第三章　使用 ArCats 進行局部性模型驗證

- 檢驗安全性質

到目前為止利用局部性分析所開發的工具相當少，比較實用的為之前所提到的 Fc2Tools。但利用局部性分析的基礎在於分析的軟體本身必須具有良好的階層式系統架構，Fc2tools 對於此問題並沒有提供足夠的解決方案。在 ArCats 中有提供 model architecture Refactoring 的功能，能在不改變原始系統行為的情況中，轉變系統的架構，進而可能提供好的階層為局部分析所使用，這使欲檢驗的系統不再受限於系統本身階層式架構的不良。在目前的研究發現，利用 CCS（Calculus of Communicating System）所建立的系統模型，在經過 Model architecture Refactoring 轉換後所得出的系統架構，將優於用 CSP（Communicating Sequential process）方式所建立的系統模型；並且較具有對稱性，得出來的結果特別有利於局部性分析。也就是說用 CCS 進行的架構重構可以產生較好的局部分析結果。故我們將利用 ArCats 來的基礎，利用 CCS 的系統模型來開發利用局部性分析檢驗軟體安全性質的工具。

3.1節利用例子Gas-Station指出局部性分析目前所遭遇的困難與極限。3.2來節介紹在ArCats上檢驗安全性質所使用的理論，使其突破原本局部性分析的極限。3.3介紹實做在ArCats的安全檢驗引擎。3.4介紹ArCats的新型態狀態合成方式。3.5將利用實做於ArCats的安全檢驗引擎來測試Gas-Station例子。

3.1 使用局部性模型驗證安全性質

一般的局部性分析原理在於在每個子系統中，盡可能的隱藏內部行為（internal action）來達到最小化的效果，藉以減緩組態爆炸的發生。但在檢驗安全性質時，某些欲分析的系統性質，因為系統的同步行為，必須保留到全域階層才能作檢驗，此時將大幅違反局部性分析的原理。Figure 3-1為常見的 Gas-Station例子（子系統模型將在3.5節詳記介紹），將欲檢驗的安全性質R_Change或R_Service加入到系統的適當階層，接者利用bottom-up的方式從level-1 到level-4 開始一層層的將子系統組合。由於R_Change所檢驗的某些安全性質和其他子系

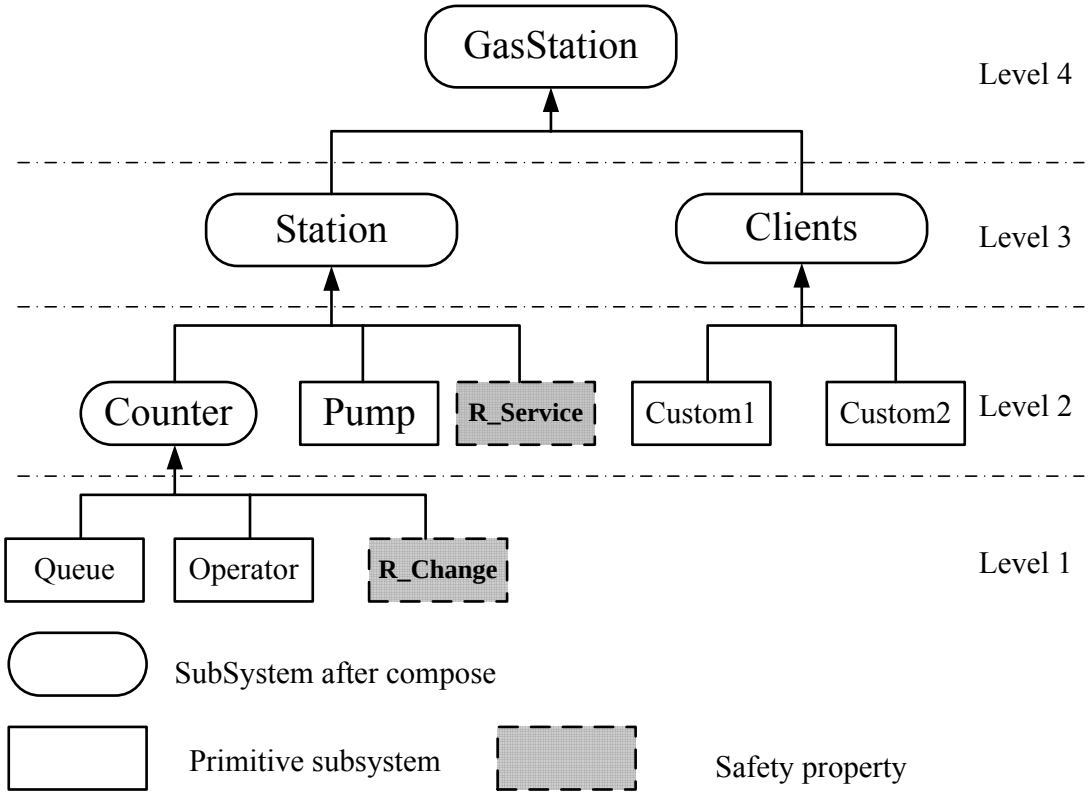


Figure 3-1 利用 button-up 的方式平行合成 Gas-Station 系統

統（Clients）有同步溝通的行為，必須到 level-4 時才能同步執行，這表示我們無法在 level-1 時就將子系統的內部行為最小化，故違反了局部性分析的原則。如果欲檢驗的安全性質較龐大時，原本可以減化的內部行為就必須保留到其他階

層，顯然的這將加速組態爆炸的發生。

對於這問題，Cheung 提出了一個技巧，在 multi-way rendezvous 的情況下，可以解決這種事情的發生。下一小節我們將利用此技巧套用於 two-way rendezvous 的系統中，並將其實做於 ArCats 來解決此問題。

3.2 使用加強版局部性模型驗證安全性質

這裡我們要先介紹本篇論文如何描述欲檢驗的系統特性。一般檢驗的系統性質稱為property automata，如Figure 3-2(a)，其相對應的系統應該遵照其描述的系統特性在運行正常情況下，為了顯示property automata的違反狀態，我們使用另一種有限狀態機稱為image automata如Figure 3-2(b)，他是由property automata所衍生出來的狀態機。當相對應的子系統執行記錄無法符合property automata時出現，image automata將會進入π狀態。

