Towards Formal Interfaces for Web Services with Transactions *

Zhenbang Chen, Ji Wang, Wei Dong, and Zhichang Qi

National Laboratory for Parallel and Distributed Processing,
Changsha 410073, China
{z.b.chen,jiwang,dong.wei}@mail.edu.cn

Abstract. The accuracy of interface description is very important to service composition and dynamic selection of service-oriented systems. It is desirable to describe Web service formally so as to improve the ability of service orchestration. This paper presents a formal interface theory for specifying Web service by extending the existing with the ability to model interface behaviour with transactions at the levels of signature, conversation and protocol. Signature interface and conversation interface model the static invocation relations in Web service interfaces, and protocol interface describes the temporal invocation information. A formal semantics of protocol interface is presented. Based on the semantics, the protocol interface can be transformed into a Labeled Transition System (LTS). Additionally, the compatibility and substitutivity relation conditions between Web services are also proposed.

Key words: Web service, Interface theory, Composition, Transaction

1 Introduction

Service orientation is a new trend in software engineering [1]. It makes the separation of service provider and requester possible, and allows the run-time composition of services. Meanwhile, it is a challenge to understand and ensure a high confidence of service-oriented software systems.

Web service is emerging as a standard framework for service-oriented computing. Business integration raises the need for Web service composition languages such as BPEL4WS (BPEL) [2]. There are many works on formalization and verification for Web service composition languages, and their main aim is to ensure the correctness of Web service composition.

In service-oriented architecture, service providers publish the service interface descriptions to the service registry. It is important to ensure the interface accuracy for service-oriented computing. As a formal foundation of component-based design, de

* Supported by the National Natural Science Foundation of China under Grant No.60233020, 60303013, 90612009; the National High-Tech Research and Development Plan of China under Grant No.2005AA113130; the National Grand Fundamental Research 973 Program of China under Grant No.2005CB321802; Program for New Century Excellent Talents in University under Grant No.NCET-04-0996.

Alfaro and Henzinger [3] proposed a theory of interface automata for specifying the component interfaces. Recently, Beyer et al. [4] presented a Web service interface description language, which can describe the Web service interfaces in three levels, i.e. signature, consistency and protocol. However, transactions are not considered in the existing interface theories, though it is one of the essential features in distributed computing such as Web service systems. How to describe transaction information of Web service is a problem. Web service-based transactions differ from traditional transactions in that they execute over long periods, require commitments to the transactions to be "negotiated" at runtime, and isolation levels must be relaxed [5].

The main contribution of this paper is to extend the formalism of Web service interfaces proposed in [4] for describing the transaction information in all three levels of signature, conversation and protocol. In each level, we separate the transaction description from the normal behaviour description. This separation makes the transaction information of Web service interfaces to be easier to describe and maintain. The compatibility and substitutivity relation conditions of the Web service interfaces are proposed for supporting Web service system development.

The rest of this paper is organized as follows. Section 2 presents the framework of the interface theory for Web services with transactions, including the signature interface, conversation interface and protocol interface. As a key element, Section 3 gives a complete semantics of protocol interface. Section 4 exemplifies the theory by a case study. In Section 5, related work is reviewed and compared with ours. Section 6 concludes the paper and presents some future work.

2 Web Service Interface Theory

A Web service interface description contains some method declarations, and clients can use the functionalities of Web services through method calls. A Web service may provide or request some methods which may return some different values. An *action* is one case of a method call. From the perspective of *action*, the interface behaviour of Web service contains *three parts*. The *first part* is the normal behaviour of action invocations. If an exception action is invoked and completes, it will be handled by its corresponding fault handling behaviour, which is the *second part* of the interface behaviour. If an exception action can invoke some successful actions before the exception occurrence, the successful actions should be compensated by the corresponding compensation behaviour, which is the *third part* of the interface behaviour.

There are different detailed interface descriptions from Web service providers. For this reason, we propose the interface theory for describing the transaction information at three different abstract levels of signature, conversation and protocol. Inspired by the ideas of Aspect-Oriented Programming (AOP) [6], we separate the descriptions of fault handling and compensation behaviour from those of normal behaviour in the interface description. In the interface semantics, the fault handling and compensation behaviour can be weaved with the normal behaviour to describe the transaction information.

Let $\mathcal{M}$ be a finite set of Web methods, $\mathcal{O}$ be a finite set of outcomes, and $dom(f)$ denote the domain of the function f. Each level of the interface theory will be presented as follows.

Definition 1 (Signature Interface, SI). *A signature interface $\mathcal{P}$ = ($\mathcal{A}$, $\mathcal{S}$, $\mathcal{S}_\mathcal{C}$, $\mathcal{S}_\mathcal{F}$), where $\mathcal{A} \subseteq \mathcal{M} \times \mathcal{O}$ is a set of actions that can appear in $\mathcal{P}$, $\mathcal{S} : \mathcal{A} \to 2^\mathcal{A}$ is a partial function that assigns to an action a a set of actions that can be invoked by a, $\mathcal{S}_\mathcal{C} : \mathcal{A} \to 2^\mathcal{A}$ is a partial function that assigns to an action a a set of actions that can be invoked by the compensation for a, $\mathcal{S}_\mathcal{F} : \mathcal{A} \to 2^\mathcal{A}$ is a partial function that assigns to an action a a set of actions that can be invoked by the fault handling for a, and $dom(\mathcal{S}_\mathcal{C}) \cap dom(\mathcal{S}_\mathcal{F}) = \emptyset$, $dom(\mathcal{S}_\mathcal{C}) \cup dom(\mathcal{S}_\mathcal{F}) = dom(\mathcal{S})$.*

Signature interface describes the direct invocation relation of Web service interfaces. An action may have different *types*. An action $a \in \mathcal{A}$ is a *supported action* if $\mathcal{S}(a)$ is defined. A Web method $m \in \mathcal{M}$ is a *supported method* if there exists a supported action $a = \langle m, o \rangle$. An action a is a *success action* if $\mathcal{S}_\mathcal{C}(a)$ is defined. An action a is an *exception action* if $\mathcal{S}_\mathcal{F}(a)$ is defined. An action a is a *required action* if it can be invoked by a supported action or compensation or fault handling, which can be expressed by the formula defined as follows:

$$required(a') = (\exists a \in dom(\mathcal{S}).\ a' \in \mathcal{S}(a)) \vee (\exists a \in dom(\mathcal{S}_\mathcal{C}).\ a' \in \mathcal{S}_\mathcal{C}(a)) \vee (\exists a \in dom(\mathcal{S}_\mathcal{F}).\ a' \in \mathcal{S}_\mathcal{F}(a)).$$

Service registries often require service providers to publish solid interface descriptions. Well-formedness can be used to assure the integrity. A signature interface is *well-formed* if the following conditions hold: every required action whose method is a supported method is a supported action, and no exception action can be invoked in compensation or fault handling.

