类型化的 Web 服务组合形式化模型*)

辜希武　卢正鼎
(华中科技大学计算机科学技术学院　武汉 430074)

摘　要　Web服务组合的正确性包括动态行为的匹配性和数据类型的一致性。本文定义了一个扩充的Pi-演算类型系统,同时利用该系统对BPEL4WS Web服务组合规范建立了一个类型化的形式化模型,通过该模型能够对Web服务组合的正确性进行验证。最后通过一个案例,给出了对Web服务组合动态行为的匹配性和数据类型的一致性的验证方法。

关键词　类型化模型,Web服务组合,Web服务商业流程执行语言

A Typed Formal Model for Web Services Composition

GU Xi-Wu　LU Zheng-Ding
(College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074)

Abstract　The correctness of Web service composition includes dynamic behavior compatibility and data type consistency. This paper defines an extended type system based on Pi-calculus, and presents a typed formal model of BPEL4WS specification on the basis of extended type system. This typed formal model can be utilized to verify the correctness of Web service composition. At the end of this paper, the verification method of dynamic behavior compatibility and data type consistency of Web service composition are introduced through a case study.

Keywords　Typed model, Web service composition, Business process execution language for Web services

1　引言

Web服务组合是实现软件复用甚至增值的一种有效方式。目前工业界已经针对Web服务组合发布了一系列的规范,如:Web服务商业流程执行语言[1](Business Process Execution Language for Web Services, BPEL4WS)、Web服务编排描述语言[2](Web Services Choreography Description Language, WS-CDL)等。但这些规范仅仅是基于XML的描述性规范,因此对于业务逻辑复杂的Web服务组合,这些规范无法保证Web服务组合的正确性。

利用形式化的方法,如进程代数来描述、验证Web服务组合,以保证各个被组合的子服务间的动态交互的正确性,是目前在Web服务组合研究领域的一个重要方向。但Web服务组合的不良性为除了动态行为不匹配而导致死锁等现象外,数据类型的不一致性也是出错的重要因素。因此,一种行之有效的方法是在传统的形式化模型中加上类型系统[3],这样可以同时保证Web服务的动态行为的匹配性和数据类型的一致性。目前很多学者在这方面开展了工作:Meredish和Bjorg[4]阐述了类型化的形式化模型对Web服务的重要性;Sangiorgi和Walker[5]在Pi-演算的基础上定义了一个简单的类型系统Simply Typed Pi-calculus,用以描述并发系统的行为和数据类型的一致性;Gay和Hole[6]将一个进程的动态行为按照通道划分为一个个的会话(session),并将会话类型化,在此基础上进行动态行为和数据类型的一致性检查;Igarashi和Kobayashi[7]在Pi-演算的基础上提出了一个通用类型系统的框架,该类型系统将进程的动态行为类型化并进行验证,与Gay和Hole的工作不同的是,Igarashi和Kobayashi没有将进程的动态行为再细分。Pahl[8]则提出了一个基于Pi-演算的软件组件组合框架,在该框架中,利用PortType和Contract来形式化描述组件之间的交互。

本文在Simply Typed Pi-calculus的基础上,提出了进程类型假设集的外延和进程类型假设集相容性的概念,并且给出了进程类型假设集的合并算法。在此基础上,定义了一个扩充的Pi-演算类型系统,在该系统中,对进程类型良好性判定规则和捕获运行时错误的规则进行了重新定义。同时利用扩充的Pi-演算类型系统针对BPEL4WS规范建立了一个类型化的BPEL4WS形式化模型。最后通过一个案例给出了模型的验证方法。

本文包括以下内容:第2节给出扩充的Pi-演算类型系统的形式化定义;第3节定义了类型化的BPEL4WS形式化模型;第4节通过案例分析给出了类型化的BPEL4WS形式化模型的验证方法;最后给出全文总结。

2　扩充的Pi-演算类型系统

在Simply Typed Pi-calculus的基础上,我们定义了一个扩充的Pi-演算类型系统,包括基本定义、基本语法、类型判定规则和进程操作语义。特别是我们提出了进程类型假设集的外延和进程类型假设集相容性的概念,并且给出了进程类型假设集的合并算法。在此基础上,对进程类型良好性判定规则和捕获运行时错误的规则进行了重新定义。

2.1　基本定义

定义1(类型假设)　给一个名字a指定一个类型T称为

*)国家自然科学基金资助项目(项目编号:60403027)、湖北省自然科学基金(项目编号:2005ABA258)、软件工程重点实验室开放基金(项目编号:SKLSE05-07)。**辜希武**　博士研究生,研究方向为Web服务、语义Web和中间件。**卢正鼎**　教授,博士生导师,研究方向为分布式系统集成、Web数据管理、信息安全、数据库系统。

一个类型假设，记为：$a : T$。

定义 2(类型假设集)　一个有限的类型假设的集合称为类型假设集，记为：Γ。其中，$\Gamma = \{ a_i : T_i \mid i$，为有限索引且 $i \neq j \Rightarrow a_i \neq a_j \}$。$\Gamma, a : T$ 表示在类型假设集 Γ 中加入一个新的类型假设 $a : T$。空的类型假设集记为 Φ_Γ。

定义 3(类型函数)　类型假设集 Γ 可以看作从名字到类型的一个函数映射关系：$\forall a \in Dom(\Gamma), \Gamma(a) = T$。其中，$Dom(\Gamma)$ 为函数的定义域，$Dom(\Gamma) = \{a \mid a : T \in \Gamma\}$。

定义 4(子类型关系)　对于类型集合 **T** 中的类型 T_1 和 T_2，T_1 为类型 T_2 的子类型记为：$T_1 \leqslant T_2$。这时称序对 $\langle T_1, T_2 \rangle$ 满足子类型关系 $\leqslant$。

定义 5(子类型关系集合)　类型集合 **T** 上的子类型关系集合记为 $\leqslant_{\mathbf{T}}$，$\leqslant_{\mathbf{T}} = \{ T_i \leqslant T_j \mid T_i \in \mathbf{T}$，且 $i \neq j \Rightarrow T_i \neq T_j \}$。空的子类型关系集合记为 $\Phi_{\leqslant}$。

定义 6(类型判定)　如果名字 a 在类型假设集 Γ 和子类型关系集合 $\leqslant_{\mathbf{T}}$ 下，能够推导出其类型为 T，则称名字 a 被判定为类型 T，记为：$\Gamma \leqslant \mathbf{T} \vdash a : T$。如果名字 a 在类型假设集 Γ 和子类型关系集合 $\leqslant_{\mathbf{T}}$ 下，能判定其类型不为 T，则记为：$\Gamma, \leqslant_{\mathbf{T}} \overline{\vdash} a : T$。

定义 7(子类型判定)　如果在子类型关系集合 $\leqslant_{\mathbf{T}}$ 下，能够推导出类型 S 为类型 T 的子类型，则称 S 被判定为 T 的子类型。记为 $\leqslant_{\mathbf{T}} \vdash S \leqslant T$。如果在子类型关系集合 $\leqslant_{\mathbf{T}}$ 下，类型 S 不为类型 T 的子类型，则记为：$\leqslant_{\mathbf{T}} \vdash S \overline{\leqslant} T$。

定义 8(类型良好进程)　如果一个进程在类型假设集 Γ 和子类型关系集合 $\leqslant_{\mathbf{T}}$ 下，不会出现任何类型错误，则称该进程为类型良好的，记为：$\Gamma, \leqslant_{\mathbf{T}} \vdash P$。如果一个进程在类型假设集 Γ 和子类型关系集合 $\leqslant_{\mathbf{T}}$ 下，出现了类型错误，则称该进程为类型不良的，记为：$\Gamma, \leqslant_{\mathbf{T}} \overline{\vdash} P$。