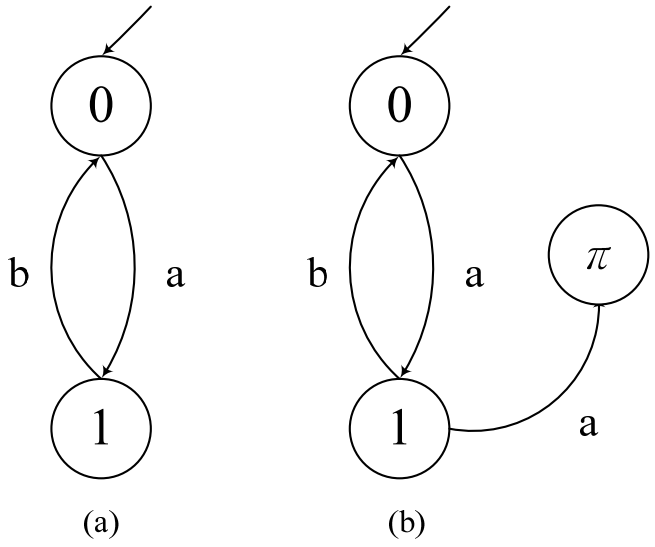


Figure 3-2 表示安全性質的有限狀態機
(a) Property Automata T (b) image Automata T'

我們利用tr(T)代表property automata中的trace，而tr(T `) 表示會含有π的

image automata其不合法的trace，正常來說如果系統行為都屬於合法狀態，檢驗的紀錄應該都會包含於tr(T)中，否則會出現如tr(T `)有進入違反狀態的不合法紀錄。因此我們可以將image automata加入系統中進行平行合成，看其是否會出現違反狀態。舉例來說Figure 3-2當T`加入系統檢驗安全性質時，只要系統的行為符合a,b,a,b,a…..而不是出現連續兩個a（ex. a,b,a,a,…）行動時，我們可以稱系統符合此安全性質；否則系統違反property automata T時，將進入所謂的死結狀態 π 。

我們用更精確的方程式表示如下

T =< S , A , △ , q >

T'=< S∪{ π } , A , △' , q >

其中 T'可由 T 推導出，即

△'= △∪{(s, a, π) | (s,a)∈ $S \times A \wedge \nexists s' \in S : (s, a, s') \in \Delta$ }

舉例來說，我們也可以將上列方程式描述如下：

（1）T 和 T`有相同的非陷阱記錄（nontrapping traces）

並且

（2）系統中的任意行程 P，當 P||T`不包括陷阱記錄時若且唯若當

tr(P ↑ α T)⊆tr(T)

利用（2）的原理，我們可以將檢驗安全性質的工作對應成偵測系統中是否產生死結。這裡要注意的是情況是，當子系統中發現死結的產生時，並不代表全域系統中也會產生死結。畢竟我們是利用列舉的方式將子系統中所有可能的情況條列出來，然在全域系統中，卻不一定會發生進入死結的情況。接下來我們將更詳細的介紹所開發的安全檢驗引擎，它可以在不破壞系統原有特性，利用加入死結的方式來檢驗安全性質。

3.3 安全性質檢驗引擎

欲檢驗的安全性質將透過安全檢驗引擎（Figure 3-3）來執行，輸入端分別為子系統模型和image automata(安全性質模型)，皆以CCS檔案格式作為輸入端格式，安全檢驗引擎以image automata作為判斷條件，當發現違反系統性值時將會在系統中加入死結狀態。如果檢驗都沒錯誤，表示系統符合此安全狀態。安全檢驗引擎採用BFS (breadth first search)演算法配合Queue資料結構來完成運作，其中Queue用來記錄子系統中所有的行程和檢驗的image automata目前的狀態，剛開始將會存入每個行程的初始狀態（包含image automata）。當平行合成在子系統中可以展開新的狀態時，將會同時檢查目前所處的image automata狀態是否也

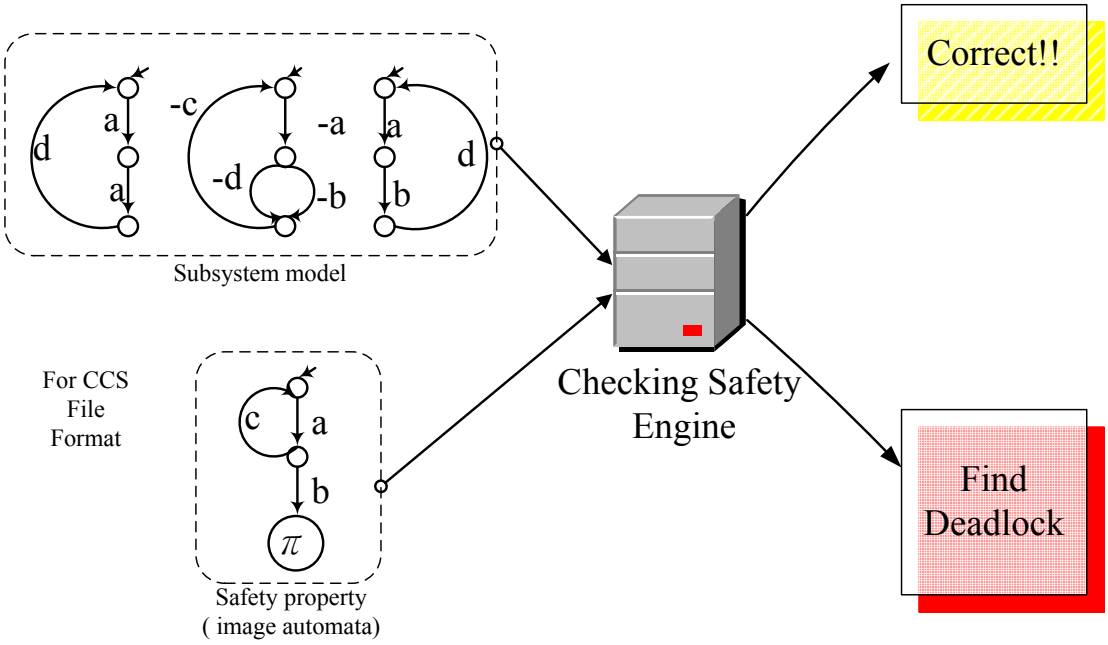


Figure 3-3 安全檢驗引擎架構圖。

可進行至下一個狀態。安全性質的狀態轉移方式將和原本的同步移轉不太一樣，此時 image automata 的狀態轉移不再侷限於必須在同步行動完成後，反而不論是否完成同步動作，只要是相同的行動(不在侷限於 two-way rendezvous 必須兩兩配對來同步轉移到新的狀態)，都可以轉移到下一個可達狀態。由於利用 BFS 演

算法，我們可以完整的列舉出所有系統的狀態。在列舉的過程中，只要 image automata 進入了違反狀態（即所謂的π），我們會將子系統目前所轉移到的新狀態標記為死結狀態，由於死結狀態代表系統進入一個違反的狀態，此死結狀態將不將此加入 Queue 中做擴展。

在Figure 3-4(a)有三個行程P1、P2 和P3，我們將用其解說安全檢驗引內部運作情況。首先Figure 3-5(a)為整個系統合成階層圖，利用buttom-up的方式，子系統是將先結合P1 和P2，接著再去結合(P1||P2)和P3 來完成整個系統；另有一個S1為image automata(安全性質模型Figure 3-4(b))，我們會將其加入適當階

