Modelling Coordination and Compensation

He Jifeng*

Shanghai Key Laboratory of Trustworthy Computing
East China Normal University, China

Abstract. Transaction-based services are increasingly being applied in solving many universal interoperability problems. Exception and failure are the typical phenomena of the execution of long-running transactions. To accommodate these new program features, we extend the Guarded Command Language [10] by addition of compensation and coordination combinators, and enrich the standard design model [15] with new healthiness conditions. This paper shows that such an extension is conservative one because it preserves the algebraic laws for designs, which can be used to reduce all programs to a normal form algebraically. We also explore a Galois link between the standard design model with our new model, and show that the embedding from the former to the latter is actually a homomorphism.

1 Introduction

With the development of Internet technology, web services play an important role to information systems. The aim of web services is to achieve the universal interoperability between different web-based applications. In recent years, in order to describe the infrastructure for carrying out long-running transactions, various business modelling languages have been introduced, such as XLANG, WSFL, BPEL4WS (BPEL) and StAC [25,16,9,7].

Coordination and compensation mechanisms are vital in handling exception and failure which occur during the execution of a long-running transaction. Butler *et al.* investigated the compensation feature in a business modelling language StAC (Structured Activity Compensation) [6]. Further, Bruni *et al.* studied the transaction calculi for StAC programs, and provided a process calculi in the form of Java API. [4]. Qiu *et al.* have provided a deep formal analysis of the coordination behaviour for BPEL-like processes [23]. Pu *et al.* formalised the operational semantics for BPEL [22], where bisimulation has been considered. The π-calculus has been applied in describing various compensable program models. Lucchi and Mazzara defined the semantics of BPEL within the framework of the π-calculus [19]. Laneve and Zavattaro explored the application of the π-calculus in the formalisation of the compensable programs and the standard pattern of composition [17]. We introduced the notation of design matrix to describe various irregular phenomena of compensable programs in [12,13].

* This work was supported by the National Basic Research Program of China (Grant No. 2005CB321904) and Shanghai Leading Academic Discipline Project B412.

T. Margaria and B. Steffen (Eds.): ISoLA 2008, CCIS 17, pp. 15–36, 2008.

This paper is an attempt at taking a step forward to gain some perspectives on long-running transactions within the design calculus [15]. Our novel contributions include

- an enriched design model to handle exception and program failure.
- a set of new programming combinators for compensation and coordination
- an algebraic system in support of normal form reduction.
- a Galois link between the standard design model with our new model

The paper is organised as follows: Section 2 provides a mathematical framework to describe the new program features. Section 3 extends the Guarded Command Language by addition of compensation and coordination combinators to synchronise the activity of programs. It also investigates the algebraic properties of our new language. We introduce normal form in Section 4, and show that all programs can be reduced to a normal form algebraically. Section 5 establishes a Galois link between the standard design model with our new model, and prove that the embedding from the former to the latter is a homomorphism. The paper concludes with a short summary.

2 An Enriched Design Model

In this section we work towards a precise characterisation of the class of *designs* [15] that can handle new programming features such as program failure, coordination and compensation.

A subclass of designs may be defined in a variety of ways. Sometimes it is done by a syntactic property. Sometimes the definition requires satisfaction of a particular collection of algebraic laws. In general, the most useful definitions are these that are given in many different forms, together with a proof that all of them are equivalent. This section will put forward additional healthiness conditions to capture such a subclass of designs. We leave their corresponding algebraic laws in Section 3.

2.1 Exception Handling

To handling exception requires a more explicit analysis of the phenomena of program execution. We therefore introduce into the alphabet of our designs a pair of Boolean variables $eflag$ and $eflag'$ to denote the relevant observations:

- $eflag$ records the observation that the program is asked to start when the execution of its predecessor halts due to an exception.
- $eflag'$ records the observation that an exception occurs during the execution of the program.

The introduction of error states has implication for sequential composition: all the exception cases of program P are of course also the exception cases of $P;Q$. Rather than change the definition of sequential composition given in [15], we enforce these rules by means of a healthiness condition: if the program Q is asked to start in an exception case of its predecessor, it leaves the state unchanged

($\mathbf{Req}_1$) $Q \;=\; II \lhd eflag \rhd Q$

when the design II adopts the following definition

$$II \;=_{df}\; true \vdash (s' = s)$$

where s denotes all the variables in the alphabet of Q.

A design is $\mathbf{Req}_1$-healthy if it satisfies the healthiness condition $\mathbf{Req}_1$. Define

$$\mathcal{H}_1(Q) \;=_{df}\; (II \lhd eflag \rhd Q)$$

Clearly $\mathcal{H}_1$ is idempotent. As a result, Q is $\mathbf{Req}_1$ healthy if and only if Q lies in the range of $\mathcal{H}_1$.

The following theorem indicates $\mathbf{Req}_1$-healthy designs are closed under conventional programming combinators.

Theorem 2.1

(1) $\mathcal{H}_1(P \sqcap Q) \;=\; \mathcal{H}_1(P) \sqcap \mathcal{H}_1(Q)$

(2) $\mathcal{H}_1(P \lhd b \rhd Q) \;=\; \mathcal{H}_1(P) \lhd b \rhd \mathcal{H}_1(Q)$

(3) $\mathcal{H}_1(P; \mathcal{H}_1(Q)) \;=\; \mathcal{H}_1(P); \mathcal{H}_1(Q)$

2.2 Rollback

To equip a program with compensation mechanism, it is necessary to figure out the cases when the execution control has to rollback. By adopting the technique used in the exception handling model, we introduce a new logical variable $forward$ to describe the status of control flow of the execution of a program:

- $forward' = true$ indicates successful termination of the execution of the forward activity of a program. In this case, its successor will carry on with the initial state set up by the program.
- $forward' = false$ indicates it is required to undo the effect caused by the execution of the program. In this case, the corresponding compensation module will be invoked.

As a result, a program must keep idle when it is asked to start in a state where $forward = false$, i.e., it has to meet the following healthiness condition:

($\mathbf{Req}_2$) $Q \;=\; II \lhd \neg forward \rhd Q$

This condition can be identified by the idempotent mapping

$$\mathcal{H}_2(Q) \;=_{df}\; II \lhd \neg forward \rhd Q$$

in the sense that a program meets $\mathbf{Req}_2$ iff it is a fixed point of $\mathcal{H}_2$.

We can charecterise both $\mathbf{Req_1}$ and $\mathbf{Req}_2$ by composing $\mathcal{H}_1$ and $\mathcal{H}_2$. To ensure that their composition is an idempotent mapping we are going to show that

Theorem 2.2

$\mathcal{H}_2 \circ \mathcal{H}_1 \;=\; \mathcal{H}_1 \circ \mathcal{H}_2$

Proof: From the fact that

$$\mathcal{H}_1(\mathcal{H}_2(Q)) \;=\; II \lhd eflag \vee \neg foward \rhd Q \;=\; \mathcal{H}_2(\mathcal{H}_1(Q))$$

Define $\mathcal{H} =_{df} \mathcal{H}_1 \circ \mathcal{H}_2$..

Theorem 2.3

A design is healthy (i.e., it satisfies both $\mathbf{Req_1}$ and $\mathbf{Req_2}$) iff it lies in the range of $\mathcal{H}$.

The following theorem indicates that healthy designs are closed under the conventional programming combinators.

Theorem 2.4

(1) $\mathcal{H}(P \sqcap Q) \;=\; \mathcal{H}(P) \sqcap \mathcal{H}(Q)$

(2) $\mathcal{H}(P \lhd b \rhd Q) \;=\; \mathcal{H}(P) \lhd b \rhd \mathcal{H}(Q)$

(3) $\mathcal{H}(P; \mathcal{H}(Q)) \;=\; \mathcal{H}(P); \mathcal{H}(Q)$

In the following sections, we will confine ourselves to healthy designs only.

3 Programs

This section studies a simple programming language, which extends the Guarded Command Language [10] by adding coordination constructs. The syntax of the language is as follows:

$$\begin{aligned} P \;::=\; & \texttt{skip} \mid \texttt{fail} \mid \texttt{throw} \mid \bot \mid x := e \mid \\ & P \sqcap P \mid P \lhd b \rhd P \mid P; P \mid b *_{\mathcal{H}} P \mid \\ & P \;\texttt{cpens}\; P \mid P \;\texttt{else}\; P \mid P \;\texttt{catch}\; P \mid P \;\texttt{or}\; P \mid P \;\texttt{par}\; P| \end{aligned}$$

In the following discussion, v will represent the program variables referred in the alphabet of the program.