2.2 基本语法

扩充的 Pi-演算类型系统语法包括类型、值、进程、类型假设集和子类型关系集合几个部分的定义，下面分别给出它们的定义。

2.2.1 类型定义

类型 $T ::= ch(T) \mid in(T) \mid out(T) \mid B \mid \widetilde{T}_n \mid \widetilde{T}_n[i]$

其中：$ch(T)$ 为输入输出通道类型，该类型的通道输入输出值类型为 T；$in(T)$ 为输入通道类型，该类型通道的输入值类型为 T；$out(T)$ 为输出通道类型，该类型通道的输出值类型为 T；B 为基本类型；$\widetilde{T}_n$ 为 n 维类型向量；$\widetilde{T}_n[i]$ 为 n 维类型向量的第 i 个分量投影。n 维类型向量：$\widetilde{T}_n ::= T_1 \times T_2 \times \cdots \times T_n (n \geqslant 1)$。基本类型 $B ::= Bool \mid Int \mid String \mid \cdots$。

2.2.2 值定义

值 $V ::= bv \mid x \mid \widetilde{X}_n \mid \widetilde{X}_n[i]$

其中：bv 为基本值；x 为名字；$\widetilde{X}_n$ 为 n 维名字向量；$\widetilde{X}_n[i]$ 为 n 维名字向量的第 i 个分量投影。n 维名字向量 $\widetilde{X}_n ::= (x_1, x_2, \cdots, x_n)\ (n \geqslant 1)$。基本值 $bv ::= true \mid false \mid 0 \mid$ "hello" $\mid \cdots$。

2.2.3 进程定义

进程 $P ::= a(\widetilde{X}_n : \widetilde{T}_n) \cdot P \mid \bar{a}(\widetilde{X}_n) \cdot P \mid \tau \cdot P \mid (\nu x : T)P \mid \sum_{i \in I} P_i \mid [x = y]P \mid [x \neq y]P \mid (P_1 \mid P_2) \mid !P \mid A(\widetilde{X}_n) \mid 0 \mid Err$ 其中：

(1) $a(\widetilde{X}_n : \widetilde{T}_n)$ 为输入前缀，其含义为从通道 a 输入类型为 $\widetilde{T}_n$ 的名字向量 $\widetilde{X}_n$。注意，在无类型的 Pi-演算语法里，输入前缀是不需要对输入名字做类型假设的。进程 $a(\widetilde{X}_n : \widetilde{T}_n) \cdot P$ 在执行完输入动作后，其行为表现为进程 P。

(2) $\bar{a}(\widetilde{X}_n)$ 为输出前缀，其含义为从通道 a 输出名字向量 $\widetilde{X}_n$。进程 $\bar{a}(\widetilde{X}_n) \cdot P$ 在执行完输出动作后其行为表现为进程 P。

(3) 前缀 τ 表示一个进程外部不可见的内部动作(或称为哑动作)。进程 $\tau \cdot P$ 在执行完内部动作 τ 后，其行为表现为进程 P。

(4) $(\nu x : T)P$ 表示在进程 P 内创建一个类型假设为 T 的约束名 x，名字在 P 进程外不可见。

(5) $\sum_{i \in I} Pi$ 表示选择执行，I 为有限索引集合。例如 $P_1 + P_2$ 表示选择执行 P_1 或 P_2。

(6) 匹配表达式 $[x = y]P$ 表示一个进程，当名字 x 和 y 为同一个名字时，其行为表现为进程 P。匹配表达式 $[x \neq y]P$ 表示一个进程，当名字 x 和 y 不为同一个名字时，其行为表现为进程 P。

(7) $P_1 \mid P_2$ 表示 2 个进程的并行执行。

(8) $!P$ 表示 P 被重复复制无穷多次。

(9) $A(\widetilde{X}_n)$ 称为进程标识符。每个进程标识符有其定义。假如进程标识符 $A(\widetilde{Y}_n)$ 的定义为 $A(\widetilde{X}_n) ::= P(i \neq j \Rightarrow \widetilde{X}_n[i] \neq \widetilde{X}_n[j])$。其含义为：当用 $\widetilde{Y}_n[i]$ 替换 $\widetilde{X}_n[i]$ 时，$A(\widetilde{Y}_n)$ 的行为将表现为进程 P。

(10) 空进程 0 表示不执行任何动作。Err 表示一个进程由于类型错误而导致运行时错误。

另外，一个进程里的输入名和约束名称为绑定名，其他的名字称为自由名。P 的绑定名的集合记为 $bn(P)$，自由名的集合记为 $fn(P)$。特别地，如果名字向量 $\widetilde{X}_n$ 是绑定的，则 $\widetilde{X}_n$ 的 n 个分量都属于 $bn(P)$；如果名字向量 $\widetilde{X}_n$ 不是绑定的，则 $\widetilde{X}_n$ 的 n 个分量都属于 $fn(P)$。同时，进程 P 的名字替换记为：$P\{\widetilde{Y}_n / \widetilde{X}_n\}$。其中用 $\widetilde{Y}_n[i]$ 替换 $\widetilde{X}_n[i]$。

2.2.4 类型假设集和子类型关系

类型假设集 $\Gamma ::= \Phi_\Gamma \mid \Gamma, V : T$

子类型关系集合 $\leqslant_{\mathbf{T}} ::= \Phi_{\leqslant} \mid \leqslant_{\mathbf{T}}, T_1 \leqslant T_2$

其中：$T_1, T_2 \in \mathbf{T}$

2.3 类型判定规则

2.3.1 值类型的判定规则

$$\frac{\Gamma(bv) = B}{\Gamma, \leqslant_{\mathbf{T}} \vdash bv : B}\text{(T-BV)}$$

$$\frac{\Gamma(x) = T}{\Gamma, \leqslant_{\mathbf{T}} \vdash x : T}\text{(T-NAME)}$$

$$\frac{\Gamma, \leqslant_{\mathbf{T}} \vdash \widetilde{X}_n[i] : \widetilde{T}_n[i] (1 \leqslant i \leqslant n)}{\Gamma, \leqslant_{\mathbf{T}} \vdash \widetilde{X}_n : \widetilde{T}_n}\text{(T-TUPLE)}$$

$$\frac{\Gamma, \leqslant_{\mathbf{T}} \vdash \widetilde{X}_n : \widetilde{T}_n}{\Gamma, \leqslant_{\mathbf{T}} \vdash \widetilde{X}_n[i] : \widetilde{T}_n[i] (1 \leqslant i \leqslant n)}\text{(T-PROJ)}$$

其中：规则 T-TUPLE 指如果一个名字向量 $\widetilde{X}_n$ 的第 i 个分量的类型正好是某个类型向量 $\widetilde{T}_n$ 的第 i 个分量，则该名字向量的类型为 $\widetilde{T}_n$。规则 T-PROJ 则正好相反。

2.3.2 子类型关系判定规则

$$\frac{}{\leqslant_{\mathbf{T}} \vdash T \leqslant T}\text{(ST-SELF)}$$

$$\frac{\leqslant_{\mathbf{T}} \vdash T_1 \leqslant T_2 \wedge \leqslant_{\mathbf{T}} \vdash T_2 \leqslant T_3}{\leqslant_{\mathbf{T}} \vdash T_1 \leqslant T_3}\text{(ST-TRANS)}$$

$$\frac{\Gamma, \leqslant_{\mathbf{T}} \vdash x : S \wedge \leqslant_{\mathbf{T}} \vdash S \leqslant T}{\Gamma, \leqslant_{\mathbf{T}} \vdash x : T}\text{(ST-SUB)}$$

$$\frac{\forall i (1 \leqslant i \leqslant n) \leqslant_{\mathbf{T}} \vdash \widetilde{S}_n[i] \leqslant \widetilde{T}_n[i]}{\leqslant_{\mathbf{T}} \vdash \widetilde{S}_n \leqslant \widetilde{T}_n}\text{(ST-TUPLE)}$$

$$\frac{}{\leqslant_T \vdash ch(T)\leqslant in(T)}(\text{ST-IN1})$$

$$\frac{}{\leqslant_T \vdash ch(T)\leqslant out(T)}(\text{ST-OUT1})$$

$$\frac{\leqslant_T \vdash S\leqslant T}{\leqslant_T \vdash in(S)\leqslant in(T)}(\text{ST-IN2})$$

$$\frac{\leqslant_T \vdash S\leqslant T}{\leqslant_T \vdash out(T)\leqslant out(S)}(\text{ST-OUT2})$$

$$\frac{\exists i(1\leqslant i\leqslant n)\leqslant_T \vdash \tilde{S}_n[i]\leqslant\tilde{T}_n[i]}{\leqslant_T \vdash \tilde{S}_n\leqslant\tilde{T}_n}(\text{NST-TUPLE})$$

