使用高階派翠網路驗證工作結構
A High-Level Petri Nets Approach to verifying Task Structures

賴聯福 Lien F. Lai
資訊管理系
建國技術學院
lflai@cc.ckit.edu.tw

李允中 Jonathan Lee　楊鎮華 Stephen J. Yang
資訊工程系
國立中央大學
{yjlee,jhyang}@se01.csie.ncu.edu.tw

潘健一 Jiann-I Pan
資訊管理系
建國技術學院
jipan@cc.ckit.edu.tw

摘要

隨著知識庫系統的技術被廣泛的接受，為了增進系統的可靠度和品質，知識庫系統的驗證更有其必要性。傳統上，知識庫系統的驗證都只著重在知識的層次，而知識層次的驗證只能找出法則庫中的重複、矛盾及循環等的結構性錯誤，並無法偵測出需求規格發生不一致狀況的語意錯誤。在本論文中，我們提出在需求規格的層次上，使用工作結構來模組知識庫系統中的使用者需求和領域知識，以及利用高階派翠網路來表達與驗證此工作結構規格。使用高階派翠網路來表示工作結構時，將會遭遇到幾個困難點，例如，工作分解的表示方式、限制和狀態模型的表示法、跟隨運算子與立即跟隨運算子的差別、以及條件選擇運算子中互斥概念的表示方式等。此外，在工作結構的驗證上，我們利用高階派翠網路的特性以完成下列兩項工作：(1) 驗證工作結構的模型規格：藉由限制滿足演算法來檢查限制網路的一致性，並使用限制放鬆方法來修復限制違反的情況；以及 (2) 驗證工作結構的程序規格：利用可達性及特定性的概念，來檢查多個階層間的一致性以及同一個階層內的一致性。

關鍵詞：驗證、工作結構、高階派翠網路。

一、緒論

隨著知識庫系統 (knowledge-based system) 的技術被廣泛的接受，為了增進系統的可靠度和品質，知識庫系統的驗證 (verification) 更有其必要性 [1,13]。傳統上，知識庫系統的驗證都只著重在知識的層次 (knowledge level) [4,5,15,20]。而知識層次的驗證只能找出法則庫中的重複 (redundancy)、矛盾 (conflict) 及循環 (circularity) 等的結構性錯誤。這些傳統的驗證技術有一個共通的缺點和限制，亦即，大部份的方法都無法在需求規格的層次上 (requirements sp ecification level) 來驗證知識庫系統。

在需求規格層次完成知識庫系統的驗證有許多的優點：

(1) 驗證需求規格可以偵測出那些傳統知識層次的驗證所無法找出的語意錯誤 (semantic error)。
(2) 由於軟體系統必須由需求規格一步一步發展而來，每個產品的品質及可靠度跟它們的需求規格的正確性有很大的關係。
(3) 在軟體的需求規格層次上，比較容易讓系統發展者檢查和驗證問題領域的知識，而在軟體的程式碼是很難去檢查及驗證問題領域知識。
(4) 驗證知識庫系統的需求規格可以讓我們在軟體發展生命週期的早期階段，提早偵測錯誤的發生，有助於我們減少設計上的錯誤及降低軟體的成本。

Tsai [19] 以及 Slage [16] 等研究者針對知識庫系統的需求規格，引導了一個實驗性的專案研究，這些研究指出了在發展知識庫系統時，需求規格的重要性。同時，Chandrasekaran 及許多其它的研究者也提出使用工作結構 (task structures) 作為一般化的模型方法來模組知識庫系統[2,17,18]。工作結構是由工作 (task)、方法 (method) 及子工作 (subtask) 的樹狀結構所組成，並依此架構，以逐漸精煉 (refinement) 的方式來組織知識庫系統中所有的知識。可惜的是，這些方法都沒有提出一個正規 (formal) 的方式來驗證知識庫系統的預期功能和動態行為。

我們已經提出一個正規化的工作結構，使得知識庫系統中的功能規格及動態行為可以經由工作結構來正規地模組和描述 [10,12]。我們的方法可以透過工作的一般化概念來擷取以及組織領域知識、功能需求、和解問題方法。對於由工作結構所模組的需求規格來說，規格中的每個部分都可以在單一層中或在層與層之間重複地精煉 (refine) 和驗證。如同在 [9,11] 中所指出的，我們的方法提供了許多優點，這在需求規格層次建構以及驗證知識庫系統是非常有用的：

- 導入工作結構觀念，提供一個詳細的功能分解技術來組織及精煉功能和行為規格。
- 軟性 (soft) 及硬性 (rigid) 條件的差異，可以來描述矛盾 (conflict) 的功能規格。
- 工作狀態表示式 (Task State Expressions，TSE) 不僅有助於描述預期的控制流程和模組之間的互動關係。同時，可以用來驗證行為規格與系統功能規格之間的一致性。
- 狀態模型 (state model) 使得複雜狀態的描述變得更為容易。在狀態模型中所定義的術語名詞，可以重複地用來描述不同工作的功能。如果缺少了狀態模型，則每次都要描述系統執行前後的狀態條件，如此將會太繁雜而不切實際的。

本論文主要是著重在需求規格的層次上，使用工作結構 (task structures) 來模組知識庫系統，以及利用高階派翠網路 (high-level Petri nets) 來驗證此工作結構規格。高階派翠網路是一個圖形化而且具有數學基礎的模組工具，它具有強大的表達能力可以來描述系統的結構和動態方面的特性。在我們的方法中，選擇使用高階派翠網路來描述知識庫系統是因為：(1) 它的視覺化及正規的表示式，有助於在需求規格層次上模組工作結構，(2) 它具有推理能力，有助於模擬工作結構的動態行為，以及 (3) 它的特性分析機制，有助於在需求規格層次上驗證工作結構。

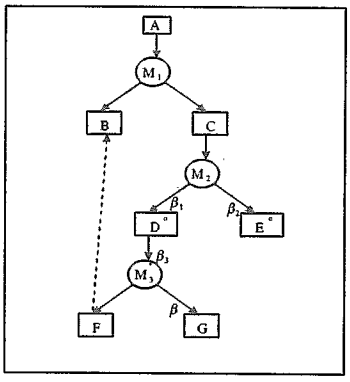
在工作結構的驗證方面，我們利用高階派翠網路的特性以完成下列兩項工作：

(1)驗證工作結構的模型規格 (model specifications)：藉由限制滿足演算法 (constraint satisfaction algorithm) 來檢查限制網路的一致性，並使用限制放鬆方法 (constraint relaxation method) 來修復限制違反的情況。

