PLANNING AND ACTING IN THE REAL WORLD

11

現實世界的規劃與行動

本章中我們將看到，更有表達力的表示法和更具交互性的代理人結構，兩者是如何導致在現實世界中有用的規劃器。

前一章介紹了規劃中的最基本概念、表示和演算法。被使用於眞實世界中，負責太空船、工廠與軍事操演之作業的規劃與排程所用的規劃器更形複雜；它們同時擴充了表示語言與規劃器與環境互動的方式。本章將說明這是如何實現的。第 11.1 節進一步延伸用於規劃之經典語言來談論動作之持續時間與資源限制。第 11.2 節描述如何以階層組織的方式來構築規劃的方法。這得以使人類專家與規劃器能相互溝通他們所知道有關如何求解問題方面的事。階層也是有效率的規劃結構，因爲規劃器能夠於深入細節之前，在一個抽象的層次求解一個問題。第 11.3 節呈現代理人架構，它能夠應付不確定的環境並與執行情形交相審酌，並給予一些眞實世界系統的例子。第 11.4 節展示了當環境包含其他代理人時如何規劃。

11.1 時間、排程和資源

經典規劃表示談論該做什麼，以及依什麼次序，但是該表示不討論時間方面的事：一個動作需時多久以及它什麼時候會發生。舉例來說，第 10 章的規劃器能夠產生一個時程表，表中會說哪一架飛機預計分配到哪一個航次，但是我們實際上也需要知道起飛與抵達時間。這就是**排程**的主要課題。眞實世界也加諸許多的**資源限制**；舉例來說，一架飛機的機組人員數目是有限的——在某航次的機組人員不可能同一時間會出現在另一個航次。這一節內容涵蓋規劃問題加入時序與資源限制後的表示與求解方法。

這一節中我們採用的方法是「先規劃，後排程」：也就是說，我們將整個問題分成一個規劃階段和一個排程階段，規劃階段選擇行動並對其進行偏序排序以滿足問題的目標，之後是排程安排階段，時序資訊被添加到規劃中以確保滿足資源和最終期限的限制。這種方法在現實世界的製造業和後勤安排中是常見的，其中規劃階段通常由人類專家完成。第 10 章自動的方法也能夠用於規劃階段，只要它們所產生的規劃具有正確性所需的最小排序限制。GRAPHPLAN(第 10.3 節)，SATPLAN(第 10.4.1 節)，以及偏序規劃器(第 10.4.4 節)能夠做到這件事；搜尋型的方法(第 10.2 節)產生的是全序規劃，但是這些能夠很容易地被轉換爲最小排序限制的規劃。

11.1.1 表示時序與資源限制

一個典型的**加工車間排程問題**，首見於第 6.1.2 節，由一組**作業**所組成，每個作業由一群具有排序限制的**動作**所組成。每個動作有一個**持續時間**以及該動作所需的一組資源限制。每個限制敘明一個資源的類型(例如，螺栓，扳手，或飛行員)，資源所需之數目，以及資源是否屬於**可消耗的**(例如，螺栓不再可用)或是**可再用的**(例如，在某個航次的飛行員雖抽不了身，但是當該航次結束後，該飛行員又可以供差遣)。資源能夠也可以由負向消耗的動作所產生，包括製造、種植與重新供應等等動作。一個加工車間排程問題的解答必須敘明每個動作的開始時間，並且必須滿足所有的時序排序限制與資源限制。正如搜尋與規劃問題，解答能夠根據某個代價函數被計算出來；這可能會非常的複雜，具有非線性的資源成本，與時間有關的延遲成本，等等。爲求簡化，我們假設代價函數就是該規劃的全部持續時間，稱之爲**完工時間**(makespan)。

圖 11.1 中爲一我們已熟知的例子：一個有關兩輛車之組合的問題。該問題由兩項作業所組成，每項作業的形式是[*AddEngine*, *AddWheels*, *Inspect*]。然後 *Resources* 語句宣告了四種類型的資源，並賦予每個類型在開始的時候可用的資源數目：1 引擎吊繩，1 具車輪載台，2 個檢驗器，500 枚螺母。行動模式賦予每個動作所需的持續時間與資源。當輪子掛到車輛後，螺母會消耗掉，然而其他的資源在動作開始是「被借用的」而於動作結束後被歸還。

$$
\begin{aligned}
&Jobs(\{AddEngine1 \prec AddWheels1 \prec Inspect1\},\\
&\quad \{AddEngine2 \prec AddWheels2 \prec Inspect2\})\\
&Resources(EngineHoists(1), WheelStations(1), Inspectors(2), LugNuts(500))\\
&Action(AddEngine1, \text{DURATION:}30,\\
&\quad \text{USE:}EngineHoists(1))\\
&Action(AddEngine2, \text{DURATION:}60,\\
&\quad \text{USE:}EngineHoists(1))\\
&Action(AddWheels1, \text{DURATION:}30,\\
&\quad \text{CONSUME:}LugNuts(20), \text{USE:}WheelStations(1))\\
&Action(AddWheels2, \text{DURATION:}15,\\
&\quad \text{CONSUME:}LugNuts(20), \text{USE:}WheelStations(1))\\
&Action(Inspect_i, \text{DURATION:}10,\\
&\quad \text{USE:}Inspectors(1))
\end{aligned}
$$

圖 11.1　資源限制下、裝配兩輛汽車的加工車間排程問題。符號 $A \succ B$ 意指動作 A 必須先於動作 B

資源表示爲數值量，如 *Inspectors*(2)，而不是作爲名稱實體，諸如 *Inspectors*(I_1)和 *Inspectors*(I_2)，這是一種稱爲**聚集**的非常通用技術的一個例子。聚集的核心想法是當物件從其即刻效用的角度看完全不可區分時，把個體的物件聚合成數量。在我們的裝配問題中，哪名檢查員檢查汽車是無關緊要的，所以不需要進行區分(同樣的想法在習題 3.9 之傳教士和野人問題中一樣有用)。聚集對降低複雜度很重要。考慮當假設一個排程表有 10 個同時發生的 *Inspect*(檢查)行動，但是只有 9 名檢查員可用的情況下會發生什麼。把檢查員表示爲數量，失敗馬上被檢測出來，演算法回溯轉而嘗試另一個排程表。在把檢查員表示爲個體下，演算法回溯嘗試對行動的所有 10!個分配檢查員的方法。