Given two Web service interfaces, it is desirable to check whether they can cooperate properly. First, two Web services cannot support the same actions. Second, the new Web service interface, which is composed of them, should be well-formed. Formally, given two signature interfaces $\mathcal{P}_1 = (\mathcal{A}_1, \mathcal{S}_1, \mathcal{S}_{\mathcal{C}1}, \mathcal{S}_{\mathcal{F}1})$ and $\mathcal{P}_2 = (\mathcal{A}_2, \mathcal{S}_2, \mathcal{S}_{\mathcal{C}2}, \mathcal{S}_{\mathcal{F}2})$, they are *compatible* (denoted by $comp(\mathcal{P}_1, \mathcal{P}_2)$) if the following conditions are satisfied: $dom(\mathcal{S}_1) \cap dom(\mathcal{S}_2) = \emptyset$, and $\mathcal{P}_c = \mathcal{P}_1 \cup \mathcal{P}_2 = (\mathcal{A}_1 \cup \mathcal{A}_2, \mathcal{S}_1 \cup \mathcal{S}_2, \mathcal{S}_{\mathcal{C}1} \cup \mathcal{S}_{\mathcal{C}2}, \mathcal{S}_{\mathcal{F}1} \cup \mathcal{S}_{\mathcal{F}2})$ is well-formed. If two signature interfaces $\mathcal{P}_1$ and $\mathcal{P}_2$ are compatible, their composition (denoted by $\mathcal{P}_1 \parallel \mathcal{P}_2$) is $\mathcal{P}_c$. The composition operator is commutative and associative.

To enable top-down design, it is desirable to replace a Web service in a system (environment) with a new Web service without affecting the running of the system. After replacement, all parts of the system can still cooperate properly as before. Intuitively, the supported, success and exception actions are the *guarantees* of the Web service, and the required actions are the *assumptions* of the environment. The replacing Web service should guarantee more and assume fewer than the replaced Web service.

Given two signature interfaces $\mathcal{P}_1 = (\mathcal{A}_1, \mathcal{S}_1, \mathcal{S}_{\mathcal{C}1}, \mathcal{S}_{\mathcal{F}1})$ and $\mathcal{P}_2 = (\mathcal{A}_2, \mathcal{S}_2, \mathcal{S}_{\mathcal{C}2}, \mathcal{S}_{\mathcal{F}2})$, $\mathcal{P}_2$ *refines* $\mathcal{P}_1$ ($\mathcal{P}_2 \preccurlyeq \mathcal{P}_1$) if the following conditions are satisfied:for every $a \in \mathcal{A}$, if $\mathcal{P}_1$ supports a, then $\mathcal{P}_2$ supports a; for every $a \in \mathcal{A}$, if a is a success action in $\mathcal{P}_1$, then a is a success action in $\mathcal{P}_2$; for every $a \in \mathcal{A}$, if a is an exception action in $\mathcal{P}_1$, then a is an exception action in $\mathcal{P}_2$; for every $a, a' \in \mathcal{A}$, and $a \in dom(\mathcal{S}_1)$, if $a' \in \zeta_2(a)$, where $\zeta_2 \in \{\mathcal{S}_2, \mathcal{S}_{\mathcal{C}2}, \mathcal{S}_{\mathcal{F}2}\}$, then $a' \in \zeta_1(a)$; for every unsupported Web method $m \in \mathcal{M}$ in $\mathcal{P}_2$, if $\langle m, o \rangle$ is a required action in $\mathcal{P}_2$, then $\langle m, o \rangle$ is a required action in $\mathcal{P}_1$.

The first three conditions ensure that the replacing Web service guarantees every action guaranteed by the replaced one. The last two conditions ensure that every required action in $\mathcal{P}_2$ is required by $\mathcal{P}_1$, and they describe that $\mathcal{P}_2$ assumes fewer actions which

are supported by environment than $\mathcal{P}_1$. Given three signature interfaces $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$, if $comp(\mathcal{P}_1, \mathcal{P}_3)$, $comp(\mathcal{P}_2, \mathcal{P}_3)$, and $\mathcal{P}_2 \preccurlyeq \mathcal{P}_1$, then $\mathcal{P}_2 \parallel \mathcal{P}_3 \preccurlyeq \mathcal{P}_1 \parallel \mathcal{P}_3$.

An action may invoke different action sets in different conditions. Signature interface cannot describe this feature. Conversation interface is proposed for specifying different cases of action invocation. A conversation is a set of actions that are invoked together. Propositional formulae are used to represent different conversations. The set of *conversation expressions* over an action set $\mathcal{A}$ is given by the following grammar, where $a \in \mathcal{A}$.

$$\omega :: \top \mid a \mid \omega_1 \sqcup \omega_2 \mid \omega_1 \sqcap \omega_2$$

$\top$ is the propositional constant, which represents no action is needed to be invoked. Action a represents a single action is needed to be invoked. The expression $\omega_1 \sqcup \omega_2$ represents that each conversation represented by ω_1 or ω_2 can be invoked. The expression $\omega_1 \sqcap \omega_2$ represents that one conversation must contain each conversation from ω_1 and ω_2. The set of all conversation expressions on the action set $\mathcal{A}$ is denoted by $\omega(\mathcal{A})$.

A conversation interface $\mathcal{I}$ = ($\mathcal{A}$, $\mathcal{E}$, $\mathcal{E}_\mathcal{C}$, $\mathcal{E}_\mathcal{F}$), where $\mathcal{A} \subseteq \mathcal{M} \times \mathcal{O}$ is a set of actions that can appear in $\mathcal{I}$, $\mathcal{E}$, $\mathcal{E}_\mathcal{C}$ and $\mathcal{E}_\mathcal{F}$ are partial functions, whose definitions are same as $\mathcal{A} \rightarrow \omega(\mathcal{A})$. The meanings of functions are similar to those of signature interface, except that each function assigns to an action a conversation expression to describe the interface behaviour.