(2)驗證工作結構的程序規格 (process specifications)：利用可達性 (reachability) 及特定性 (specificity)，來檢查多個階層間的一致性以及同一個階層內的一致性。

二、工作結構

工作結構是由工作、方法、以及子工作的樹狀結構所組成，並依此架構，以逐漸精煉 (refinement) 的方式來組織知識庫系統中所有的知識。我們已提出了一個正規的工作結構表示方式，可以正規地使用工作結構來模組知識庫系統的功能規格及動態行為[10,12]。用工作結構來模組的知識庫系統規格由兩個部份所構成：模型規格和程序規格。模型規格描述系統的靜態特性，而程序規格則描述系統的動態特性，系統的靜態特性可以兩個模型表達：領域模型（domain model）描述領域內的實體物件和實體物件之間的關係；狀態模型（state model）則描述解問題的狀態。系統的動態特性可由兩個部份描述：(1)使用狀態的改變來描述一個工作的功能性 (2)使用工作狀態表示法（TSE）來描述子工作的優先順序和子工作之間的互動關係，亦即描述系統之行為。TSE使用的運算子可以區分為三組不同的運算：(1)順序運算：包括 follow (以";"表示)與 immediately follow (以"," 表示)運算子，(2)分支運算：包括 selectional、optional 以及 condition a 運算子(以"∨"表示)，(3)重覆運算：iteration (以"*"表示)運算子。模型規格和程序規格都提供了逐步修正的方法，首先在較高抽象階層描述模型規格和程序規格，然後它們可在下一個階層進一步地修正為更詳細的規格。



圖一、工作結構的動態行為

圖一顯示工作結構圖中的 TSE 運算子的圖形表示法。方法 M_1 的子工作 B 及 C 之間的關係是順序運算(即(B，C))。分支運算以方法 M_2 的子工作間的關係為例，條件的符號為β_1 與β_2，以及 "o" 代表分支選擇的情況(如β_1D∨β_2E)。重覆有兩種情況：(1)重覆的條件與整個表示式有關(如β_3(F，G))。(2)重覆的條件與方法的子工作有關(如(F，β_4G))。我們稱工作 F 及 B 是 global interaction (以虛線表示)，因為互動是在不同階層的工作間。這種的關係可以用 follo w運算子表示(如 F；G)。

以下的例子是採用 R1/SOAR [20] 的問題領域，舉例說明工作結構的各個重要部份：

R1/SOAR 著重在 R1 組織 (configuration) 工作的 unibus 組織部份上。在此例子中我們著重在兩個工作來描述：configure modules 和 configure a module。configure modules 工作之功能，是以最佳化的順序，盡可能地放入最多 的modules 到目前的 backplane 上。我們可以由重複以下兩個子工作來完成 configure modules 的工作：(1) configure a module 工作，嘗試將目前的 module 放到目前的 backplane 上，(2) obtain the next module 工作，依照最佳化的順序取得目前 module 的下一個 module。當目前的 module 無法放入目前的 backplane 時，知識工程師可能會犯下一個錯誤。一個正確的規格必須指出，在這個情況下應該要跳過 obtain the next module 工作，因為我們尚未將目前的 module 放入中，不能立刻去取下一個 module。同時，一個正確的規格也必須指出，系統在放入下一個 module 之前，應該先執行 find a backplane suitable for the current module 工作。本例中的工作和方法規格如圖二、圖三、及圖四所示。

Task: *Configure-Module*

- **Model Specification**:
  - **Domain Model**:
    module's power drawn, room for current module in the backplane.
  - **State Model**:
    *A **moduled backplane** is a backplane that (1) has as many as possible been placed in, and (2) satisfies the constrained of maintainin the optimal ordering of modules that are configured into the backplane.
- **Process Specification**:
  - **Precondition**: There exists a current backplane, and a sequence of unconfigured modules.
  - **Protection**: The optimal ordering of modules should be maintained (inherited)
  - **Postcondition**:
    *(rigid) The chosen backplane is a **module backplane**.

圖二、工作 Configure-Modules

Method:

- **Parent Task**: Configure-Modules
- **Guard Condition**: TRUE
- **Subtasks**:
  - T_1:Configure-A-Module
  - T_2:Obtain-The-Next-Module.
- **TSE**: $(T_1,(\beta_1\wedge\beta_2)T_2\vee\neg(\beta_1\wedge\beta_2)T_{ex})^*$
  - β_1:The current module is compatible with the current backplane and is configured into the backplane.
  - β_2:There exists a remaining unconfigured module.

圖三、工作 Configure-Modules 的方法

三、使用高階派翠網路來模組工作結構

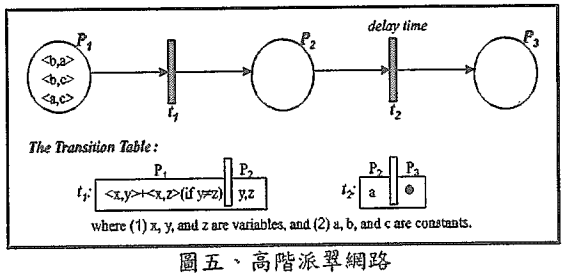
高階派翠網路[6,7,14]為由兩種節點所構成的有向圖。這兩種節點為 place 和 transitio 並由 arc 使其相互連結。帶有 token 的 place (以圓形表示) 反映系統的狀態，而 transition (以矩形表示) 的激發 (fire) 則表達狀態改變的觀念。每個 place 可以含有數個 token，而每個 token 都帶有 token color 的值。transition tabl 指定哪些 token