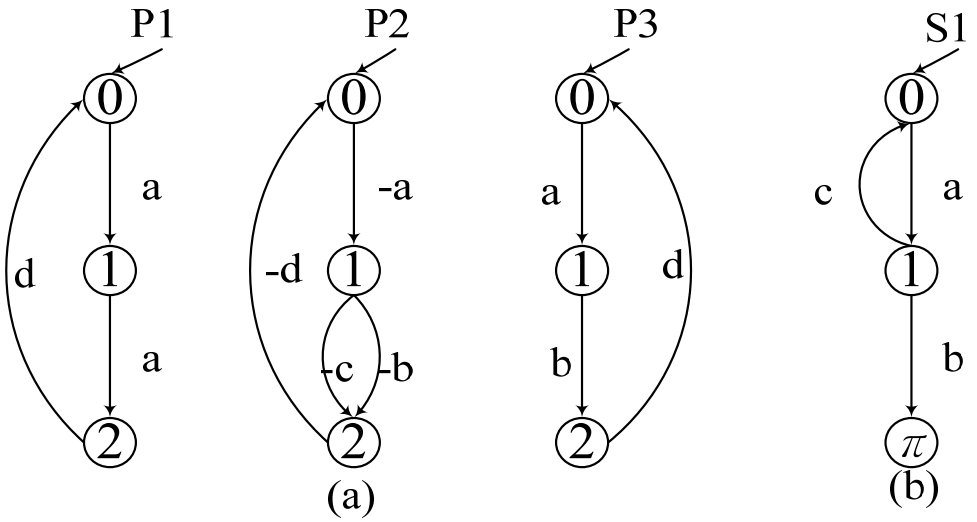
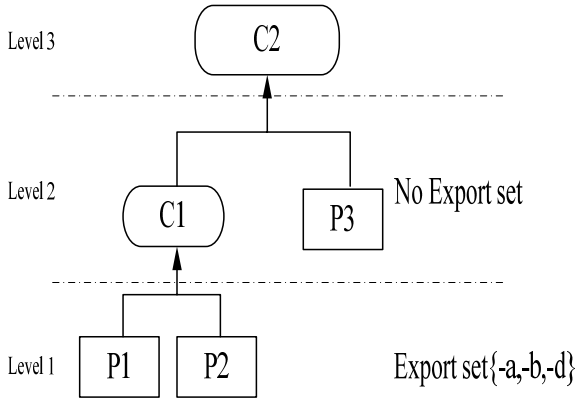
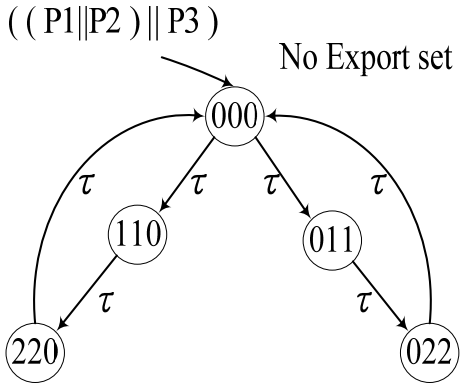


Figure 3-4 子系統 P1,P2,P3 和安全性質 S1
(a) P1,P2 和 P3 的 LTS .
(b) 安全性質表示成image automata S1

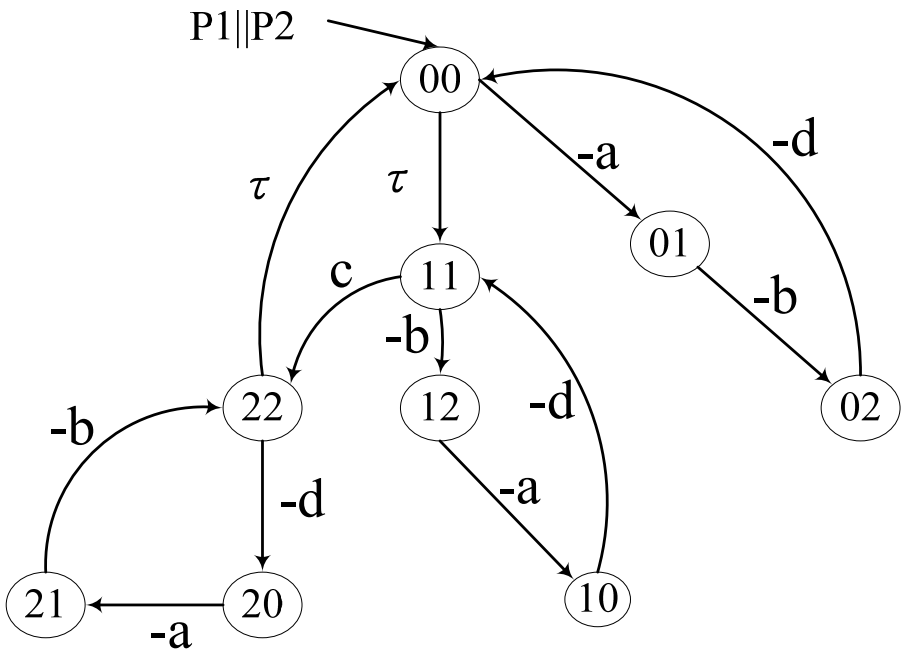
層，並利用局部性分析來檢驗是否違反安全性質S1。我們假設當系統發生某項記錄後如Figure 3-4 (b)發生連續的行動a,b 或 a,c,a,b 或a,c,…..,a,b時，系統將違反安全性質，即系統會進入死結狀態。



(a)



(c)



(b)

Figure 3-5　(P1||P2)||P3 平行合成詳細圖表

(a)　P1、P2 和 P3 的階層式合成架構圖

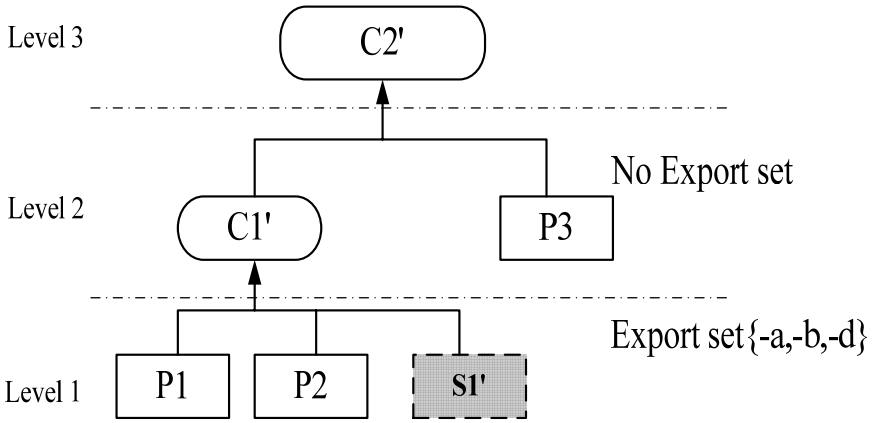
(b) P1 和 P2平行合成後的LTS

(c) 全域的LTS C2.