11.1.2 求解排程問題

我們先考慮只有時序排程的問題，不考慮資源限制。欲最小化完工時間(規劃的持續時間)，我們必須尋找出所有動作的最早開始時間並須滿足問題所提出的排序限制。將這些排序限制視爲一個與動作有關聯之有向圖是很有幫助的，如圖 11.2 所示。我們能夠對於這個圖形運用**要徑法**(CPM)來決定每個動作可能的開始與結束時間。經過偏序規劃的一條**路徑**是一個從 *Start* 開始到 *Finish* 結束的行動的線性有序序列(例如，在圖 11.2 的偏序規劃中有兩條路徑)。

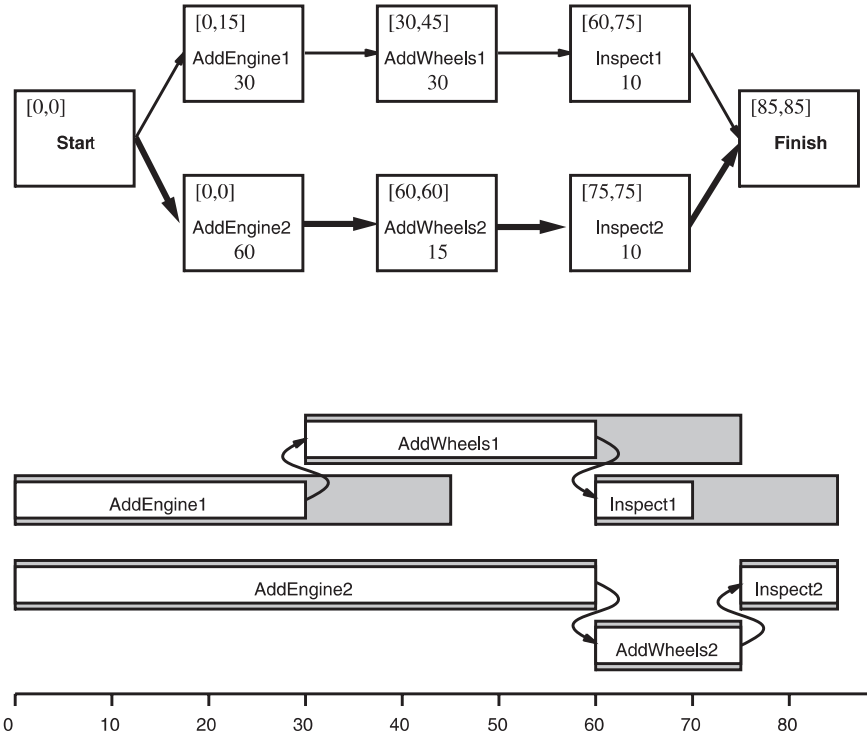

圖 11.2 **上半部：圖 11.1 的加工車間排程問題的時序限制表示。每個動作的持續時間已標明在每個方塊的底部。求解該問題時，我們計算最早的與最晚的開始時間為[ES, LS]對，其顯示於左上角。這兩個數字間的差值是行動的鬆弛；具有零鬆弛的行動是處於關鍵路徑上，用粗箭頭表示**

下半部：以一條時間線來展示的相同解答。灰色矩形表示時間區間，一個行動可以在這區間內執行，倘若遵守定序限制的話。灰色矩形中沒有被佔用的部分表示鬆弛

要徑(關鍵路徑)是總持續時間最長的路徑。路徑是「關鍵」的，這是因爲它確定了整個規劃的持續時間——縮短其他規劃不會縮短整個規劃，但是延遲關鍵路徑上任何行動的開始將使整個規劃慢下來。關鍵路徑以外的行動有一些迴旋餘地——它們可以在一個時間視窗內執行。這個視窗根據最早可能開始時間 *ES* 和最遲可能開始時間 *LS* 來指定。數量 *LS* – *ES* 被稱爲行動的**鬆弛**。於圖 11.2 中我們看到整個規劃費時 85 分鐘，位在上層的每個作業的動作有 15 分鐘的鬆弛，位在要徑上的每個動作並沒有任何鬆弛時間(依定義)。所有行動的 *ES* 時間和 *LS* 時間一起構成了問題的一個**排程表**。

下面的公式可以作爲 *ES* 和 *LS* 的定義以及描述計算它們的動態規劃演算法的輪廓：*A* 與 *B* 是動作，而 $A \prec B$ 意指 *A* 發生在 *B* 之前：

$$ES(Start) = 0$$
$$ES(B) = \max_{A \prec B} ES(A) + Duration(A)$$

$$LS(Finish) = ES(Finish)$$
$$LS(A) = \min_{A \prec B} LS(B) - Duration(A)$$

思維是我們從把 *ES*(*Start*)指派值爲 0 開始。然後一旦我們得到一個行動 *B*，所有直接出現在 *B* 之前的行動的 *ES* 都已經指派過值，我們就可以設 *ES*(*B*)爲那些直接前繼行動的最早完成時間的最大值，其中一個行動的最早完成時間定義爲最早開始時間加上持續時間。這個過程重複進行直到每個行動被指派予一個 *ES* 值。*LS* 值用類似的方式計算，從 *Finish* 行動逆向進行。

關鍵路徑演算法的複雜度僅僅是 $O(Nb)$，其中 N 是行動的個數，b 是進入或離開行動的最大分支因數。(爲了瞭解這點，注意對每個行動的 *LS* 和 *ES* 只計算一次，每個計算最多疊代於 b 個其他行動)。因此，要尋找出最小持續時間的排程，在偏序已知並且沒有資源限制的情況下，是非常的容易。

