \rightarrow \mathbf{seq2}(\mathbf{seq1}(R_k, compen(C)), F)\}\end{aligned}$$

Now we do further analysis. Assume that $X \neq \boxtimes$ or $X \neq \boxminus$. Then we have:

If $X = II$, then
$\mathbf{seq2}(\mathbf{seq1}(X, compen(C)), F) = compen(C)$.
Otherwise, then
$\mathbf{seq2}(\mathbf{seq1}(X, compen(C)), F) = X\ ;\ compen(C)$.
By using the derivation strategy, P can have the transitions below:

(l-1)
$\langle\{A?C, F\}, \sigma, L, Cpens\rangle \longrightarrow \langle compen(C), \sigma, L, Cpens\rangle$, if $b_i(\sigma)$ and $P_i = II$
(l-2)
$\langle\{A?C, F\}, \sigma, L, Cpens\rangle \longrightarrow \langle\{A'?C, F\}, \sigma, L, Cpens\rangle$, if $b_i(\sigma)$ and $P_i \neq II$
(l-3)
$\langle\{A?C, F\}, \sigma, L, Cpens\rangle \longrightarrow \langle compen(C), \sigma, L, Cpens\rangle$, if $hold_j(g_j)$ and $Q_j = II$
(l-4)
$\langle\{A?C, F\}, \sigma, L, Cpens\rangle \longrightarrow \langle\{A'?C, F\}, \sigma, L, Cpens\rangle$, if $hold_j(g_j)$ and $Q_j \neq II$
(l-5)
$\langle\{A?C, F\}, \sigma, L, Cpens\rangle \xrightarrow{a_k.m_k} \langle compen(C), \sigma_k, L, Cpens\rangle$, if $R_k = II$
(l-6)
$\langle\{A?C, F\}, \sigma, L, Cpens\rangle \xrightarrow{a_k.m_k} \langle\{A'?C, F\}, \sigma_k, L, Cpens\rangle$, if $R_k \neq II$
(l-7)
$\langle compen(C), \sigma, L, Cpens\rangle \longrightarrow \langle II, \sigma, L, Cpens^\frown\langle C\rangle\ \rangle$

Now we want to prove that the above transitions (l-1–l-7) also exist in the transition system. This can be directly achieved from the transition system of scope. □

Now we present the main result of this section.

Theorem 3.7 *Regarding the derived operational semantics for BPEL, the derivation strategy is equivalent with the transition system.*

This result demonstrates that the transitions from the derivation strategy are the same as those in the transition system. Together with the theorems achieved in last subsection, it also shows that our transition system (operational semantics) for BPEL is sound and complete with respect to the derivation strategy in the last subsection.

4 Deriving denotational semantics from algebraic semantics for BPEL

In this section, we study the derivation of denotational semantics for BPEL programs; i.e., calculating the denotational semantics from the head normal form. This gives us a way to reason about program equivalence easily.

4.1 Denotational semantic model for BPEL

This section considers the denotational semantic model for BPEL. Our approach is based on the relational calculus [25]. In order to represent the execution state of a program, we introduce a pair of variables $\overleftarrow{st}$ and $\overrightarrow{st}$ into our semantic model. Variable $\overleftarrow{st}$ stands for the initial execution state of a program before its activation, whereas $\overrightarrow{st}$ stands for the final execution state of the program during the current observation.

A program may have five execution states: (1) *divergent state*, (2) *completed state*, (3) *waiting state*, (4) *error state*, (5) *undo state*. The first three states are similar to those in reactive systems [12,35,36,50]. For "error state", a program may encounter a fault during its execution, where compensation may be executed via fault handling. Further, "undo state" is introduced to distinguish the termination of a process itself and the termination of the execution of compensating programs.

For a parallel process, messages can be transformed from one service to another. A Boolean guard can be fired by the update of its associated links. For considering these two mechanisms, a trace variable tr is introduced, whose elements has two forms: (1) the message transmission from different services; (2) the update of the shared-labels. For message transmission, the element recorded in the trace variable can be expressed in the form $a.m$. Here, a is the channel name where communication may go through and m is the message for transmitting. For the update of a shared-label, the associated element in the trace variable can be expressed as a snapshot in the form $l : (\overleftarrow{l}, \overrightarrow{l})$, where l is the label name, and $\overleftarrow{l}$ and $\overrightarrow{l}$ stand for the initial and final values of label l respectively. We use $\overleftarrow{tr}$ and $\overrightarrow{tr}$ to represent the initial and final trace of variable tr.