3.1 Primitive Commands

The behaviour of the chaotic program $\bot$ is totally unpredictable

$$\bot \;=_{df}\; \mathcal{H}(\mathbf{true})$$

The execution of `skip` leaves program variables intact.

$$\texttt{skip} \;=_{df}\; \mathcal{H}(\mathbf{success})$$

where $\mathbf{success} \;=_{df}\; true \vdash ((v' = v) \wedge forward' \wedge \neg eflag')$

The execution of `fail` rollbacks the control flow.

$$\texttt{fail} \;=_{df}\; \mathcal{H}(\mathbf{rollback})$$

where $\mathbf{rollback} \;=_{df}\; true \vdash ((v' = v) \wedge \neg forward' \wedge \neg eflag')$

An exception case arises from the execution of `throw`

$$\texttt{throw} =_{df} \mathcal{H}(\mathbf{error})$$

where $\mathbf{error} =_{df} true \vdash ((v' = v) \wedge eflag')$

3.2 Nondeterministic Choice and Sequential Composition

The nondeterministic choice and sequential composition have exactly the same meaning as the corresponding operators on the single predicates defined in [15].

$$P; Q =_{df} \exists m \bullet (P[m/s'] \wedge Q[m/s])$$
$$P \sqcap Q =_{df} P \vee Q$$

The change in the definition of $\perp$ and `skip` requires us to give a proof of the relevant laws.

Theorem 3.1

(1) $\texttt{skip}; P = P = P; \texttt{skip}$

(2) $\perp; P = \perp$

(3) $\perp \sqcap P = \perp$

Proof: Let $s = (v, forward, eflag)$.

(1) $\texttt{skip}; P$ $\quad \{\text{Theorem 2.4(3)}\}$

$= \mathcal{H}(\mathbf{success}; P)$ $\quad \{\mathcal{H}(Q) = \mathcal{H}((forward \wedge \neg eflag)^\top; Q)\}$

$= \mathcal{H}((true \vdash (s' = s)); P)$ $\quad \{(true \vdash (s' = s); D = D\}$

$= \mathcal{H}(P)$ $\quad \{P \text{ is healthy}\}$

$= P$

Besides the laws presented in [15] for composition and nondeterministic choice, there are additional left zero laws for sequential composition.

Theorem 3.2

(1) $\texttt{throw}; P = \texttt{throw}$

(2) $\texttt{fail}; P = \texttt{fail}$

Proof:

(1) $\texttt{throw}; P$ $\quad \{\text{Theorem 2.4(3)}\}$

$= \mathcal{H}(\mathbf{error}; P)$ $\quad \{\text{Def of } \mathbf{error}\}$

$= \mathcal{H}(\mathbf{error}; (eflag)_\perp; P)$ $\quad \{P = \mathcal{H}(P)\}$

$= \mathcal{H}(\mathbf{error}; (eflag)_\perp; \mathcal{H}(P)[true/eflag])$ $\quad \{\text{Def of } \mathcal{H}\}$

$= \mathcal{H}(\mathbf{error}; (eflag)_\perp)$ $\quad \{\text{Def of } \texttt{throw}\}$

$= \texttt{throw}$

3.3 Assignment

Successful execution of an assignment relies on the assumption that the expression will be successfully evaluated.

$$x := e \;=_{df}\; \texttt{skip}[e/x] \lhd \mathcal{D}(e) \rhd \texttt{throw}$$

where the boolean condition $\mathcal{D}(e)$ is true in just those circumstances in which e can be successfully evaluated [21]. For example we can define

$$\mathcal{D}(c) =_{df} true \quad \text{if } c \text{ is a constant}$$

$$\mathcal{D}(e_1 + e_2) =_{df} \mathcal{D}(e_1) \wedge \mathcal{D}(e_2)$$

$$\mathcal{D}(e_1/e_2) =_{df} \mathcal{D}(e_1) \wedge \mathcal{D}(e_2) \wedge e_2 \neq 0$$

$$\mathcal{D}(e_1 \lhd b \rhd e_2) =_{df} \mathcal{D}(b) \wedge (b \Rightarrow \mathcal{D}(e_1)) \wedge (\neg b \Rightarrow \mathcal{D}(e_2))$$

Notice that $\mathcal{D}(e)$ is always well-defined, i.e., $\mathcal{D}(\mathcal{D}(e)) = true$.

Definition 3.1

An assignment is *total* if its assigning expression is well-defined, and all the variables of the program appear on its left hand side.

3.4 Conditional

The definition of conditional and iteration take the well-definedness of its Boolean test into account

$$P \lhd b \rhd Q \;=_{df}\; (\mathcal{D}(b) \wedge b \wedge P) \;\vee\; (\mathcal{D}(b) \wedge \neg b \wedge Q) \;\vee\; \neg\mathcal{D}(b) \wedge \texttt{throw}$$

$$b *_{\mathcal{H}} P \;=_{df}\; \mu_{\mathcal{H}} X \bullet (P; X) \lhd b \rhd \texttt{skip}$$

where $\mu_{\mathcal{H}} X \bullet F(X)$ stands for the weakest **Req**− healthy solution of the equation $X = F(X)$.

The alternation is defined in a similar way

$$\mathbf{if}(b_1 \to P_1, .., b_n \to P_n)\mathbf{fi} \;=_{df}\; \begin{pmatrix} \bigvee_i (\mathcal{D}(b) \wedge b_i \wedge P_i) \vee \\ \mathcal{D}(b) \wedge \neg b \wedge \bot \;\vee \\ \neg\mathcal{D}(b) \wedge \texttt{throw} \end{pmatrix}$$

where $b =_{df} \bigvee_i b_i$.

The following theorem illustrates how to convert a conditional into an alternation with well-defined boolean guards.

Theorem 3.3

$P \lhd b \rhd Q \;=$

$\mathbf{if}((b \lhd \mathcal{D}(b) \rhd false) \to P,\; (\neg b \lhd \mathcal{D}(b) \rhd false) \to Q,\; \neg\mathcal{D}(b) \to \texttt{throw})\mathbf{fi}$

A similar transformation can be applied to an assignment.

Theorem 3.4

$x := e \; = \; (x, y, ..z \; := \; (e, y, .., z) \lhd \mathcal{D}(e) \rhd (x, \; y, .., z)) \lhd \mathcal{D}(e) \rhd \texttt{throw}$

The previous theorems enable us to confine ourselves to well-defined expressions in later discussion. For total assignment, we are required to reestablish the following laws.

Theorem 3.5

(1) $(x := e; x := f(x)) \; = \; (x := f(e))$

(2) $x := e; (P \lhd b(x) \rhd Q) \; = \; (x := e; P) \lhd b(e) \rhd (x := e; Q)$

(3) $(x := e) \lhd b \rhd (x := f) \; = \; x := (e \lhd b \rhd f)$

(4) $(x := x) \; = \; \texttt{skip}$

The following laws for alternation will be used in later normal form reduction.

Theorem 3.6

Let $\underline{G}$ denote a list of alternatives.

(1) $\mathbf{if}(b_1 \rightarrow P_1, ... P_2, .. b_n \rightarrow P_n)\mathbf{fi} \; = \; \mathbf{if}(b_{\pi(1)} \rightarrow P_{\pi(1)}, .., b_{\pi(n)} \rightarrow P_{\pi(n)})\mathbf{fi}$

where π is an arbitrary permutation of $\{1, .., n\}$.