A conversation is a set of actions, which do not have any sequence information. [4] presented *protocol interface* to depict the sequences of actions. In this paper, the protocol interface is extended to enable transaction description. In Web service, the modes of action invocations include thread creation, choice, parallel executions, join after parallel execution, and sequence etc. We use terms to represent these different modes. The set of terms over an action set $\mathcal{A}$ is given by the following grammar, where $a,\ b \in \mathcal{A}$.

$$term :: \tau \mid a \mid a \sqcup b \mid a \sqcap b \mid a \boxplus b \mid a\ ;\ b \mid [term]$$

The set of all terms over $\mathcal{A}$ is denoted by $Term(\mathcal{A})$, and $[[term]] = [term]$. The term τ is an ***Empty term***, which represents that no action is invoked. The term a=$\langle m, o \rangle$ is a ***Call term***, which represents a call to Web method m with expected outcome o. The term $a \sqcup b$ is a ***Choice term***, which represents a nondeterministic choice between actions a and b. The term $a \sqcap b$ is a ***Fork term***, which represents parallel invocations of actions a and b, and the parent waits for both actions to return. If any action is an exception action, the parent fails. The term $a \boxplus b$ is a ***Fork-Choice term***, which represents parallel invocations of actions a and b, while the return of any action will return the parent. $a \boxplus b$ fails only when both actions are exception actions. The term $a\ ;\ b$ is a ***Sequence term***, which represents two sequential calls. The term $[term]$ is a ***Transaction term***, which represents that any exception action invoked from the term in the brackets will cause the compensation or fault handling to the actions which are invoked before from the the term in the brackets. For the sake of simplicity, it is assumed that the term of transaction term must result in exceptions.

The sequences of invocations between Web services can be specified in automata. To indicate the place where exceptions occur, we propose *extended protocol automata* as follows.

Definition 2 (Extended Protocol Automata, EPA). *An extended protocol automaton G is a triple $(\mathcal{A}, \mathcal{L}, \delta)$, where $\mathcal{A}$ is a set of actions, $\mathcal{L}$ is a set of locations, there are two special locations $\perp, \boxtimes$ in $\mathcal{L}$, $\perp \in \mathcal{L}$ is the return location, and $\boxtimes \in \mathcal{L}$ is the exception location, $\delta \subseteq (\mathcal{L} \setminus \{\perp, \boxtimes\}) \times Terms(\mathcal{A}) \times \mathcal{L}$ is the transition relation set.*

A location is terminating in *EPA* if there exists a trace starting from the location and ending with $\perp$ or $\boxtimes$. Based on *EPA*, we define protocol interface as follows.

Definition 3 (Protocol Interface, PI). *A protocol interface $\mathcal{T}$ is 4-tuple $(G, \mathcal{R}, \mathcal{R}_\mathcal{C}, \mathcal{R}_\mathcal{F})$, where G is an extended protocol automaton to specify interface behaviour, $\mathcal{R} : \mathcal{A} \to \mathcal{L}$ is a partial function that assigns to an action the start location in G, $\mathcal{R}_\mathcal{C} : \mathcal{A} \to \mathcal{L}$ is a partial function that assigns to an action a the start location in G of the compensation for a, $\mathcal{R}_\mathcal{F} : \mathcal{A} \to \mathcal{L}$ is a partial function that assigns to an action a the start location in G of the fault handling for a, and $dom(\mathcal{R}_\mathcal{C}) \cap dom(\mathcal{R}_\mathcal{F}) = \emptyset$, $dom(\mathcal{R}_\mathcal{C}) \cup dom(\mathcal{R}_\mathcal{F}) = dom(\mathcal{R})$.*

A location is terminating in *PI* if it is terminating in *EPA* and the start location of each action in the terminating trace is also terminating in *PI*. Given a protocol interface $\mathcal{T} = (G, \mathcal{R}, \mathcal{R}_\mathcal{C}, \mathcal{R}_\mathcal{F})$, the *underlying signature interface* of $\mathcal{T}$ (denoted by $psi(\mathcal{T})$) is $(\mathcal{A}_s, \mathcal{S}, \mathcal{S}_\mathcal{C}, \mathcal{S}_\mathcal{F})$, where $\mathcal{A}_s = \mathcal{A}$; $\mathcal{S}(a) = sigl(\mathcal{R}(a))$ if $\mathcal{R}(a)$ is defined, otherwise $\mathcal{S}(a)$ is undefined; $\mathcal{S}_\mathcal{C}(a) = sigl(\mathcal{R}_\mathcal{C}(a))$ if $\mathcal{R}_\mathcal{C}(a)$ is defined, otherwise $\mathcal{S}_\mathcal{C}(a)$ is undefined; $\mathcal{S}_\mathcal{F}(a) = sigl(\mathcal{R}_\mathcal{F}(a))$ if $\mathcal{R}_\mathcal{F}(a)$ is defined, otherwise $\mathcal{S}_\mathcal{F}(a)$ is undefined. The function $sigl : \mathcal{L} \to 2^\mathcal{A}$ is defined as follows:

$sigl(\perp) = \emptyset, sigl(\boxtimes) = \emptyset, sigl(q) = \bigcup_{\exists(q, term, q') \in \delta} \varphi(term) \cup sigl(q')$,

$\varphi(\tau) = \emptyset, \varphi(a) = \{a\}, \varphi([term]) = \varphi(term), \varphi(a \square b) = \{a, b\}, \square \in \{\sqcup, \sqcap, \boxplus, ;\}$.

A protocol interface $\mathcal{T}$ is *well-formed* if the following conditions hold: $psi(\mathcal{T})$ is well-formed; if $a \in dom(\mathcal{R})$, then $\mathcal{R}(a)$ is terminating; if $a \in dom(\mathcal{R}_\mathcal{C})$, then $\mathcal{R}_\mathcal{C}(a)$ is terminating; if $a \in dom(\mathcal{R}_\mathcal{F})$, then $\mathcal{R}_\mathcal{F}(a)$ is terminating. For the sake of simplicity, it is assumed that: no transaction term can be invoked by exception action, nor can it be invoked by compensation or fault handling; transaction term cannot be invoked recursively or parallelly. The types of an action a in a protocol interface $\mathcal{T}$ are same as those of a in $psi(\mathcal{T})$.