Further, we introduce a variable $Cpens$ in our model, which records a sequence of programs in the form of *stack* (i.e., first installing last compensating). Here, we use $\overleftarrow{Cpens}$ and $\overrightarrow{Cpens}$ to denote the initial and final compensating sequence, respectively. Channels are applied in our model for transmitting messages between different services. Like CSP [22], we also apply a set variable ref in our semantic model, which indicates that the elements in the set are refused by the process currently. Similarly, $\overleftarrow{ref}$ and $\overrightarrow{ref}$ stand for the initial and final refusal set, respectively. Now we start to consider the healthiness conditions a BPEL program should satisfy. Two trace variables tr and $Cpens$ are introduced in our semantic model. They cannot be shortened. A formula P which identifies a program must satisfy the healthiness condition below.

$$(H1) \quad P = P \wedge Inv(tr, Cpens)$$

where, $Inv(tr, Cpens) =_{df} (\overleftarrow{tr} \preceq \overrightarrow{tr}) \wedge (\overleftarrow{Cpens} \preceq \overrightarrow{Cpens})$. Here, $s \preceq t$ denotes that sequence s is a prefix of sequence t.

If a program is initially at the divergent state, the program cannot be started and its initial values are even unobservable. Therefore, another healthiness condition is required for BPEL programs.

$$(H2) \quad P = P \vee (\overleftarrow{st} = div \wedge Inv(tr, Cpens))$$

If a program is activated and finally at the divergent state, this indicates that the program is totally unpredictable. Therefore, a healthiness condition should be satisfied:-

$$(H3) \quad P = P; III$$

where, $III =_{df} (\overleftarrow{st} = div \Rightarrow Inv(tr, Cpens)) \wedge (\overleftarrow{st} \neq div \Rightarrow Id)$. Here Id is the identity relation.

For sequential composition "$R; P$", if P is initially at the "*error*" state, this means that the predecessor of P (i.e., R) encounters a fault. Then P cannot have the chance to be scheduled. Similar considerations also apply to "*wait*" and "*undo*" states. Therefore, it satisfies a further healthiness condition.

$$(H4) \quad P = III \lhd (\overleftarrow{st} = wait \vee \overleftarrow{st} = error \vee \overleftarrow{st} = undo) \rhd P$$

where, Boolean expression $P \lhd b \rhd Q =_{df} b \wedge P \vee \neg b \wedge Q$. Next we give the definition for $\mathcal{H}$-function:

$$\mathcal{H}(X) =_{df} (X \wedge Inv(tr, Cpens)\ ;\ III) \lhd \overleftarrow{st} = completed \rhd (Inv(tr, Cpens) \lhd \overleftarrow{st} = div \rhd III)$$

From this definition, we know that $\mathcal{H}(X)$ satisfies all the healthiness conditions $(H1)$–$(H4)$. The $\mathcal{H}$-function can be used in defining the denotational semantics for BPEL programs.

4.2 Deriving denotational semantics from algebraic semantics

Firstly, we introduce the alphabet for command c, which is denoted as:

$$(\mathbf{Var}(c), \mathbf{SLink}(c), \mathbf{TLink}(c))$$

where

- $\mathbf{Var}(c)$ is the set of program variables owned by c.
- $\mathbf{SLink}(c)$ stands for the set of links which can only be read by c; i.e., the set containing all the source links of command c.
- $\mathbf{TLink}(c)$ represents the set of links which can only be updated by c; i.e., the set containing all the target links of command c. Sometimes, without causing confusions, $\mathbf{Var}(c)$, $\mathbf{SLink}(c)$ and $\mathbf{TLink}(\mathbf{c})$ are simply written as **Var**, **SLink** and **TLink** for command c respectively.

Further, we use $\mathbf{Comm}(C)$ to record all the snapshots which stand for the update of all the links in set C; i.e., $\mathbf{Comm}(C) =_{df} \{l : (e_1, e_2) \mid l \in C\}$. We use $\mathbf{beh}(c)$ to represent the meaning of command c. Next we give some preliminary definitions.

(1) $Rec\_Comm =_{df} \exists t \in \textbf{Comm}(\textbf{SLink})^* \bullet \overrightarrow{tr} = \overleftarrow{tr} \cdot t$
(2) $same(A) =_{df} \bigwedge_{x \in A} \overrightarrow{x} = \overleftarrow{x}$
(3) $\textbf{VTC} =_{df} \textbf{Var} \cup \textbf{Tlink} \cup \{Cpens\}$

Formula Rec_Comm indicates that the source links of the process may receive updates from other parallel components. These updates are recorded in the trace variable. Formula $same(A)$ indicates that all variables in set A remain unchanged during the execution of the program. Further, **VTC** stands for the set of the union of all local variables, target links of a process, together with the denotational variable $Cpens$.