(2) $\mathbf{if}(b \rightarrow \mathbf{if}(c_1 \rightarrow Q_1, .., c_n \rightarrow Q_n)\mathbf{fi}, \underline{G})\mathbf{fi} \; =$

$\mathbf{if}(b \wedge c_1 \rightarrow Q_1, .., b \wedge c_n \rightarrow Q_n, \underline{G})\mathbf{fi}$

provided that $\bigvee_k c_k \; = \; true$

(3) $\mathbf{if}(b \rightarrow P, \; b \rightarrow Q, \; \underline{G})\mathbf{fi} \; = \; \mathbf{if}(b \rightarrow (P \sqcap Q), \; \underline{G})\mathbf{fi}$

(4) $\mathbf{if}(b \rightarrow P, \; c \rightarrow Q, \; \underline{G})\mathbf{fi} \; = \; \mathbf{if}(b \vee c \rightarrow (P \lhd b \rhd Q) \sqcap (Q \lhd c \rhd P), \; \underline{G})\mathbf{fi}$

(5) $\mathbf{if}(b_1 \rightarrow P_1, .., b_n \rightarrow P_n)\mathbf{fi}\,; \, Q \; = \; \mathbf{if}(b_1 \rightarrow (P_1; Q), .., b_n \rightarrow (P_n; Q))\mathbf{fi}$

(6) $\mathbf{if}(b_1 \rightarrow P_1, .., b_n \rightarrow P_n)\mathbf{fi} \sqcap Q \; = \; \mathbf{if}(b_1 \rightarrow (P_1 \sqcap Q), .., b_n \rightarrow (P_n \sqcap Q)\mathbf{fi}$

(7) $\mathbf{if}(b_1 \rightarrow P_1, .., b_n \rightarrow P_n)\mathbf{fi} \wedge Q \; = \; \mathbf{if}(b_1 \rightarrow (P_1 \wedge Q), .., b_n \rightarrow (P_n \wedge Q))\mathbf{fi}$

provided that $\bigvee_k b_k \; = \; true$

(8) $\mathbf{if}(false \rightarrow P, \; \underline{G})\mathbf{fi} \; = \; \mathbf{if}(\underline{G})\mathbf{fi}$

(9) $\mathbf{if}(b_1 \rightarrow P_1, .., b_n \rightarrow P_n)\mathbf{fi} \; = \; \mathbf{if}(b_1 \rightarrow P_1, .., b_n \rightarrow P_n, \; \neg \vee_i b_i \rightarrow \bot)\mathbf{fi}$

(10) $\mathbf{if}(true \rightarrow P)\mathbf{fi} \; = \; P$

3.5 Exception Handling

Let P and Q be programs. The notation P `catch` Q represents a program which runs P first, and if its execution throws an exception case then Q is activated.

$$P \; \texttt{catch} \; Q \; =_{df} \; \mathcal{H}(P; \phi(Q))$$

where $\phi(Q) \; =_{df} \; II \lhd \neg eflag \rhd Q[false, true/eflag, forward]$

Theorem 3.7

(1) $P \ \texttt{catch}\ (Q \ \texttt{catch}\ R) \ = \ (P \ \texttt{catch}\ Q) \ \texttt{catch}\ R$

(2) $(\texttt{throw}\ \texttt{catch}\ Q) \ = \ Q \ = \ (Q \ \texttt{catch}\ \texttt{throw})$

(3) $P \,\texttt{catch}\, Q \ = \ P$ if $P \in \{\bot, \texttt{fail}, (v := e)\}$

(4) $\mathbf{if}(b_1 \to P_1, .., b_n \to P_n)\mathbf{fi}\ \texttt{catch}\ Q \ =$
$\mathbf{if}(b_1 \to (P_1 \ \texttt{catch}\ Q), .., b_n \to (P_n \ \texttt{catch}\ Q))\mathbf{fi}$

(5) $(P \sqcap Q) \ \texttt{catch}\ R \ = \ (P \ \texttt{catch}\ R) \sqcap (Q \ \texttt{catch}\ R)$

(6) $P \ \texttt{catch}\ (Q \sqcap R) \ = \ (P \ \texttt{catch}\ Q) \sqcap (P \ \texttt{catch}\ R)$

Proof:

(1) LHS $\{\text{Def of } \texttt{catch}\}$

$= \mathcal{H}(\mathcal{H}(P; \phi(Q)); \phi(R))$ $\{\text{Def of } \mathcal{H}\}$

$= \mathcal{H}((forward \wedge \neg eflag)^\top;$
$\mathcal{H}(P; \phi(Q)); \phi(R))$ $\{Q \lhd false \rhd P \ = \ P\}$

$= \mathcal{H}(P; \phi(Q); \phi(R))$ $\{\phi(Q); \phi(R) \ = \ \phi(Q; \phi(R))\}$

$= \mathcal{H}(P; \phi(Q; \phi(R)))$ $\{\phi(S) \ = \ \phi(\mathcal{H}(S))\}$

$= \mathcal{H}(P; \phi(\mathcal{H}(Q; \phi(R))))$ $\{\text{Def of } \texttt{catch}\}$

$= \mathcal{H}(P; \phi(Q \,\texttt{catch}\, R))$ $\{\text{Def of } \texttt{catch}\}$

$= RHS$

(2) $\texttt{throw}\,\texttt{catch}\, Q$ $\{\text{Def of } \texttt{catch}\}$

$= \mathcal{H}(\texttt{throw}; \phi(Q))$ $\{\text{Def of } \texttt{throw}\}$

$= \mathcal{H}(Q[false, true/eflag, forward])$ $\{\text{Def of } \mathcal{H}\}$

$= \mathcal{H}(Q)$ $\{Q \ = \ \mathcal{H}(Q)\}$

$= Q$ $\{\phi\texttt{throw} \ = \ \texttt{skip}\}$

$= Q \,\texttt{catch}\,\texttt{throw}$

(3) LHS $\{\text{Def of } \texttt{catch}\}$

$= \mathcal{H}((v := e); \phi(Q))$ $\{\text{Def of } \mathcal{H}\}$

$= \mathcal{H}((forward \wedge \neg efalg)^\top; (v := e); \phi(Q))$ $\{e \text{ is well-defined}\}$

$= \mathcal{H}((forward \wedge \neg efalg)^\top; (v := e);$
$(forward \wedge \neg eflag)_\bot; \phi(Q))$ $\{\text{Def of } \phi\}$

$= \mathcal{H}((forward \wedge \neg efalg)^\top; (v := e);$
$(forward \wedge \neg eflag)_\bot)$ $\{(v := e) = \mathcal{H}(v := e)\}$

$= RHS$

(5) LHS $\{\text{Def of } \texttt{catch}\}$

$= \mathcal{H}(\mathbf{if}(b_1 \rightarrow P_1,\ b_n \rightarrow P_n)\mathbf{fi}\,;\ \phi(R))$ $\{\text{Theorem 3.6(5)}\}$

$= \mathcal{H}(\mathbf{if}(b_1 \rightarrow (P_1; \phi(R),$

$b_n \rightarrow (P_n; \phi(R))\mathbf{fi}$ $\{\text{Theorem 2.4(2)}\}$

$= RHS$

3.6 Compensation

Let P and Q be programs. The program $P\,\texttt{cpens}\,Q$ runs P first. If its execution fails, then Q is invoked as its compensation.

$$P\ \texttt{cpens}\ Q\ =_{df}\ \mathcal{H}(P; \psi(Q)$$

where $\psi(Q) =_{df} (II \lhd forward \vee eflag \rhd Q[true/forward])$

Theorem 3.8

(1) $P\ \texttt{cpens}\ (Q\ \texttt{cpens}\ R)\ =\ (P\ \texttt{cpens}\ Q)\ \texttt{cpens}\ R$

(2) $P\ \texttt{cpens}\ Q\ =\ P$ if $P \in \{\texttt{throw}, \bot, (v := e)\}$

(3) $(\texttt{fail}\,\texttt{cpens}\,Q)\ =\ Q\ =\ (Q\ \texttt{cpens}\ \texttt{fail})$

(4) $\mathbf{if}(b_1 \rightarrow P_1, .., b_n \rightarrow P_n)\mathbf{fi}\,\texttt{cpens}\,Q\ =$

$\mathbf{if}(b_1 \rightarrow (P_1\,\texttt{cpens}\,Q), .., b_n \rightarrow (P\,\texttt{cpens}\,Q))\mathbf{fi}$

(5) $(P \sqcap Q)\,\texttt{cpens}\,R\ =\ (P\,\texttt{cpens}\,R) \sqcap (Q\,\texttt{cpens}\,R)$

(6) $P\,\texttt{cpens}\,(Q \sqcap R)\ =\ (P\,\texttt{cpens}\,Q) \sqcap (P\,\texttt{cpens}\,R)$

(7) $(v := e; P)\,\texttt{cpens}\,Q\ =\ (v := e); (P\,\texttt{cpens}\,Q)$

Proof:

Let $B\ =_{df}\ (forward \wedge \neg eflag)$.