就數學上，要徑問題不難求解，因它們被定義成一個開始與結束時間之線性不等式的連言。當引進資源限制後，加諸於開始與結束時間的限制變得更複雜。舉例來說，*AddEngine* 動作，於圖 11.2 中在同一個時間開始，需要相同的 *EngineHoist* 以至於無法重疊。這個「無法重疊」的限制是兩個線性不等式的選言，各是針對每個可能的排序。引進選言造成具有資源限制的排程成爲 NP-hard。

圖 11.3 顯示了具有最快完成時間的解，115 分鐘。這比沒有資源限制的排程表需要的 85 分鐘長。注意，沒有同時需要兩名檢查員的時刻，所以我們能夠立即將兩名檢查員中的一名轉移到更有生產價值的位置。

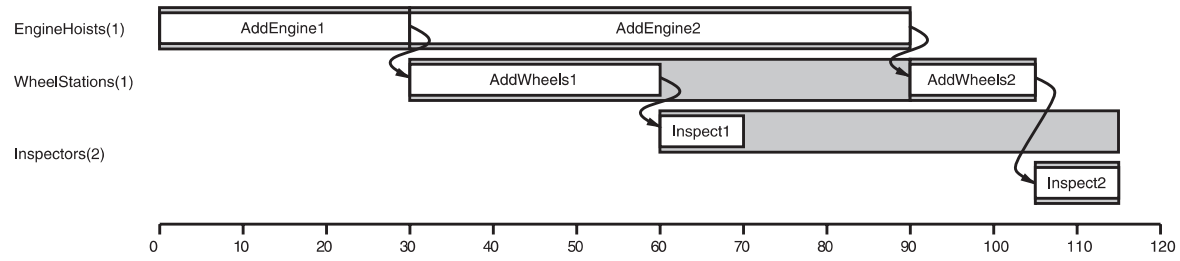

圖 11.3　圖 11.1 之加工車間排程問題的一個解答，資源限制有列入考慮。左邊列出了 3 種可重複使用資源，行動與其使用的資源則水平對齊地顯示。有兩個可能的排程表，取決於哪個裝配先使用發動機起重機；我們顯示了最短時間解，其需要花費 115 分鐘

資源限制下的排程複雜度在實務中是與理論中一樣常見。1963 年提出的一個具有挑戰性的問題——爲一個正好包含 10 台機器和每項工作有 100 個行動的 10 項工作的問題尋找最佳排程表——在 23 年內一直沒有得到解決(Lawler 等人，1993)。已經有許多方法被嘗試過，包括分支界限、類比退火、禁止搜尋、限制滿足及第二部分的其他技術。一個簡單而流行的啓發式是最小鬆弛演算法。在每個疊代過程，任何以未排程好的動作開始之最早可能時程會將所有它的動作排程好並且有**最少鬆弛**；然後更新每個受到影響到之動作的 ES 與 LS 時間並再重複一次。該啓發式頗類似於限制條件滿足性問題的 minimum-remaining-values(MRV)啓發式。在實際應用中它通常工作得很好，但是對於我們的裝配問題它產生了一個 130 分鐘的解，而不是圖 11.3 中的 115 分鐘的解。

到目前這個地步，我們已經假設動作集與排序限制都是固定不變的。在這些假設之下，每一個排程問題能夠經由避免所有資源衝突之非重疊序列來解出，只要各個動作本身是可行的。不過，如果一個排程問題已證明是非常困難的，以這樣的方式求解它或許不是好的構想——假若會引導至一個更簡單的排程問題的話，最好重新考慮動作與限制。如果是那樣的話，透過在建構偏序規劃期間考慮持續時間和重疊，而將規劃和排程安排整合起來是有意義的。第 10 章中的一些規劃演算法可以被擴充以處理這種資訊。例如，透過與用因果連接檢測衝突非常相似的方式，偏序規劃器能夠檢測對資源限制的破壞。啓發式能夠被用來估計一個規劃的全部完成時間。這是當前很活躍的一個研究領域。

11.2 分層規劃

前面幾章之問題求解與規劃方法都是在固定的一組原子動作下運作。動作能夠被串成序列或是具有分枝的網路；最先進的演算法能夠產生含有數以千計的動作的解答。

由人腦執行的規劃，原子動作是肌肉致動的。粗略的估算，我們有大約 10^3 肌肉可抽動(有些人算的是 639，但是其中許多有數個子組織)；我們能以約每秒 10 次的頻率來控制它們的抽動；我們活著且醒著的時間大約有 10^9 秒。因此，一個人終其生包含大約 10^{13} 個動作，增或減兩個量級。即使我們限制我們規劃於短促得許多的時間範圍——舉例來說，夏威夷度假兩週——一個詳細的行動規劃包含約 10^{10} 動作。這比 1000 多上許多。

要跨越這個鴻溝，AI 系統必須做出人類顯然會做的事情：在更高的抽象層次做規劃。一個合理的夏威夷假期規劃可能是「前往舊金山機場；搭乘夏威夷航空公司的第 11 次航班去檀香山；享受兩週假期；搭乘夏威夷航空公司第 12 次航班回舊金山；回家。」給予這樣的規劃，「前往舊金山飛機場」這個動作本身就能夠被看成是一個規劃任務，其解答像是「駕車去長期停車場；停車；乘機場巴士去航空站。」這其中的每一個動作，反過來，能夠被更進一步分解，直到我們達到讓動作能夠被不假思索地執行來製造出所需的行動控制序列的層次。

