第五章　探討 Refactoring 轉移之系統等價關係

Refactoring 方法分析子系統之行為，歸納行為以產生新的子系統，使得這些新的子系統對內隱藏了複雜的介面行為，對外只有簡單的介面，因此有效地縮減了系統可到達的狀態，而可防止組態爆炸。但轉移子系統行為的過程中，必需保持行為的等價性質，確保所架構之新系統與原系統是等價的。

驗證系統的行為時，通常都是拿已建構完整的系統來分析。我們首先舉一個已建構完整的系統作為例子，它的元件間有複雜的介面行為，因此不能適用於局部性分析。然後說明 Refactoring 如何將它轉移成元件間只有簡單的介面行為。

接著再以這個例子連同第三章及第四章的方法，並依據 Refactoring 轉移系統的規則，設計一個程序代數模型，探討經 Refactoring 轉移後之新系統與原系統的等價關係，使得這個等價關係是可取代的等價關係。

第一節　Refactoring 轉移的例子

Refactoring 提出一系列的轉移規則，用來轉移已架構完好的系統，這些系統因內部元件間複雜的介面行為，使得無法利用局部性分析來分析。Refactoring 不像局部性分析僅單純的將系統分解成小元件、取代成簡單的介面行為、再循序漸進的組合起來。Refactoring 在將系統分解成小元件時，歸納元件間複雜的介面產生新的小元件，再將仍有複雜的介面的小元件結合起來成新的小元件，使其對外僅有簡單的介面行為並保持行為的等價。Refactoring 的優點不但擴展局部性分析的範圍，而且對於某些參數化系統很容易延展(scalable)

接下來舉一實例來說明 Refactoring 的用法。考慮圖 5-1 的系統，它由四個程序 R、S1、S2、S3 所組成，其中 S1、S2、S3 是一樣的，它們有相同的行為，R 的大小因 S 的個數而變動，目前圖 5-1 僅畫出三個 S。圖 5-1 是

以 CCS 的語意畫的[1]，因此只允許兩個程序間的傳輸，可傳輸的行為分別以 a 及 $\bar{a}$ 表示，在此圖以-a 表示 $\bar{a}$。這個例子的行為是這樣的，R 不斷地重複在 in 通道(channel)讀取 id，一當收到某一個 S 的 id，即與該 S 進行一連串的同步傳輸，完成後又回到初始狀態。Ack 是當某一個 S，它的 id 被 R 讀取時向外界發出的確認訊息。

這個例子程序間的介面比較複雜，如果一下子就將所有程序以並行組合方式架構起來成為(R || S1 || S2 || S3)來分析時，會因 S 個數增加而產生組態爆炸(state explosion)的問題，以致分析失敗。若以其它各種不同的分割再組合方式，如((R || S1) || S2 || S3)或((R || S1 || S2) || S3)等，結果都一樣會產生組態爆炸的問題。像這樣元件間介面複雜且可參數化的例子，就不是局部性分析(compositional analysis)所能分析的。

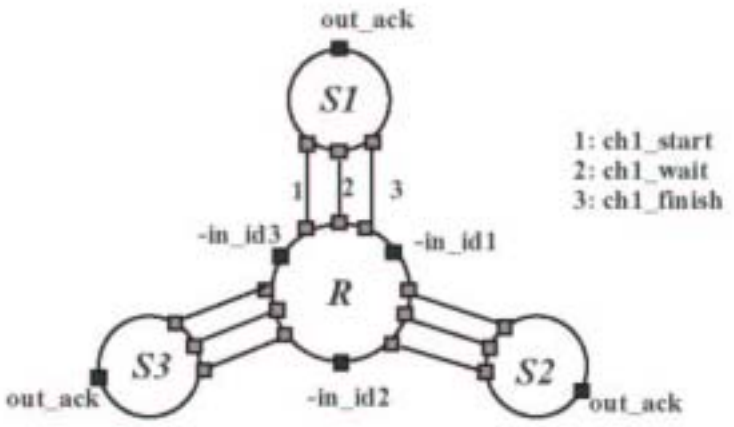

圖 5-1　複雜介面的例子

若應用 refactoring 將圖 5-1 轉移到圖 5-2 時，圖 5-1 中 R 與 S_i 間複雜的行為已分別被重新組合到圖 5-2 之 RS_i 與 S_i 中。在圖 5-2 中每一個 RS_i 與 S_i 的組合被視為一個元件，因此隱藏了內部複雜的行為，使得整個元件所有的行為就是原先已有的 $-in\_id_i$ 及 out_ack。同時為了保持行為的等價，增加同步的”-lock”及”-lease” 行為。