(1) RHS $\{\text{Def of } \texttt{cpens}\}$

$= \mathcal{H}(\mathcal{H}(P; \psi(Q)); \psi(R))$ $\{\text{Def of } \mathcal{H}\}$

$= \mathcal{H}(B^{\top}; \mathcal{H}(P; \psi(Q)); \psi(R))$ $\{Q \lhd false \rhd P\ =\ P\}$

$= \mathcal{H}(P; \psi(Q); \psi(R))$ $\{\psi(Q; \psi(R)\ =\ \psi(Q; \phi(R))\}$

$= \mathcal{H}(P; \psi(Q; \psi(R)))$ $\{\psi(Q)\ =\ \psi(\mathcal{H}(Q))\}$

$= \mathcal{H}(P; \psi(\mathcal{H}(Q; \psi(R))))$ $\{\text{Def of } \texttt{cpens}\}$

$= LHS$

(7) LHS $\{\text{Def of } \texttt{cpens}\}$

$= \mathcal{H}(v := e; P; \psi(Q))$ $\{B^{\top}; (v := e) \;=\; B^{\top}; (v := e); B_{\perp}\}$

$= \mathcal{H}(v := e; B_{\perp}; P; \psi(Q))$ $\{\text{Def of } \mathcal{H}\}$

$= \mathcal{H}(v := e; B_{\perp}; \mathcal{H}(P; \psi(Q)))$ $\{B^{\top}; (v := e) \;=\; B^{\top}; (v := e); B_{\perp}\}$

$= \mathcal{H}(v := e; \mathcal{H}(P; \psi(Q)))$ $\{\text{Theorem 2.4(3)}\}$

$= RHS$

3.7 Coordination

Let P and Q be programs. The program $P \,\texttt{else}\, Q$ behaves like P if its execution succeeds. Otherwise it behaves like Q.

$$P \;\texttt{else}\; Q \;=_{df}\; (P; forward^{\top}) \vee (\exists t' \bullet P[false/forward'] \wedge Q)$$

where t denotes the vector variable $< ok,\, eflag,\, v >$.

Theorem 3.9

(1) $P \;\texttt{else}\; P \;=\; P$

(2) $P \;\texttt{else}\; (Q \,\texttt{else}\, R) \;=\; (P \;\texttt{else}\; Q) \,\texttt{else}\, R$

(3) $P \;\texttt{else}\; Q \;=\; P$ if $P \in \{\perp,\, (v := e),\, (v := e; \texttt{throw})\}$

(4) $(x := e\; \texttt{fail})\; \texttt{else}\, Q \;=\; Q$

(5) $\mathbf{if}(b_1 \rightarrow P_1, ..., b_n \rightarrow P_n)\mathbf{fi} \;\texttt{else}\; R \;=$

$\mathbf{if}(b_1 \rightarrow (P_1 \;\texttt{else}\; R), .., b_n \rightarrow (P_n \;\texttt{else}\; R))\mathbf{fi}$

(6) $P \;\texttt{else}\; \mathbf{if}(c_1 \rightarrow Q_1, ..., c_n \rightarrow Q_n)\mathbf{fi} =$

$\mathbf{if}(c_1 \rightarrow (P \;\texttt{else}\; Q_1), .., c_n \rightarrow (P \;\texttt{else}\; Q_n))\mathbf{fi}$

provided that $\bigvee_k c_k \;=\; true$

(7) $(P \sqcap Q) \;\texttt{else}\; R \;=\; (P \;\texttt{else}\; R) \sqcap (Q \;\texttt{else}\; R)$

(8) $P \;\texttt{else}\; (Q \sqcap R) \;=\; (P \;\texttt{else}\; Q) \sqcap (P \;\texttt{else}\; R)$

Proof:

(1) LHS $\{\text{Def of } \texttt{else}\}$

$= P; forward^{\top} \vee \exists t' \bullet P[false/forward'] \wedge P$ $\{\text{predicate calculus}\}$

$= (\exists t' \bullet P[false/forward'] \;\vee\; \neg \exists t' \bullet P[false/forward'] \wedge \exists t' \bullet P[true/forward']) \wedge$
$(P; forward^{\top}) \;\vee \exists t' \bullet P[false/forward'] \wedge P$ $\{forward^{\top} \vee II \;=\; II\}$

$= (\neg \exists t' \bullet P[false/forward'] \wedge \exists t' \bullet P[true/forward']) \wedge (P; forward^{\top}) \vee$
$\exists t' \bullet P[false/forward'] \wedge P$ $\{P; II \;=\; P\}$

$= (\neg \exists t' \bullet P[false/forward'] \wedge \exists t' \bullet P[true/forward']) \wedge P \vee$
$\exists t' \bullet P[false/forward'] \wedge P$ $\{\text{predicate calculus}\}$

$= (\exists t' \bullet P[true/forward'] \vee \exists t' P[false/forward']) \wedge P$ $\{\exists t',\, forward' \bullet P \;=\; true\}$

$= RHS$

(6) LHS {Def of `else`}

$= P; forward^{\top} \vee \exists t' \bullet P[false/forward'] \wedge$

$\quad \mathbf{if}(c_1 \to Q_1, .., c_n \to Q_n)\mathbf{fi}$ {Theorem 3.6(7)}

$= P; forward^{\top} \vee \mathbf{if}(c_1 \to \exists t' \bullet P[false/forward'] \wedge Q_1, ...$

$\quad c_n \to \exists t' \bullet P[false/forward'] \wedge Q_n)\mathbf{fi}$ {Theorem 3.6(6)}

$= \mathbf{if}(c_1 \to (P; forward^{\top} \vee \exists t' \bullet P[false/forward'] \wedge Q_1), ..$

$\quad c_n \to (P; forward^{\top} \vee \exists t' \bullet P[false/forward'] \wedge Q_n))\mathbf{fi}$ {Def of `else`}

$= RHS$

The choice construct $P \,\texttt{or}\, Q$ selects a successful one between P and Q. When both P and Q succeed, the choice is made nondeterministically.

$$P \texttt{ or } Q \;=_{df}\; (P \texttt{ else } Q) \sqcap (Q \texttt{ else } P)$$

Theorem 3.10

(1) $P \,\texttt{or}\, P \;=\; P$

(2) $P \,\texttt{or}\, Q \;=\; Q \,\texttt{or}\, P$

(3) $(P \,\texttt{or}\, , Q) \,\texttt{or}\, R \;=\; P \,\texttt{or}\, (Q \,\texttt{or}\, R)$

(4) $\mathbf{if}(b_1 \to P_1, .., b_n \to P_n)\mathbf{fi} \,\texttt{or}\, Q \;=\; \mathbf{if}(b_1 \to (P_1 \,\texttt{or}\, Q), ..., b_n \to (P_n \,\texttt{or}\, Q))\mathbf{fi}$

provided that $\bigvee_k b_k \;=\; true$

(5) $(P \sqcap Q) \,\texttt{or}\, R \;=\; (P \,\texttt{or}\, R) \sqcap (Q \,\texttt{or}\, R)$

Proof:

(1) From Theorem 3.9(1)

(2) From the symmetry of $\sqcap$

(3) From Theorem 3.9(2)

(4) From Theorem 3.9(7) and (8)

(5) From Theorem 3.9(9) and (10)

Let P and Q be programs with disjoint alphabets. The program $P \,\texttt{par}\, Q$ runs P and Q in parallel. It succeeds only when both P and Q succeed. Its behaviour is described by the *parallel merge* construct defined in [15]:

$$P \texttt{ par } Q \;=_{df}\; (P\|_M Q)$$

where the parallel merge operator $\|_M$ is defined by

$$P \|_M Q \;=_{df}\; (P[0.m'/m']\|Q[1.m'/m']); M(ok, 0.m, 1.m, m', ok')$$

where m represents the shared variables $forward$ and $eflag$ of P and Q, and $\|$ denotes the disjoint parallel operator

$$(b \vdash R)\|(c \vdash S) \;=_{df}\; (b \wedge c) \vdash (R \wedge S)$$

and the merge predicate M is defined by

$$M =_{df}$$

$$true \vdash \begin{pmatrix} (eflag' = 0.eflag1 \vee 1.eflag) \wedge \\ (\neg 0.eflag \wedge \neg 1.eflag) \Rightarrow (forward' = 0.forward1 \wedge 1.forward) \wedge \\ (v' = v) \end{pmatrix}$$

We borrow the following definition and lemma from [15] to explore the algebraic properties of `par`.