Task: *Configure-A-Module*

- **Model Specification**:
  - –**Domain Model**: module's unibus load, module's pin type.
  - –**State Model**:
    - * A current module is said to be **compatible** with the current backplane if
      1. The current module's unibus load is less or equal to the unibus load left.
      2. The current module's pin type is the same as the current backplane's pin type.
      3. The current module is a kmc11 and th backplane is either 4 slot or 9 slot.
      4. There is enough room for the curren module in the backplane.
- **Process Specification**:
  - –**Precondition**: There exists a current module, an a current backplane.
  - –**Protection**: The optimal ordering of modules should be maintained (inherited)
  - –**Postcondition**:
    - * (rigid)
      1. β_1:The current module is **compatible** with the current backplane and is configured into the current backplane. Or,
      2. $\neg\beta_1$:The current module is not **compatible** with the current backplane and the curren backplane is a moduled backplane.
  - –**TSE**: $(T_1;\neg\beta_1 T_3)$ [T_3: Find-A-Backplane-Suitable-For-Current-Module]

圖四、工作 Configure-A-Module

color 及其數量會從 transitio 的 input place 移除並加到 transition 的 output place。transitio 被稱為致能的 (enabled) 代表該 transitio 的每個 input place 都擁有足夠 token 而且是正確的 token color。一個致能的 transition 可以被激發。transition 的激發會將致能的 token color 從 input place 中移除，並在 output place 加入正確的 token color。在所有 place 中 token 的分佈狀態被稱為 marking。Initial marking (M_0) 代表初始狀態。一個致能 transition 的激發會改變派翠網路中 token 的分佈 (亦即，改變狀態)。一連串的激發會產生一連串的 marking。如果存在一連串的激發使得 M_0 轉換成 M_n。則 marking M_n 被稱為可抵達的(reachable)，可達性圖(reachability graph)包含所有可能的可抵達 marking。



圖五、高階派翠網路

圖五是一個高階派翠網路的例子。其中(1) P_1，P_2，及 P_3 是 place，(2) <b，a>，<b，c>，<a，c>是 place P_1 的 token color，(3) t_1，t_2 是 transition，(4) t_1 的 transition table 的左邊描述會從 P_1 移除哪些 token color，而右邊描述會增加哪些 token color 到 P_2，(5) t_2 的 transition table 描述會從 P_2 移除哪些 token color，以及增加哪些 token color 到 P_3，(6) t_2 有一個延遲時間。初始 marking M_0 是 [P_1(<b，a>，<b，c>，<a，c>)]。透過一致性的取代{b\|x，a\|y，c\|z}之後，t_1 為致能的，因為 P_1 有足夠的 token 並且是正確的 token color (即<b，a>，<b，c>)。

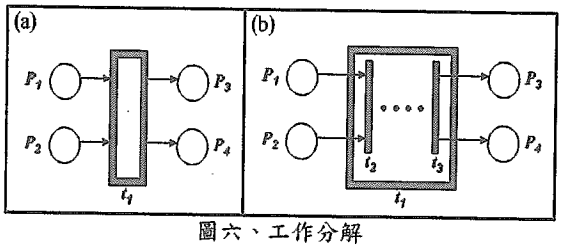
階層式著色派翠網路(hierarchical colored Petri nets)[6,7] 提供五種階層式建構：substitution of transition、substitution of place、invocation of transition、fusion of place 以及 fusion of transition。substitution of transition 允許使用者以一個較複雜的著色派翠網路來取代一個 transition 及其 arc，如此，可以提供一個較為詳細的描述給被取代的 transition。substitution of place 則是用一個較詳細的子網路來取代 place。invocation of transition 允許被取代 transition 的子網路中使用本身的 transition。fusion of place 允許使用者描述一群 place 為同一個 place。相同地，fusion of transiti o 允許使用者描述一群相同的 transition。時間性的派翠網路(timed Petri nets)允許 place 和 transitio 有延遲時間。延遲時間描述在週期性時間的派翠網路中每個 transitio 多快可以起始激發。

高階派翠網路具有多個有用的特性，適合用來將工作結構對映到相對的派翠網路。舉例來說，一個致能 transition 的激發可以表達在工作結構中工作被執行的狀態轉變，一連串 transitio 的激發對應於在 TSE 中的工作序列 (task sequence)。然而，在使用高階派翠網路來表示工作結構時，仍有幾個重要的問題尚待解決：

- 工作 (task)、方法 (method)、和工作分解 (task decomposition) 等概念是工作結構的重要元素，必須能由高階派翠網路表示。少了這樣的概念，我們無法 refine 複雜的工作，就更別說要完全描述工作結構。
- 狀態模型 (state model) 是關於解問題狀態的模型。透過限制的滿足，工作狀態可以被改變。狀態模型的高階派翠網路表示方法必須能表達限制網路的滿足 (satisfaction) 及放鬆 (relaxation) 的語意。
- 工作結構中的方法是透過 TSE 來描述系統的動態行為。用高階派翠網路來表示 TSE 時，有幾個問題必須要解決：follo w運算子與 immediately follo w運算子的差別，在 selectiona 與 conditional 運算子中相互互斥的概念，以及 iteratio n運算子的實作。

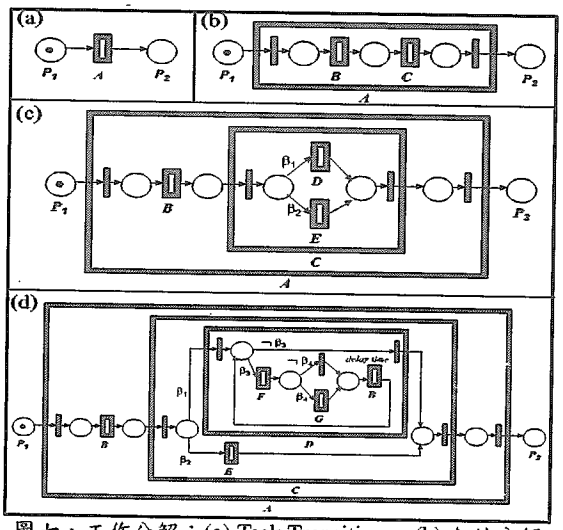
A.將工作分解對映至高階派翠網路

在工作結構中，一個工作可以被視為一個狀態轉換，並且由 precondition、protection、和 postconditi o 來描述它。為了表達狀態轉換的概念，我們可以利用高階派翠網路的 transition 來表示工作 (稱為 task transition)，其中 input place 及 output place 分別描述工作的 precondition 和 postcondition。工作的 protection 則在 input place 及 output place 都必須成立，因為 protectio 是不能被改變的。task transition (以▯表示) 可經由描述它子工作行為的方法或由描述限 滿足的狀態模型來 refine 它。