在執行檢驗安全性質前，我們將利用Figure 3-5來回顧如何執行階層式結合的步驟。為了簡化表示方式我們定義簡單的式子如下：

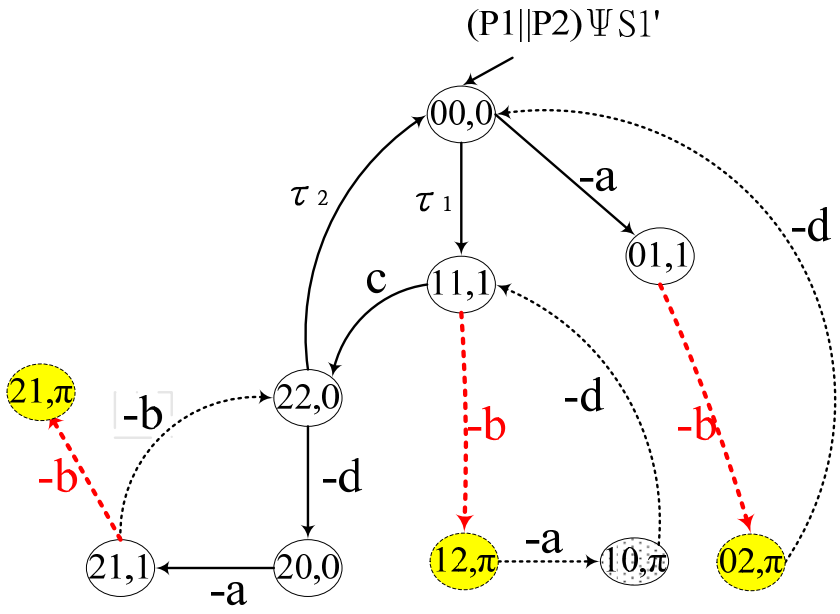
$$_{\text{process name}}(\text{from state}, \text{action}, \text{to state})$$

如果式子為 $_{P1}(0,a,1)$ 代表在P1 中，狀態 0 經由行動轉移到狀態 1；同樣的我們也可以套用在平行合成中，例如：$_{P1\|P2}(0\|0,a\|-a,1\|1)$ 表示P1 和P2 目前都在狀態 0，透過同步溝通a,-a可以轉移到自己的各自狀態 1。利用這樣的式子配合 export set，我們知道哪些action是需要展開讓下個階層可以做同步的溝通。首先 P1 和 P2 的起始狀態都為 0， 可以展開兩個不同的狀態表示為 $_{P1\|P2}(0\|0,a\|-a,1\|1)$ 和 $_{P1\|P2}(0\|0,\ \|-a,0\|1)$，接著我們看後面一條式子，又可以展開一個新的狀態表示為 $_{P1\|P2}(0\|1,\ \|-b,0\|2)$，然後再展開一個狀態表示為 $_{P1\|P2}(0\|0,\ \|-d,0\|0)$，此時回到了最原始的狀態，因此這條式子將不需要再展開。利用相同的方式開頭第一條式子也可以擴展下去，最後完成level-1 的合成結果如Figure 3-5(b)所示。level-1 合成完畢後的結果和P3 繼續做平行合成，最後可以得到level-3 的C2，所有的狀態都被列舉出來。

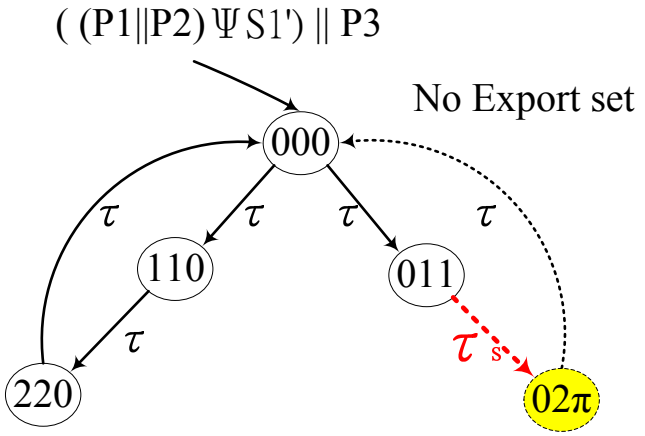


$$(((P1\|\|P2)\Psi S1') \ \|P3)$$

(a)



(b)



(c)

Figure 3-6 加入安全性質後的平行合成內部圖表

(a)將安全性質利用階層式方式加入系統中檢驗

(b)子系統C1' = (P1|| P2)ΨS1'

(c) 包含死結的全域系統C2' =((P1|| P2)ΨS1')||P3

Figure 3-6表示在子系統中加入安全性質來做檢測，我們將其記做(P1|| P2)ΨS1，讀作P1 compose P2 psi S1。Ψ這個動作有點類似合成的意味，但他並不屬於two-way or multi-way的同步模式。圖中我們可以發現每個合成狀態旁都跟著一個