於這例子，我們看到規劃能夠出現於規劃執行之前及之間；舉例來說，一個人或許會暫時擱置從長期停車場停車位置到機場巴士站之間路線的規劃問題，直到執行期間發現一個特別的停車場。因此，該特別動作在進入執行階段之前仍然是處於抽象層次。我們推遲這個課題的討論到第 11.3 節。這裡，我們致力於**階層分解**方面，所有觸及管理複雜性的概念都會論及此。複雜軟體從副程式或物件類的層次體系建立出來，軍隊作爲單位的等級體系而運轉，政府和企業有部門、子部門和分支辦公室的分層體系。層次結構的關鍵好處是，在層次的每一層上，一個計算任務、軍事任務或管理功能都被還原爲下一個較低層次的少量行動，所以對當前問題尋找正確的方法來安排這些行動的計算消耗是小的。在另一方面，非層次方法將一個任務還原爲大量單個行動；對大規模問題，這是完全不切實際的。

11.2.1 高層次動作

我們採用來了解階層分解的基本方法論，是來自於**分層任務網路**(hierarchical task networks)或稱HTN 規劃的領域。一如於經典規劃(第 10 章)，我們假設有一組具有全部的可見性、確定性與可用性的動作，現在稱之爲原始動作，具有標準的前提-效果模式。另外的關鍵概念是**高層次動作**或 HLA——舉例來說，於先前提到的例子，「前往舊金山飛機場」的動作。每個 HLA 有一個以上可能的**強化調整**，爲一序列的動作[1]，每個調整可能是個 HLA 或是一個原始動作(依定義它不會有任何調整)。舉例來說，「前往舊金山飛機場」的動作，以 *Go*(*Home*, *SFO*)正式地來表示，可能有兩個可能的調整，如圖 11.4 所示。同一張圖展示航行在眞空吸塵器世界的**遞迴**調整：欲到達目的地，採取一個步驟，然後前往該目的地。

$Refinement(Go(Home, SFO),$
  STEPS: $[Drive(Home, SFOLongTermParking),$
    $Shuttle(SFOLongTermParking, SFO)]$)
$Refinement(Go(Home, SFO),$
  STEPS: $[Taxi(Home, SFO)]$)

$Refinement(Navigate([a, b], [x, y]),$
  PRECOND: $a = x \ \wedge \ b = y$
  STEPS: [])
$Refinement(Navigate([a, b], [x, y]),$
  PRECOND: $Connected([a, b], [a - 1, b])$
  STEPS: $[Left, Navigate([a - 1, b], [x, y])]$)
$Refinement(Navigate([a, b], [x, y]),$
  PRECOND: $Connected([a, b], [a + 1, b])$
  STEPS: $[Right, Navigate([a + 1, b], [x, y])]$)
...

圖 11.4 兩個高層次動作所用之可能調整的定義：前往舊金山機場與在真空吸塵器世界駕駛。在後面的例子，請注意該調整的遞迴本質與前提的使用

這些例子展示高層次動作與它們的調整使如何做事情的知識具體化了。例如，*Go*(*Home*, *SFO*)之調整指出，你能夠開車或坐計程車到達機場；買牛奶，坐下，及移動西洋棋的騎士至 e4 位置等等都不需要列入考慮。

一個只包含原始動作的 HLA 調整稱之爲該 HLA 的一個**實作**。舉例來說，於眞空吸塵器世界，序列[*Right*, *Right*, *Down*]與[*Down*, *Right*, *Right*]同時採用 HLA *Navigate*([1, 3], [3, 2])。一個高層次規劃(一群 HLA 序列)是依序實作該序列中的各個 *HLA*。在每個原始動作之前提-效果都已定義的情況下，要判斷任何一個高層次規劃之實作是否能達成目標是很直截了當的事。如果從一個給定的狀態，高層次規劃的實作中至少有一個會達成目標，那麼，我們就能夠說，這個高層次規劃可以從一個給定的狀態達成目標。於這個定義中，「至少一個」這個字眼是非常關鍵的——並非所有的實作都必須達成目標，因爲代理人會判斷它將執行哪一個實作。因此，於 HTN 規劃中可能的實作之集合——集合中的每個實作可能會有不同的結果——是不同於不確定性規劃之可能結果的集合。那裡，我們所需的是一個適用於所有結果的規劃，因爲代理人不用想該如何選出結果；老天會。

最簡單的情況是 HLA 就只有一個實作。那樣的話，我們能夠從該實作的前提與效果來計算 HLA 的前提與效果(見習題 11.2)然後將 HLA 本身視如一個原始動作。可以證明使用一組正確的 HLA，盲目搜尋的時間複雜性能夠自解答深度的指數量級跌至解答深度的線性量級，儘管設計這樣的一組 HLA 本身不見得是個輕鬆的工作。當 HLA 同時有數個可能的實作時，可有兩個選擇：選擇之一是，從實作之中找出一個能用的，一如於第 11.2.2 節者；另一個選擇是，直接對 HLA 做推理——儘管實作的多重性——如第 11.2.3 節所解釋者。後面這個方法會讓推導可證明正確抽象規劃一事，不需要考慮它們的實作。

11.2.2 搜尋原始解答

HTN 規劃常常用一個稱之爲 *Act* 的單一「上層」動作來公式化，這麼做的目標是尋找一個達成目標的 *Act* 實作。這是個全然地一般性的方法。舉例來說，經典規劃問題能夠被定義如下：對每個原始動作 a_i，以步驟[a_i，*Act*]提供一個 *Act* 的調整。這樣創造了一個讓我們增加動作的 *Act* 的遞迴定義。但是我們需要一些中斷遞迴的方法；我們的辦法是對 *Act* 提供一個額外的調整，它具有空步驟表且其前提等於問題之目標。這是說如果目標已經達成，那麼那個對的實作什麼都不會做。