其中：规则 ST-SELF 指一个类型是它本身的子类型；规则 ST-TRANS 指子类型关系具有传递性；ST-SUB 指如果一个值的类型已判定为 S，那么可以判定该值的类型为 S 的父类型 T；规则 ST-TUPLE 指如果类型向量 $\tilde{S}_n$ 的所有分量分别是类型向量 $\tilde{T}_n$ 的对应分量的子类型，则类型向量 $\tilde{S}_n$ 是类型向量 $\tilde{T}_n$ 的子类型（类似地，如果类型向量 $\tilde{S}_n$ 是类型向量 $\tilde{T}_n$ 的子类型，则 $\tilde{S}_n$ 的所有分量是 $\tilde{T}_n$ 的对应分量的子类型）；规则 NST-TUPLE 指只要类型向量 $\tilde{S}_n$ 有一个分量不是类型向量 $\tilde{T}_n$ 的对应分量的子类型，则类型向量 $\tilde{S}_n$ 不是类型向量 $\tilde{T}_n$ 的子类型；规则 ST-IN1 和 ST-OUT1 指 $ch(T)$ 是 $in(T)$ 和 $out(T)$ 的子类型（因为 $ch(T)$ 为输入输出通道类型，它比 $in(T)$ 和 $out(T)$ 有着更多的功能）；规则 ST-IN2 指明了输入通道的类型同向性：如果 $S\leqslant T$，则 $in(T)\leqslant in(T)$，即对于 $in(S)$ 类型的通道，总是可以认为输入值的类型是 S 的父类型；规则 ST-OUT2 指明了输出通道的类型反向性：如果 $S\leqslant T$，则 $out(T)\leqslant out(S)$，即 $out(T)$ 类型的通道总是可以输出类型 T 和 T 的子类型。

2.3.3 类型假设集的构造和合并

对于一个 Pi-演算进程 P，Γ_P 为其类型假设集。Γ_P 以如下方式得出：

$\Gamma_P=\Phi_\Gamma$

For each $x\in fn(P)$

$\Gamma_P=\Gamma_P, x:T_x$

Next x

由上面的 Γ_P 的构造算法，我们有 $Dom(\Gamma_P)=fn(P)$。

下面我们给出类型假设集合有关的二个引理，在此基础上给出类型假设集的外延的定义。

引理 1（弱化） 如果 $\Gamma_P,\leqslant_T \vdash P$ 且 $x\notin Dom(\Gamma_P)$，则 $\Gamma_P, x:S,\leqslant_T \vdash P$。

引理 1 指出如下事实：如果进程 P 在其类型假设集 Γ_P 下类型良好，则进程 P 在 Γ_P 因增加了新名字的类型假设而得到的更大的类型假设集下也类型良好，只要新增加的名字和 Γ_P 中已有的名字不重复。

引理 2（子类化） 如果 $\Gamma_P, x: S,\leqslant_T \vdash P$ 且 $\leqslant_T \vdash T\leqslant S$，则 $\Gamma_P, x: T,\leqslant_T \vdash P$。

引理 2 指出如下事实：如果进程 P 在其类型假设集 Γ_P 下类型良好，则进程 P 在把 Γ_P 中所有名字的类型替换为其子类型而得到的进程假设集下也类型良好。

定义 9（类型假设集的外延） 类型假设集 Γ_P 的类型假设集合的外延，记为：$\Gamma_P{}^E$。$\Gamma_P{}^E$ 是满足下列条件的假设集合：

①$Dom(\Gamma_P)\subset Dom(\Gamma_P{}^E)$。

②$V_x\in Dom(\Gamma_P)\cap Dom(\Gamma_P{}^E),\leqslant_T \vdash \Gamma_P{}^E(x)\leqslant\Gamma_P(x)$。

根据引理 1，2，$\Gamma_P{}^E$ 可以有以下性质：$\Gamma_P,\leqslant_T \vdash P\Rightarrow\Gamma_P^E,\leqslant_T \vdash P$，其证明如下。

外延性质的证明：根据 $\Gamma_P{}^E$ 的定义，我们有 $\Gamma_P{}^E=\Gamma,\Delta$ 其中 $Dom(\Gamma)=Dom(\Gamma_P)$、$\forall x\in Dom(\Delta)$ 有 $x\notin Dom(\Gamma_P)$ 和 $\forall x\in Dom(\Gamma)$ 有 $\Gamma(x)=\Gamma_P{}^E(x)$。

现假设 $\Gamma_P,\leqslant_T \vdash P$，由于 $\forall x\in Dom(\Delta)$ 有 $x\notin Dom(\Gamma_P)$，根据引理 1，我们有 $\Gamma_P,\Delta,\leqslant_T \vdash P$。

同时，对于 $Dom(\Gamma_P)$ 中的任何一个 x，我们有 $\Gamma_P-(x:\Gamma_P(x)\},x:\Gamma_P(x),\Delta,\leqslant_T \vdash P$。又根据 $\Gamma_P{}^E$ 定义中的第 2 个条件，我们有 $\leqslant_T \vdash\Gamma(x)=\Gamma_P{}^E(x)\leqslant\Gamma_P(x)$。根据引理 2，可以得到 $\Gamma_P-(x:\Gamma_P(x)\},x:\Gamma(x),\Delta\leqslant_T \vdash P$。对 $Dom(\Gamma_P)$ 定义中的所有名字做同样的子类型替换，最后得到 $\Gamma,\Delta,\leqslant_T \vdash P$，因此 $\Gamma_P{}^E,\leqslant_T \vdash P$。

由于进程选择操作"+"和进程并行操作"|"的存在，我们必须在一个统一的类型假设集下判断进程 $P+Q$ 和 $P|Q$ 的类型良好性，因此必须将类型假设集 Γ_P 和 Γ_Q 合并。在 Γ_P 和 Γ_Q 中，如果有同名的名字，则可能存在类型假设冲突，有的冲突是矛盾的，这种情况则称 Γ_P 和 Γ_Q 不相容；有的冲突则可以通过子类型关系消去，因此下面我们给出二个类型假设集相容性的定义和相容的类型假设集的合并算法。

定义 10（类型假设集相容性） 二个类型假设集 Γ_P 和 Γ_Q 为相容的，记为 $\Gamma_P\lhd\rhd\Gamma_Q$。如果它们满足下列条件之一：

①$Dom(\Gamma_P)\cap Dom(\Gamma_Q)=\Phi$。

②$Dom(\Gamma_P)\cap Dom(\Gamma_P)\neq\Phi$，且 $\forall x\in Dom(\Gamma_P)\cap Dom(\Gamma_Q)$，$x$ 满足下列情况之一：

(1)$\leqslant_T \vdash\Gamma_P(x)\leqslant\Gamma_Q(x)\vee\leqslant_T \vdash\Gamma_Q(x)\leqslant\Gamma_P(x)$。

(2)$\Gamma_P(x)=\text{in}(T)\wedge\Gamma_Q(x)=\text{out}(S)\wedge\leqslant_T \vdash S\leqslant T$。

(3)$\Gamma_Q(x)=\text{in}(T)\wedge\Gamma_P(x)=\text{out}(S)\wedge\leqslant_T \vdash S\leqslant T$。