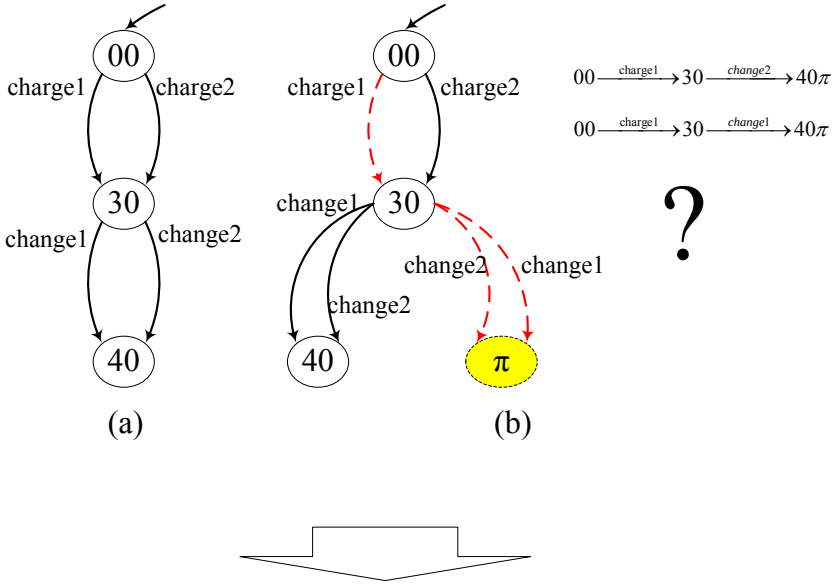
目前可達的image automata狀態，舉例來說Figure 3-6(b)一開始的起使狀態都為 0 安全檢驗引擎將所有行程的初始狀態都放入Queue中進行BFS的展開方式，P1 和 P2 在經過行動-a的狀態轉移時(記作 $_{P1\|P2}(0\|0, \|-a,0\|1)$)，由於-a也包含於image automataS1 欲檢驗的行動，因此S1 其目前狀態也會平行的由狀態 0 轉移入狀態 1，接著 $_{P1\|P2}(0\|0, \|-b,0\|2)$ 進行轉移時，S1 同樣可以經過行動-b轉移到下一個狀態，此時S1 發現自己進入了π狀態，他將通知安全檢驗引擎要將剛剛新產生的狀態標示為π狀態（如Figure 3-6(b)狀態編號$(12,\pi)$），代表系統在此處有機會進入死結狀態，、接著安全檢驗引擎會將其從BFS的Queue中剔除，原本會擴展的狀態也將不會在系統出現（如Figure 3-6(b)狀態編號$(10,\pi)$）。子系統C1'中最後顯示找出三個死結狀態，利用此子系統C1'再去和P3 做平行合成，將發現全域系統最後會出現一個死結狀態。這提供了一個重要訊息，雖然在C1'中有三個死結狀態，但這是在子系統的行為，如果擴展到全域系統中，由於其他行程的影響，全域系統不一定會進入所有的死結狀態，如Figure 3-6(c)。

3.4 新型態合成引擎

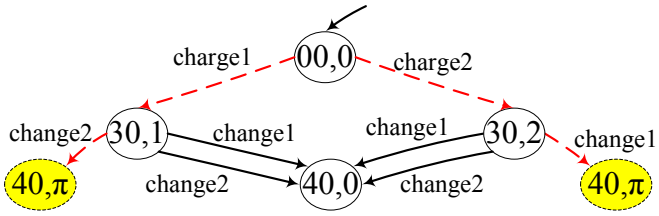
在一般的檢驗安全性質作法中，有另一種不破壞系統原有性質的作法，他是利用分裂行動的方式來增加死結狀態，而在死結產生的同時還能繼續保持系統原有系統特性，但實際上這種作法會發生一個狀況－錯亂的死結狀態。在Figure 3-7 (a)我們利用兩名顧客在加油站找零錢的例子來說明這種情況的產生。正確的情況是當顧客 1 在索價後，加油站系統必須正確的找錢給顧客 1，同樣的加油站系統也必須提供正確的服務給顧客 2。當應該找給顧客 2 的錢最後卻找給顧客 1，此時發生找錯錢的狀況，此時將違反系統的合法行為。這樣的安全檢驗引擎同樣的可以檢驗出不合法狀態，並加入死結狀態，但卻發現在顧客 1 原本正確的行為卻也進入了死結狀態（此死結狀態是由顧客 2 索價後，加油站系統卻錯將應該給

顧客 1 的錢找給了顧客 2，如圖Figure 3-7(b)）。

會發生錯亂的死結狀態在於原系統中設計的不良，當系統設計者認為在某種狀態下透過不同的行動(行動>1 時)可以轉移到相同的狀態，而欲檢驗的安全性質透過不同的行動會轉移到不同的狀態時，雖然利用安全性質檢驗引擎可以找出系統錯誤，但卻沒法精確的表示如何出錯，甚至原本正確的系統行為卻反而變成違反的狀況。但這樣的情況在我們的安全檢驗引擎中並不會發生，原因在於我們的



使用 ArCats的結合安全檢驗引擎將會自動化的分裂必要的狀態



(c)

Figure 3-7 死結錯亂圖
(a) 部分 Gas-station 系統中的行為
(b) 加入死結後的行為圖
(c)使用 ArCats 安全檢驗引擎所產生的行為

合成引擎將會加入安全性質狀態 ID 作為製作新狀態 ID 的元素，這樣做的好

處在於能自動分辨出何時該分裂狀態，也避免了死結錯亂的產生。底下我們將介紹改良過的 ArCats 合成引擎，其如何產生新的狀態 ID。

3.4.1 狀態 ID 的資料結構

首先我們從資料結構介紹。ArCats狀態ID的型態為unsigned long long int 由表格 3-1可知，其可以儲存到相當大的狀態ID，由於目前 64bit的作業系統和硬體漸漸普及，未來在移植到 64bit的系統也將是相對的容易。

表格 3-1 狀態 ID 資料結構示意圖

Type	Bits	Possible Values
unsigned long long int	64	0 to 18,446,744,073,709,551,615

3.4.2 產生新的狀態 ID

製作新的狀態 ID 的流程總共有兩個部分，如下所示：

（1） .根據子系統中每個 LTS 和將檢驗的安全性質 LTS 的最大狀態 ID 來決定位移數。

（2） .新的狀態 ID 是根據(1)所得到的位移數用 left bit-shift 運算子做移位動作，並將每個 LTS 的目前狀態 ID 利用 bitwise OR 運算子和新的狀態做結合，此時將獲得部分新的 ID。

如Figure 3-8所表式，總共有n 個行程進行平行合成並同時檢驗一個image automata，在Step1 時，LTS1 首先根據getshift()來獲得位移數並作移位動作和利