Next we introduce the behaviours of some fundamental statements. The execution of $x := e$ terminates immediately. Variable x is updated, while all other variables remain unchanged. Further, assignment can also receive the updates of all its source links. From denotational view, we regard $\textbf{beh}(assign(x, e)) = \textbf{beh}(x := e)$.

$\textbf{beh}(x := e)$

$$=_{df} \mathcal{H}\begin{pmatrix} \overrightarrow{st} = completed \wedge \overrightarrow{x} = \overleftarrow{e} \wedge \\ Rec\_Comm \wedge same(\textbf{VTC} \setminus \{x\}) \end{pmatrix}$$

The execution of rec $a\,x$ can initially be at the waiting state. At this state, channel a is ready for communication and the process can receive the updates of all source links. On the other hand, a message is received via this channel finally and the process terminates. All variables remain unchanged except variable x.

$\textbf{beh}(\texttt{rec}\, a\, x)$

$$=_{df} \mathcal{H}\begin{pmatrix} \overrightarrow{st} = wait \wedge Rec\_Comm \wedge a \notin \overrightarrow{ref} \vee \\ \overrightarrow{st} = completed \wedge same(\textbf{VTC} \setminus \{x\}) \wedge \\ \exists s, t \in \textbf{Comm}(\textbf{SLink})^*, \exists m \in \textbf{Type}(a) \bullet \\ (\overrightarrow{x} = m \wedge \overrightarrow{tr} = \overleftarrow{tr} \cdot s \cdot \langle a.m \rangle \cdot t)) \end{pmatrix}$$

Similar to rec $a\,x$, the execution of rep $a\,x$ also has two states. The message for output is stored in the trace variable tr, which can be received by the corresponding partner.

$\textbf{beh}(\texttt{rep}\, a\, x)$

$$=_{df} \mathcal{H}\begin{pmatrix} \overrightarrow{st} = wait \wedge Rec\_Comm \wedge a \notin \overrightarrow{ref} \vee \\ \overrightarrow{st} = completed \wedge same(\textbf{VTC}) \wedge \\ \exists s, t \in \textbf{Comm}(\textbf{SLink})^* \bullet \overrightarrow{tr} = \overleftarrow{tr} \cdot s \cdot \langle a.\overleftarrow{x} \rangle \cdot t \end{pmatrix}$$

The execution of throw makes the process at the $error$ state. Meanwhile, it can initially receive the updates of all its source links, while all the corresponding variables remain unchanged.

$\textbf{beh}(\texttt{throw})$

$$=_{df} \mathcal{H}(\overrightarrow{st} = error \wedge Rec\_Comm \wedge same(\textbf{VTC}))$$

When the undo statement is scheduled, all the compensated processes will be executed in the reverse order. Finally, the process will be at the $undo$ state.

$\textbf{beh}(\texttt{undo}) =_{df} \textbf{e}xec(\overleftarrow{Cpens})$,

where

$\textbf{exec}(\overleftarrow{Cpens})$

$$=_{df} \begin{pmatrix} \overleftarrow{Cpens} = \epsilon \Rightarrow \mathcal{H}\begin{pmatrix} \overrightarrow{st} = undo \wedge \\ Rec\_Comm \wedge \\ same(\textbf{VTC}) \end{pmatrix} \wedge \\ \overleftarrow{Cpens} \neq \epsilon \Rightarrow \\ \textbf{beh}(\textbf{final}(\overleftarrow{Cpens}))\ ;\ \textbf{exec}(\textbf{front}(\overleftarrow{Cpens})) \end{pmatrix}$$

If event $@(g)$ does not hold, the process is at the waiting state. On the other hand, when $@(g)$ is satisfied finally, the process is at the terminating state, with the variables unchanged. No matter what state the process is at, the process can receive the updates of all the source links.

$\textbf{beh}(g)$

$$=_{df} \mathcal{H}\begin{pmatrix} \overrightarrow{st} = wait \wedge Rec\_Comm \wedge \neg\, \textbf{holds}(g) \vee \\ \overrightarrow{st} = completed \wedge Rec\_Comm \wedge \\ same(\textbf{VTC}) \wedge \textbf{holds}(g) \end{pmatrix}$$

Similar to local variable assignment, $b \rhd l$ terminates immediately. The label update is recorded in the trace variable, which can be shared by its parallel partner. From denotational view, we also regard $\textbf{beh}(assign(l, b)) = \textbf{beh}\ (b \rhd l)$.