Given two protocol interfaces $\mathcal{T}_1 = (G_1, \mathcal{R}_1, \mathcal{R}_{\mathcal{C}1}, \mathcal{R}_{\mathcal{F}1})$ and $\mathcal{T}_2 = (G_2, \mathcal{R}_2, \mathcal{R}_{\mathcal{C}2}, \mathcal{R}_{\mathcal{F}2})$, they are *compatible* if the following conditions are satisfied: $psi(\mathcal{T}_1)$ and $psi(\mathcal{T}_2)$ are compatible, and $\mathcal{L}_1 \cap \mathcal{L}_2 = \{\perp, \boxtimes\}$; $\mathcal{T}_c = \mathcal{T}_1 \cup \mathcal{T}_2 = (G_1 \cup G_2, \mathcal{R}_1 \cup \mathcal{R}_2, \mathcal{R}_{\mathcal{C}1} \cup \mathcal{R}_{\mathcal{C}2}, \mathcal{R}_{\mathcal{F}1} \cup \mathcal{R}_{\mathcal{F}2})$ is well-formed, where $G_1 \cup G_2 = (\mathcal{A}_1 \cup \mathcal{A}_2, \mathcal{L}_1 \cup \mathcal{L}_2, \delta_1 \cup \delta_2)$. If $\mathcal{T}_1$ and $\mathcal{T}_2$ are compatible (denoted by $comp(\mathcal{T}_1, \mathcal{T}_2)$), their composition (denoted by $\mathcal{T}_1 \parallel \mathcal{T}_2$) is $\mathcal{T}_c$. The composition operator is commutative and associative. The substitutivity relation between protocol interfaces should be defined based on the semantics to ensure the temporal correctness.

Signature interface and conversation interface describe the static invocation relations of Web service interfaces, and their semantics are simple and show the static aspects of the Web service interface. Protocol interface describes dynamic invocations in Web service interfaces. The interface behaviour of protocol interface should ensure not only the invocation process should been recorded for compensation or fault handing, but also the sequence of compensation and fault handling should agree to the long-running transaction model.

3 The Semantics of Protocol Interface

The action invocation process is a *pushdown* system which can continue only after the completion of every invoked action. The sequence of compensation and fault handling should be reverse of the sequence of the previous invocations, and the recorded actions should be first in last out. Therefore, we use a binary tree nested by a stack to interpret the protocol interface behaviour.

A *binary tree* over a finite set of labels L is a partial function $t : \mathbb{B}^* \rightarrow L$, where $\mathbb{B}^*$ denotes the set of finite words over $\mathbb{B} = \{0, 1\}$, and ρ denotes the empty word. $\mathbb{T}(L)$ denotes the set of all trees over a finite label set L. A *stack* over a finite set of labels L is a partial function $s(m) : \mathbb{N} \rightarrow L$, where $\mathbb{N}$ is the natural number set, and $dom(s(m)) = \{n \mid n < m \wedge n \in \mathbb{N}\}$. $s(0)$ is the empty stack. $s(m)(m-1)$ is the top element of the stack $s(m)$. $\mathbb{S}(L)$ denotes the set of all stacks over a finite label set L.

Given a protocol interface $(G, \mathcal{R}, \mathcal{R}_\mathcal{C}, \mathcal{R}_\mathcal{F})$, its semantics is defined by a *labeled transition system* (LTS). The set of states is $\mathbb{T}(Q_t) \times \mathbb{B}^* \times \mathbb{S}(Term(\mathcal{A}))$, that is, the Cartesian products of trees over $Q_t = \mathcal{L} \times \mathcal{A}^* \times \wp$, the set of tree nodes $\mathbb{B}^*$, and stacks over $Term(\mathcal{A})$, where $\mathcal{L}$ is the location set of $EPA\ G$, $\mathcal{A}^*$ is the set of words over the action set $\mathcal{A}$ in G, $\wp = \{\circ, \circledcirc, \circledcirc_c, \circledcirc_d, \boxplus, \boxplus_c, \boxplus_d, \triangle, \nabla, \odot, \boxdot\}$ is the node type set, and the tree node in $\mathbb{B}^*$ is the corresponding node of the stack. The *underlying transition relation* of $\mathcal{T}$ is a transition relation $\rightarrow_\mathcal{T} \subseteq (\mathbb{T}(Q_t) \times \mathbb{B}^* \times \mathbb{S}(Term(\mathcal{A}))) \times 2^{\mathcal{A} \cup \{ret, exp, cfstart, end\}} \times (\mathbb{T}(Q_t) \times \mathbb{B}^* \times \mathbb{S}(Term(\mathcal{A})))$. The label of state transition is the set of elements from $\mathcal{A} \cup \{ret, exp, cfstart, end\}$. We write $\nu \rightarrow_\mathcal{B} \nu'$ for $(\nu, \mathcal{B}, \nu') \in \rightarrow_\mathcal{T}$, where $\nu = (t, \Psi, s(n))$ and $\mathcal{B} \subseteq \mathcal{A} \cup \{ret, exp, cfstart, end\}$. The transition rules have two parts, the first part consists of the rules for normal behaviour transitions, and the second part consists of the rules for transitions made by compensation or fault handling. The second part weaves the transaction behaviour into the normal behaviour.

The beginning of normal behaviour is the invocation of a supported action. The initial state is a tree that has only one node and a stack whose content is decided by the type of the supported action. Supposing we start from invoking a supported action a, if a is a success action, the initial state is $\nu_{inital} = (t_{inital}, \Psi, s(0)) = (\{(\rho, (\mathcal{R}(a), \rho, \circ))\}, \rho, \emptyset)$, else if a is an exception action, the initial state is $\nu_{inital} = (t_{inital}, \Psi, s(1)) = (\{(\rho, (\mathcal{R}(a), \rho, \circ))\}, \rho, \{(0, a)\})$. The operations in normal transaction rules can be divided into two parts: *tree operations* and *stack operations*. The tree operations depict a pushdown system. Only leaf nodes of the tree can be operated. Call, Choice and Sequence terms lead to pushing down the leaf node. Fork and Fork-Choice terms lead to branching the leaf node. Transaction term leads to pushing down the leaf node with a transaction node. For pushdown operations, if current node location is reached from a success supported action, or the leaf node is not pushed from a transaction term node, no stack operations is needed. If current node location is reached from a supported exception action, or the node is branched from a transaction term node and the transaction node is not under compensation or fault handling, stack operations is needed. Main operations in the normal rules are illustrated in Figure 1.