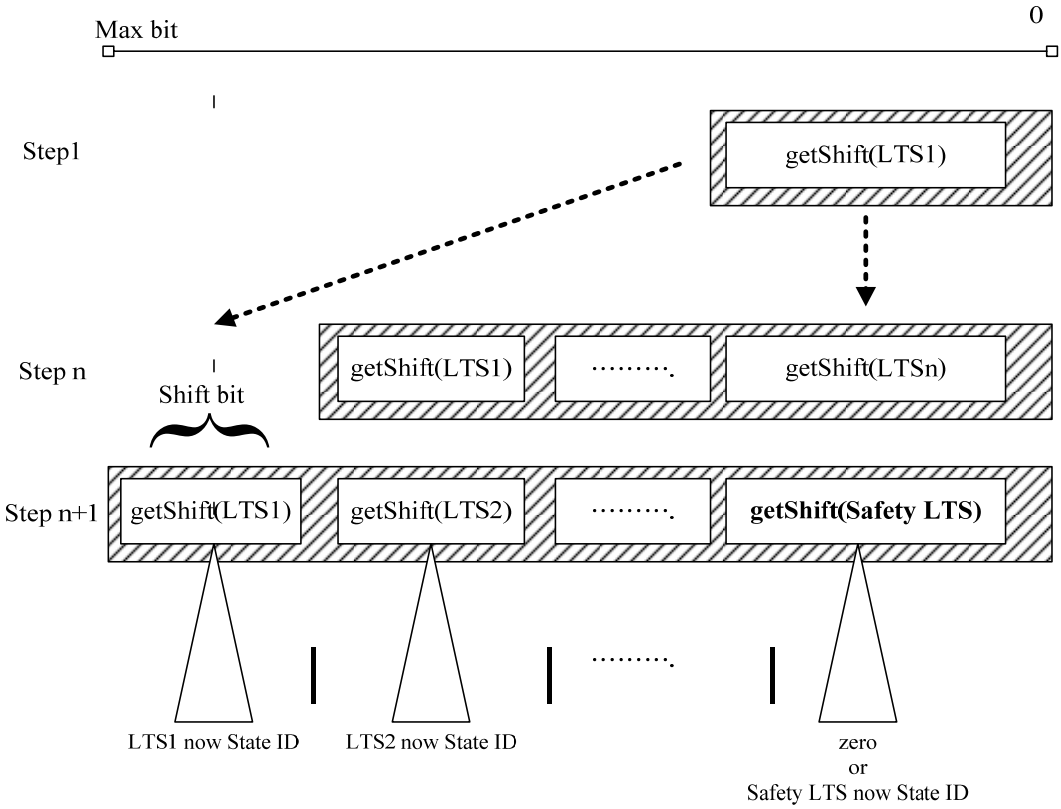


Figure 3-8 安全行程製作新狀態 ID 示意圖

用 bitwise OR 來和新狀態作結合，如此動作重複到 Step n+1，最後的安全性質行程也加入製作新的狀態 ID。

3.4.3 重新對應(Re-Mapping) 狀態 ID

由於多加入 LTS 製作新的狀態 ID，儘管我們使用了 64bit 的數字系統，但還是有可能在製作新狀態 ID 的過程中，造成數字系統的溢位(overflow)。實際上在作局部性分析時只要同時以 64 個 LTS 來合成新的狀態 ID 時，幾乎數字系統就會發生 overflow，此時將降低驗證工具的更能，或許以更彈性的數字系統來取代舊有的數字系統，可以解決此問題。

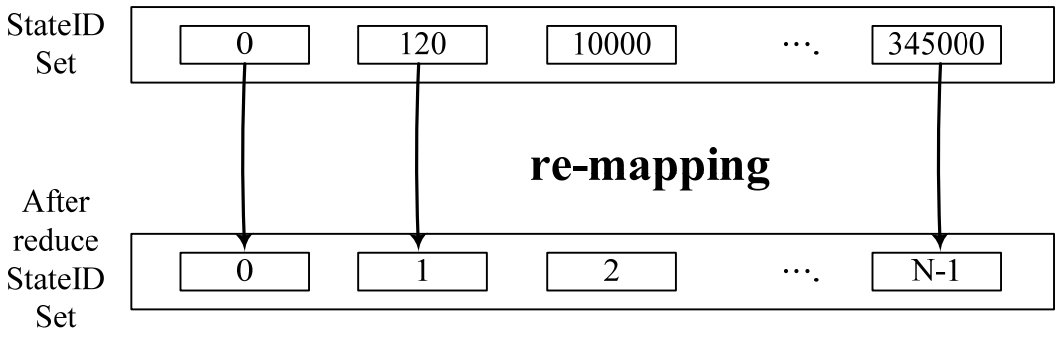


Figure 3-9 re-mapping 示意圖

為了降低 overflow 的發生，我們通常在每個子系統完成所有的狀態列舉後，就會做一次 re-mapping 的動作，所謂 re-mapping 就是將原本排列鬆散的狀態 ID 集，透過重設狀態 ID，將每個 ID 縮小到適當的位數。透過 re-mapping 類似壓縮的方式將狀態 ID 縮小，可利於當下一個階段在製造新的狀態 ID 時，將

能減少因為狀態ID過大或同時合成狀態個數過多造成數字系統overflow，導致compose engine的失效。Figure 3-9為remapping的示意圖，假設合成後共有N個新的狀態ID，如果N=100，表示系統中有些狀態ID過大，如例如最大的狀態ID為345000，事實上如果壓縮狀態ID，應該使用 99 這個號碼就可以表示此狀態ID，故經過re-mapping後將可以化減到適當數字N-1。

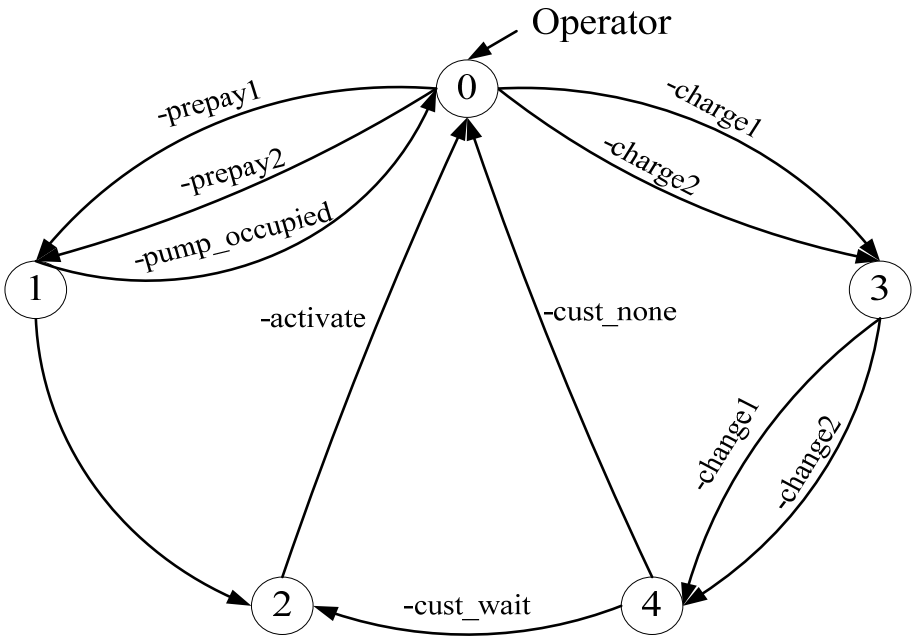
雖然分裂狀態可能會造成佔用更多的儲存空間，但卻可以解決死結錯亂的發生，如此系統偵測出來的錯誤才有意義，也更精確。事實上要減少狀態分裂的發生，必須由改善欲檢驗的安全性質來著手，也就是說如何寫出一個好的安全性質使其能符合檢驗系統將是模型驗證中一門重要的課題。