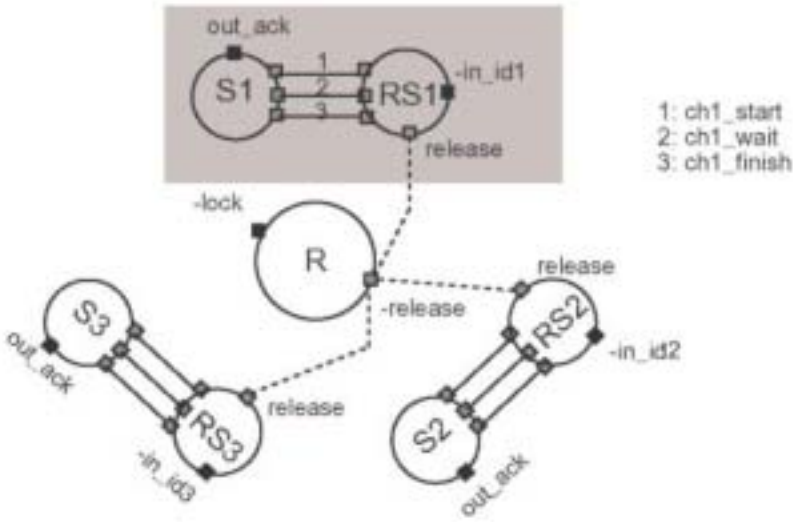

圖 5-2　經 Refactoring 轉移後的系統

圖 5-2 的系統有一些很好的性質是原系統沒有的，例如陰影部分的元件具有緊密的耦合(tightly coupled)而對外界僅有簡單的介面，因此隱藏子系統間傳輸，有效的縮減系統的狀況空間。

第二節　以程序代數表示系統

圖 5-1 對外界的行為有 in_id_i 及 out_ack，為了說明這些行為與外界互動的情形，以及使用 Refactoring 轉移時會新增與外界互動的行為，我們還必需加入一個環境程序 P。在不影響我們討論的前提下，盡可能的簡化我們使用的例子，因此我們忽略 out_ack，並使得程序 P 越簡單越好。

我們把省略 out_ack 之後的圖 5-1 稱為原系統，原系統有四個程序分別為 R、S1、S2 及 S3，S1-S3 以 S_i 來表示，我們以 S1、S2 討論即可，因此 i，$i \in \{1,2\}$，再來-在 R 或 S_i 對我們的討論不重要，都把它歸到 R 這一邊，因此 R、S_i 的代數表示如下：

$R = -in\_id_i \cdot -ch_i\_start \cdot -ch_i\_wait \cdot -ch_i\_finish \cdot R$

$S_i = ch_i\_start \cdot ch_i\_wait \cdot ch_i\_finish \cdot S_i$

$P = in\_id_1 \cdot P + in\_id_2 \cdot P$

同時 R、S_i 再進一的以 a、b、c 代表它的 action，因此簡化為：

$R = \bar{i}_i \cdot \bar{a}_i \cdot \bar{b}_i \cdot \bar{c}_i \cdot R$

$S_i = a_i \cdot b_i \cdot c_i \cdot S_i$

$P = i_1 \cdot P + i_2 \cdot P$

原系統為$(R \| S_1 \| S_2)$，若以$[a_i]$表示 a_i 與 $\bar{a}_i$ 發生傳輸行為，就是定義 $a_i \mid \bar{a}_i = [a_i]$，令 SR=$(R \| S_1 \| S_2)$，則 SR 有下列的結果：

$SR = [i_1] \cdot [a_1] \cdot [b_1] \cdot [c_1] \cdot SR + [i_2] \cdot [a_2] \cdot [b_2] \cdot [c_2] \cdot SR$---------- (1)