圖六、工作分解

我們採用階層式著色派翠網路[6,7]中的 substitution of transiti o 來表示工作結構中工作分解的概念。一個 task transition 的分解如圖六所示。考慮圖六(a)中的 task transition t_1，t_1 有兩個 input place (P_1 和 P_2) 及兩個 output place (P_3 及 P_4)。task transition t_1 可由圖六(b)中子網路的取代來作分解，其中我們把 transition t_2 當成輸入埠來連接 t_1 所有的 input place，把 transition t_3 當成輸出埠來連接 t_1 所有的 output place。對每次的 refine 來說，輸出入的 place 及其箭頭必須維持相同。這種 balancing 的概念可以確保被取代的 task transiti o 和它的子網路是保持一致的。如此，被取代的 task transitio n可被視為是一個黑盒子，而子網路則是描述如何完成它的 parent task 的白盒子。

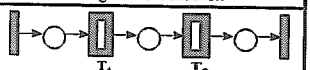
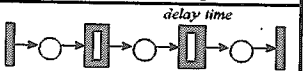
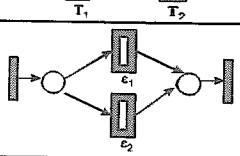
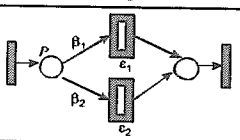
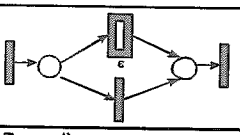
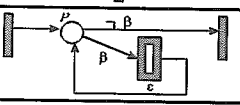


圖七、工作分解：(a) Task Transition (b) A 的分解 (c) C 的分解 (d) D 的分解

以圖一中的工作結構為例，首先我們用一個 task transition 來表示工作 A (見圖七(a))。由於工作 A 可由它的方法 M_1 來 refine，對應的 task transition 會被分解成一個包含 A 的兩個子工作 B 和 C 的子網路 (見圖七(b))。接著，task transition C 會由 M_2 refine 並由含 D 和 E 的子網路所取代 (見圖七(c))。最後，task transition 會分解成含有 F、G 和 B 的子網路 (見圖七(d))。

B.將工作狀態表示式(TSE)對映至高階派翠網路

TSE 是正規表示式的擴充，可以用狀態轉換來表達。為了能將 TSE 轉換為高階派翠網路，我們將 TSE 中的工作序列視為一連串 transition 的激發，每個 transition 的激發相當於一個工作的被執行。表一中描述如何使用高階派翠網路來表示 TSE 的運算子。首先，follo w運算子與 immediately foll o 運算子的差異可由延遲時間來區分，對一個有延遲時間的 task transitio 而言，transition 的激發將會延遲一段時間。所以在 follo w運算子之後的工作 transitio 有延遲時間，而 immediately foll o 運算子則沒有。其次，互斥概念可以由矛盾結構 (conflict structure) 來表達。在派翠網路的矛盾結構中，當其中一個致能的 transition 開始激發時，token 不能被用來致能另一個 transition。因此，selectional 運算子可以對映至一個矛盾結構來表達互斥的語意。在 condition a 運算子的情況，兩個條件β_1 及β_2 可由不同的 token color 來表示。如果 place P 有一個 token color β_1，則 E_1 的 transition table 會致能 E_1。另一方面，如果 P 含有β_2，E_2 的 transition table 會致能 E_2。在 optiona 運算子的情況中，不是 E 就是一個空 (dummy) 的 transition 會被激發。Iteratio n運算子可以轉換成一個 self-loop transition，它的輸入箭頭帶有一個 token color 來表達重覆的條件。如果 place P 有一個 token color β，task transition E 會被激發然後加一個新的 token color 到 place P，另一方面，如果 P 有$\neg\beta$，則就會離開重覆結構。因此，除非$\neg\beta$被滿足，否則重覆結構將不會停止。

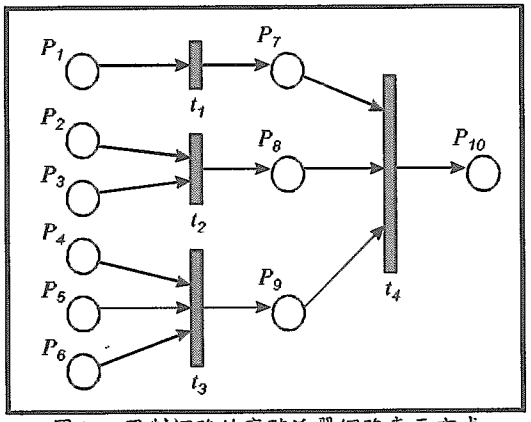
TSE Operators	Syntax	In High-Level Petri Nets
Immediately Follow	(T_1,T_2)	
Follow	$(T_1;T_2)$	
Selectional	$(\varepsilon_1 \vee \varepsilon_2)$	
Conditional	$(\beta_1\varepsilon_1 \vee \beta_2\varepsilon_2)$	
Optional	$[\varepsilon]$	
Iteration	$(\beta\varepsilon)^*$	

表一、TSE 運算子與高階派翠網路的對應

根據以上的討論 (見表一)，圖一中的工作結構可被轉成圖七(d)。首先，A 的方法 M_1 的 TSE 是(B，C)，根據 immediately follo o 運算子的對映可轉成圖七(b)。其次，C 的方法 M_2(也就是(β_1D$\vee\beta_2$E))，根據 conditional 運算子可轉成 task transition C 的子網路(見圖七(c))。最後，D 的方法 M_3 和 F 的 TSE 的組合 (也就是(β_3(F,β_4G;B))*)，根據 iteration、conditional、和 follo w運算子會被轉換成圖七(d)中 task transitio n 的子網路。