$\textbf{beh}(b \rhd l)$

$$=_{df} \mathcal{H}\begin{pmatrix} \overrightarrow{st} = completed \wedge same(\textbf{VTC} \setminus \{l\}) \wedge \\ \exists s, t \in \textbf{Comm}(\textbf{SLink})^* \bullet \\ \overrightarrow{tr} = \overleftarrow{tr} \cdot s \cdot \langle l : (\overleftarrow{l}, \overrightarrow{l}) \rangle \cdot t \wedge \overrightarrow{l} = \overleftarrow{b} \end{pmatrix}$$

Now we consider the derivation strategy for deriving denotational semantics from algebraic semantics. We use the notation $A(P)$ to represent the derived denotational semantics from the algebraic semantics. Here, our derivation approach is only limited to finite programs.

Definition 4.1 (*Derivation Strategy*)

(1) If $\mathcal{HF}(P) = \bigoplus_{i \in I} P_i$,
then $A(P) = \bigvee_{i \in I} A(P_i)$
(2) Otherwise,

(a) If $\mathcal{HF}(P)$
$= []_{i \in I}\{b_i \& \texttt{skip} \rightarrow P_i\} [][]_{j \in J}\{@(g_j) \rightarrow Q_j\}$
$[][]_{l \in L}\{\texttt{comm}\ a_l\ x_l \rightarrow R_l\}$
then

$$A(P) =_{df}$$
$$\mathcal{H}\left(\begin{array}{l}\left(\begin{array}{l}\overrightarrow{st} = wait \wedge (I = \emptyset) \wedge \bigwedge_{j \in J} \mathbf{beh}(g_j) \\ \wedge \bigwedge_{l \in L} \mathbf{beh}(\mathbf{comm}\ a_l\ x_l)\end{array}\right) \\ \vee \\ \left(\begin{array}{l}\bigvee_{i \in I}(\, b_i \wedge \mathbf{beh}(\mathtt{skip})\ ;\ A(P_i))\vee \\ \bigvee_{j \in J}(\ \overrightarrow{st} = completed \wedge \\ \mathbf{beh}(g_j)\ ;\ A(Q_j)\ )\ \vee \\ \bigvee_{l \in L}(\ \overrightarrow{st} = completed \wedge \\ \mathbf{beh}(\mathbf{comm}\ a_l\ x_l)\ ;\ A(R_l)\ )\end{array}\right)\end{array}\right)$$

(b) If $\mathcal{HF}(P) = assign(u, v) \to Q$,
then $A(P) = \mathbf{beh}(assign(u, v)) \to A(Q)$

(c) If $\mathcal{HF}(P) = compen(C) \to Q$,
then $A(P) = \mathbf{beh}(compen(C)) \to A(Q))$

(d) If $\mathcal{HF}(P) = \mathtt{throw}$,
then $A(P) = \mathbf{beh}(\mathtt{throw})$

(e) If $\mathcal{HF}(P) = \mathtt{undo}$,
then $A(P) = \mathbf{beh}(\mathtt{undo})$

where, the behaviour of $compen(C)$ can be defined as:
$\mathbf{beh}(compen(C))$
$$=_{df} \left(\begin{array}{l}\overrightarrow{st} = completed\ \wedge\ Rec\_Comm \wedge \\ same(\mathbf{Var} \cup \mathbf{Tlink})\ \wedge \overrightarrow{Cpens} = \overleftarrow{Cpens}^\frown\langle C\rangle\end{array}\right)$$

□

Furthermore, we also need the consideration $A(II) =_{df} III$ when deriving denotational semantics from algebraic semantics. If the head normal form of a program can be expressed as the summation of a set of processes, the behaviour of the program is just the disjunction of all these processes. Otherwise, the head normal form of the program can be expressed as one of the four typical forms. The analysis can be proceeded via these four forms. If the head normal form is expressed in the form of guarded choice, the process can be either at the waiting state provided that there are no `skip` guards in the choice. At this state, the behaviour of the process is just the conjunction of all the event guards and communication guards. Further, if any guard is scheduled or fired, the subsequent behaviour is the following process after the corresponding guard. The analysis of other forms is similar.

This gives us a way to calculate the denotational semantics from the head normal form of a program. Next we want to study the compositional properties for this derivation strategy.

Theorem 4.2 (1) $A(P; Q) = A(P); A(Q)$
(2) $A(P \lhd b \rhd Q) = A(P) \lhd b \rhd A(Q)$
(3) $A(P \sqcap Q) = A(P) \vee A(Q)$
(4) $A(\{P\ ?\, C,\ F\}) = A(P); \mathbf{deal}(C, F)$,
where,

$\mathbf{deal}(C, F)$
$$=_{df} \left(\begin{array}{l}\overleftarrow{st} = completed\ \Rightarrow\ \mathbf{beh}(compen(C)) \wedge \\ \overleftarrow{st} = error\ \Rightarrow\ A(F)[completed/\overleftarrow{st}\,] \wedge \\ \overleftarrow{st} \in \{wait,\ undo\}\ \Rightarrow\ \mathbf{beh}(\mathtt{skip})\end{array}\right)$$

□

Here we use formula $\mathbf{deal}(C, F)$ to deal with compensation and fault handler. If P terminates successfully, this can be represented by "$\overleftarrow{st} = completed$" in formula $\mathbf{deal}(C, F)$. At this case, process C is recorded in the compensation list. On the other hand, if P is at the error state, this can be represented by "$\overleftarrow{st} = error$" in formula $\mathbf{deal}(C, F)$. At this case, fault handler F is scheduled. Moreover, if process P is currently at the waiting or undo state, $\mathbf{deal}(C, F)$ behaves like `skip`.