Definition 3.2 (valid merge)

A merge predicate $N(ok,\, 0.m,\, 1.m,\, m',\, ok')$ is valid if it is a design satisfying the following properties

(1) N is symmetric in its input $0.m$ and $1.m$

(2) N is associative

$$N3(1.m,\, 2.m,\, 0.m/0.m,\, 1.m,\, 2.m] = N3$$

where $N3$ is a three-way merge relation

$$N3 =_{df} \exists x,\, t \bullet N(ok,\, 0.m,\, 1.m,\, x,\, t) \wedge N(t,\, x,\, 2.m,\, m',\, ok')$$

(3) $N[m,\, m,\, /0.m,\, 1.m] = true \vdash (m = m') \wedge (v' = v)$

where m represents the shared variables of parallel components.

Lemma 3.1

If N is valid then the parallel merge $\|_N$ is symmetric and associative.

From the definition of the merge predicate M we can show that M is a valid merge predicate.

Theorem 3.11

(1) $(P \;\texttt{par}\; Q) = (Q \;\texttt{par}\; P)$

(2) $(P \;\texttt{par}\; Q) \;\texttt{par}\; R = P \;\texttt{par}\; (Q \;\texttt{par}\; R)$

(3) $\bot \;\texttt{par}\; Q = \bot$

(4) $\textbf{if}(b_1 \rightarrow P_1, ..., b_n \rightarrow P_n)\textbf{fi} \;\texttt{par}\; Q =$

$\quad \textbf{if}(b_1 \rightarrow (P_1 \;\texttt{par}\; Q), ..., b_n \rightarrow (P_n \;\texttt{par}\; Q))\textbf{fi}$

(5) $(P \sqcap Q) \;\texttt{par}\; R = (P \;\texttt{par}\; R) \sqcap (Q \;\texttt{par}\; R)$

(6) $(v := e; P) \;\texttt{par}\; Q = (v := e); (P \;\texttt{par}\; Q)$

(7) `fail par throw` $=$ `throw`

(8) `fail par fail` $=$ `fail`

(9) `throw par throw` $=$ `throw`

(10) $\texttt{skip}_A \,\texttt{par}\, Q = Q_{+A}$

$$(b \vdash R)_{+\{x,..,z\}} =_{df} b \vdash (R \wedge x = x' \wedge .. \wedge z' = z)$$

Proof:

(1) and (2): From Lemma 3.1.

(3) From the fact that $\bot \| Q \ = \ \bot$ and $\bot ; M \ = \ \bot$

(4) From Theorem 3.6(5) and the fact that

$$\mathbf{if}(b_1 \to P_1, ..., b_n \to P_n)\mathbf{fi} \| Q \ = \ \mathbf{if}(b_1 \to (P_1 \| Q), .., b_n \to (P_n \| Q))\mathbf{fi}$$

(5) From the fact that $(P \sqcap Q) \| R \ = \ (P \parallel R) \sqcap (Q \parallel R)$

(6) From the fact that $(v := e; P) \| Q \ = \ (v := e); (P \| Q)$

4 Normal Form

The normal form we adopt for our language is an alternation of the form:

$$\mathbf{if}(b1 \to \sqcap_{i \in S1}(v := e_i),\ b2 \to \sqcap_{j \in S2}(v := f_j; \mathtt{fail}),\ b3 \to \sqcap_{k \in S3}(v := g_k; \mathtt{throw})\mathbf{fi}$$

where all expressions are well-defined, and all assignments are total, and all the index sets S_i are finite. The objective of this section is to show that all finite programs can be reduced to normal form. Our first step is to prove that normal forms are closed under the programming combinators defined in the previous section.

Theorem 4.1

Let $P \ = \ \mathbf{if}\,(b1 \to P1,\ b2 \to P2,\ b3 \to P3)\,\mathbf{fi}$

and $Q \ = \ \mathbf{if}\,(c1 \to Q1,\ c2 \to Q2,\ c3 \to Q3)\,\mathbf{fi}$, where

$P1 \ = \sqcap_{i \in S1}(v := e1_i)$	$Q1 \ = \sqcap_{i \in T1}(v := f1_i)$
$P2 \ = \sqcap_{j \in S2}(v := e2_j); \mathtt{fail}$	$Q2 \ = \sqcap_{j \in T2}(v := f2_j); \mathtt{fail}$
$P3 \ = \sqcap_{k \in S3}(v := e3_k); \mathtt{throw}$	$Q3 \ = \sqcap_{k \in T3}(v := f3_k); \mathtt{throw}$

Then $P \sqcap Q \ =$

$$\mathbf{if} \begin{pmatrix} (b1 \wedge c \vee c1 \wedge b) \to \\ \quad \sqcap_{i \in S1,\, j \in T1}(v := (e1_i \lhd b_1 \rhd f1_j)) \sqcap (v := (f1_j \lhd c_1 \rhd e1_i)) \\ (b2 \wedge c \vee c2 \wedge b) \to \\ \quad \sqcap_{i \in S2,\, j \in T2}(v := (e2_i \lhd b_2 \rhd f2_j)) \sqcap (v := (f2_j \lhd c_2 \rhd e2_i)); \mathtt{fail} \\ (b3 \wedge c \vee c3 \wedge b) \to \\ \quad \sqcap_{i \in S3,\, j \in T3}(v := (e3_i \lhd b_1 \rhd f3_j)) \sqcap (v := (f3_j \lhd c_1 \rhd e3_i)); \mathtt{throw} \end{pmatrix} \mathbf{fi}$$

where $b \ =_{df} \ b_1 \vee b_2 \vee b_3$ and $c \ =_{df} \ c_1 \vee c_2 \vee c_3$

Proof: LHS $\{\text{Theorem 3.6(6)}\}$

$$= \ \mathbf{if}(b1 \to (P1 \sqcap Q),\ b2 \to (P2 \sqcap Q),\ b3 \to (P3 \sqcap Q))\mathbf{fi}$$

$\{\text{Theorem 3.6(6)}\}$

$$= \ \mathbf{if}\begin{pmatrix} b1 \to \mathbf{if}(c1 \to (Q1 \sqcap P1),\ c2 \to (Q2 \sqcap P1),\ c3 \to (Q3 \sqcap P1))\mathbf{fi} \\ b2 \to \mathbf{if}(c1 \to (Q1 \sqcap P2),\ c2 \to (Q2 \sqcap P2),\ c3 \to (Q3 \sqcap P2))\mathbf{fi} \\ b3 \to \mathbf{if}(c1 \to (Q1 \sqcap P3),\ c2 \to (Q2 \sqcap P3),\ c3 \to (Q3 \sqcap P3))\mathbf{fi} \end{pmatrix}\mathbf{fi}$$

$\{\text{Theorem 3.6(2) and (9)}\}$

$$= \ \mathbf{if}\begin{pmatrix} (b1 \wedge c) \to P1, & (b2 \wedge c) \to P2, & (b3 \wedge c) \to P3, \\ (b \wedge c1) \to Q1, & (b \wedge c2) \to Q2, & (b \wedge c3) \to Q3 \end{pmatrix}\mathbf{fi}$$

$\{\text{Theorem 3.6(4)}\}$

$$= \ \mathbf{if}\begin{pmatrix} (b1 \wedge c \vee b \wedge c1) \to (P1 \lhd b1 \rhd Q1) \sqcap (Q1 \lhd c1 \rhd P1) \\ (b2 \wedge c \vee b \wedge c2) \to (P2 \lhd b2 \rhd Q2) \sqcap (Q2 \lhd c2 \rhd P2) \\ (b3 \wedge c \vee b \wedge c3) \to (P3 \lhd b3 \rhd Q3) \sqcap (Q3 \lhd c3 \rhd P3) \end{pmatrix}\mathbf{fi}$$

$\{\text{Theorem 3.5(3)}\}$

$$= \ RHS$$

Let

$$W \ =_{df} \ \mathbf{if}(b1 \to (x := e1),\ b2 \to (x := e2); \texttt{fail},\ b3 \to (x := e3); \texttt{throw})\mathbf{fi}$$

$$R \ =_{df} \ \mathbf{if}(c1 \to (x := f1),\ c2 \to (x := f2); \texttt{fail},\ c3 \to (x := f3); \texttt{throw})\mathbf{fi}$$