C.將限制網路對映至高階派翠網路

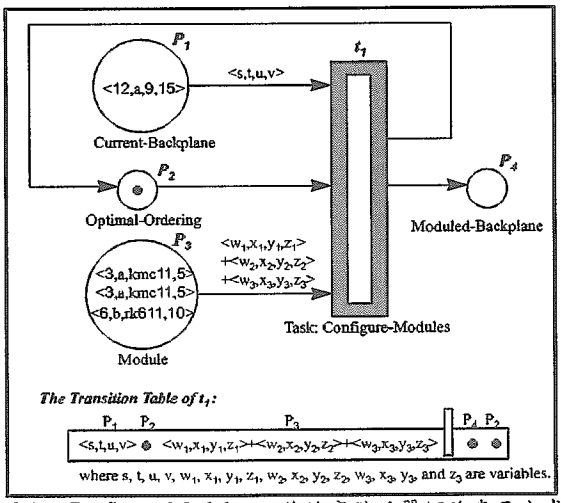
在工作結構中，狀態模型是描述解問題的模型。透過限制的滿足，工作的狀態可以被改變。因此，狀態模型可由限制網路來描述。限制網路是由一些必須同時滿足的限制集合所構成[3,8]。傳統上，限制網路可以表示成一個變數集合和一個述詞 (predicate) 集合所組成，其中每個變數必須可以有適當的值來同時滿足所有的 predicate。在本論文中，我們將限制轉換成一個 transition 及其輸出入 place，其中(1)每個 input place 表示一個變數，(2)input place 裡的 token color 代表它實際的變數值，(3) transition table 表達一個述詞，(4)在 output place 加入一個 token color 表達述詞被滿足的語意。



圖八、限制網路的高階派翠網路表示方式

如圖八所示，我們可以使用高階派翠網路來表示狀態模型的限制網路。其中單元限制(unary)可以表示成 transition t_1，其中 t_1 的激發表達限制滿足的語意。只有當 input place P_1 有正確的 token color，限制才會被滿足，並在 output place P_7 加入 token color。相同地，transition t_2 和 t_3 表示二元(binary)和三元(ternary)的限制。transition t_4 表示一個限制網路中包含三個限制：t_1、t_2 和 t_3。如果 P_7、P_8 和 P_9 有正確的 token color(也就是說，三個限制同時滿足)，t_4 會被激發並加入 token color 到 P_{10}。所以，我們可以藉由檢查 P_{10} 來檢查限制網路的滿足。

考慮 R1/SOAR 的例子(見圖二、三和四)，Configure-Modules 工作可以轉成圖九，其中(1)工作 Configure-Modules 可以表示成 task transition t_1，(2) precondition 由 input place P_1 和 P_3 來描述，(3) postconditi o 由 output place P_4 來描述，(4) protecti o 由同時為 t_1 的 input place 和 output place 的 P_2 來描述。Configure-Modules 的方法是 $(T_1 \ , \ (\beta_1 \wedge \beta_2)T_2 \vee \neg(\beta_1 \wedge \beta_2)T_{exit})^*$，而工作 T_1 的 TSE 是 $(T_1 ; \neg\beta_1 T_3)$。我們可以根據 T_1 來組合這兩個 TSE 以獲得完整的 TSE：$((T_1 \ , \ (\beta_1 \wedge \beta_2)T_2 \vee \neg(\beta_1 \wedge \beta_2)T_{exit})^* ; \neg\beta_1 T_3)$。因此，Configure-Modules 可以根據它的 TSE 分解成圖十。圖十中的 Configure-A-Module (即 t_5)可更進一步根據它的狀態模型中的限制網路來分解(見圖十一)。其中，T_1 狀態模型中的限制 1、2、3 和 4 (見圖四)可分別轉成 transition t_{11}、t_{12}、t_{13} 和 t_{14}。



圖九、Configure-Modules 工作的高階派翠網路表示方式

四、規格的驗證

驗證知識庫系統的需求規格可以讓我們在軟體發展生命週期的早期階段，提早偵測錯誤的發生，有助於我們減少設計上的錯誤及降低軟體的成本。使用高階派翠網路來驗證工作結構可以應用在 (1) 每個工作的模型規格，使用限制滿足演算法 [8] 以檢查限制網路的一致性，並且使用限制放鬆方法 [3] 修正限制違反的情況；以及 (2) 每個工作的程序規格，根據高階派翠網路的可達性 [7,14] 及特定性的概念 [10] 的比較來檢查不同階層間的一致性。。以下是我們提出的演算法，從工作結構中最上層的工作開始往下，一次處理一個工作，直到所有的工作規格都被驗證過為止。

演算法:以高階派翠網路驗證工作規格 T，包含

1.驗證 T 的模型規格：

(a)對 T 中所有的限制建立限制階層。

(b)根據每個限制的強度，一層一層地建立限制網路。

(c)藉由限制滿足演算法來檢查限制網路的一致性，以及使用限制放鬆方法來修復被違反的限制。

2.驗證 T 的程序規格：

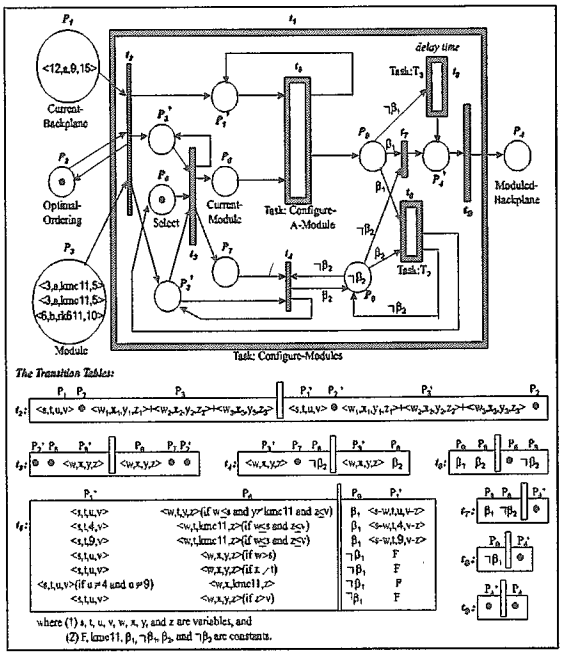
(a)使用高階派翠網路的可達性（reachabilit ），來比較 T 的程序規格與包含 T 的方法(稱為 T 的父方法)和方法中其它子工作，來證明 T 的 precondition 的成立與否。