In this section, we have explored the calculation of denotational semantics from the algebraic semantics for BPEL. This can help us in reasoning the properties of BPEL programs easily, especially for parallel programs. Further, this approach also shows the linking theories between algebraic semantics and denotational semantics for BPEL programs.

5 Related work

Compensation is one typical feature for long-running transactions. Butler et al. have explored the compensation feature in the style of process algebra CSP [21,44], namely *compensating* CSP. The operational semantics and trace semantics have been studied [5,9]. The compensation was introduced via a construct $P \div Q$, where P is the forward process and Q is its associated compensation behavior. Structured Activity Compensation (StAC) [6] is another business process modeling language, where compensation acts as one of its main features. Its operational semantics has also been studied in [7]. Meanwhile, the combination of StAC and B method [1] has been explored in [8], which provides the precise description of business transactions. Bruni et al. have studied the transaction calculi for Sagas [4]. The long-running transactions were discussed and a process calculi was proposed in the form of Java API, namely Java Transactional Web Services [3].

Transaction-based services are increasingly being applied in showing many universal interoperability problems. Compensation and failure are typical phenomena of the execution of long-running transactions. To accommodate these new program features, He [19] extended the Guarded Command Language [14] by adding the compensation and coordination combinators. Such extension has been shown as conservative because it preserves all the algebraic laws of formula *design* [25]. A Galois link between the standard design with this new model has been established. Lanotte et al. [29] have studied the design and verification of long-running transactions in a timed framework. They have developed a model of Communicating Hierarchical Timed Automata, which allows the verification of long-running transaction's properties by model checking.

Compensation and fault handing are also the two main features of BPEL. Qiu et al. have provided a deep formal analysis of the fault behavior for BPEL-like processes [43]. Pu et al. have formalized the operational semantics for BPEL [42], where bisimulation has been considered. Further, Cerone et al. explored the security issues in BPEL framework [10]; i.e., Role Based Access Control (RBAC) has been integrated into BPEL. In this framework, model checking has been applied to verify the satisfaction of security issues. π-calculus [38] has been applied in describing various web services models. Lucchi and Mazzara formalized the semantics of BPEL using π-calculus [32]. Laneve and Zavattaro [28] also explored the application of π-calculus in the formalization of the semantics of the transactional construct of BPEL. Breugel and Koshkina proposed a language called BPE-calculus, which is based on BPEL4WS for expressing web service orchestration. They have applied the Concurrency Workbench in supporting the verification of web services [27]. Dead-Path-Ellimination is a key ingredient of BPEL4WS. As DPE may give rise to unintended side effects. BPE-calculus was modeled both in the absence and in the presence of DPE [47]. Recently, Luo et al. have studied the verification of BPEL programs using Hoare logic [33,34]. A set of proof rules were proposed in Hoare logic style. They were proven sound with respect to the formalized semantics.

We have also done research in applying formal approaches to BPEL4WS. We have investigated a formal model for BPEL-like language. Its denotational semantics [20] was studied via the UTP approach and the operational semantics [51] was also explored. A set of algebraic laws has been explored, and their soundness can be verified via these two semantics respectively. We have also studied the link between the operational semantics and denotational semantics for BPEL [52]. Our approach is to derive the denotational semantics from operational semantics. The concept of transition condition and phase semantics was defined for each type of transition for building the link between these two semantics. This paper investigates the algebraic approach for linking the semantics for BPEL, where our approach is to derive the operational semantics and denotational semantics from algebraic semantics.

Unifying Theories of Programming (abbreviated as *UTP*) was developed by Hoare and He in 1998 [25]. *UTP* covers wide areas of fundamental theories of programs in a formalized style and acts as a consistent basis for the principles of programming language. For relating operational and algebraic semantics, Hoare and He have studied the derivation of operational semantics from the algebraic semantics [24,25]. An operational semantics of CSP [22] was derived, based on CSP's algebraic laws according to a derivation strategy (called the action transition relation). An operational semantics of Dijkstra's Guarded Command Language (GCL) was also derived based on GCL's algebra according to the derivation strategy (called the step relation). The total correctness of the derived GCL's operational semantics was also discussed in [26].