Theorem 4.2

$W; R \ =$

$$\mathbf{if}\begin{pmatrix} (b1 \wedge c1[e1/x]) \to (x := f1(e1)) \\ (b2 \wedge (\neg b_1 \vee c[e1/x]) \vee b1 \wedge c2[e1/x]) \to \\ \quad (x := (e2 \lhd b2 \rhd f2[e1/x]) \sqcap x := (f2[e1/x] \lhd c2[e1/x] \rhd e2)); \texttt{fail} \\ (b3 \wedge (\neg b1 \vee c[e1/x]) \vee b1 \wedge c3[e1/x]) \to \\ \quad (x := (e3 \lhd b3 \rhd f3[e1/x]) \sqcap x := (f3[e1/x] \lhd c3[e1/x] \rhd e3)); \texttt{throw} \end{pmatrix}\mathbf{fi}$$

Proof:

LHS $\{\text{Theorem 3.6(5)}\}$

$$= \ \mathbf{if}(b1 \to (x := e1); R,\ b2 \to (x := e2); \texttt{fail},\ b3 \to (x := e3); \texttt{throw})\mathbf{fi}$$

$\{\text{Theorem 3.5(2)}\}$

$$= \mathbf{if}\begin{pmatrix} b1 \to \mathbf{if}\begin{pmatrix} c1[e1/x] \to (x := f1[e1/x]), \\ c2[e1/x] \to (x := f2[e1/x]); \texttt{fail}, \\ c3[e1/x] \to (x := f3[e1/x]); \texttt{throw} \end{pmatrix}\mathbf{fi} \\ b2 \to (x := e2); \texttt{fail} \\ b3 \to (x := e3); \texttt{throw} \end{pmatrix}\mathbf{fi}$$

$\{\text{Theorem 3.6(2) and (3)}\}$

$$= \textbf{if}\begin{pmatrix} b1 \wedge c1[e1/x] \rightarrow (x := f1[e1/x]), \\ b1 \wedge c2[e1/x] \rightarrow (x := f2[e1/x]); \texttt{fail}, \\ b2 \wedge \neg(b1 \wedge \neg c[e1/x]) \rightarrow (x := e2); \texttt{fail} \\ b1 \wedge c3[e1/x] \rightarrow (x := f3[e1/x]); \texttt{throw}, \\ b3 \wedge \neg(b1 \wedge \neg c[e1/x]) \rightarrow (x := e3); \texttt{throw} \end{pmatrix} \textbf{fi}$$

{Theorem 3.6(4)}

$= RHS$

Theorem 4.3

$W\,\texttt{catch}\,R =$

$$\textbf{if}\begin{pmatrix} (b1 \wedge (\neg b3 \vee c[e3/x]) \vee b3 \wedge c1[e3/x]) \rightarrow \\ \quad (x := (e1 \lhd b1 \rhd f1[e3/x]) \sqcap x := (f1[e3/x] \lhd c1[e3/x] \rhd e1)) \\ (b2 \wedge (\neg b3 \vee c[e3/x]) \vee b3 \wedge c2[e3/x]) \rightarrow \\ \quad (x := (e2 \lhd b2 \rhd f2[e3/x]) \sqcap x := (f2[e3/x] \lhd c2[e3/x] \rhd e2)); \texttt{fail} \\ (b3 \wedge c3[e3/x]) \rightarrow (x := f3[e3/x]); \texttt{throw} \end{pmatrix} \textbf{fi}$$

Proof:

LHS {Theorem 3.7(2) and (3)}

$= \textbf{if}(b1 \rightarrow (x := e1),\ b2 \rightarrow (x := e2); \texttt{fail},\ b3 \rightarrow (x := e3); R)\textbf{fi}$

{Theorem 4.6(2)}

$$= \textbf{if}\begin{pmatrix} b1 \rightarrow (x := e1), \\ b2 \rightarrow (x := e2); \texttt{fail}, \\ b3 \rightarrow \textbf{if}\begin{pmatrix} c1[e3/x] \rightarrow (x := f1[e3/x]), \\ c2[e3/x] \rightarrow (x := f2[e3/x]); \texttt{fail}, \\ c3[e3/x] \rightarrow (x := f3[e3/x]); \texttt{throw} \end{pmatrix} \textbf{fi} \end{pmatrix} \textbf{fi}$$

{Theorem 4.6(2) and (3)}

$$= \textbf{if}\begin{pmatrix} b1 \wedge (\neg b3 \vee c[e3/x]) \rightarrow (x := e1), \\ b3 \wedge c1[e3/x] \rightarrow (x := f1[e3/x]), \\ b2 \wedge (\neg b3 \vee c[e3/x]) \rightarrow (x := e2); \texttt{fail} \\ b3 \wedge c2[e3/x] \rightarrow (x := f2[e3/x]); \texttt{fail}, \\ b3 \wedge c3[e3/x] \rightarrow (x := f3[e3/x]); \texttt{throw} \end{pmatrix} \textbf{fi}$$

{Theorem 3.6(4)}

$= RHS$

Theorem 4.4

$W\ \texttt{cpens}\ R\ =$

$$\textbf{if}\left(\begin{array}{l}(b1 \wedge (\neg b2 \vee c[e2/x]) \vee b \wedge c1[e2/x]) \rightarrow \\ \quad (x := (e1 \lhd b1 \rhd f1[e2/x]) \sqcap x := (f1[e2/x] \lhd c1[e2/x] \rhd e1)) \\ (b2 \wedge c2[e2/x]) \rightarrow (x := f2[e3/x]); \texttt{fail} \\ (b3 \wedge (\neg b2 \vee c[e3/x]) \vee c3[e2/x] \wedge b) \rightarrow \\ \quad (x := (e3 \lhd b3 \rhd f3[e2/x]) \sqcap x := (f3[e2/x] \lhd c3[e2/x] \rhd e3); \texttt{throw}\end{array}\right)\textbf{fi}$$

Proof: Similar to Theorem 4.3.

Theorem 4.5

$W\ \texttt{else}\ R\ =$

$$\textbf{if}\left(\begin{array}{l}(b1 \wedge c \vee b \wedge c1) \rightarrow (x := (e1 \lhd b1 \rhd f1) \sqcap x := (f1 \lhd c1 \rhd e1)) \\ (b2 \wedge c2) \rightarrow (x := f2); \texttt{fail} \\ (b3 \wedge c \vee c3 \wedge b) \rightarrow (x := (e3 \lhd b3 \rhd f3) \sqcap x := (f3 \lhd c3 \rhd e3)); \texttt{throw}\end{array}\right)\textbf{fi}$$

Proof: LHS {Theorem 4.9(2) and (3)}

$$= \quad \textbf{if}(b1 \rightarrow (x := e1),\ b2 \rightarrow R,\ b3 \rightarrow (x := e3); \texttt{throw})\textbf{fi}$$

{Theorem 4.6(2)}

$$= \quad \textbf{if}\left(\begin{array}{l}b1 \wedge (\neg b2 \vee c) \rightarrow (x := e1), \\ b2 \wedge c1 \rightarrow (x := f1), \\ b2 \wedge c2 \rightarrow (x := f2); \texttt{fail}, \\ b2 \wedge c3 \rightarrow (x := f3); \texttt{throw}, \\ b3 \wedge (\neg b2 \vee c) \rightarrow (x := e3); \texttt{throw}\end{array}\right)\textbf{fi}$$

{Theorem 3.6(4)}

$$= \quad RHS$$

Theorem 4.6

$W \lhd d \rhd R\ =$

$$\textbf{if}\left(\begin{array}{l}(\hat{d} \wedge b1 \vee \widehat{\neg d} \wedge c1) \rightarrow \\ \qquad (x := (e1 \lhd b1 \rhd f1) \sqcap x := (f1 \lhd c1 \rhd e1)) \\ (\hat{d} \wedge b2 \vee \widehat{\neg d} \wedge c2) \rightarrow \\ \qquad (x := (e2 \lhd b2 \rhd f2) \sqcap x := (f2 \lhd c2 \rhd e2)); \texttt{fail} \\ (\hat{d} \wedge b2 \vee \widehat{\neg d} \wedge c2 \vee \neg \mathcal{D}(d)) \rightarrow \\ \qquad (x := ((e3 \lhd b3 \rhd f3) \lhd \mathcal{D}(d) \rhd x) \sqcap \\ \qquad x := ((f3 \lhd c3 \rhd e3) \lhd \mathcal{D}(d) \rhd x)); \texttt{throw}\end{array}\right)\textbf{fi}$$

where $\hat{d}\ =_{df}\ d \lhd \mathcal{D}(d) \rhd false$

Proof: LHS {Theorem 3.3}

$$= \quad \mathbf{if}(\hat{d} \to W,\ \hat{\neg d} \to R,\ \neg\mathcal{D}(d) \to \mathtt{throw})\mathbf{fi}$$