3.5 使用 ArCats 檢驗 Gas-station 系統

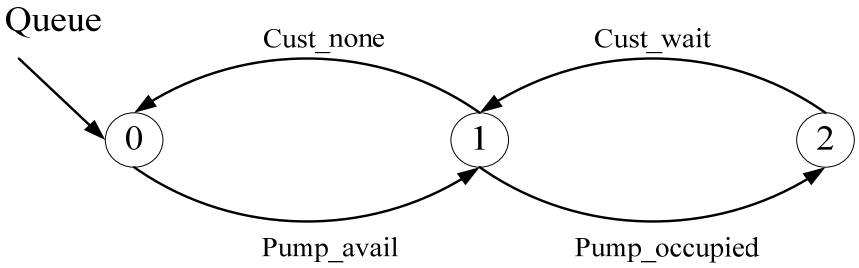
在Figure 3-1中顯式了加油站的局部性分析系統架構，本小節中將利用加油站的例子，說明如何利用我們實做於ArCats上的安全檢驗引擎來檢驗系統的安全性質。3.5.1將先介紹加油站系統架構，並模型化系統和所要檢測的系統性質。3.5.2將利用ArCats檢測出系統的錯誤。

3.5.1 模型化 Gas-station

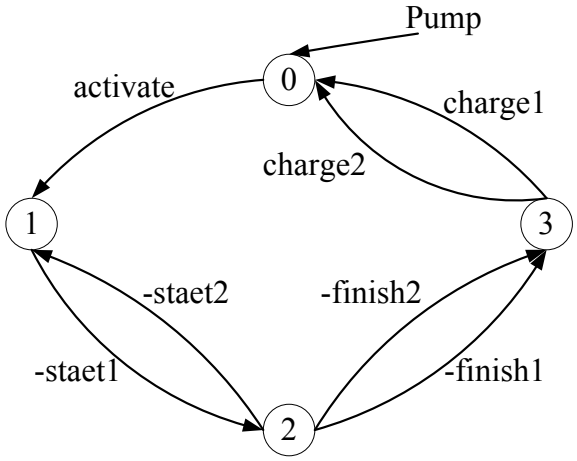
加油站系統最早由Helmbold和Luckham[13]所提出，我們利用Cheung[29]所提供的CSP原圖改寫為CCS的方式。



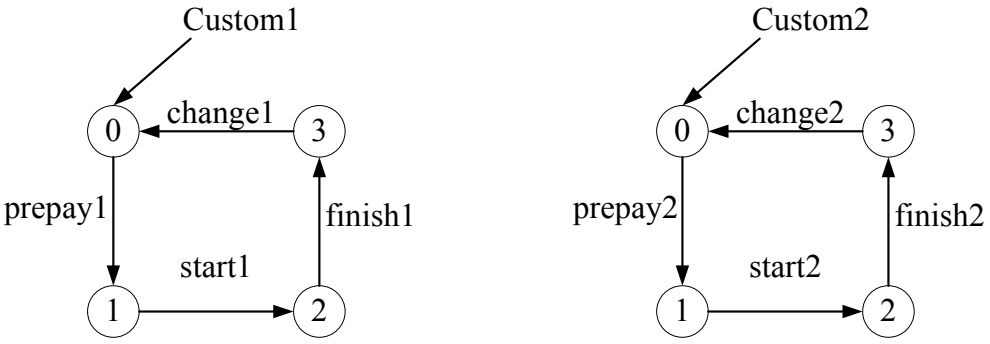
(a)Behavior of Operator.



(b)Behavior of Queue.



(c).Behavior of Pump.

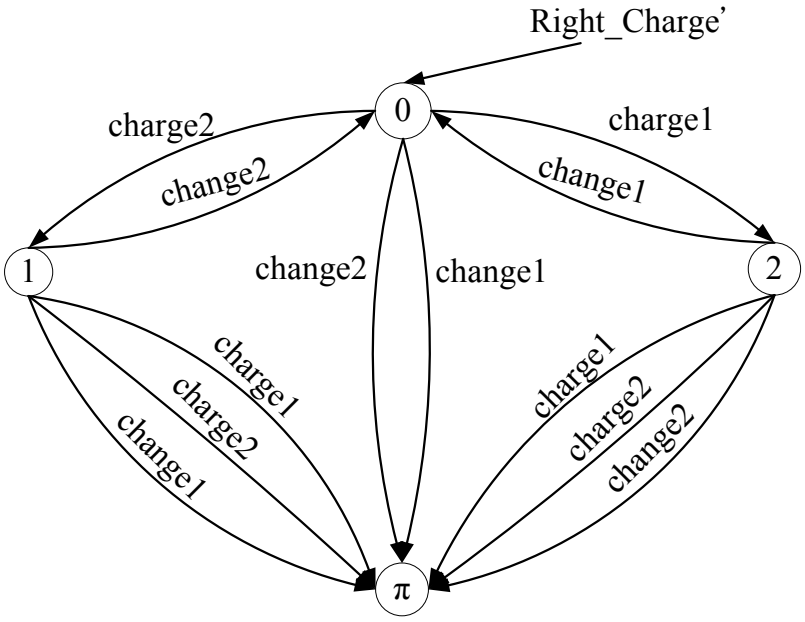


(d).Behavior of two customers Custom1 and Custom2.

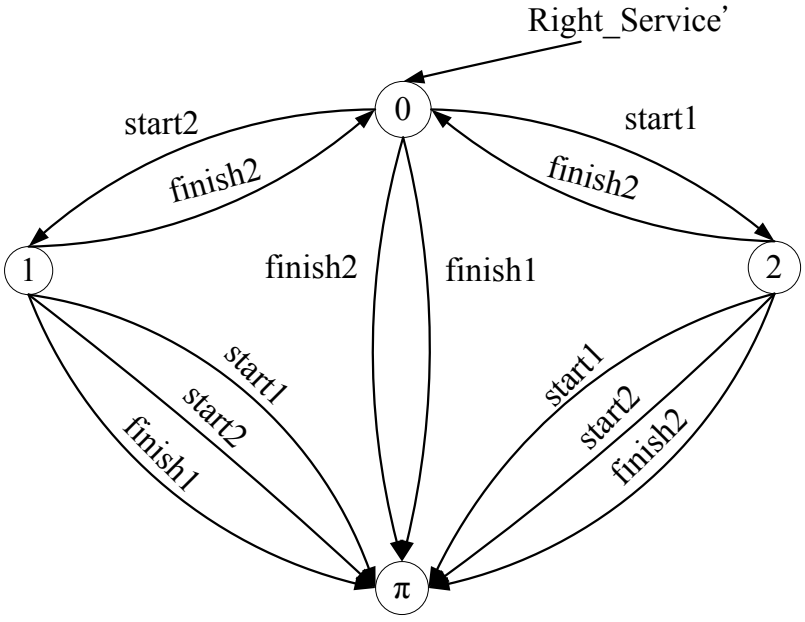
Figure 3-10 利用 two-way rendezvous 的方式製作 Gas-station 的五個子系統
(a)Operator (b)Queue (c)Pump (d)Custom1 和 Custom2