該方法引申出一個簡單的演算法：於目前的規劃中重複地選取一個 HLA 並以它的調整取代它，直到該規劃達成目標。一個建立於廣度優先樹搜尋方法的可能實作展示於圖 11.5。規劃是依調整之巢狀深度的順序來考慮，而非依原始步驟之數目。設計一個演算法的圖形搜尋版本以及深度優先與疊代加深版本是非常簡單的。

```
function HIERARCHICAL-SEARCH(problem, hierarchy) returns a solution, or failure
  frontier ← a FIFO queue with [Act] as the only element
  loop do
    if EMPTY?(frontier) then return failure
    plan ← POP(frontier)  /* chooses the shallowest plan in frontier */
    hla ← the first HLA in plan, or null if none
    prefix,suffix ← the action subsequences before and after hla in plan
    outcome ← RESULT(problem.INITIAL-STATE, prefix)
    if hla is null then  /* so plan is primitive and outcome is its result */
      if outcome satisfies problem.GOAL then return plan
    else for each sequence in REFINEMENTS(hla, outcome, hierarchy) do
      frontier ← INSERT(APPEND(prefix, sequence, suffix), frontier)
```

圖 11.5 分層式前向規劃搜尋的一個廣度優先實作。提供予演算法之初始規劃是[*Act*]。REFINEMENTS **函數傳回一組行動序列給** HLA **的各個調整，**HLA **其前提被指定狀態** outcome **所滿足。**

從本質上講，這種形式的分層搜尋探索了與 HLA 庫中關於事情如何完成的知識相符的空間序列。大量的知識能夠被編碼進去，而不侷限那些載明於每個調整之動作序列，也包括用於該調整的前提。對於某些領域，HTN 規劃器已經能夠以非常少的搜尋製造出龐大的規劃。例如，O-規劃(Bell 和 Tate，1995)把 HTN 規劃和排程安排結合起來，它已經被用於爲日立公司開發產品規劃。一個典型的問題涉及 350 種不同產品的生產線，35 台裝配機器和超過 2000 個不同操作。規劃器產生一個每天 3 班 8 小時輪換的 30 日排程表，包括上百萬的步驟。HTN 規劃另一個重要的特徵是，根據定義，它們是分層地架構出來的；這使得它們更容易被我們所了解。

藉由審視一個理想化的情況，能看出分層搜尋在計算上的優點。。假設一個規劃問題有一個具有 d 個原始動作的解答。對於一個非分層的，每個狀態具有 b 個允許行動的前向狀態空間規劃器，成本是 $O(b^n)$，如第 3 章所解釋。對 HTN 規劃器，讓我們假設一個非常規則的分解結構：每個非原始行動有 d 種可能的分解，每種分解都在下一個低層把行動分爲 k 個行動。我們希望知道在這架構下有多少不同的調整樹。如果在原始層有 n 個行動，那麼根以下的層數爲是 $\log_k n$，所以內部分解節點數爲 $1+k+k^2+...+k^{\log_k d}-1=(d-1)/(k-1)$。每個內部節點有 r 個可能的調整，所以能夠建構出 $r^{(d-1)/(k-1)}$ 棵正常的分支樹。檢查這個公式，我們看到保持 d 小、k 大能夠導致巨大節省：本質上，如果 b 和 d 是可比較的，我們在採用非分層的第 k 個根的成本。小的 r 與大的 k 意指一個調整數量不多的 HLA 庫，每個調整都會得到很長的動作序列(儘管如此可以讓我們解出任何問題)。這不一定都是可能的：可以運用於廣泛問題的長動作序列是極其珍貴的。

那麼 HTN 規劃的關鍵是構造一個包含實現複雜的高層行動的已知方法的規劃庫。構造規劃庫的一種方法是學習來自問題求解經驗的方法。在從零做起構造一個規劃的痛苦經歷之後，代理人能夠將規劃保存在規劃庫中，作爲實現任務定義的高層行動的一種方法。用這種方式，代理人隨著時間會變得越來越有能力，而新方法建立在舊方法之上。這個學習過程的一個重要方面是對被構造出的方法進行一般化的能力，消除問題實例特定的細節(例如，施工人員的姓名和地皮所在的位址)，只保留規劃的關鍵元素。如何獲得這種一般化的方法將在第十九章中描述。人類能夠在沒有這些機制的情況與它們一樣有能力，在我們看來這似乎是不可思議的。

11.2.3 搜尋抽象解答

於前面一節的分層式搜尋演算法中全面的調整 HLA 成原始動作序列來判斷某個規劃是否派得上用場。這與常識有違：你應該能夠決定出兩層-HLA 高層次規劃

[*Drive*(*Home*, *SFOLongTermParking*), *Shuttle*(*SFOLongTermParking*, *SFO*)]

將某人送到機場而不必決定一條精確路線、選擇停車地點等等。解答似乎是顯而易見的：寫出 HLA 的前提-效果描述，一如我們寫下該原始動作做什麼。從該描述，應該很容易證明該高層次規劃達成了目標。可以這麼說，這是分層規劃的聖杯，因爲如果我們推導出一個高層次規劃，經過證明於高層次動作的小範圍搜尋空間中是可達成該目標的，那麼我們能夠承諾該規劃，且著力於調整該規劃的個步驟的問題。這給了我們所渴求的指數量級的縮減量。要使這個能發生作用，就必須是達成目標的每一個「鏈接」(憑藉它的步驟描述)的高層次規劃事實上是以符合稍早前定義的意義來達成目標的情況：它必須至少有一個實作確能達成目標。這個性質被稱爲 HLA 描述的**向下強化調整性質** (downward refinement property)。

寫出滿足向下調整性質的 HLA 描述，原則上，是容易的：只要描述爲眞，則任何聲稱可以達成目標之高層次規劃必須事實上能做到該件事——否則，該描述是製造一些有關 HLA 所做之事的不實聲明。我們已經看過如何爲恰有一個實作的 HLA 寫出眞描述(習題 11.2)；當 HLA 有數個實作，就會出現問題。我們如何能夠描述一個能夠以許多不同的方式實作之動作的效果呢？