There are five normal transition rules. As a shorthand, the rules (**Pushdown**) and (**Exception**) are listed in Appendix. The rule (**Pushdown**) describes the operations of different invoked terms. The rule (**Exception**) describes that when an exception location is reached, some coordination should be taken. There are two complicated

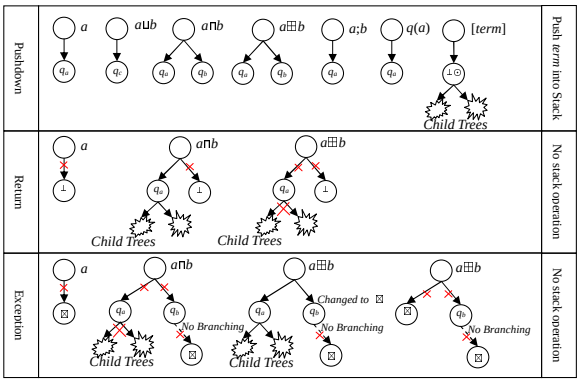

Fig. 1. Illustrations of operations in normal transition rules.

cases. The first case is that the exception location is reached from a Fork term, and it will cause the global exception and the other branch should be terminated. Another case is that the exception location is reached from a Fork-Choice term, and whether it can cause the global exception is determined by the other branch. If the other branch returns successfully, the parent is successful. If the other branch does not return, this branch should wait until the return of the other branch. If exception also occurs in the other branch, the global exception occurs. The invocation of the unsupported action is supposed to return immediately.

The compensation and fault handling occur in the condition that some exception actions have been invoked. After normal transitions, actions in the execution must return, and the state of the execution must reach $(\{(\rho, (\bot, \rho, \circ))\}, \rho, s(0))$ or $(\{(\rho, (\boxtimes, \rho, \circ))\}, \rho, s(n))$ with $n > 0$. If it is $(\{(\rho, (\bot, \rho, \circ))\}, \rho, s(0))$, the invocation is successful, and no compensation or fault handling is needed. If the state is $(\{(\rho, (\boxtimes, \rho, \circ))\}, \rho, s(n))$, some exceptions occur in the invocation process, and compensation or fault handling should be taken. After compensation or fault handling, the state must finally reach $(\{(\rho, (\bot, \rho, \circ))\}, \rho, s(0))$, which represents that the whole invocation process completes. If the invocation returns on a node whose corresponding label is $(\boxtimes, \rho, \odot)$, $\Phi \neq \rho$, and the stack is not empty, it represents that a transaction term invocation has returned, and compensation or fault handling should be taken.

It is assumed that no exception will appear in compensation or fault handling. The compensation and fault handling do not need stack operation. The operations in compensation and fault handling transition rules are simpler than those in normal transition rules. The transition rules specify that the recorded term in the stack should be popped out in sequence, and whether compensation or fault handling will be taken is determined by the action type. After completing compensation and fault handling, whether the invocation process terminates is determined by the exception reason. If the beginning action is a supported exception action, the invocation process terminates. If the exception is resulted from a *Transaction term*, the invocation process will continue.

Based on the transition rules, we can use the *LTS* simulation relation to define the substitutivity relation of protocol interfaces. A labeled transition system is a 4-tuple (S, I, L, Δ), where S is the set of states, $I \subseteq S$ is the set of initial states, L is the set

of labels, and $\Delta \subseteq S \times L \times S$ is the transition relation set. Given a protocol interface $\mathcal{T} = (G, \mathcal{R}, \mathcal{R}_\mathcal{C}, \mathcal{R}_\mathcal{F})$, and a supported action a, the *underlying labeled transition system* of invoking a (denoted by $LTS(\mathcal{T}, a)$) is $(S_a, I_a, L_a, \Delta_a)$, which can be given as follows: $S_a = \mathbb{T}(Q_t) \times \mathbb{B}^* \times \mathbb{S}(Term(\mathcal{A}))$; if a is a success action, $I_a = \{(\{(\rho, (\mathcal{R}(a), \rho, \circ))\}, \rho, \emptyset)\}$; if a is an exception action, $I_a = \{(\{(\rho, (\mathcal{R}(a), \rho, \circ))\}, \rho, \{(0, a)\})\}$; $L_a = 2^{\mathcal{A} \cup \{ret, exp, cfstart, end\}}$; Δ_a is the underlying transition relation set of $\mathcal{T}$ using the transition rules.

Because ret, exp, $cfstart$, and end are not external Web service actions, the transitions labeled by them do not assume to the environment. The simulation relation of the underlying labeled transition systems can be extended to relax the conditions that substitutivity should satisfy. We denote $(s_1, a, s_1') \in \Delta$ as $s_1 \rightarrow_a s_1'$. If $t = a_1 a_2 ... a_n \in L^*$, $s_1 \rightarrow_{a_1} \rightarrow_{a_2} ... \rightarrow_{a_n} s_1'$ is denoted as $s_1 \rightarrow_t s_1'$.

Given two *LTSs* $M_1 = (S_1, I_1, L_1, \Delta_1)$ and $M_2 = (S_2, I_2, L_2, \Delta_2)$ and a label set W, M_2 is *weakly simulated* by M_1 over the label set W if there exists a relation $\preccurlyeq_W \subseteq S_1 \times S_2$ such that: for every $s_1 \in M_1, s_2 \in M_2$, if $s_2 \preccurlyeq_W s_1$, then for every $(s_2, a, s_2') \in \Delta_2$, there exists $s_1 \rightsquigarrow_a s_1'$ in Δ_1, such that $s_2' \preccurlyeq_W s_1'$, where $s_1 \rightsquigarrow_a s_1'$ represents $s_1 \rightarrow_t s_1'$, in which $t = a_1 a_2 ... a_n$, and there exists only one a_i that $a_i = a$, and all the other labels are all in W; for every $s_2 \in M_2$, there exists $s_1 \in M_1$ such that $s_2 \preccurlyeq_W s_1$.