(b)使用特定性（specificity），比較工作 T 的程序規格與完成 T 的方法(稱為 T 的子方法)和它的子工作之一致性檢查。

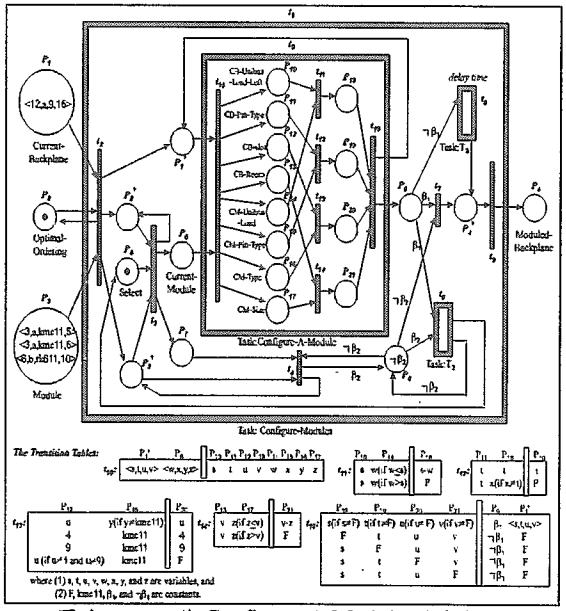
(1)T 的子方法的前置狀態描述必須比 T 的 preconditi o 在語意上更為特定。

(2)T 的 protectio 沒有被任何 T 的子工作違反。

(3)T 的子方法的後置狀態描述必須比 T 的 postconditi o 在語意上更為特定。



圖十、工作 Configure-Modules 的分解圖



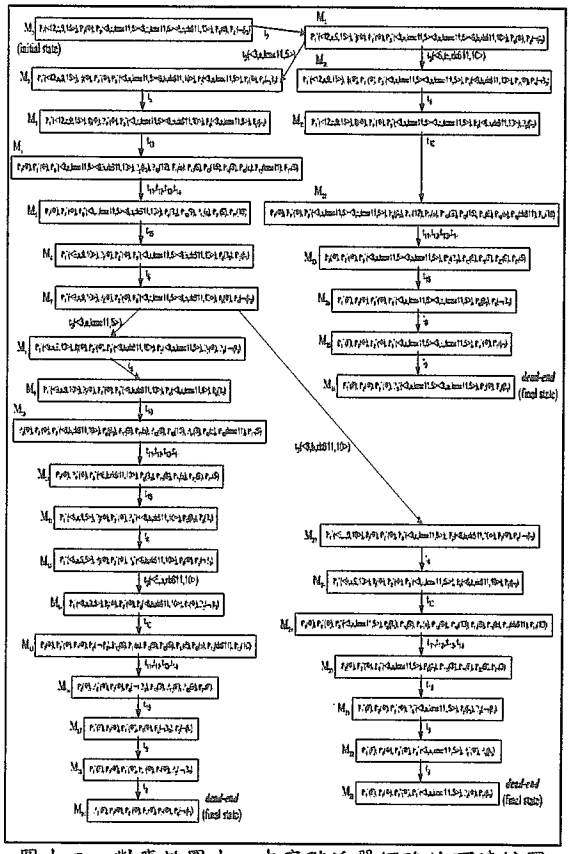
圖十一、工作 Configure-A-Module 的分解圖

A.驗證模型規格

限制階層是根據每個限制的強度(以 C_0 到 C_n 表示)建立[3,12]。限制網路從高階往低階一層一層地的建立。可以用來檢查所有限制之間的一致性。我們依循下列三點來建立限制網路：(1)將每個限制轉換成一個 transition，(2)將限制網路視為一個限制，表達所有的限制需同時被滿足，和(3)將限制網路轉換成一個 transition，其 input place 是所有限制 transitio 的 output place。最強的限制(以 C_0 表示)首先會被加入，以形成初始的限制網路。如果所有的 C_i 限制都被檢查為一致，則 C_{i+1}(比 C_i 弱)限制會被加入限制網路中，然後再一次檢查限制網路的一致性。如果發生了不一致的情況，會運用限制放鬆技術並將強度最弱的限制移除掉。當所有的限制都被檢驗過並加入限制網路中，則限制網路的驗證就完成了。

使用高階派翠網路驗證工作的狀態模型，可以轉換成限制滿足的問題，把限制視為一個 transitio 及其輸出入 place，其中(1)每個 input place 代表一個變數，(2)input place 裡的 token color 代表它的變數值，(3) transition table 表達一個 predicate，(4)在 output place 中加入一個 token color 代表滿足一個限制的 predicate。所以，限制網路可用一個 transitio 來表示，這個 transitio 其 input place 是所有限制 transitio 的 output place。Marking M 稱為滿足一個限制網路 CN 若且唯若 M 可以激發 CN 而且不會加入錯誤值(以 F 表示)到 CN 的任一個 output place。也就是說，限制網路只有在所有的 input place 都同時有正確的 token color 時才會被滿足。而且，限制網路 CN 為一致的若且唯若至少存在一個可達的前置狀態來滿足 CN。

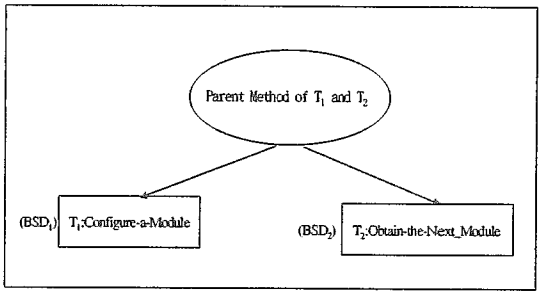
考慮 R1/SOAR 的例子(見圖十一)，它的可達性圖如圖十二所示。為驗證工作 Configure-A-Module(t_5)的狀態模型，我們檢查是否至少有一個 t_{10} 的前置狀態能滿足限制網路 t_{15}。檢查可達性圖中所有可以激發 t_{10} 的 marking，有五個可達 t_{10} 的前置狀態：M_3、M_9、M_{14}、M_{21}、和 M_{28}。每個都可激發 t_{15} 而分別達到 M_6、M_{12}、M_{17}、M_{24}、和 M_{31}。因為 M_6 和 M_{12} 沒有'F' token color(即錯誤值)，因此，M_3 和 M_9 可滿足 t_{15} 的事實，確保了限制網路是一致的。