一個安全的答案是(至少對所有前提與目標都是正的問題而言)是只要納入已被 HLA 的每個實作達成之正效果以及任一實作的負效果。那麼向下調整性質就會被滿足。不幸的是，這對 HLA 而言是過於保守的語意。再一次考慮 HLA *Go*(*Home*, *SFO*)，它有兩個調整，爲了論證起見，假設爲一個簡單的世界，於此你一定能夠開車到機場並停車，但是搭乘計程車時則需要 *Cash* 做爲前提。那樣的話，*Go*(*Home*, *SFO*)不見得一定會送你到機場。尤其，如果 *Cash* 是僞造的時候它會失敗，所以我們無法斷言 *At*(*Agent*, *SFO*)爲 HLA 的效果。這沒有意義，不過；如果代理人沒有 *Cash*，它將要自行開車。要求某個效果對每一個實作都會成立等同於假設其他人——對手——將會選取該實作。它對待 HLA 的數個結果有如 *HLA* 是個**不確定性**的動作，一如於第 4.3 節。對於我們的情況來說，代理人本身會選取該實作。

程式設計語言社群稱對手作出選擇的情況爲**惡魔般的不確定性**，相對之下，代理人本身作出選擇的情況，則稱之爲**天使般的不確定性**。我們借用這名詞來定義 HLA 描述的天使般語意。了解天使般語意所需的基本的概念是 HLA 的**可到達集**(reachable set)：給定一個狀態 s，一個 HLA h 的可到達集，寫成 REACH(s, h)，是任何 HLA 的實作都可達到之狀態的集合。關鍵之處在於代理人能夠選取它執行 HLA 時所達的可到達集個元素；因此，一個具有數個調整的 HLA 較諸相同但具有較少調整的 HLA 更「有力」。我們也能夠定義一個 HLA 序列的可到達集。舉例來說，序列[h_1, h_2]的可到達集是施加 h_2 於 h_1 之可到達集中的每個狀態所得到搭所有可到達集的聯集：

$$\text{REACH}(s,[h_1,h_2]) = \bigcup_{s'\in\text{REACH}(s,h1)} \text{REACH}(s',h_2)$$

由這些定義，一個高層次的規劃——一個 HLA 的序列——達成目標，如果它的可到達集與目標狀態集相交(比較這個與條件強烈許多的惡魔語意，其中可到達集的每個成員必須都是個目標狀態)。相反地，如果可到達集與目標沒有交集，那麼該規劃肯定沒有任何作用。圖 11.6 展示了這些概念。

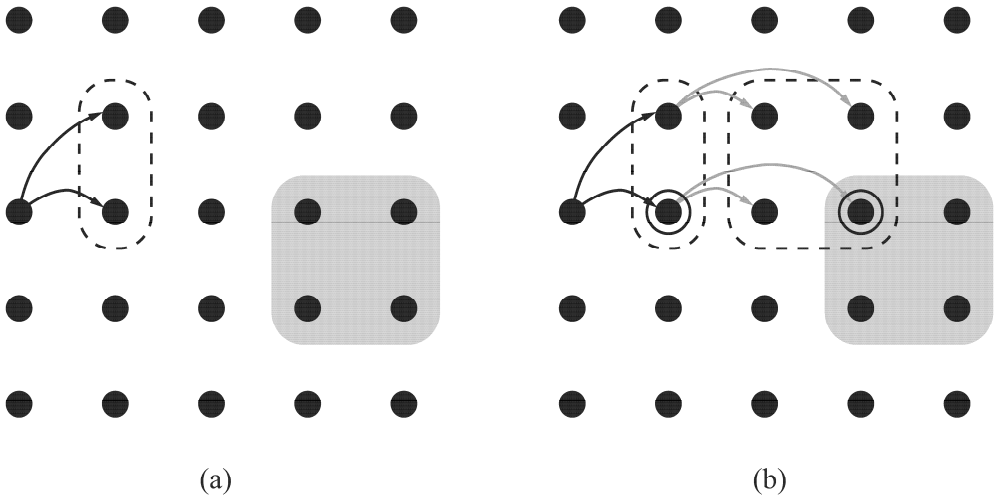

圖 11.6　可到達集的圖解例子。目標狀態集為灰影。黑色與灰色箭頭分別指出 h_1 與 h_2 的可能實作。(a) 於狀態 s 中的 HLA h_1 的可到達集。(b) 序列[h_1, h_2]的可到達集。因為這與目標集相交，序列達成該目標

可到達集的概念引出了直截了當的演算法：於高層次規劃之間搜尋，尋找哪一個規劃的可到達集與目標有交集；一旦出現這個現象，演算法便能夠對該抽象規劃做出承諾，知道它是有作用的，並進一步專注於調整該規劃。我們稍後將會回到演算法的議題；首先，我們考慮一個 HLA 之效果——每個可能初始狀態之可到達集——如何表示的問題。正如第 10 章的經典行動模式，我們表示出對於每個流所作的變更。想像一個流爲狀態變數。一個原始動作能夠加入一個變數或刪除一個變數或不作任何更動。[在條件效果(見 11.3.1 節)之下，會有四個可能性：倒反變數爲它的相反值]。

一個在天使語意之下的 HLA 能夠做的更多：它能夠控制一個變數的值，根據選取了哪一個實作來設定變數爲眞或爲假。事實上，一個 HLA 能夠有九個不同的效果作用於一個變數：如果該變數開始時爲眞，它能夠一直使之爲眞，一直使之爲假，或是有所選擇；如果該變數開始時爲假，它能夠一直使之爲假，一直使之爲眞，或是有所選擇；各個情況的三個選擇可以任意搭配，共作出九種。就符號性而言，這是有些挑戰性。我們將使用符號 ~ 來表示「可能地，如果代理人如此選擇的話。」因此，一個效果 $\tilde{+}A$ 意指「可能加入 A」，也就是說，要麼不更動 A 不然就是使之爲眞。類似地，$\tilde{-}A$ 意指「可能刪除 A」而 $\tilde{\pm}A$ 意指「可能地加入或刪除 A。」舉例來說，HLA $Go(Home, SFO)$，具有如圖 11.4 所示的兩個調整，可能刪除 $Cash$(如果代理人決定搭計程車)，所以它應該有效果 $\tilde{-}Cash$ 。因此，我們看到 HLA 的描述是可以，原則上，從它們調整的描述來推導的——事實上，如果我們要眞正的 HLA 描述，使得向下調整性質成立的話，這是必要的。現在，假設我們對於 HLAs h_1 與 h_2 有下述模式：

$$Action(h_1, \text{PRECOND: } \neg A, \text{EFFECT: } A \wedge \tilde{-} B)$$
$$Action(h_2, \text{PRECOND: } \neg B, \text{EFFECT: } \tilde{+} A \wedge \tilde{\pm} C)$$