Several approaches have been investigated for the equivalence of operational semantics and denotational semantics. Brookes has given a new denotational semantics for a shared-variable parallel programming language [2]. The denotational semantics is proved to be fully abstract with respect to the operational-based partial correctness behaviour. His semantics is adaptable to deal with different levels of granularity or atomicity. In each of these cases, full abstraction is always achieved. Equivalence has also been investigated in the book *Control Flow Semantics* (abbreviated as *CFS*) [13]. *CFS* is devoted to studying the equivalence of operational semantics and denotational semantics for 27 languages using the theory of Metric Space. For the equivalence investigation, the Banach Space theorem is applied.

Glabbeek has investigated the topic of comparative concurrency semantics, which aims at the classification of process semantics [16–18]. He proposed four different kinds of identification categories; i.e., linear time versus branching time, interleaving semantics versus partial order semantics, different treatments of abstraction from internal actions and different approaches to infinity. For the classification of process semantics in the category of linear time versus branching time [18], there are 12 semantics in total. These semantics have been compared, with a resulting comparison hierarchy. The coarsest semantics is the trace semantics, whereas the finest semantics is the bisimulation semantics.

6 Conclusion

In this paper, we have explored the linking theories between the three semantics of BPEL, namely, algebraic semantics, operational semantics and denotational semantics, where our starting point is algebraic semantics. Our consideration is to derive the operational semantics and denotational semantics from algebraic semantics.

Firstly, we considered the algebraic laws for BPEL programs. Our approach is new. We introduced four types of typical forms. Among these, a new form "$compen(C)$" is introduced, which acts as recording the compensation program C in the compensation list. Scope activity is dealt with using the typical form "$compen(C)$" and the new introduced construct "$\circ$". Based on the explored algebraic laws, we have defined the concept of head normal form, by which each

BPEL program is expressed as the summation of a set of initially deterministic processes.

The derivation of the operational semantics for BPEL has been studied from its algebraic semantics. We have presented a general definition of the derivation strategy. Then a transition system (i.e., operational semantics) for BPEL can be derived via the derivation strategy. This gives us the confidence for the consistency between the operational semantics and the algebraic semantics. Further, the relationship between the derivation strategy and the derived operational semantics has also been investigated. We have proved that the derived operational semantics is equivalent with the derivation strategy. The result achieved here shows that our transition system is sound and complete with respect to the algebraic semantics.

As every program can be expressed as a summation of a set of initially deterministic processes, the denotational semantics for BPEL programs has also been derived based on the head normal form of each program. The derivation strategy is defined and the denotational semantics for every program can be calculated. This gives us a way to reason about program properties easily, especially for parallel programs.

For the future, we continue to explore the unifying theories for web services [25,48], as well as the further web service models, including the probabilistic web service models [37] and web service transaction models [31].

Acknowledgments The authors gratefully acknowledge support from the Danish National Research Foundation and the National Natural Science Foundation of China (Grant No. 61061130541) for the Danish-Chinese Center for Cyber Physical Systems. This work was also supported by National Basic Research Program of China (No. 2011CB 302904), National High Technology Research and Development Program of China (No. 2011AA010101) and National Natural Science Foundation of China (No. 61021004).