Given two protocol interfaces $\mathcal{T}_1 = (G_1, \mathcal{R}_1, \mathcal{R}_{\mathcal{C}1}, \mathcal{R}_{\mathcal{F}1})$ and $\mathcal{T}_2 = (G_2, \mathcal{R}_2, \mathcal{R}_{\mathcal{C}2}, \mathcal{R}_{\mathcal{F}2})$, we say $\mathcal{T}_2$ *refines* $\mathcal{T}_1$ ($\mathcal{T}_2 \preccurlyeq \mathcal{T}_1$) if the following conditions are satisfied: $psi(\mathcal{T}_2) \preccurlyeq psi(\mathcal{T}_1)$, and for every $a \in dom(\mathcal{R}_1)$, $LTS(\mathcal{T}_2, a)$ is weakly simulated by $LTS(\mathcal{T}_1, a)$ over $2^{\{ret, exp, cfstart, end\}}$. Given three protocol interfaces $\mathcal{T}_1$, $\mathcal{T}_2$, and $\mathcal{T}_3$, if $comp(\mathcal{T}_1, \mathcal{T}_3)$, $comp(\mathcal{T}_2, \mathcal{T}_3)$, and $\mathcal{T}_2 \preccurlyeq \mathcal{T}_1$, then $\mathcal{T}_2 \parallel \mathcal{T}_3 \preccurlyeq \mathcal{T}_1 \parallel \mathcal{T}_3$.

4 Case Study

Figure 2 shows a classical Web service-based system, supply chain management system. The system is composed of six Web services. Each labeled arrow from one service to another indicates the Web method call from the caller to the callee. *Shop* supports the Web method *SellItem* that can be called by the *Client* to start the selling process. When the selling process starts, the *Shop* will first check the availability of items to be sold by calling the method *ChkAvail*, which requires the Web method *ChkStore* implemented by *Store* to check whether the desirable items are in stock and deduct the number of items if the stock checking is successful. If the stock is inadequate, the selling process fails. If the stock is inadequate or the stock after deducting is below a certain amount, the *Store* department will make an order from the *Supplier* and get some new items. If the availability checking is successful, the *Shop* will parallelly process the payment by calling the method *ProcPay* and the delivery by calling the method *ShipItem*. *ProcPay* is implemented by the *Bank* and its success can be compensated by calling the method *Compensate*. *ProcPay* is implemented by *Transport* and its success can be compensated by calling the method *Withdraw*. If all the above steps are successful, the selling process is successful, otherwise the successful steps before failure should be compensated and the failed steps should be handled. For instance, the *Shop* will call the method *Apologize* implemented by itself to send an apologetic letter to the *Client* because of the failure of the selling process. After composition, the supply chain management system contains six Web services. The protocol interface for the system and a trace of the action

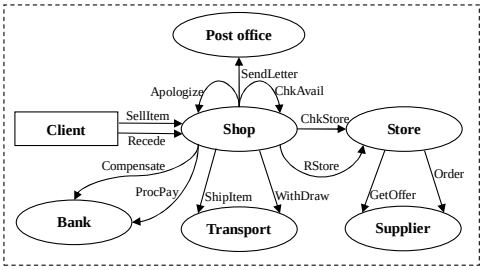

Fig. 2. Supply chain management system.

⟨SellItem, FAIL⟩ invocation is shown in Figure 3, and the reason of purchasing failure is the invocation of action ⟨ProcPay, FAIL⟩.

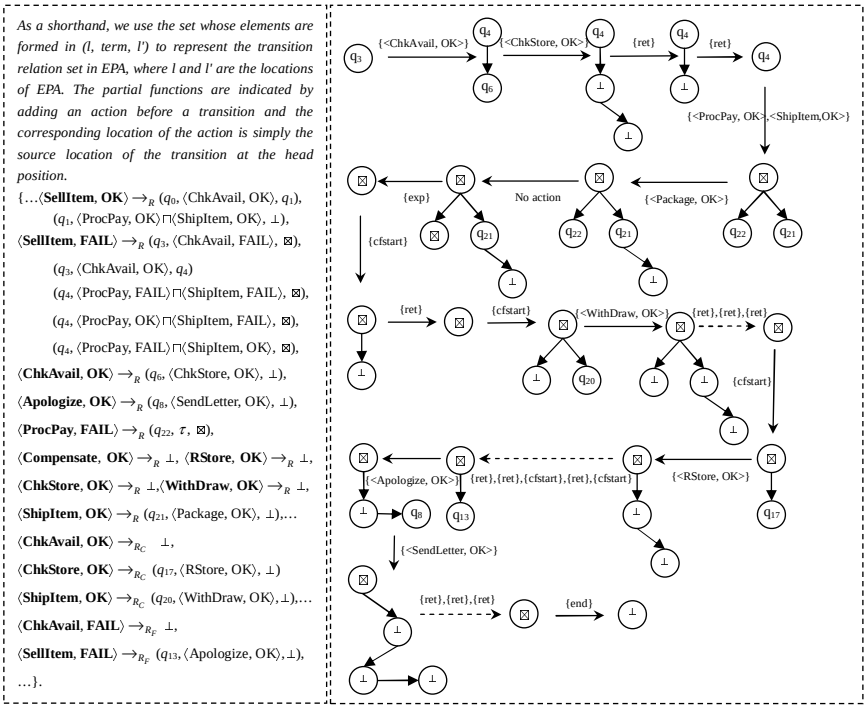

Fig. 3. The protocol interface for the supply chain management system and a trace of the action ⟨*SellItem,FAIL*⟩ invocation.

5 Related Work

There are many researches on formalization and verification of Web service interface behaviour, which can be divided by the underlying semantic theory as follows:

Petri net. [8] used Petri net to formalize the Web service compositions. [9] used WF-nets (workflow nets) to formalize the BPEL description, and a mapping from BPEL process model to WF-net was proposed.

Process algebra. Foster [10] used finite state process (FSP) to formalize BPEL and WS-CDL [11], and the specifications are verified on LTSA WS-Engineer, which can perform safety and liveness analysis, and interface compatibility checking. In [12], BPEL-Calculus was proposed to formalize BPEL, and the description can be verified on Concurrency WorkBench (CWB) by a syntax compiler plug-in for BPEL-Calculus.

Automata. In [13], guarded finite state automata (GFSA) was used to describe service composition, and the description can be translated to Promela which can be verified on SPIN. In [14], hierarchical state machine (HSM) was used to specify Web service interfaces, and Java PathFinder (JPF) was used to verify that the implementation of each peer of Web service system conforms to its interface, and the interface behaviour models can be verified on SPIN.