也就是說，h_1 加入 A 且可能刪除 B，而 h_2 可能加入 A 且能完全的控制 C。現在，如果於初始狀態只有 B 爲眞且目標是 $A \wedge C$ 那麼序列$[h_1, h_2]$達成該目標：我們選取一個使得 B 爲假的 h_1 實作，那麼選取一個 h_2 實作，將 A 置於眞且使 C 爲眞。

前面的討論假設一個 HLA 的效果——任一個給定之初始狀態的可到達集——能夠藉由描述出作用於每個變數的效果而被完整無誤的描述出來。如果這恆爲眞當然是很好的事，但是在許多情況我們僅能夠近似該效果，因爲一個 HLA 可能會有無限多個實作而且可能產生任意地扭曲可到達集——非常雷同於圖 7.21 示範說明過的 *wiggly*-信度狀態問題。舉例來說，我們說 $Go(Home, SFO)$可能刪除 $Cash$；也可能加入 $At(Car, SFOLongTermParking)$；但是它沒無法同時做這兩件事——事實上，它必須擇一來做。正如信度狀態，我們可能必須寫出近似的描述。我們將使用兩個種近似方式：一個 HLA h 的**樂觀描述** $\text{REACH}^+(s, h)$可能誇大了可到達集，然而**悲觀描述** $\text{REACH}^-(s, h)$可能低估了可到達集。因此，我們有：

$$\text{REACH}^-(s, h) \subseteq \text{REACH}(s, h) \subseteq \text{REACH}^+(s, h)$$

例如，$Go(Home, SFO)$的樂觀描述說它可能刪除 $Cash$ 且可能加入 $At(Car, SFOLongTermParking)$。另一個好的例子源於八方塊遊戲，該遊戲的狀態半數是無法從任何已知狀態達到的(見習題 3.5)：Act 的樂觀描述可能相當的涵蓋了整個狀態空間，因爲完整無誤的可到達集是非常的扭曲。

透過近似性的描述，一個規劃是否達成目標的測試必須稍微地予以修改。如果爲該規劃之樂觀可到達集與目標並沒有交集，那麼該規劃並沒有用；如果悲觀可到達集與目標有交集，那麼該規劃就有用(圖 11.7a)。對於確切的描述，一個規劃要麼有用要麼沒有用，但是對於樂觀描述，會有一個灰色地帶：如果樂觀的集合與目標有交集，但是悲觀的集合並沒有交集，那麼我們無法辨別出該規劃是否有用(圖 11.7b)。當這情況出現，此不確定性能夠經由調整該規劃而泯除。這在人類做推理實是個非常常見的情景。舉例來說，於規劃前述的兩週夏威夷假期，你可能會提議在七座島的每一座島上各度假兩天。審愼的人會指出這個富有野心的規劃必須予以修改而加入島際間的交通往返細節。

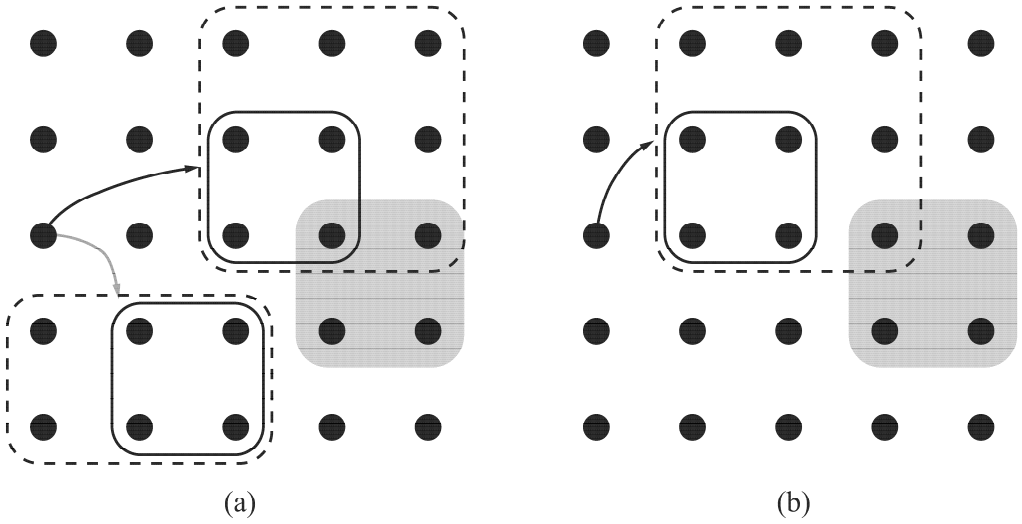

圖 11.7　搭配近似描述的高層次規劃之目標達成。目標狀態集被畫上斜線。對於每個規劃，樂觀的(實線)與悲觀的(虛線)可到達集都顯示出來。(a) 黑色箭頭所指的規劃一定會達成目標，而灰色箭頭所指的規劃則一定不會。(b) 一個必須被更進一步調整的規劃，以判斷它是否真能達成目標