定义 11（相容类型假设集的合并） 如果二个类型假设集 Γ_P 和 Γ_Q 满足 $\Gamma_P\lhd\rhd\Gamma_Q$，则它们的合并记为：$\Gamma_P\oplus\Gamma_Q$，且规定运算符"$\oplus$"的优先级高于运算符","。$\Gamma_P\oplus\Gamma_Q$ 以如下算法得到：

$\Gamma_P\oplus\Gamma_Q=\Phi\Gamma$

For each $x\notin Dom(\Gamma_P)\cap Dom(\Gamma_Q)$

if $(x\in Dom(\Gamma_P))\Gamma_P\oplus\Gamma_Q=\Gamma_P\oplus\Gamma_Q,x:\Gamma_P(x)$

if $(x\in Dom(\Gamma_Q))\ \Gamma_P\oplus\Gamma_Q=\Gamma_P\oplus\Gamma_Q, x:\Gamma_Q(x)$

Next x

For each $x\in Dom(\Gamma_P)\cap Dom(\Gamma_Q)$

If $(\leqslant_T \vdash\Gamma_P(x)\leqslant\Gamma_Q(x))\ \Gamma_P\oplus\Gamma_Q=\Gamma_P\oplus\Gamma_Q, x:\Gamma_P(x)$

If $(\leqslant_T \vdash\Gamma_Q(x)\leqslant\Gamma_P(x))\Gamma_P\oplus\Gamma_Q=\Gamma_P\oplus\Gamma_Q, x:\Gamma_Q(x)$

If$(\Gamma_P(x)=\text{in}(T)\wedge\Gamma_Q(x)=\text{out}(S)\wedge\leqslant_T \vdash S\leqslant T)$

$\Gamma_P\oplus\Gamma_Q=\Gamma_P\oplus\Gamma_Q, x:ch(T)$

End If

If$(\Gamma_Q(x)=\text{in}(T)\wedge\Gamma_P(x)=\text{out}(S)\wedge\leqslant_T \vdash S\leqslant T)$

$\Gamma_P\oplus\Gamma_Q=\Gamma_P\oplus\Gamma_Q, x:ch(T)$

End If

Next x

定义 10，11 指明下列事实：如果二个进程的类型假设集 Γ_P 和 Γ_Q 的定义域有交集，只要交集中的名字 x 分别在二个类型假设集中的类型 $\Gamma_P(x)$ 和 $\Gamma_Q(x)$ 满足子类型关系，或者存在公共的子类型，则将 x 的类型假设为子类型就可以消除类型假设冲突。因此这二个类型假设集是相容的，从而这二个类型假设集可以合并为一个更大的类型假设集 $\Gamma_P\oplus\Gamma_Q$。而且根据引理 1，2，可以证明 $\Gamma_P\oplus\Gamma_Q$ 不会破坏进程 P、Q 的类型良好性，证明方法和类型假设集外延的性质的证明类似。特别要提到的是，如果名字 x 在 Γ_P 和 Γ_Q 中的假设类型分别是 $in(T)$ 和 $out(S)$ 且 $S\leqslant T$，$in(T)$ 和 $out(S)$ 不满足子类型关

系，但它们有公共的子类型：$ch(T)\leqslant out(T)\leqslant out(S)$，同时$ch(T)\leqslant in(T)$，这时将名字 x 的类型假设为 $ch(T)$即可消除冲突。

在 2.3.3 节的基础上，下面给出判断进程是否类型良好的规约规则。

2.3.4 进程类型良好判定规则

以下判断进程类型良好的规则被定义。

$$\frac{}{\Gamma,\leqslant_{\mathrm{T}}\vdash 0}(\text{T-0})$$

$$\frac{\Gamma_P,\leqslant_{\mathrm{T}}\vdash P}{\Gamma_P,\leqslant_{\mathrm{T}}\vdash \tau\cdot P}(\text{T-}\tau)$$

$$\frac{\Gamma_P,\leqslant_{\mathrm{T}}\vdash P\wedge(\exists\Gamma_P{}^E,\text{满足:}\Gamma_P{}^E,\leqslant_{\mathrm{T}}\vdash x:T\wedge\Gamma_P{}^E,\leqslant_{\mathrm{T}}\vdash y:T)}{\Gamma_P{}^E,\leqslant_{\mathrm{T}}\vdash[x=y]P}(\text{T-MATCH})$$

$$\frac{\Gamma_P,\leqslant_{\mathrm{T}}\vdash P\wedge\Gamma_P(\widetilde{X}_n)=\widetilde{T}_n\wedge(\exists\Gamma_P{}^E,\text{满足:}\Gamma_P{}^E,\leqslant_{\mathrm{T}}\vdash\widetilde{Y}_n:\widetilde{S}_n\wedge\leqslant_{\mathrm{T}}\vdash\widetilde{S}_n\leqslant\widetilde{T}_n}{\Gamma_P{}^E,\leqslant_{\mathrm{T}}\vdash P\{\widetilde{Y}_n/\widetilde{X}_n\}}(\text{T-SUBST})$$

$$\frac{\Gamma_P,\widetilde{X}_n:\widetilde{T}_n,\leqslant_{\mathrm{T}}\vdash P\wedge(\exists\Gamma_P{}^E,\text{满足:}\Gamma_P{}^E,\leqslant_{\mathrm{T}}\vdash a:in(\widetilde{S}_n)\wedge\leqslant_{\mathrm{T}}\vdash\widetilde{S}_n\leqslant\widetilde{T}_n)}{\Gamma_P{}^E,\leqslant_{\mathrm{T}}\vdash a(\widetilde{X}_n:\widetilde{T}_n)\cdot P}(\text{T-INPUT})$$

$$\frac{\Gamma_P,\leqslant_{\mathrm{T}}\vdash P\wedge(\exists\Gamma_P{}^E,\text{满足:}\Gamma_P{}^E,\leqslant_{\mathrm{T}}\vdash a:out(\widetilde{S}_n)\wedge\Gamma_P{}^E,\leqslant_{\mathrm{T}}\vdash\widetilde{X}_n:\widetilde{T}_n\wedge\leqslant_{\mathrm{T}}\vdash\widetilde{T}_n\leqslant\widetilde{S}_n)}{\Gamma_P{}^E,\leqslant_{\mathrm{T}}\vdash\bar{a}(\widetilde{X}_n)\cdot P}(\text{T-OUT})$$

$$\frac{\Gamma_P,\leqslant_{\mathrm{T}}\vdash P\wedge Q,\leqslant_{\mathrm{T}}\vdash Q}{\Gamma_P\oplus\Gamma_Q,\leqslant_{\mathrm{T}}\vdash P|Q}(\text{T-PAR})$$

$$\frac{\Gamma_P,\leqslant_{\mathrm{T}}\vdash P\wedge\Gamma_Q,\leqslant_{\mathrm{T}}\vdash Q}{\Gamma_P\oplus\Gamma_Q,\leqslant_{\mathrm{T}}\vdash P+Q}(\text{T-SUM})$$

$$\frac{\Gamma_P,x:T,\leqslant_{\mathrm{T}}\vdash P}{\Gamma_P,\leqslant_{\mathrm{T}}\vdash(\nu x:T)P}(\text{T-RES})$$

$$\frac{\Gamma_P,\leqslant_{\mathrm{T}}\vdash P}{\Gamma_P,\leqslant_{\mathrm{T}}\vdash !\ P}(\text{T-REP})$$