圖十二、對應於圖十一中高階派翠網路的可達性圖

B. 驗證一個工作的程序規格與其父方法是否一致

如果工作 T 的 precondition，可以從 T 的前置狀態描述中推論出來，則 T 的程序規格為一致的。也就是說，T 的 preconditio 在 T 的前置狀態描述必須是真。相同地，如果從工作的前置狀態描述可以推論出工作的 precondition 為假，則程序規格是不一致的。藉由檢查可達性圖，可以獲得每個工作 T 的可達前置狀態描述。所以，我們可以藉由檢查是否每個 T 的可達前置狀態描述含有 T 的 precondition 來驗證工作與其父方法的一致性。



圖十三、工作 Configure-A-Module 及其父方法

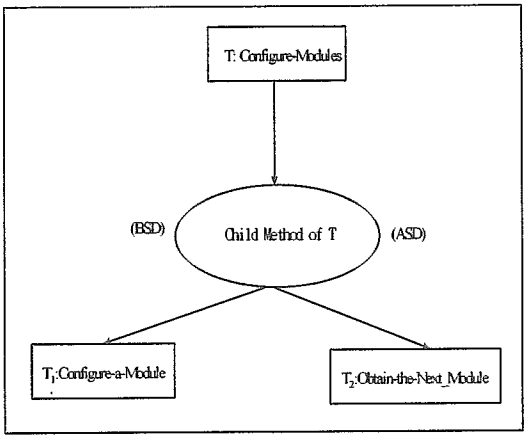
在圖十三的例子中，T_1 的前置狀態描述(BSD_1)是它的父方法中 T_1 執行之前的狀態。T_2 的前置狀態描述(BSD_2)是 BSD_1 經過父方法 TSE 的執行後，在 T_2 執行之前的狀態。為驗證其一致性，我們需要檢查是否 T_1 和 T_2 的前置狀態分別在 BSD_1 和 BSD_2 中為真。在可達性圖中 BSD_1 是激發 transition t_{10} 之前的 marking(見圖十二)，BSD_2 是激發 transition t_6 之前的 marking。

檢查在可達性圖中有可能激發 t_{10} 的 marking，有五個可達的 BSD_1：M_3、M_9、M_{14}、M_{21}、和 M_{28}。另一方面，工作 T_1 的 precondition 是由 t_5 的 transition table 的左邊所描述(見圖十二)。我們首先檢查在 M_3 中 T_1 的 precondition 是否為真。透過對[P_1(<s，t，u，v>)，P_6(<w，t，kmc11，z>)](若 w≥s 且 z≤v)情況的一致性取代{12|s，a|t，9|u，15|v，3|w，5|z}，可知在 M_3 中 T_1 的 precondition 為真，因為 M_3 含有[P_1(<12，a，9，15>)，P_6(<3，a，kmc11，5>)]。相同地，分別透過取代{9|s，a|t，9|u，10|v，3|w，5|z}、{6|s，a|t，9|u，5|v，6|w，b|x，rk611|y，10|z}、{12|s，a|t，9|u，15|v，6|w，b|x，rk611|y，10|z}、和{9|s，a|t，9|u，10|v，6|w，b|x，rk611|y，10|z}，在 M_9、M_{14}、M_{21}、和 M_{28} 中 T_1 的 precondition 都為真。由於 T_1 的 precondition 在所有的 BSD_1 中都為真，可知工作 Configure-A-Module 與其父方法是一致的。

接著，我們檢查在所有可達的 BSD_2 中 T_2 的 precondition 是否都為真。檢驗在可達性圖中有可能激發 t_6 的 marking，找出兩個可達的 BSD_2：M_6 和 M_{12}。工作 T_2 的 precondition 是由 t_6 的 transition table 的左邊所描述(也就是，[$P_9\beta_1$)，$P_8(\beta_2)$]，其中 β_1 和 β_2 是變數)。因為 M_6 和 M_{12} 都含有[$P_9\beta_1$)，$P_8(\beta_2]$，所以在 M_6 和 M_{12} 中 T_2 的 precondition 都為真。因此，我們知道工作 Obtained-The-Next-Module 的 precondition 和它的父方法是一致的。

C.驗證一個工作的程序規格與其子方法是否一致

驗證一個工作 T 與其子方法是否一致，有三個步驟：(1)T 的子方法的前置狀態描述必須比 T 的 precondition 在語意上更為特定，(2)T 的 protection 未被任何 T 的子工作所違反，(3)T 的子方法的後置狀態描述必須比 T 的 postcondition 在語意上更為特定。S_1 狀態比狀態 S_2 更為特定若且唯若 S_1 包含 S_2。



圖十四、工作 Configure-Module 及其子方法

考慮工作 Configure-Modules(T)與其子方法和兩個子工作 T_1、T_2(見圖十四)。BSD 和 ASD 表示它子方法的前置狀態描述和後置狀態描述。為驗證一致性，我們必須檢查是否(1)BSD 比 T 的 precondition 更為特定，(2)T 的 protection 未被任何的子工作所違反，(3)ASD 比 T 的 postcondition 更為特定。BSD 是在可達性圖中 transition t_2 被激發前的 marking(見圖十二)，而 ASD 包含 TSE 中所有的 dead-end 以及 transition t_9 被激發後的 marking。

檢查在可達性圖中所有可能激發 t_2 的 marking，只有一個可達的 BSD：M_0。另一方面，工作 T 的 precondition 是由 t_1 的 transition table 的左邊所描述(也就是，[P_1(<s，t，u，v>)，P_2(・)，P_3(<w_1，x_1，y_1，z_1>+<w_2，x_2，y_2，z_2>+<w_3，x_3，y_3，z_3>)]，其中 s、t、u、v、w_1、x_1、y_1、z_1、w_2、x_2、y_2、z_2、w_3、x_3、y_3、和 z_3 都是變數)。透過一致的取代{12|s，a|t，9|u，15|v，3|w_1，a|x_1，kmc11|y_1，5|z_1，3|w_2，a|x_2，kmc11|y_2，5|z_2，6|w_3，b|x_3，rk611|y_3，10|z_3}之後，可知 M_0 比 T 的 precondition 更為特定。因為 M_0 包含[P_1(<12，a，9，15>)，P_2(・)，P_3(<3，a，kmc11，5>+<3，a，kmc11，5>+<6，b，rk611，10>)]。所以，我們可知工作 Configure-Modules 的 precondition 與它子方法的前置狀態描述是一致的。