{Theorem 3.6(2) and (3)}

$$= \quad \mathbf{if}\begin{pmatrix} \hat{d} \wedge b1 \wedge c \to (x := e1), & \hat{\neg d} \wedge c1 \wedge b \to (x := f1), \\ \hat{d} \wedge b2 \wedge c \to (x := e2); \mathtt{fail}, & \hat{\neg d} \wedge c2 \wedge b \to (x := f2); \mathtt{fail}, \\ \hat{d} \wedge b3 \wedge c \to (x := e3); \mathtt{throw}, & \hat{\neg d} \wedge c3 \wedge b \to (x := f3); \mathtt{throw}, \\ \neg\mathcal{D}(d) \to \mathtt{throw} & \end{pmatrix}\mathbf{fi}$$

{Theorem 3.6(4)}

$$= \quad RHS$$

Theorem 4.7

$$(x := e)\ \mathtt{par}\, R\ = \mathbf{if}\begin{pmatrix} c1 \to (x,\, y\ :=\ e,\ f1), \\ c2 \to (x,\, y := e,\ f2); \mathtt{fail}, \\ c3 \to (x,\, y :=\ e,\ f3); \mathtt{throw} \end{pmatrix}\mathbf{fi}$$

Proof:

LHS {Theorem 4.11(4)}

$$= \mathbf{if}\begin{pmatrix} c1 \to ((x := e)\, \mathtt{par}\, (y := f1)), \\ c2 \to ((x := e)\, \mathtt{par}\, (y := f2; \mathtt{fail})), \\ c3 \to ((x := e)\, \mathtt{par}\, (y := f3; \mathtt{throw})) \end{pmatrix}\mathbf{fi}$$

{Theorem 4.11(6) and (10)}

$$= RHS$$

Theorem 4.8

$$(x := e; \mathtt{fail})\ \mathtt{par}\ R\ =$$

$$\mathbf{if}\begin{pmatrix} c1 \vee c2 \to \\ \quad (x,\, y\ :=\ (e,\ f1) \lhd c1 \rhd (e,\ f2) \sqcap x,\, y :=\ (e,\ f2) \lhd c2 \rhd (e,\ f1)); \mathtt{fail} \\ c3 \to (x,\, y :=\ e,\ f3); \mathtt{throw} \end{pmatrix}\mathbf{fi}$$

Proof: Similar to Theorem 4.7.

Theorem 4.9

$$(x := e; \mathtt{throw})\ \mathtt{par}\ R\ =$$

$$\mathbf{if}\begin{pmatrix} c1 \vee c2 \vee c3 \to \\ \quad \begin{pmatrix} (x,\, y\ :=\ ((e,\ f1) \lhd c1 \rhd (e,\ f2)) \lhd c1 \vee c2 \rhd (e,\ f3)) \sqcap \\ (x,\, y\ :=\ ((e,\ f2) \lhd c2 \rhd (e,\ f1)) \lhd c1 \vee c2 \rhd (e,\ f3)) \sqcap \\ (x,\, y\ :=\ (e,\ f3) \lhd c3 \rhd ((e,\ f1) \lhd c1 \rhd (e,\ f2))) \sqcap \\ (x,\, y\ :=\ (e,\ f3) \lhd c3 \rhd ((e,\ f2) \lhd c2 \rhd (e,\ f1))) \end{pmatrix}; \mathtt{throw} \end{pmatrix}\mathbf{fi}$$

Proof: Similar to Theorem 4.7

Now we are going to show that all primitive commands can be reduced to a normal form.

Theorem 4.10

$\texttt{skip} \ = \ \textbf{if}(true \rightarrow (v := v))\textbf{fi}$

Proof: $\texttt{skip}$ $\{$Theorem 3.5(4)$\}$

$= \quad v := v$ $\{$Theorem 4.6(10)$\}$

$= \quad \textbf{if}(true \rightarrow v := v)\textbf{fi}$

Theorem 4.11

$\texttt{fail} \ = \ \textbf{if}(true \rightarrow (v := v); \texttt{fail})\textbf{fi}$

Proof: Similar to Theorem 4.10.

Theorem 4.12

$\texttt{throw} \ = \textbf{if}(true \rightarrow (v := v); \texttt{throw})\textbf{fi}$

Proof: Similar to Theorem 4.10.

Theorem 4.13

$\bot \ = \ \textbf{if}()\textbf{fi}$

Proof: From Theorem 4.6(10).

Theorem 4.14

$x := e \ = \ \textbf{if}(\mathcal{D}(e) \rightarrow (x,\, y, .., z \ := \ (e \lhd \mathcal{D}(e) \rhd x), y, .., z),\, \neg\mathcal{D}(e) \rightarrow \texttt{throw})\textbf{fi}$

Proof: From Theorem 4.4.

Finally we reach the conclusion.

Theorem 4.15

All finite program can be reduced to a normal form.

Proof: From Theorem 4.1–4.14.

5 Link with the Original Design Model

This section explores the link between the model of Section 2 with the original design model given in [15].

For any design P and **Req**-healthy design Q we define

$$\mathcal{F}(P) =_{df} \mathcal{H}(P; \texttt{success})$$

$$\mathcal{G}(Q) =_{df} Q[true,\, false/forward,\, eflag]; (forward \wedge \neg eflag)_{\bot}$$

Theorem 5.1

$\mathcal{F}$ and $\mathcal{G}$ form a Galois connection:

(1) $\mathcal{G}(\mathcal{F}(P)) \;=\; P$

(2) $\mathcal{F}(\mathcal{G}(Q)) \sqsubseteq Q$

Proof: $\mathcal{G}(\mathcal{F}(P))$ $\{$Def of $\mathcal{F}$ and $\mathcal{G}\}$

$= \quad P; \mathtt{success}; (true \vdash (v' = v)) \lhd forward \wedge \neg eflag \rhd \bot)$

$\{$Def of $\mathtt{success}\}$

$= \quad P; (true \vdash (v' = v))$ $\{$unit law of ; $\}$

$= \quad P$

$\mathcal{F}(\mathcal{G}(Q))$ $\{$Def of $\mathcal{F}$ and $\mathcal{G}\}$

$= \; \mathcal{H}(Q[true, \neg false/forward, eflag];$

$(true \vdash (v' = v) \lhd forward \wedge \neg eflag \rhd \bot); \mathtt{success})$

$\{$Def of $\mathcal{H}$, $(P \lhd b \rhd Q); R \;=\; (P; R) \lhd b \rhd (Q; R)\}$

$= \; Q; (\mathtt{success} \lhd forward \wedge \neg eflag \rhd \bot)$ $\{$Def of $\mathtt{sucess}\}$

$= \; Q; ((true \vdash (v' = v \wedge forwared' = forward \wedge eflag' = eflag))$

$\lhd forward \wedge \neg eflag \rhd \bot)$ $\{\bot \sqsubseteq D\}$

$\sqsubseteq \; Q; (true \vdash (v' = v \wedge forwared' = forward \wedge eflag' = eflag))$

$\{$unit law of ; $\}$

$= \; Q$

$\mathcal{F}$ is a homomorphism.

Theorem 5.2

(1) $\mathcal{F}(true \vdash (v' = v)) \;=\; \mathtt{skip}$

(2) $\mathcal{F}(true \vdash (x' = e \wedge y' = y \wedge z' = z)) \;=\; (x := e)$

provided that e is well-defined.

(3) $\mathcal{F}(true) \;=\; \bot$

(4) $\mathcal{F}(P1 \sqcap P2) \;=\; \mathcal{F}(P1) \sqcap \mathcal{F}(P2)$

(5) $\mathcal{F}(P1 \lhd b \rhd P2) \;=\; \mathcal{F}(P1) \lhd b \rhd \mathcal{F}(P2)$

provided that b is well-defined.