其中：T-PAR 和 T-SUM 指如果 P 和 Q 在各自的类型假设集下类型良好，则 $P|Q$ 和 $P+Q$ 在 $\Gamma_P\oplus\Gamma_Q$ 下类型良好；规则 T-SUBST 指名字替换$\{\widetilde{Y}_n/\widetilde{X}_n\}$发生时，如果进程 P 在其类型假设集 Γ_P 下类型良好，并且存在一个 Γ_P 的外延 $\Gamma_P{}^E$，使得在 $\Gamma_P{}^E$ 中 $\widetilde{Y}_n$ 的类型是 $\widetilde{X}_n$ 在 Γ_P 中的类型的子类型，则 $P\{\widetilde{Y}_n/\widetilde{X}_n\}$在 $\Gamma_P{}^E$ 下类型良好；规则 T-INPUT 指如果进程 P 在类型假设集 $\Gamma_P,\widetilde{X}_n:\widetilde{T}_n$ 下类型良好，并且存在类型假设集 Γ_P 的外延 $\Gamma_P{}^E$，使得 a 在 $\Gamma_P{}^E$ 中的类型为 $in(\widetilde{S}_n)$，且 $\widetilde{S}_n$ 为 $\widetilde{T}_n$ 的子类型，则 $a(\widetilde{X}_n:\widetilde{T}_n)\cdot P$ 在 $\Gamma_P{}^E$ 下类型良好；规则 T-OUT 指如果进程 P 在类型假设集 Γ_P 下类型良好，并且存在类型假设集 Γ_P 的外延 $\Gamma_P{}^E$，使得 a 在 $\Gamma_P{}^E$ 中的类型为 $out(\widetilde{S}_n)$，$\widetilde{X}_n$ 在 $\Gamma_P{}^E$ 中的类型为 $\widetilde{T}_n$，且 $\widetilde{T}_n$ 为 $\widetilde{S}_n$ 的子类型，则 $\bar{a}(\widetilde{X}_n)\cdot P$ 在 $\Gamma_P{}^E$ 下类型良好。

2.4 进程操作语义

进程的操作语义反映了进程的变迁，描叙了一个进程的动态行为。进程的操作语义由一套形如 $P\xrightarrow{\alpha}Q$ 的推演规则来表示：其含义为进程 P 经过动作 α 变迁(演化)成为进程 Q，其中动作 α 可以为内部哑动作(τ)、输入动作和输出动作。例如：输入动作 $a(\widetilde{X}_n)$和输出动作 $\bar{a}\widetilde{U}_n$ 正好是一对互补动作，输出向量 $\widetilde{U}_n$ 沿着通道 a 正好被输入动作 $a(\widetilde{X}_n)$输入以替换名字向量 $\widetilde{X}_n$，导致进程的名字替换$\{\widetilde{U}_n/\widetilde{X}_n\}$，同时这对互补的动作互相抵消而成为一个内部动作 τ。这种情况用推演规则 COMM 来描述：

$$\frac{P\xrightarrow{a(\widetilde{X}_n)}P',Q\xrightarrow{\bar{a}\widetilde{U}_n}Q'}{(P+M)|(Q+N)\xrightarrow{\tau}P'\{\widetilde{U}_n/\widetilde{X}_n\}|Q}(\text{R-COMM})$$

和无类型的 Pi-演算相比，由于我们在进程表达式的基本语法语法里增加了 Err 子项以表示进程由于类型不匹配而产生了运行时错误，因此必须要有适当的规则来捕获运行时错误 Err。在给出捕获运行时错误规则以前，我们首先给出 *IsName* 函数的定义以方便表达。函数 *IsName*(x)对 x 在运行时的值进行判断，如果在进程变迁过程中 x 为一个名字，则 *IsName*(x)返回 true；如果在进程变迁过程中 x 成为为一个基本数值，则 IsName(x)返回 false。同时记当前运行时的类型假设集为 Γ_{Run}。在此基础上定义捕获运行时错误的规则如下：

$$\frac{IsName(a)=\text{false}}{\bar{a}(\widetilde{X}_n)\cdot P\xrightarrow{\tau}Err}(\text{ERR-OUT1})\qquad\frac{\Gamma_{Run},\leqslant_{\mathrm{T}}\bar{\vdash}P}{\bar{a}(\widetilde{X}_n)\cdot P\xrightarrow{\tau}Err}(\text{ERR-OUT2})$$

$$\frac{IsName(a)=\text{true}\wedge\Gamma_{Run},\leqslant_{\mathrm{T}}\bar{\vdash}a:out(\widetilde{S}_n)}{\bar{a}(\widetilde{X}_n)\cdot P\xrightarrow{\tau}Err}(\text{ERR-OUT3})$$

$$\frac{\Gamma_{Run},\leqslant_{\mathrm{T}}\vdash P\wedge\Gamma_{Run},\leqslant_{\mathrm{T}}\vdash a:out(\widetilde{S}_n)\wedge\Gamma_{Run},\leqslant_{\mathrm{T}}\vdash\widetilde{X}_n:\widetilde{T}_n\wedge\leqslant_{\mathrm{T}}\vdash\widetilde{T}_n\leqslant\widetilde{S}_n}{\bar{a}(\widetilde{X}_n)\cdot P\xrightarrow{\tau}Err}(\text{ERR-OUT4})$$

$$\frac{IsName(a)=\text{false}}{a(\widetilde{X}_n:\widetilde{T}_n)\cdot P\xrightarrow{\tau}Err}(\text{ERR-IN1})\qquad\frac{\Gamma_{Run},\widetilde{X}_n:\widetilde{T}_n\leqslant_{\mathrm{T}}\bar{\vdash}P}{a(\widetilde{X}_n:\widetilde{T}_n)\cdot P\xrightarrow{\tau}Err}(\text{ERR-IN2})$$

$$\frac{IsName(a)=\text{true}\wedge\Gamma_{Run},\leqslant_{\mathrm{T}}\bar{\vdash}a:in(\widetilde{S}_n)}{a(\widetilde{X}_n:\widetilde{T}_n)\cdot P\xrightarrow{\tau}Err}(\text{ERR-IN3})$$

$$\frac{\Gamma_{Run},\widetilde{X}_n:\widetilde{T}_n\leqslant_{\mathrm{T}}\vdash P\wedge\Gamma_{Run},\leqslant_{\mathrm{T}}\vdash a:in(\widetilde{S}_n)\wedge\leqslant_{\mathrm{T}}\vdash\widetilde{S}_n\bar{\leqslant}\widetilde{T}_n}{a(\widetilde{X}_m:\widetilde{T}_n)\cdot P\xrightarrow{\tau}Err}(\text{ERR-IN4})$$

$$\frac{m\neq n\vee(m=n\wedge\Gamma_{Run},\leqslant_{\mathrm{T}}\vdash\widetilde{Y}_n:\widetilde{S}_n\wedge\leqslant_{\mathrm{T}}\vdash\widetilde{S}_n\bar{\leqslant}\widetilde{T}_m}{a(\widetilde{X}_m:\widetilde{T}_m)\cdot P|\bar{a}(\widetilde{Y}_n)\cdot Q\xrightarrow{\tau}Err}(\textit{ERR-COM})$$

$$\frac{P \xrightarrow{\tau} Err \vee Q \xrightarrow{\tau} Err}{P+Q \xrightarrow{\tau} Err}\text{(ERR-SUM)} \qquad \frac{P \xrightarrow{\tau} Err}{(\nu x:T)P \xrightarrow{\tau} Err}\text{(ERR-RES)}$$

$$\frac{}{Err|P \xrightarrow{\tau} Err}\text{(ERR-PAR)} \qquad \frac{P \xrightarrow{\tau} Err}{!\ P \xrightarrow{\tau} Err}\text{(ERR-REP)}$$