接著，我們檢查 T 的子方法所有的前置、進行中和後置狀態以確定 T 的 protection 未被違反。T 的 protection 是[P_2(・)]。由於在可達性圖中每個 marking 都含有 P_2(・)，因此，工作 Configure-Modules 的 protection 未被任何子工作所違反。

最後，我們驗證 T 的 postcondition。考慮在可達性圖中所有 dead-ends 和 t_9 激發後的每個 marking，共有三個可達的 ASD：M_{19}、M_{26} 和 M_{33}。另一方面，T 的 postcondition 是由 t_1 的 transition table 的右邊所描述的(也就是，[P_2(・)，P_4(・)])。由於三個可達的 ASD 都含有[P_2(・)，P_4(・)]，可知 T 的子方法的後置狀態描述比 T 的 postcondition 更為特定。因此，我們知道工作 Configure-Modules 的 postcondition 與它的子方法的後置狀態描述是一致的。

五、結論

利用高階派翠網路來驗證工作結構規格提供下列幾項優點，有助於建構及驗證知識庫系統：

- 我們的方法在需求規格層次上驗證知識庫系統，所以不只是在法則庫中的重複、矛盾、循環性等結構性錯誤可以被檢驗，傳統知識層次的驗證所無法找出的語意錯誤，包括靜態模型中的不一致性和動態行為的不一致性也可以被偵測出來。
- 整合高階派翠網路與工作結構來模組需求規格比單獨使用工作結構或高階派翠網路能獲得更豐富的語意。高階派翠網路的動態分析能力可以描述狀態轉換的語意和模擬系統的動態行為。另一方面，工作結構的工作分解機制可以用來組織和精煉需求規格，這是高階派翠網路難以獨自完成的。
- 不同觀點的需求規格 (例如狀態模型、功能模型、和行為模型) 可以使用單一的規格表示方式 (高階派翠網路) 來表達，並且這些不同的觀點可以在工作的概念下緊密地結合。此外，每個模型所產生的規格是可分享的，如：工作的狀態模型可以在行為模型中的 TSE 中被重複使用。

我們未來的研究將朝下列的方向進行：(1) 利用高階派翠網路的工作結構作為設計模式 (design pattern) 以重複使用 (reuse) 需求規格；(2) 擴充目前的架構，導入模糊邏輯方法以應用在不明確 (imprecise) 的軟體需求上。

參考文獻

[1] T. Bench-Capon, F. Coenen, H. Nwana, R. Paton, and M. Shave. Two aspects of the validation and verification of knowledge -based systems. *IEE Expert*, pages 76-81, Jun. 1993.

[2] B. Chandrasekaran, T.R. Johnson, and J.W. Smith. Task-structure analysis for knowledge modeling. *Communications of the ACM* , 35(9):124-137, Sep.1992.

[3] B.N. Freeman-Benson, J. Maloney, and A. Borning. An incremental constraint solver. *Communications of the ACM*, 33(1):54-63, January 1990.

[4] S.W. French and D. Hamilton. A comprehensive framework for knowle dge-base verification and validation. *International Journal of Intelligent Systems*, 9(9):809-837, 1994.

[5] C. Ghezzi, D. Mandrioli, S. Morasca, and M. Pezze. A unified high -level petri net formalism for time-critical systems. *IEEE Trans. on Software Engineering*, 17(2):160-171, Feb. 1991.

[6] P. Hubler, K. Jenson, and R.M. Sharpiro. Hierarchies in coloured petri nets. In G. Rozenberg, editor *Advances in Petri nets 1990* , pages 313 -341. Springer-Verlag, 1990.

[7] K. Jenson. Coloured petri nets: a high level language for system design and analysis. In G. Rozenberg, editor, *Advances in Petri nets 1990*, pages 342 -416. Springer-Verlag, 1990.

[8] V. Kumar. Algorithms for constraint -satifaction problems: A survey. *AI Magazine*, 13(1): -44, Spring 1992.

[9] L.F. Lai, J. Lee, and S.J. Yang. Fuzzy logic as a basis for reusing task-based specifications. *International Journal of Intelligent Systems*, 14:331-357,Apr. 1999.

[10] J. Lee. Task structures as a basis for modelin knowledg -based systems. *International Journal of Intelligent Systems*, 12:167-190, March 1997.

[11] J. Lee and L.F. Lai. Verifying task-based specifications in conceptual graphs. *Information and Software Technology*, 39: -923,Feb. 1998.

[12] J. Lee, L.F, Lai, and W.T. Huang. Task -based Specifications through conceptual graphs. *IEE Expert*, 11(4):60-70, Aug. 1996.

[13] S. Lee and R.M. O'Keefe. Developing a strategy for expert system verification and validation. *IEE Transactions on systems, Man, and cybernetics* , 24(4):643-655, Apr. 1994.

[14] T. Murata. Petri nets: properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541-579,Apr. 1989.

[15] D.L. Narazeth. Investigating the applicability of petri nets for rule -based system verification. *IEE Transactions on Knowledge and Data engineering* , 4(3):402-415, Jun.1993.

[16] J.R. Slagle, D.A. Gardiner and K. Han. Knowledge specification of expert system. *IEEE expert*, 5(4):29-38, Aug.1990.

[17] M.C. Tanner and A.M. Keuneke. The roles of the task structure and domain functional models. *IEE Expert*, 6(3): -57, June 1991.

[18] A.H.M. ter Hofstead and E.R. Nieuwia nd. Task structure semantics through process algebra. *Software Engineering Journal*, pages 14-20, Jan. 1993.

[19] W.T. Tsai, P.E. Johnson, J.R. Slage, I.A. Zualkernan, K.G. Heisler, K. Jamal, K. Han, D. Volovik, D.A. Gardiner, and T.L.A. Yang. Requirements specification for expert systems? A case study. Technical Report TR 88-44, University of Minnesota, Minneapolis, MN, 1988.

[20] C.H. Wu and S.J. Lee. Enhanced high -level peri nets with multiple colors for knowledge verification/validation of rule-based expert systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 27(5):760-773, Oct. 1997.