經 Refactoring 轉移後，原系統被轉移成圖 5-2(捨去 out_ack)，圖 5-2 的系統我們稱它為新系統，新系統捨去 S3 後有五個程序分別為 R、RS1、RS2、S1 及 S2，S1-S2 與 RS1-RS2 分別以 S_i、RS_i 來表示，$i \in \{1,2\}$，因此 R、RS_i 及 S_i 的代數表示如下：

$R = -lock \cdot -release \cdot R$

$RS_i = -in\_id_i \cdot -ch_i\_start \cdot -ch_i\_wait \cdot -ch_i\_finish \cdot release \cdot RS_i$

$S_i = ch_i\_start \cdot ch_i\_wait \cdot ch_i\_finish \cdot S_i$

$P = lock \cdot in\_id_1 \cdot P + lock \cdot in\_id_2 \cdot P$

同樣的 R、RS_i 及 S_i 進一簡化為：

$R = \bar{l} \cdot \bar{r} \cdot R$

$RS_i = \bar{i}_i \cdot \bar{a}_i \cdot \bar{b}_i \cdot \bar{c}_i \cdot r \cdot RS_i$

$S_i = a_i \cdot b_i \cdot c_i \cdot S_i$

$P = l \cdot i_1 \cdot P + l \cdot i_2 \cdot P$

新系統為$(R \| RS_1 \| RS_2 \| S_1 \| S_2)$，同樣的以$[a_i]$表示 a_i 與 $\bar{a}_i$ 發生傳輸行為，令 DR=$(R \| RS_1 \| RS_2 \| S_1 \| S_2)$，則 DR 有下列的結果：

$DR = [l] \cdot [i_1] \cdot [a_1] \cdot [b_1] \cdot [c_1] \cdot [r] \cdot DR + [l] \cdot [i_2] \cdot [a_2] \cdot [b_2] \cdot [c_2] \cdot [r] \cdot DR$----------(2)

比較(1)、(2)兩個式子，第(2)個式子在每一個子項前後都分別有[l]、[r]護衛著，除此之外兩式之間沒有甚麼不同。若以一個外界觀察者的角度觀察兩個系統執行情形，原系統在兩個選項中挑一項來執行，新系統亦是如此。若在系統內觀察，新系統在開始及回到初始狀態會延遲一下。

第三節　系統的公設

第三章第七節討論程序間的傳輸(communication)行為時，定義了一個函數 f 表示兩個原子 action 是否發生傳輸行為，現在我們將它擴充，擴充之後的 f 稱為γ函數，γ與 f 為一的差異在γ的值域多了一個ε。我們將所有原子 action 記成 A，γ的定義為：γ：A$\rightarrow$ A$\cup\{\varepsilon\}$，因此γ(a,b)有定義時，γ(a,b)=a|b=c 才成立，否則γ(a,b)= δ，在我們的系統所謂γ(a,b)有定義一定是在 a=$\overline{b}$ 或 b=$\overline{a}$。

A1	$x + y = y + x$	A6	$x + \delta = x$
A2	$(x + y) + z = x + (y + z)$	A7	$\delta x = \delta$
A3	$x + x = x$	A8	$\varepsilon x = x$
A4	$(x + y)z = xz + yz$	A9	$x\varepsilon = x$
A5	$(xy)z = x(yz)$	BE	$a(\tau(x + y) + x) = a(x + y)$
CTM1	$x \between y = x \lfloor\!\lfloor y + y \lfloor\!\lfloor x + x \mid y + \surd(x)\ \surd(y)$		
TM2	$\varepsilon \lfloor\!\lfloor x = \delta$	TM5	$\varepsilon \mid x = \delta$
TM3	$ax \lfloor\!\lfloor y = a(x \between y)$	TM6	$x \mid \varepsilon = \delta$
TM4	$(x + y) \lfloor\!\lfloor z = x \lfloor\!\lfloor z + y \lfloor\!\lfloor z$	CF0	$\gamma(l, \overline{l}) = \gamma(r, \overline{r}) = \varepsilon$
CM7	$ax \mid by = (a \mid b) \cdot (x \between y)$	CF1	$a \mid b = \gamma(a,b)$　若 f(a,b)已定義
CM8	$(x + y) \mid z = x \mid z + y \mid z$	CF2	$a \mid b = \delta$　若 f(a,b)未定義
CM9	$x \mid (y + z) = x \mid y + x \mid z$	TI0	$\tau_I(\varepsilon) = \varepsilon$
TI1	$\tau_I(a) = a$　if $a \notin I$	TI3	$\tau_I(x + y) = \tau_I(x) + \tau_I(y)$
TI2	$\tau_I(a) = \tau$　if $a \in I$	TI4	$\tau_I(x \cdot y) = \tau_I(x) \cdot \tau_I(y)$
TE1	$\surd(\varepsilon) = \varepsilon$	TE3	$\surd(x + y) = \surd(x) + \surd(y)$
TE2	$\surd(a) = \delta$	TE4	$\surd(xy) = \surd(x) \cdot \surd(y)$