其中：ERR-OUT1 指当 a 不是一个名字时，$\bar{a}(\tilde{X}_n)\cdot P$ 会导致运行时错误而变迁为进程 Err（如果 a 不是一个名字而是一个基本值如 $true$，$\overline{true}(\tilde{X}_n)\cdot P$ 显然是没有意义的）；ERR-OUT2 指当进程 P 在当前运行的类型假设集下类型不良时，$\bar{a}(\tilde{X}_n)\cdot P$ 会导致运行时错误而变迁为进程 Err；ERR-OUT3 指当 a 是名字但 a 不是输出类型通道时，$\bar{a}(\tilde{X}_n)\cdot P$ 会导致运行时错误而变迁为进程 Err；ERR-OUT4 指当 a 为名字、进程 P 在当前运行的类型假设集下类型良好、a 为输出 $\tilde{S}_n$ 类型值的通道，但要输出的值 $\tilde{X}_n$ 的类型不是 $\tilde{S}_n$ 的子类型时，$\bar{a}(\tilde{X}_n)\cdot P$ 会导致运行时错误而变迁为进程 Err；ERR-IN1、ERR-IN2、ERR-IN3、ERR-IN4 含义类似。ERR-COM 指当二个并发进程 $a(\tilde{X}_m:\tilde{T}_m)\cdot P|\bar{a}(\tilde{Y}_n)\cdot Q$ 互相通信时，如果输入、输出的名字向量维数不相等或者输入、输出的名字向量维数相等但输出的值的类型不是要输入值的子类型时，这二个进程间的并发通信会导致运行时错误。

除了捕获运行时错误的规则外，其他的推演规则和无类型 Pi-演算的推演规则类似，这里就不一一列出。

表 1　类型化的 BPEL4WS 形式化模型

BPEL4WS 行为	对应的 Pi-演算进程	进程类型假设集
⟨*invoke* *inputVariable*="*ncname*" *outputVariable*="*ncname*" /*invoke*⟩	$\|Invoke\rangle = \overline{cho}\,inputVariable.\ chi(outputVariable:T_2).0$	$\Gamma_{\|Invoke\rangle} = \{inputVariable:T_1,\ cho:o(T_1),\ chi:in(T_2)\}$
⟨*receive* *variable*="*ncname*" / *receive* ⟩	$\|Re\,ply\rangle = \overline{cho}chi(variable:T).0$	$\Gamma_{\|Re\,ply\rangle} = \{chi:in(T)\}$
⟨*reply* *variable*="*ncname* /*reply*⟩	$\|Re\,ply\rangle = variable.0$	$\Gamma_{\|Re\,ply\rangle} = \{variable:T,\ cho:out(T)\}$
⟨*assign* ⟩⟨*copy*⟩ ⟨*fromvariable*="*ncname1*"⟩ ⟨*to variable*="*ncname2*"⟩ ⟨/*copy*⟩⟨/*assign*⟩	$\|Assign\rangle = (\nu assign:ch(T))$ $(\ ncname1.0$ $\|\ assign\langle ncname2:T.0)$	$\Gamma_{\|Assign\rangle} = \{\ ncname1:T\ \}$
⟨*throw* *faultName*="*qname*" *faultVariable*="*ncname*" ⟩ ⟨/*throw*⟩	$\|Throw\rangle = \overline{faultName}\ faultVariable.0$	$\Gamma_{\|Throw\rangle} = \{\ faultVariable:T,\ faultName:ch(T)\}$
⟨*compensate* *scope*="*ncname*" /⟩	$\|Compensate\rangle = \overline{Scopename}\ ncame.0$	$\Gamma_{\|compensate\rangle} = \{\ ncame:T,\ Scopename:ch(T)\}$
⟨*sequence*⟩ *activity*+ ⟨/*sequence*⟩	$\|Sequence\rangle = \|A_1\rangle.\ \cdots.\ \|A_n\rangle$	$\Gamma_{\|sequence\rangle} = \Gamma_{\|A_1\rangle}\oplus\Gamma_{\|A_2\rangle}\cdots\oplus\Gamma_{\|A_n\rangle}$
⟨*switch* ⟩ ⟨*case condition*="*bool-expr*"⟩+ *activity*⟨/*case*⟩ ⟨*otherwise*⟩ *activity* ⟨/*otherwise*⟩ ⟨/*switch*⟩	$\|Switch\rangle = \sum_{i=1}^{n}[x=N_i]\|A_i\rangle + [x\neq N_1]\cdots[x\neq N_i]$	$\Gamma = \{x:T,\ N_i:T\mid i=1\cdots n\}$ $\Gamma_{\|Switch\rangle} = \Gamma\oplus\Gamma_{A_1}\rangle\cdots\oplus\Gamma_{\|A_n} = \oplus\Gamma_{\|D\rangle}$
⟨*while condition*="*bool-expr*"⟩ *activity* ⟨/*while*⟩	$\|While\rangle = [x=N]\|A\rangle\cdot + \|While\rangle[x\neq N].0$	$\Gamma = \{x:T,\ N:T\}$ $\Gamma_{\|While\rangle} = \Gamma\oplus\Gamma_{\|A\rangle}$
⟨*pick* ⟩ ⟨ *onMessage variable* = "*ncname*"?⟩+*activity* ⟨/*onMessage*⟩ ⟨/*pick*⟩	$\|Pick\rangle = \sum_{i=1}^{n} che_i(var_i:T_i)\cdot\|A_i\rangle$	$\Gamma = \{chei:in(T_i)\mid i=1\cdots n\}$ $\Gamma_{\|Pick\rangle} = \Gamma\oplus\Gamma_{\|A_1}\rangle\cdots\oplus\Gamma_{\|A_n}\rangle$
⟨*flow*⋯⟩ *activity*+⟨/*flow*⟩	$\|Flow\rangle = \|A_1\rangle\cdots\|A_n\rangle$	$\Gamma_{\|Flow\rangle} = \Gamma_{\|A_1\rangle}\oplus\Gamma_{A_2}\rangle\cdots\oplus\Gamma_{\|A_n\rangle}$
⟨*faultHandlers*⟩ ⟨*catch* *faultName*="*qname*" *faultVariable*="*ncname*"⟩+ *activity*⟨/*catch*⟩ ⋯ ⟨/*faultHandlers*⟩	$\|FautHandler\rangle = \sum_{i=1}^{n} faultName_i(var_i:T_i)\cdot\|A_i\rangle$	$\Gamma = \{\ faultName_i:ch(T_i)\mid i=1\cdots n\}$ $\Gamma_{\|FaultHandler\rangle} = \Gamma\oplus\Gamma_{\|A_1\rangle}\cdots\oplus\Gamma_{\|A_n\rangle}$
⟨*ventHandlers*⟩ ⟨ *onMessage variable* = "*ncname*"?⟩+ *activity* ⟨/*onMessage*⟩ ⟨/*eventHandlers*⟩	$\|EventHandler\rangle = \sum_{i=1}^{n}!\ che_i(var_i:T_i)\cdot\|A_i\rangle$	$\Gamma = \{faultName_i:ch(T_i)\mid i=1\cdots n\}$ $\Gamma_{\|FaultHandler\rangle} = \Gamma\oplus\Gamma_{\|A_1\rangle}\cdots\oplus\Gamma_{\|A_n\rangle}$
⟨*compensationHandler*>⟩ *activity* ⟨/*compensationHandler*⟩	$\|CompensationHandler\rangle = Scopename(ncname:T).\|A\rangle$	$\Gamma = \{Scopename:ch(T)\}$ $\Gamma_{\|CompensationHandler\rangle} = \Gamma\oplus\Gamma_{\|A\rangle}$

3 类型化的 BPEL4WS 形式化模型

BPEL4WS 作为一种描述和执行 Web 服务组合工作流的语言，正在成为 Web 服务组合的事实上的标准。BPEL4WS 定义了各种不同的原子行为(atom activities)、结构化的行为(structural activities)和并发行为(flow activities)，来描述一个组合的商业流程。同时，它还提供了异常处理器(faulthandler)、事件处理器(eventhandler)和补偿(compensationhandler)来处理异常、事件和事务。在文[9]中，我们已经用无类型的 Pi-演算为 BPEL4WS 建立了一个形式化的模型，该模型可以从 Web 服务的动态行为的角度以形式化的方法来描述用 BPEL4WS 规范所表示的一个组合的 Web 服务，并且可以形式化地验证 Web 服务组合动态行为匹配性。但该模型没有数据类型检查机制，无法判断和捕获因数据类型不匹配而导致的运行时错误。因此，我们在以前的工作基础上，利用扩充的 Pi-演算类型系统为 BPEL4WS 建立一个类型化的 BPEL4WS 形式化模型。