(6) $\mathcal{F}(P1; P2) \;=\; \mathcal{F}(P1); \mathcal{F}(P2)$

(7) $\mathcal{F}(b * P) \;=\; b *_{\mathcal{H}} \mathcal{F}(P)$

Proof:

(6) $\mathcal{F}(P1; P2)$ $\{$Def of $\mathcal{F}\}$

$= \mathcal{H}(P1; P2; \texttt{success}\}$ $\{\texttt{success}; P2; \texttt{success} = P2; \texttt{success}\}$

$= \mathcal{H}((P1; \texttt{success}; P2; \texttt{success}))$ $\{(forward \wedge \neg eflag)^\top; \texttt{success}; Q = (forward \wedge \neg eflag)^\top; \texttt{success}; \mathcal{H}(Q)\}$

$= \mathcal{H}((P1; \texttt{success}); \mathcal{H}(P2; \texttt{success}))$ $\{$Theorem 2.4$\}$

$= \mathcal{H}(P1; \texttt{success}); \mathcal{H}(P2; \texttt{success})$ $\{$Def of $\mathcal{F}\}$

$= \mathcal{F}(P1)\,;\,\mathcal{F}(P2)$

(7) LHS $\{$fixed point theorem$\}$

$= \mathcal{F}((P; b * P) \lhd b \rhd (true \vdash (v' = v)))$ $\{$Conclusion (1), (5), (6)$\}$

$= (\mathcal{F}(P); LHS) \lhd b \rhd \texttt{skip}$

which implies that $LHS \sqsupseteq RHS$

$\mathcal{G}(RHS)$ $\{$fixed point theorem$\}$

$= \mathcal{G}((\mathcal{F}(P); RHS) \lhd b \rhd \texttt{skip})$ $\{\mathcal{G}$ distributes over $\lhd b \rhd\}$

$= \mathcal{G}(\mathcal{F}(P); RHS) \lhd b \rhd \mathcal{G}(\texttt{skip})$ $\{$Def of $\mathcal{G}\}$

$= (\mathcal{F}(P)[true,\ false/forward,\ eflag]; RHS;$
$(foward \wedge \neg eflag)_\perp) \lhd b \rhd (true \vdash (v' = v))$ $\{$Def of $\mathcal{F}\}$

$= (P; \mathbf{success}; RHS;$
$(forward \wedge \neg eflag)_\perp) \lhd b \rhd (true \vdash (v' = v))$ $\{$Def of **success**$\}$

$= (P; RHS[true,\ false/forward,\ eflag];$
$(forward \wedge \neg eflag)_\perp) \lhd b \rhd (true \vdash (v' = v))$ $\{$Def of $\mathcal{G}\}$

$= (P; \mathcal{G}(RHS)) \lhd b \rhd (true \vdash (v' = v))$

which implies

$\mathcal{G}(RHS) \sqsupseteq (b * P)$ $\{\mathcal{F}$ is monotonic$\}$

$\Rightarrow \mathcal{F}(\mathcal{G}(RHS)) \sqsupseteq LHS$ $\{$Theorem 5.1(2)$\}$

$\Rightarrow RHS \sqsupseteq LHS$

6 Conclusion

This paper presents a design model for compensable programs. We add new logical variables $eflag$ and $forward$ to the standard design model to deal with the features of exception and failures. As a result, we put forward new healthiness conditions $\mathbf{Req_1}$ and $\mathbf{Req_2}$ to characterise those designs which can be used to specify the dynamic behaviour of compensable programs.

This paper treats an assignment $x := e$ as a conditional (Theorem 4.1). After it is shown that `throw` is a new left zero of sequential composition, we are allowed to use the algebraic laws established for the conventional imperative language in [15] to convert finite programs to normal form. This shows that the model of Section 2 is really a conservative extension of the original design model in [15] in the sense that it preserves the algebraic laws of the Guarded Command Language.

Acknowledgement

The ideas put forward in this paper have been inspired from the discussion with Tony Hoare, and the earlier work of my colleagues.

References

1. Abadi, M., Gordon, A.D.: A calculus for cryptographic protocols: The spi calculus. Information and Computation 148(1), 1–70 (1999)
2. Alonso, G., Kuno, H., Casati, F., et al.: Web Services: Concepts, Architectures and Applications. Springer, Heidelberg (2003)
3. Bhargavan, K., et al.: A Semantics for Web Service Authentication. Theoretical Computer Science 340(1), 102–153 (2005)
4. Bruni, R., Montanari, H.C., Montannari, U.: Theoretical foundation for compensation in flow composition languages. In: Proc. POPL 2005, 32nd ACM SIGPLAN-SIGACT symposium on principles of programming languages, pp. 209–220. ACM, New York (2004)
5. Bruni, R., et al.: From Theory to Practice in Transactional Composition of Web Services. In: Bravetti, M., Kloul, L., Zavattaro, G. (eds.) EPEW/WS-EM 2005. LNCS, vol. 3670, pp. 272–286. Springer, Heidelberg (2005)
6. Bulter, M.J., Ferreria, C.: A process compensation language. In: Grieskamp, W., Santen, T., Stoddart, B. (eds.) IFM 2000. LNCS, vol. 1945, pp. 61–76. Springer, Heidelberg (2000)
7. Bulter, M.J., Ferreria, C.: An Operational Semantics for StAC: a Lanuage for Modelling Long-Running Business Transactions. LNCS, vol. 2949, pp. 87–104. Springer, Heidelberg (2004)
8. Butler, M.J., Hoare, C.A.R., Ferreria, C.: A Trace Semantics for Long-Running Transactions. In: Abdallah, A.E., Jones, C.B., Sanders, J.W. (eds.) Communicating Sequential Processes. LNCS, vol. 3525, pp. 133–150. Springer, Heidelberg (2005)
9. Curbera, F., Goland, Y., Klein, J., et al.: Business Process Execution Language for Web Service (2003), `http://www.siebei.com/bpel`
10. Dijkstra, E.W.: A Discipline of Programming. Prentice Hall, Englewood Cliffs (1976)
11. Gordon, A.D., et al.: Validating a Web Service Security Abstraction by Typing. Formal Aspects of Computing 17(3), 277–318 (2005)
12. Jifeng, H., Huibiao, Z., Geguang, P.: A model for BPEL-like languages. Frontiers of Computer Science in China 1(1), 9–20 (2007)
13. Jifeng, H.: Compensable Programs. In: Jones, C.B., Liu, Z., Woodcock, J. (eds.) Formal Methods and Hybrid Real-Time Systems. LNCS, vol. 4700, pp. 349–364. Springer, Heidelberg (2007)

14. Hoare, C.A.R.: Communicating Sequential Language. Prentice Hall, Englewood Cliffs (1985)
15. Hoare, C.A.R., Jifeng, H.: Unifying theories of programming. Prentice Hall, Englewood Cliffs (1998)
16. Leymann, F.: Web Service Flow Language (WSFL1.0). IBM (2001)
17. Laneve, C., et al.: Web-pi at work. In: De Nicola, R., Sangiorgi, D. (eds.) TGC 2005. LNCS, vol. 3705, pp. 182–194. Springer, Heidelberg (2005)
18. Jing, L., Jifeng, H., Geguang, P.: Towards the Semantics for Web Services Choreography Description Language. In: Liu, Z., He, J. (eds.) ICFEM 2006. LNCS, vol. 4260, pp. 246–263. Springer, Heidelberg (2006)
19. Lucchi, R., Mazzara, M.: A Pi-calculus based semantics for WS-BPEL. Journal of Logic and Algebraic Programming (in press)
20. Milner, R.: Communication and Mobile System: the π-calculus. Cambridge University Press, Cambridge (1999)
21. Morris, J.M.: Non-deterministic expressions and predicate transformers. Information Processing Letters 61, 241–246 (1997)
22. Geguang, P., et al.: Theoretical Foundation of Scope-based Compensation Flow Language for Web Service. LNCS, vol. 4307, pp. 251–266. Springer, Heidelberg (2006)
23. Qiu, Z.Y., et al.: Semantics of BPEL4WS-Like Fault and Compensation Handling. In: Fitzgerald, J.S., Hayes, I.J., Tarlecki, A. (eds.) FM 2005. LNCS, vol. 3582, pp. 350–365. Springer, Heidelberg (2005)
24. Tarski, A.: A lattice-theoretical fixpoint theorem and its applications. Pacific Journal of Mathematics 5, 285–309 (1955)
25. Thatte, S.: XLANG: Web Service for Business Process Design. Microsoft (2001)