The above approaches are deficient in modeling transaction behaviour of Web service interfaces, especially in compensation and fault handling. Because of the deficiency in formalism, the above verification methods cannot verify the properties which specify transaction behaviour part in Web service interfaces. By extending [4], the formalism presented in this paper can rigorously describe the transaction behaviour of Web service interfaces. The above verification methods mainly take into account the temporal interface behaviour and properties of Web services. In practice, some Web service providers cannot publish interfaces with temporal information. The approaches in [8–10, 12–14] cannot handle this situation. Additionally, there are some researches on formalization of long-running transactions. Butler et al. [16] extended communicating sequential process (CSP) to enable the description of long-running transactions. In [17], an enhanced Sagas language is proposed for specifying compensation in flow composition languages. Their approaches mainly aim at description of dynamic transactional behaviour. Our approach takes into account different abstract levels and separates the transaction description from the normal behaviour description.

6 Conclusions

Web service is the most popular implementing framework for service-oriented computing. In this paper, we aim at formalization of Web service interfaces with transactions. The final goal is to ensure the correctness of Web service interfaces. This paper presents an interface theory for specifying interface behaviour with transactions in Web services. The signature interface specifies the direct invocation relations and the conversation interface specifies different invocations for the same call. With protocol interface, temporal invocations can be specified. To serve as a dynamic semantics of interfaces, a set of operational rules are presented to transform the protocol interface behaviour into labeled transition systems. The relation conditions of compatibility and substitutivity between Web services are also presented in this paper. The separate description method used in this paper reflects some ideas of AOP, and different parts of behaviour can be weaved together in the semantics.

Through our approach, one can precisely describe Web service interface transaction information. The interface theory for Web services will form as a formal foundation of service-oriented software engineering, especially in the specification and verification of service-oriented systems.

Currently, a model checking based verification method has been proposed [7] and our interface theory has been be applied to BPEL [18], and the application raises some issues which need improvements of the interface theory, such as data handling description. Other ongoing and future works are to investigate an integrated formalism for both service orchestration and choreography by Web service interface theory, and to build the corresponding tools for specification and verification.

References

1. Humberto, C., Richard, S.H.: Technical Concepts of Service Orientation. *Service-Oriented Software System Engineering: Challenges and Practices*, pp. 1-47, 2005.
2. Curbera, F., et al.: *Business Process Execution Language For Web Services Version 1.1. ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf.*
3. de Alfaro, L., Henzinger, T.A.: Interface automata. In *Proc. of ESEC/FSE'01*, pp. 109-120, 2001.
4. Beyer, D., Chakrabarti, A., Henzinger, T.A.: Web Service Interfaces. In *Proc. of WWW'05*, pp. 148-159, ACM Press, 2005.
5. Little, M.: Transactions and Web Services. *Communication of the ACM*, 46(10):49-54, 2003.
6. Kiczales, G., et al.: Aspect-Oriented Programming. In *Proc. of ECOOP'97*, LNCS 1241, pp. 220-242, Springer, 1997.
7. Chen, Z.B., Wang, J., Dong, W., Qi, Z.C., Yeung, W.L.: An Interface Theory Based Approach to Verification of Web Services. In *Proc. of COMPSAC'06*, pp. 139-144, IEEE Press, 2006.
8. Hamadi, R., Benatallah, B.: A Petri Net-based Model for Web Service Composition. In *Proc. of ADC'03*, pp. 191-200, IEEE Press, 2003.
9. Verbeek, H.M.W., van der Aalst, W.M.P.: Analyzing BPEL Processes using Petri Nets. In *Proc. of the 2th International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management at the Petri Nets'05*, pp. 59-78, 2005.
10. Foster, H., Uchitel, S., Magee, J., Kramer, J.: Compatibility for Web Service Choreography. In *Proc. of ICWS'04*, pp. 738-741, IEEE Press, 2004.
11. Kavantzas, N., et al.: *Web Services Choreography Description Language Version 1.0. http://www.w3.org/TR/ws-cdl-10/.*
12. Koshkina, M.: Verification of Business Processes for Web Services. [MS. Thesis]. York University, 2003.
13. Fu, X., Bultan, T., Su, J.W.: Analysis of Interacting BPEL Web Services. In *Proc. of WWW'04*, pp. 621-630, ACM Press, 2004.
14. Betin-Can, A., Bultan, T.: Verifiable Web Services with Hierarchical Interfaces. In *Proc. of ICWS'05*, pp. 85-94, IEEE Press, 2005.
15. Beyer, D., Chakrabarti, A., Henzinger, T.A.: An Interface Formalism for Web Services. In *Proc. of the WorkShop FIT at CONCUR'05*, San Francisco, CA, 2005.
16. Butler, M., Ripon, S.: Executable Semantics for Compensating CSP. In *Proc. of WS-FM'05*, LNCS 3670, pp. 243-256, Springer, 2005.
17. Bruni, R., Melgratti, H., Montanari, U.: Theoretical Foundations for Compensations in Flow Composition Languages. In *Proc. of POPL'05*, pp.209-220, ACM Press, 2005.
18. Chen, Z.B., Wang, J., Dong, W., Qi, Z.C.: Interface Theory based Formalization and Verification of Orchestration in BPEL4WS. In *Proc. of the Workshop SOT at ICEC'06*, Fredericton, New Brunswick, Canada, 2006.

Appendix. Some Important Normal Transition Rules

Due to the space limitations, we only give two normal transition rule definitions, where

- Let pj denote the concatenation of the word p with $j \in \mathbb{B}$, and pp' denote the concatenation of the words p and p'. For a tree t and a node $p \in dom(t)$, $leaf(t) = \{p \in dom(t) \mid \forall j \in \mathbb{B}, pj \notin dom(t)\}$, $child(p)$=$\{q \mid \exists j \in \mathbb{B}, q$=$pj \wedge q \in dom(t)\}$, and $parent(p) = \{q \mid \exists j \in \mathbb{B}, p = qj \wedge q \in dom(t)\}$. $ancestor(p_1, p_2) = (p_1 \in parent(p_2)) \vee (\exists q \in parent(p_2).\ ancestor(p_1, q))$ denotes whether a node p_1 is the ancestor of another node p_2. $ancestor\text{-}y(p, \beta)$ denotes the node p's youngest ancestor whose node type is β;
- in a transition, the source state is defined as $\nu = (t, \Psi, s(n))$, and the target state as $\nu' = (t', \Psi', s'(m))$. $q(w)\beta$ represents (q, w, β) in Q_t, and if $w = \rho$, $q\beta$ is used to represent it. For example, $q\boxplus$ represents $(q, \rho, \boxplus)$;
- $\delta(q) = (a, q')$ denotes that there exists a transition relation (q, a, q') in the extended protocol automaton. $\xi(p) = \square$ if the type of the tree node p is $\square \in \wp$. If action c is supported by $\mathcal{R}$, $q_c = \mathcal{R}(c)$, otherwise $q_c = \perp$.