类型化的 BPEL4WS 形式化模型包括以下几个部分：类型集合 T、子类型关系集合 $\leqslant_T$、BPEL4WS 规范中各种行为所对应的 Pi-演算模型(通过一个 Pi-演算进程来描述)、每个 Pi-演算进程 P 对应的类型假设集合 Γ_P。每部分的构造方法如下。

(1)类型集合 T 通过解析 XML Schema 文档，将 XML Schema 文档中所支持的数据类型作为基本类型添加到类型集合 T 中，同时形成了子类型关系集合 $\leqslant_T$。

(2)对于一个 BPEL4WS 所描述的每一个行为 A_{bpel}，用一个 Pi-演算进程来建模，其对应的 Pi-演算进程模型记为 $|A_{bpel}\rangle$。

(3)对每一个 Pi-演算进程 $|A_{bpel}\rangle$，按 2.3.3 节的算法构造其类型假设集。同时按照类型假设集外延的定义和类型假设集的合并算法得到多个 Pi-演算进程的并发组合和选择组合所对应的更大的类型假设集。对于每个行为中输入、输出的数据类型是通过解析 Web 服务的 WSDL 接口描述中 operation 元素的子元素 input、output 来确定。

在用 Pi-演算进程 $|A_{bpel}\rangle$ 来建模 BPEL4WS 所描述的每一个行为时，我们利用 Pi-演算进程间通信来描述 BPEL4WS 的输入、输出行为，Pi-演算进程通信使用的通道是由 BPEL4WS 行为的 *partnerLink*、*portType* 和 *operation* 属性来决定：对于一个三元组 $AccessPoint=(partnerLink, portType, operation)$，函数 $Chan(AccessPoint)$ 返回一个唯一的通道名，因此对表 1 中所列的 Pi-演算进程中出现的通道名不再详细说明。BPEL4WS 的类型化形式化模型如表 1 所示。

4 案例研究和模型验证

在本节我们将通过图 1 所示的一个 Web 服务组合案例来分析，说明如何利用我们所定义的类型化 Pi-演算模型来描述一个 Web 服务组合，如何验证 Web 服务组合的动态行为的正确性以及检查在运行过程中可能出现的类型不匹配情况。

在图 1 中，Web 服务 *Rect* 和 *Circle* 分别能够计算矩形和圆的面积。Web 服务 *MathServer* 则将这二个 Web 服务组合起来。客户通过它和 *MathServer* 之间的交互通道 $op1$、$op2$ 来选择到底是访问 *Rect* 还是 *Circle* 服务：当客户需要访问 *Rect* 服务时，客户通过通道 $op1$ 向 *MathServer* 发送通道名 $ch1$(比如一个 Socket)，*MathServer* 在调用 *Rect* 服务时将通道名 $ch1$ 传递给 *Rect*，*Rect* 接着利用该通道直接和客户交互。客户需要访问 *Circle* 服务的方法类似。由于通道 $ch1$ 和 $ch2$ 是动态产生并且可以在进程之间移动，故在图中用虚线表示。

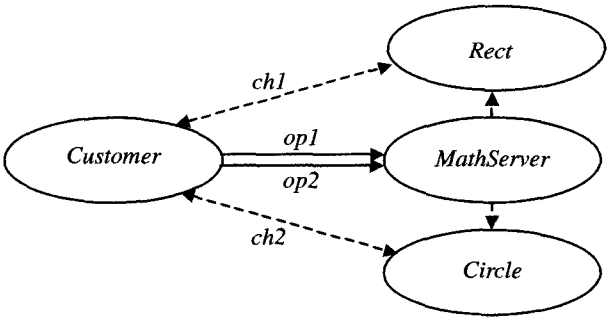

图 1 *Web* 服务组合案例

由于篇幅所限，这里不详细列出 *MathServer* 服务的 BPEL4WS 描述。根据第 3 节所定义的 BPEL4WS 模型，我们分别给出 *Rect*、*Circle*、*MathServer* 和 *Customer* 的类型化 Pi-演算描述(我们假设用户的输入参数和服务返回值为 *Float* 型)，同时为了和下面要用到的模型验证工具的语法规则保持一致，我们用 $'a\langle x\rangle$ 表示从 a 通道输出名字 x，用 υ 表示一个约束名(内部名)。

(1)*Rect* 从通道 $x1$ 输入矩形长和宽，并从 $x1$ 返回结果，用类型化的 Pi-演算进程描述如下：

$$Rect(x1, rectarea) = x1(length: Float).\ x1(width: Float).\ 'x1\langle rectarea\rangle.0$$

由于通道 $x1$ 可以输入、输出 *Float* 类型数据，故其类型为 $ch(Float)$。所以，进程 *Rect* 的类型假设集 $\Gamma_{Rect}=\{x1: ch(Float),\ rectarea: Float\}$。

(2)同样地，*Circle* 的类型化的 Pi-演算进程和类型假设集分别为

$$Circle(x2, circlearea) = x2(radious: Float).\ 'x2\langle circlearea\rangle.0$$

$$\Gamma_{Circle}=\{x2: ch(Float),\ circlearea: Float\}$$

(3)*MathServer* 从通道 $op1$(或 $op2$)等待客户的请求，如果从通道 $op1$(或 $op2$)接受到一个通道名 $ch1$(或 $ch2$)，则调用 *Rect*(或 *Circle*)，并且通过名字替换将通道名 $ch1$(或 $ch2$)传递给相应的子服务，替换掉子服务中的通道名 $x1$(或 $x2$)。*MathServer* 的类型化的 Pi-演算进程和类型假设集分别为

$$MathServer(op1, op2, rectarea, circlearea)=$$
$$op1(ch1: ch(Float)).\ Rect(ch1, rectarea)+op2(ch2: ch(Float)).\ Circle(ch2, circlearea)$$
$$\Gamma_{MathServer}=\{op1: in(ch(Float)),\ op2: in(ch(Float)),$$
$$rectarea: Float,\ circlearea: Float\}\oplus\Gamma_{Rect}\oplus\Gamma_{Circle}$$

这里特别要注意的是：$op1$ 和 $op2$ 都是输入通道，且它们输入的值的类型为一个能输入输出 *Float* 类型值的通道，因此它们的类型都为 $in(ch(Float))$。同时，$ch1$ 和 $ch2$ 在调用相应子服务进程时通过名字替换传递到子服务进程中。

(4)*Customer* 通过 $op1$ 或 $op2$ 通道向 *MathServer* 发送通道名 $ch1$ 或 $ch2$ 来选择具体的子服务，然后通过通道名 $ch1$ 或 $ch2$ 向子服务输入参数并得到结果。*Customer* 的类型化的 Pi-演算进程和类型假设集分别为

$$Customer(op1, op2, ch1, ch2, l, w, r)=$$
$$'op1\langle ch1\rangle.\ 'ch1\langle l\rangle.\ 'ch1\langle w\rangle.\ ch1(ra: Float).0+'op2$$

$\langle ch2\rangle.{}'ch2\langle r\rangle.ch2(rc:Float).0$

$\Gamma_{Cusetomer}=\{op1:out(ch(Float)),\ op2:out(ch(Float)),$
$ch1:ch(Float),\ ch2:ch(Float),$
$l:Float,\ w:Float,\ r:Float\}$

这里特别要注意的是：$\Gamma_{mathServer}$ 和 $\Gamma_{Cusetomer}$ 中 $op1$ 和 $op2$ 的类型分别为：$in(ch(Float))$和 $out(ch(Float))$，这就是 2.3.3 节所谈到的二个进程假设集会出现对同一个名字的类型假设冲突，根据定义 10，$\Gamma_{MathServer}\lhd\rhd\Gamma_{Cusetomer}$，因此在考虑进程 $MathServer|Customer$ 时，将 $op1$ 和 $op2$ 的类型假设为它们公共的子类型 $ch(ch(Float))$，就可以消除类型冲突并且不会破坏进程 $MathServer$ 和 $Customer$ 的类型良好性。