References

1. Abrial J-R (1996) The B-book: assigning programs to meanings. University Press, Cambridge
2. Brookes SD (1996) Full abstraction for a shared-variable parallel language. Inf Comput 127(2):145–163
3. Bruni R, Ferrari GL, Melgratti HC, Montanari U, Strollo D, Tuosto E (2005) From theory to practice in transactional composition of web services. In: Proceedings of EPEW/WS-FM 2005: European performance engineering workshop and international workshop on web services and formal methods, Versailles, France, September 1–3, 2005. Lecture notes in computer science, vol 3670. Springer, Berlin, pp 272–286
4. Bruni R, Melgratti HC, Montanari U (2004) Theoretical foundations for compensations in flow composition languages. In: Proceedings of POPL 2005: 32nd ACM SIGPLAN-SIGACT symposium on principles of programming languages, Long Beach, California, USA, January 12–14, 2005. ACM, USA, pp 209–220
5. Butler M, Ripon S (2005) Executable semantics for compensating CSP. In: Proceedings of EPEW 2005: international workshop on web services and formal methods, Versailles, France, September 1–3, 2005. Lecture notes in computer science, vol 3670. Springer, Berlin, pp 243–256
6. Butler MJ, Ferreira C (2000) A process compensation language. In: Proceedings of IFM 2000: 2nd international conference on integrated formal methods, Dagstuhl Castle, Germany, November 1–3, 2000. Lecture notes in computer science, vol 1945. Springer, Berlin, pp 61–76
7. Butler MJ, Ferreira C (2004) An operational semantics for StAC, a language for modelling long-running business transactions. In: COORDINATION 2004: 6th international conference on coordination models and languages, Pisa, Italy, February 24–27, 2004. Lecture notes in computer science, vol 2949. Springer, Berlin, pp 87–104
8. Butler MJ, Ferreira C, Ng MY (2005) Precise modelling of compensating business transactions and its application to BPEL. J Univers Comput Sci 11(5):712–743
9. Butler MJ, Hoare CAR, Ferreira C (2005) A trace semantics for long-running transactions. In: Communicating sequential processes: the first 25 years, symposium on the occasion of 25 years of CSP, London, UK, July 7–8, 2004. Lecture notes in computer science, vol 3525. Springer, Berlin, pp 133–150
10. Cerone A, Zhao X, Krishnan P (2006) Modelling and resource allocation planning of BPEL workflows under security constraints. Technical report 336, UNU/IIST, P.O. Box 3058, Macau SAR, China, June
11. Curbera F, Goland Y, Klein J, Leymann F, Roller D, Satish Thatte M, Weerawarana S (2003) Business process execution language for web service. http://www.siebel.com/bpel
12. Davis J (1993) Specification and proof in real-time CSP. University Press, Cambridge
13. de Bakker J, de Vink E (1996) Control flow semantics. MIT Press, Massachusetts
14. Dijkstra EW (1976) A discipline of programming. Prentice Hall International Series in Automatic Computation
15. Garcia-Molina H, Salem K (1987) Sagas. In: Proceedings of ACM SIGMOD international conference on management of data, San Francisco, California, USA, May 27–29, 1987. ACM, USA, pp 249–259
16. Glabbeek Rv (1993) The linear time—branching time spectrum II; the semantics of sequential systems with silent moves (extended abstract). In: Best E (ed) Proceedings of CONCUR'93: 4th international conference on concurrency theory, Hildesheim, Germany, August 1993. Lecture notes in computer science, vol 715. Springer, Berlin, pp 66–81
17. Glabbeek Rv (1996) Comparative Concurrency Semantics and Refinement of Actions. CWI Tract, vol 109. CWI, Amsterdam (Second edition of dissertation)
18. Glabbeek Rv (2001) The linear time—branching time spectrum I; the semantics of concrete, sequential processes. In: Bergstra J, Ponse A, Smolka S (eds) Handbook of process algebra, chapt 1. Elsevier, Amsterdam, pp 3–99
19. He J (2008) Modelling coordination and compensation. In: Proceedings of ISoLA 2008: 3rd international symposium on leveraging applications of formal methods, verification and validation, Porto Sani, Greece, 13–15 October. Communications in Computer and Information Science, vol 17. Springer, Berlin, pp 15–36
20. He J, Zhu H, Pu G (2007) A model for BPEL-like languages. Front Comput Sci China 1(1):9–19
21. Hoare CAR (1978) Communicating sequential processes. Commun ACM 21(8):666–677
22. Hoare CAR (1985) Communicating sequential processes. Prentice Hall International Series in Computer Science
23. Hoare CAR, Hayes IJ, He J, Morgan C, Roscoe AW, Sanders JW, Sørensen IH, Spivey JM, Sufrin B (1987) Laws of programming. Commun ACM 38(8):672–686