假設目前有兩位顧客於加油站加油，則加油站系統是由五個子系統所構成，分別為:Operator、Queue、Pump、Custom1、Custom2。其中Operator和Queue將持續提供顧客（Custom1,Custom2）的服務要求。每個子系統利用two-way

rendezvous的溝通方式建構出所需要的LTS模型，Figure 3-10顯示所有模型化後的子系統：



(a)Image automata of Right_Change



(b)Image automata of Right_Service

Figure 3-11 兩種安全性質 LTS 圖示
(a)表示系統是否正確找錢的 Right_Change'(b)表示系統是否提供正確服務的 Right_Service'

這裡我們將提供簡易的安全性質供ArCats檢驗，如圖Figure 3-11分別是檢驗

（a）當顧客要求找錢時，是否能夠正確的找錢給每位顧客。(b) 當顧客要求服務時，是否能正確的提供服務給要求的顧客，其中 π 代表死結狀態。實際上在檢驗安全性質時（a），原本應該找給顧客 1 的錢在某種狀態下將可能錯誤的找給顧客 2 而發生找錢錯亂，此時系統將違反正確找錢的安全性質。而（b）不會違反安全性質。

3.5.2 使用 ArCats 檢驗 Gas-Station

當我們利用ArCats按照如Figure 3-1的階層式架構去作局部性分析後，檢驗結果將會以CCS檔案格式做為輸出，此時我們可以利用ArCats的附加功能將CCS檔案透過dot轉換成圖形方便觀察。

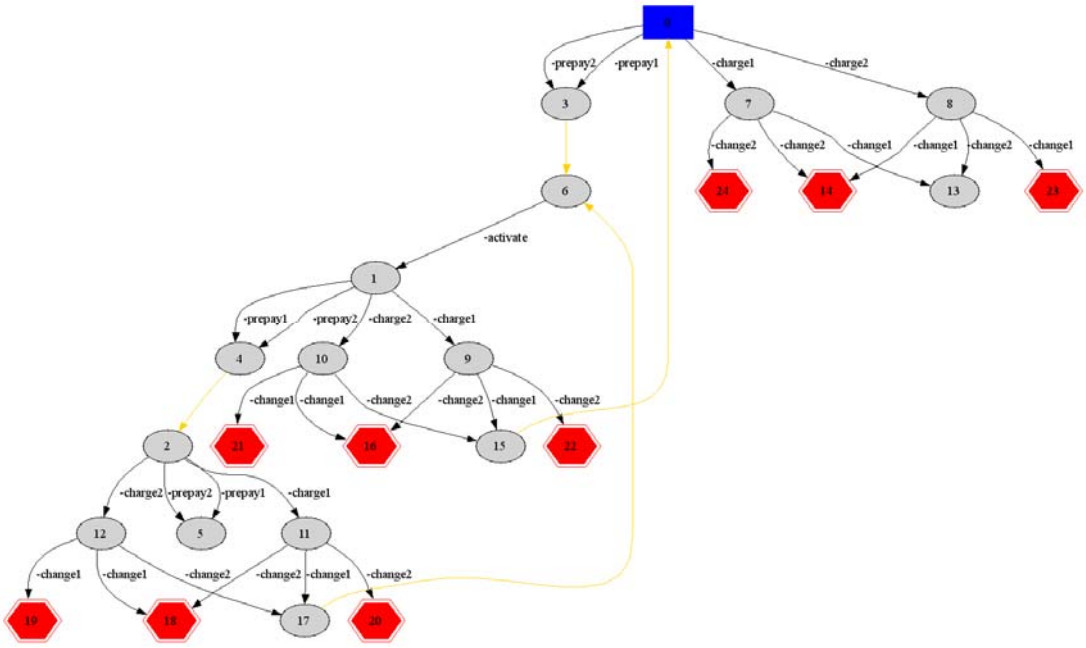


Figure 3-12 在 level_1 中做 Right_change 安全性質檢驗的子系統 Counter

Figure 3-12表示利用ArCats功能轉換出來的子系統Counter，藍色方形狀態表示初始狀態，紅色六角形狀態表示死結狀態，黃色線條表示平行合成後形成內部行為

τ。從子系統Operator和Queue在做合成並檢驗安全性質Right_change時，可發現會出現違反情況，系統有可能會進入死結狀態，而Figure 3-13中Stau為特別保留的全域行動，經由Stau所轉移到的狀態必為死結狀態，圖中顯示當整個系統進行平行合成後，系統確實會進入了死結狀態。因此我們可以追塑到當

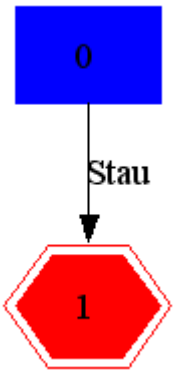


Figure 3-13 Gas-Station 經過最小化後的最終結果

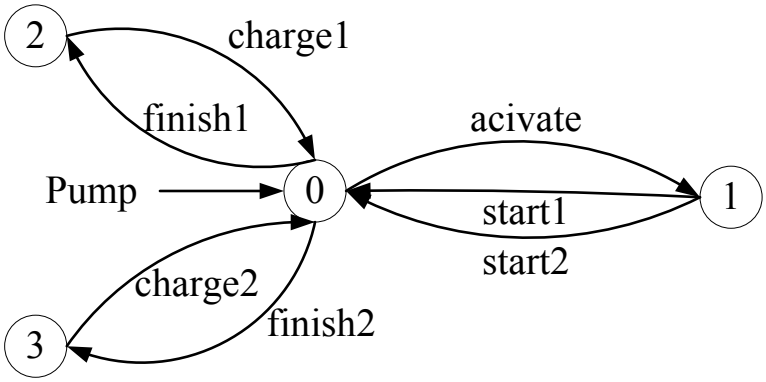


Figure 3-14 修正過後的 Pump Model

加入安全性質後，和其他子系統做平行合成卻仍然發生死結的地方，由Figure 3-1可以看出在level-2 的Pump是第一個和Counter進行平行合成後仍然有死結產生的地方，故我們可以去修正Pump的模型，修正後的Pump如Figure 3-14所示，當我們再利用修正後的模型來檢驗Right_change時，將發現子系統並不會出現死

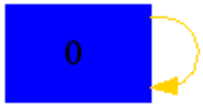


Figure 3-15 Gas-Station 使用正確的 Pump 平行合成並最小化後的最終結果

結狀態，最終全域系統經過最小化後將呈現如 Figure 3-15，系統只會留下一個行動Tau，代表系統將會正確的運行，且能夠持續不斷的運作下去而不會進入死結的狀態。

當檢驗 Right_service 時，由於系統中並未產生死結狀態，故系統最終將會正確的運行。