表 5-1　Refactoring Equivalence 的公設

有了 γ 的定義，我們就用它來定義第一節提到的 l 與 r，定義 $\gamma(l,\bar{l})=\varepsilon$、$\gamma(r,\bar{r})=\varepsilon$。另外 γ 的定義用到了 ε 及 δ，因此公設就必需包含有關規範 ε 及 δ 行為的公設。我們將用到的公設列於表 5-1，表 5-1 來自表 3-20，只比表 3-20 多了 CF0，至於表 5-1 的其他公設均於第三章討論過了。

從以上公設的討論，可以知道我們所需的代數規格與 ACP_ε^τ 是一樣的，只是增加了一個 CF0 的公設。

第四節　系統的模型

令 P 為所有程序的集合，其程序由下列方式建構而成：

i. 每一個原子 action 都是程序表示式(包含 ε、δ)。

ii. 如果 E 為遞迴規格、X 為 E 之變數，則<X|E>是程序表示式。

iii. 如果 p、q 是程序表示式，則 p+q、p・q、$p/\!\!/q$、$p\between q$、$p|q$、$\surd(p)$、$\tau_I(p)$ 都是程序表示式。

我們採用 $\leftrightarrow_{rb}$ 作為 P 上的等價關係，因為 $P_{\delta\varepsilon}/\leftrightarrow_{rb}$ 為 ACP_ε^τ 之模型*[2]*，而我們所建構的 P 和 $P_{\delta\varepsilon}$ 是一樣的，除了 CF0 外，表 5-1 的公設就是 ACP_ε^τ 的公設，因此 $P/\leftrightarrow_{rb}$ 滿足表 5-1 所有公設，也就是我們要的模型。至於 action relation 則與 ACP_ε^τ 的 action relation 完全一樣。

第五節　討論

因為第一節新系統的 R、RS_i 以及 S_i 都是 $P/\leftrightarrow_{rb}$ 中的程序，利用公設 CF0、CF1 及 CTM1，當$(R\between RS_1\between RS_2\between S_1\between S_2)$展開時，$l$ 與 $\bar{l}$、r 與 $\bar{r}$ 分別合併成 ε，再經由公設 A8、A9 消除 ε，因此新系統成為：

$DR=(R\between RS_1\between RS_2\between S_1\between S_2)$

$=\varepsilon\cdot[i_1]\cdot[a_1]\cdot[b_1]\cdot[c_1]\cdot\varepsilon\cdot DR+\varepsilon\cdot[i_2]\cdot[a_2]\cdot[b_2]\cdot[c_2]\cdot\varepsilon\cdot$

DR

$=[i_1]\cdot[a_1]\cdot[b_1]\cdot[c_1]\cdot DR+[i_2]\cdot[a_2]\cdot[b_2]\cdot[c_2]\cdot DR$

所以在表 5-1 的公設下，SR=DR。

實際上我們可以縮小範圍，在 *P* 中將所有**可定義程序集合**起來，成為 *P* 的一個子集合，這些可定義程序對公設中的算子都具有封閉性[2]，我們稱這個子集合為 *R*。*P* 中程序可以滿足表 *5-1* 的公設，*R* 中程序當然也可以，因此 $R/\underline{\leftrightarrow}_{rb}$ 就成為一個模型，我們稱 $R/\underline{\leftrightarrow}_{rb}$ 模型為 regular process model。