(Pushdown) $\nu \rightarrow_{\mathcal{M}} \nu'$

If there exists a node p such that $p \in leaf(t)$, $t(p) = q\beta$, where $\beta \in \wp$, $\delta(q) = (r, q')$, and $\Psi = \rho \vee (\Psi \neq \rho \wedge \xi(\Psi) = \odot)$:

- $r = a$: $t' = (t \setminus \{(p, q\beta)\}) \cup \{(p, q'\beta), (p0, q_a\beta)\}$, $term_1 = a$, and $\mathcal{M} = \{a\}$;
- $r = a \sqcup b$: $t' = (t \setminus \{(p, q\beta)\}) \cup \{(p, q'\beta), (p0, q_c\beta)\}$, where $c \in \{a, b\}$, $term_1 = c$, and $\mathcal{M} = \{c\}$;
- $r = a \sqcap b$: $t' = (t \setminus \{(p, q\beta)\}) \cup \{(p, q'\alpha), (p0, q_a\circledcirc_c), (p1, q_b\circledcirc_c)\}$, where $(\beta = \circ \wedge \alpha = \circledcirc) \vee (\beta = \circledcirc_c \wedge \alpha = \circledcirc_d) \vee (\beta = \boxplus_c \wedge \alpha = \nabla)$, $term_1 = a \sqcap b$, and $\mathcal{M} = \{a, b\}$;
- $r = a \boxplus b$: $t' = (t \setminus \{(p, q\beta)\}) \cup \{(p, q'\alpha), (p0, q_a\boxplus_c), (p1, q_b\boxplus_c)\}$, where $(\beta = \circ \wedge \alpha = \boxplus) \vee (\beta = \boxplus_c \wedge \alpha = \boxplus_d) \vee (\beta = \circledcirc_c \wedge \alpha = \triangle)$, $term_1 = a \boxplus b$, and $\mathcal{M} = \{a, b\}$;
- $r = a; b$: $t' = (t \setminus \{(p, q\beta)\}) \cup \{(p, q'(b)\beta), (p0, q_a\beta)\}$, $term_1 = a$, and $\mathcal{M} = \{a\}$;
- if $t(p) = q(a)\beta$, where $\beta \in \wp$, then $t' = (t \setminus \{(p, q(a)\beta)\}) \cup \{(p, q\beta), (p0, q_a\beta)\}$, and $term_1 = a, \mathcal{M} = \{a\}$.

If $n = 0 \vee (\Psi \neq \rho \wedge p \neq \Psi p')$, then $s'(m) = s(n), m = n$. If $(n > 0) \wedge ((\Psi = \rho) \vee (\Psi \neq \rho \wedge p = \Psi p' \wedge \xi(\Psi) = \odot))$, then $m = n + 1$, and $s'(m) = s(n) \cup \{(n, term_1)\}$.

(Exception) $\nu \rightarrow_{\{exp\}} \nu'$

If there exists a node $p\theta$ such that $p\theta \in leaf(t)$, where $\theta \in \mathbb{B}$, $t(p\theta) = \boxtimes\beta$, where $\beta \in \wp$, and $\Psi = \rho \vee (\Psi \neq \rho \wedge \xi(\Psi) = \odot)$:

- $\beta = \circ$ and $t(p) = q\beta$: $t' = (t \setminus \{(p\theta, \boxtimes\circ), (p, q\beta)\}) \cup \{(p, \boxtimes\beta)\}$;
- $\beta = \circledcirc_c$, $t(p_a) = q\alpha$, where $\alpha \in \{\circledcirc, \circledcirc_d, \nabla, \odot\}$, and $ancestor\text{-}y(p\theta, \alpha) = p_a$: $t' = (t \setminus \{(p_a p', q') \mid p' \in \mathbb{B}^* \wedge q' = t(p_a p')\}) \cup \{(p_a, \boxtimes\beta)\}$, where $(\alpha = \circledcirc \wedge \beta = \circ) \vee (\alpha = \circledcirc_d \wedge \beta = \circledcirc_c) \vee (\alpha = \nabla \wedge \beta = \boxplus_c) \vee (\alpha = \odot \wedge \beta = \odot)$;
- $\beta = \boxplus_c$, and $t(p_a) = q\alpha$, where $\alpha \in \{\boxplus, \boxplus_d, \triangle, \odot\}$ and $ancestor\text{-}y(p\theta, \alpha) = p_a$: if there exists $p_b \in \{p' \mid child(p_a) \wedge \neg ancestor(p', p)\}$, and $\#child(p_b) > 0 \vee \xi(p_b) = \perp$, then $t' = (t \setminus \{(p_c p', q') \mid p' \in \mathbb{B}^* \wedge q' = t(p_c p')\}) \cup \{(p_c, \boxtimes\beta)\}$, where $p_c \in child(p_a) \wedge ancestor(p_c, p)$, and $t(p_c) = q^c\beta$; if all nodes in $\{p' \mid child(p_a) \wedge \neg ancestor(p', p)\}$ have no child, and the types of all nodes are same as $\boxtimes$, then $t' = (t \setminus \{(p_a p', q') \mid p' \in \mathbb{B}^* \wedge q' = t(p_a p')\}) \cup \{(p_a, \boxtimes\beta)\}$, where $(\alpha = \boxplus \wedge \beta = \circ) \vee (\alpha = \boxplus_d \wedge \beta = \boxplus_c) \vee (\alpha = \triangle \wedge \beta = \circledcirc_c) \vee (\alpha = \odot \wedge \beta = \odot)$;
- $t(p) = q(a)\beta$, where $a \in \mathcal{A}$: $t' = (t \setminus \{(p\theta, \boxtimes\beta), (p, q(a)\beta)\}) \cup \{(p, \boxtimes\beta)\}$.

In all conditions, $s'(m) = s(n)$, and $m = n$.