利用已经建立的形式化模型，我们可以对 Web 服务组合的正确性进行验证。验证的内容包括二个方面：Web 服务交互过程中数据类型的匹配性和 Web 服务动态行为的匹配性。数据类型的匹配性利用进程类型假设集 Γ、子类型假设集$\leqslant_T$和 2.3 节所定义的类型判定规则来完成。例如：进程 $MathServer$ 的类型良好性可以通过如下推理过程来完成：

(1)$Rect$ 和 $Circle$ 是类型良好的，因为

$$\frac{\Gamma_{Rect},\leqslant_T\vdash 0\wedge\Gamma_{Rect},\leqslant_T\vdash x1:ch(Float)\wedge\Gamma_{Rect},\leqslant_T\vdash rdectarea:Float}{\Gamma_{Rect},\leqslant_T\vdash{}'x1\langle rectarea\rangle.0}(T\text{-}OUT)$$

在 $\Gamma_{Rect},\leqslant_T\vdash{}'x1\langle rectarea\rangle.0$ 结论的基础上，再利用 T-INPUT 规则，可得到

$\Gamma_{Rect},\Gamma_{Rect},\leqslant_T\vdash x1(width:Float).{}'x1\langle rectarea\rangle.0$

在 $\Gamma_{Rect},\leqslant_T\vdash x1(width:Float).{}'x1\langle rectarea\rangle.0$ 结论的基础上，再利用 T-INPUT 规则，可得到

$\Gamma_{Rect},\leqslant_T\vdash(x1(length:Float)x1(width:Float).{}'x1\langle rectarea\rangle.0$

即 $\Gamma_{Rect},\leqslant_T\vdash Rect$。同样的推导方法，我们有 $\Gamma_{Rect},\leqslant_T\vdash CVircle$。

(2)$MathServer$ 是类型良好的。首先 Γ_{Rect}、Γ_{Circle} 是相容的，并且 $\Gamma_{MathServer}$ 是按照 2.3.3 节定义的类型假设集合并算法得到，因此我们有 $\Gamma_{MathServer},\leqslant_T\vdash Rect$ 和 $\Gamma_{MathServer},\leqslant_T\vdash Circle$。在此基础上利用 T-INPUT 规则，我们有

$$\frac{\Gamma_{MathServer},\leqslant_T\vdash Rect\wedge\Gamma_{MathServer},\leqslant_T\vdash op1:in(ch(Float))\wedge\leqslant_T\vdash ch(Float)\leqslant ch(Float)}{\Gamma_{MathServer},\leqslant_T\vdash op1(ch1:ch(Float)).Rect}$$

同样的推理过程，我们有 $\Gamma_{MathServer},\leqslant_T\vdash op2(ch1:ch(Float)).Circle$。最后，利用规则 T-SUM，可以得到 $\Gamma_{MathServer},\leqslant_T\vdash MathServer$。

用同样的方法，通过合并 $\Gamma_{MathServer}$ 和 $\Gamma_{Cusetomer}$，我们可以推导出 $MathServer|Customer$ 是类型良好的。

在验证了上述几个进程都是类型良好的以后，下一步需要验证 Web 服务动态行为的匹配性。这一步通过利用进程的操作语义对进程 $MathServer|Customer$ 的交互过程进行推导，看是否存在死锁现象。如果 $MathServer|Customer$ 经过一系列前缀动作最终成为 0 进程，则说明 Web 服务的组合是正确的。利用规则 R-COMM，我们可以推导出 $MathServer|Customer$ 最终可以变迁为 0 进程而不会发生任何死锁。

同时，利用自动模型检查和验证的工具 Mobility Workbench[10](MWB)，我们也可以对 $MathServer|Customer$ 进行自动验证。通过 MWB 的 *deadlocks* 命令检查，$MathServer|Customer$ 不存在死锁；利用 *step* 命令来模拟 $MathServer|Customer$ 的运行，$MathServer|Customer$ 最终可以变迁为 0 进程。利用文[11]提出的验证方法，我们将客户进程所有行为反转(输出变输入，输入变输出)，这样就得到客户进程 $Custom$ 的镜像进程 $RevCustomer$：

$RevCustomer\ (op1,op2,ra,rc)=$
$(\tilde{c}h1,ch2,l,w,r)\ (op1(ch1).ch1(l).ch1(w).{}'ch1\langle ra\rangle.0+op2(ch2).ch2(r).{}'ch2\langle rc\rangle.0)$

通过 *weq* $MathServer$ $RevCustomer$ 命令检查，$MathServer$ 和 $RevCustomer$ 是弱相似的，因此 $MathServer$ 和 $Customer$ 的动态行为是匹配的。

最后，我们假设 $Customer$ 通过通道 $op1$ 向 $MathServer$ 发送了一个错误类型的数据(例如发送了一个 Int 型数据而不是通道类型如 Socket)。根据规则，由于 Int 类型不是通道类型的子类型，根据规则 ERR-COM，$Rect$ 不再是类型良好的，接着应用规则 ERR-SUM，我们有 $MathServerr\xrightarrow{\tau}Err$，最后应用规则 ERR-PAR，我们可以推导出 $MathServer|Customer\xrightarrow{\tau}Err$。反之，如果 $Customer$ 通过 $ch1$ 向 $Rect$ 发送 Int 型的参数，由于 $Int\leqslant Float$，我们用同样的推理得出结论：$Customer$ 和 $Rect$ 仍然是类型良好的。

总结 Web 服务组合的正确性包括动态行为的匹配和数据类型的一致性。本文在 Simply Typed Pi-calculus 的基础上定义了一个扩充的 Pi-演算类型系统，并且利用该系统建立了一个类型化的 BPEL4WS 的形式化模型。最后通过一个案例，给出了利用类型化的 BPEL4WS 的形式化模型对 Web 服务组合的建模方法和对 Web 服务组合的类型一致性和动态行为的匹配性的验证方法。

参 考 文 献

1 Andrews T, Cubera F, Dholakia H, et al. Business Process Execution Language for Web Services Version1.1. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/, 2003-05-05

2 Kavantzas N, Burdett D, Rizinger G, et al. Web Service Choreography Description Language Version 1.0. http://www.w3.org/TR/ws-cdl-10/, 2005-11

3 Pierce B C. 类型和程序设计语言. 北京：电子工业出版社，2005

4 Meredith L G, Bjorg S. Contracts and Types. Communications of the ACM, 2003, 46(10):41～47

5 Sangiorgi D, Walker D. The Pi-calculus: a Theory of Mobile Processes. Cambridge: Cambridge University Press, 2001

6 Gay S, Hole M. Subtyping for session Types in the Pi Calculus. Acta Informatica, 2005, 42(2):192～225

7 Igarashi A, Kobayashi N. A generic type system for the Pi-calculus. Theoretical Computer Science, 2004, 311(1-3):121～163

8 Pahl C. A pi-calculus based framework for the composition and replacement of components. In: Proceedings of the Workshop on Specification and Verification of Component-based Systems (OOPSLA 2001). USA, 2001

9 辜希武，卢正鼎. 基于 Pi-演算的 BPEL4WS Web 服务组合形式化模型. 计算机科学，2007, 34(3):69～74

10 Victor B, Moller F. The Mobility Workbench-A Tool for the Pi-Calculus. In: Proceedings of the 6th International Conference on Computer Aided Verification, USA, 1994

11 Salaün G, Bordeaux L, Schaerf M. Describing and Reasoning on Web Services Using Process Algebra. In: Proceedings of the IEEE International Conference on Web Services (ICWS'04). USA, 2004