24. Hoare CAR, He J (1993) From algebra to operational semantics. Inf Process Lett 45:75–80
25. Hoare CAR, He J (1998) Unifying theories of programming. Prentice Hall International Series in Computer Science
26. Hoare CAR, Jifeng H, Sampaio A (1997) Algebraic derivation of an operational semantics. In: Plotkin G, Stirling C, Tofte M (eds) Proof, language and interaction: essays in honour of Robin Milner, foundations of computer science series. The MIT Press, Massachusetts
27. Koshkina M, Breugel Fvan (2004) Modelling and verifying web service orchestration by means of the concurrency workbench. ACM SIGSOFT Softw Eng Notes 29(5):1–10
28. Laneve C, Zavattaro G (2005) Web-pi at work. In: Proceedings of TGC 2005: international symposium on trustworthy global computing, Edinburgh, UK, April 7–9, 2005. Lecture notes in computer science, vol 3705. Springer, Berlin, pp 182–194
29. Lanotte R, Maggiolo-Schettini A, Milazzo P, Troina A (2008) Design and verification of long-running transactions in a timed framework. Sci Comput Program 73(2–3):76–94
30. Leymann F (2001) Web Services Flow Language (WSFL 1.0). IBM http://www-3.ibm.com/software/solutions/webservices/pdf/WSDL.pdf
31. Li J, Zhu H, Pu G, He J (2007) Looking into compensable transactions. In: Proceedings of SEW-31: 31st IEEE software engineering workshop, Baltimore, USA. IEEE Computer Society Press, Los Angeles, pp 154–166
32. Lucchi R, Mazzara M (2007) A pi-calculus based semantics for ws-bpel. J Log Algebraic Program 70(1):96–118
33. Luo C, Qin S, Qiu Z (2008) Verifying bpel-like programs with hoare logic. In: Proceedings of TASE 2008: 2nd IEEE international symposium on theoretical aspects of software engineering, Nanjing, China, June 2008. IEEE Computer Society, Los Angeles, pp 151–158
34. Luo C, Qin S, Qiu Z (2008) Verifying bpel-like programs with hoare logic. Front Comput Sci China 2(4):344–356
35. Manna Z, Pnueli A (1992) The temporal logic of reactive and concurrent systems: specification. Springer, Berlin
36. Manna Z, Pnueli A (1995) Temporal verification of reactive systems: safety. Springer, Berlin
37. McIver A, Morgan C (2004) Abstraction, refinement and proof of probability systems. monographs in computer science. Springer, Berlin
38. Milner R (1999) Communication and mobile system: π-calculus. University Press, Cambridge
39. Moss J (1981) Nested transactions: an approach to reliable distributed computing. PhD thesis, Department of Electrical Engineering and Computer Science, MIT, April
40. Plotkin G (2004) A structural approach to operational semantics. Technical report 19, University of Aahus, 1981. J Log Algebraic Program 60–61:17–139
41. Pu G, Zhao X, Wang S, Qiu Z (2006) Towards the semantics and verification of BPEL4WS. Electron Notes Theor Comput Sci 151(2):33–52
42. Pu G, Zhu H, Qiu Z, Wang S, Zhao X, He J (2006) Theoretical foundations of scope-based compensation flow language for web service. In: Proceedings of FMOODS 2005: 8th IFIP international conference on formal methods for open object-based distributed systems, Bologna, Italy, 14–16 June, 2006. Lecture notes in computer science, vol 4307. Springer, Berlin, pp 251–266.
43. Qiu Z, Wang S, Pu G, Zhao X (2005) Semantics of BPEL4WS-Like fault and compensation handling. In: Proceedings of FM 2005: international symposium of formal methods Europe, Newcastle, UK, July 18–22, 2005. Lecture notes in computer science, vol 3582. Springer, Berlin, pp 350–365
44. Roscoe AW (1997) The theory and practice of concurrency. Prentice Hall International Series in Computer Science
45. Scott D, Strachey C (1971) Towards a mathematical semantics for computer languages. Technical report PRG-6, Oxford University Computer Laboratory
46. Thatte S (2001) XLANG: Web Service for Business Process Design. Microsoft, http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.html
47. van Breugel F, Koshkina M (2005) Dead-path-elimination in bpel4ws. In: Proceedings of ACSD 2005: fifth international conference on application of concurrency to system design, pp 192–201, St. Malo, France, June, IEEE Computer Society, Los Angeles
48. Zhu H (2005) Linking the semantics of a multithreaded discrete event simulation language. PhD thesis, London, South Bank University, February
49. Zhu H, Bowen JP, He J (2002) Soundness, completeness and non-redundancy of operational semantics for Verilog based on denotational semantics. In: Proceedings of ICFEM 2002: 4th international conference on formal engineering methods. Lecture notes in computer science, vol 2495. Springer, Berlin, pp 600–612
50. Zhu H, He J (2000) A semantics of Verilog using duration calculus. In: Proceedings of international conference on software: theory and practice, pp 421–432
51. Zhu H, He J, Bowen JP (2006) From operational semantics to denotational semantics for Verilog. In: Proceedings of ICECCS 2006: 11th IEEE international conference on engineering of complex computer systems. IEEE Computer Society Press, Los Angeles, pp 139–151
52. Zhu H, He J, Li J (2007) Unifying denotational semantics with operational semantics for web services. In: Proceedings of ICDCIT 2007: 4th international conference on distributed computing and internet technology, Bangalore, India, 17–20 December. Lecture notes in computer science, vol 4882. Springer, Berlin, pp 225–239
53. Zhu H, He J, Li J, Bowen JP (2007) Algebraic approach to linking the semantics of web services. In: Proceedings of SEFM 2007: 5th IEEE international conference on software engineering and formal methods. IEEE Computer Society Press, Los Angeles, pp 315–326
54. Zhu H, He J, Pu G, Li J (2007) An operational approach to BPEL-like programming. In: Proceedings of SEW-31: 31st IEEE software engineering workshop, Baltimore, USA. IEEE Computer Society Press, Los Angeles, pp 236–245