一個分層規劃所用之演算法採用近似天使描述示如圖 11.8。爲求簡化，我們已經保留了先前使用於圖 11.5 的同一個整體模式，也就是說，於調整空間的廣度優先搜尋。如同剛剛解釋過的，該演算法能夠藉由檢查樂觀與悲觀可到達集與目標之交集來探知該規劃會不會有用(給定每個步驟的近似描述時，如何計算某個規劃之可到達集的細節詳述於習題 11.4)。當一個可用的抽象規劃被找出時，該演算法分解原始的問題爲子問題，該規劃的每個步驟都如是。每個子問題之初始狀態與目標可以經由規劃之各個步驟的行動模式來迴歸一個保證可達目標狀態而得到(請見第 10.2.2 節有關迴歸是如何發揮功用的討論)。圖 11.6(b)示範說明了基本的概念：右手邊被圈出的狀態是保證可達目標狀態，而左手邊被圈出的狀態是中間目標，是經由最後的動作回歸目標所得到的。

具有承諾或是拒絕高層次規劃的能力能夠給予 ANGELIC-SEARCH 較諸 HIERARCHICAL-SEARCH 有可觀的計算優勢，順帶的也比平淡古老的 BREADTH-FIRST-SEARCH 更具有鉅大的優勢。舉例來說，考慮清理一個由狹窄走廊圍繞出的長方形房間所組成的眞空吸塵器世界。安排一個 HLA 予 *Navigate*(如圖 11.4 所示)以及一個 HLA 予 *CleanWholeRoom* 是有意義的(重複運用清理每行之 HLA 得以使清理房間被實作出來)。因爲於這領域中有五個動作，

BREADTH-FIRST-SEARCH 所需的成本增加約 5^d 倍，其中 d 是最短解答的長度(約略等於全部方格數的兩倍)；該演算法即使是兩個 2×2 房間也無法處理。HIERARCHICAL-SEARCH 是更有效率的，但是仍然受到以指數量級增加之苦，因爲它會儘可能的清理與該分層一致者。ANGELIC-SEARCH 的大小約與方格之數目呈線性關係——它承諾好的高層次序列並刪去其它選擇。請注意以一個接一個清理每個房間的方式來清理一群房間並非難如登天：對人類來這樣的工作說是容易的，因爲任務是分層架構的。當我們想到人們發現要解出如八方塊遊戲這樣的小謎題是如何的困難的時候，人類解決複雜問題的能力似乎很大的程度上得自於他們抽象與分解問題以消除不同組合的技巧。

```
function ANGELIC-SEARCH(problem, hierarchy, initialPlan) returns solution or fail
  frontier ← a FIFO queue with initialPlan as the only element
  loop do
    if EMPTY?(frontier) then return fail
    plan ← POP(frontier)  /* chooses the shallowest node in frontier */
    if REACH+(problem.INITIAL-STATE, plan) intersects problem.GOAL then
      if plan is primitive then return plan  /* REACH+ is exact for primitive plans */
      guaranteed ← REACH−(problem.INITIAL-STATE, plan) ∩ problem.GOAL
      if guaranteed≠{ } and MAKING-PROGRESS(plan, initialPlan) then
        finalState ← any element of guaranteed
        return DECOMPOSE(hierarchy, problem.INITIAL-STATE, plan, finalState)
      hla ← some HLA in plan
      prefix,suffix ← the action subsequences before and after hla in plan
      for each sequence in REFINEMENTS(hla, outcome, hierarchy) do
        frontier ← INSERT(APPEND(prefix, sequence, suffix), frontier)

function DECOMPOSE(hierarchy, s0, plan, sf) returns a solution
  solution ← an empty plan
  while plan is not empty do
    action ← REMOVE-LAST(plan)
    si ← a state in REACH−(s0, plan) such that sf ∈ REACH−(si, action)
    problem ← a problem with INITIAL-STATE = si and GOAL = sf
    solution ← APPEND(ANGELIC-SEARCH(problem, hierarchy, action), solution)
    sf ← si
  return solution
```

圖 11.8　一個分層規劃演算法，它使用了天使語意以辨識與承諾有用的高層次規劃而避開那些無用的高層次規劃。述詞 MAKING-PROGRESS 檢查以確認我們不會陷入無限次重新調整的處境。在頂層，以[Act]呼叫 ANGELIC-SEARCH 當作初始規劃

天使方法能夠藉由一般化可到達集而延伸至尋找代價最小的答案。與其說狀態可以達到與否，以最有效率的方式到該狀態時，它會有一個代價。(對於無法達到的狀態，該代價是∞)。樂觀的與悲觀的描述圈出了這些代價的上下範圍。於這個方式，天使搜尋能夠尋找可證明最佳抽象規劃而毋需考慮它們的實作。相同的方法能夠用於獲取有效的**分層前瞻**演算法(hierarchical lookahead algorithms)，以 LRTA*的型式(4.5.3 節)，供線上搜尋之用。在某些方面，這樣的演算法反射了人類對於任務深思熟慮的一面，例如規劃一個到夏威夷的假期——替代的想法剛開始時是經過很長的時

間在一個抽象層次上完成；規劃的某些部份會駐留在相當抽象的狀態直到執行的時候，例如如何在莫洛凱島渡過兩個慵懶的日子，然而其他的部份則規劃的很詳細，例如打算搭乘的班次以及預約好的旅店——沒有這些調整，並不保證該規劃會是可行的。

11.3 在非確定性領域中進行規劃和行動

於這第一節我們延伸規劃至處理那些局部可觀察、非確定以及未知的環境。第 4 章曾用類似的方式延伸了搜尋，而這兒所用的方法也雷同：**無感測器的規劃**(也被稱爲**一致性規劃**)適用於不做觀察的環境；**應變規劃**適用於局部可觀察的且不確定性的環境；而**線上規劃**與**重規劃**適用於未知的環境。

雖然基本的概念與第 4 章者相同，也有顯著的不同之處。這些會發生是因爲規劃器處理分解過的表示而非原子的表示。這影響我們用來表示代理人動作與觀察之能力的方式，以及我們對無法觀察與局部可觀察之環境**信度狀態**——代理人可能置身之實體狀態的集合——的表示方式。我們也能夠運用第 10 章中得到的許多領域無關方法的優點來計算搜尋啓發式。

考慮這個問題：已知一把椅子和一張桌子，目標是將它們配對起來——有相同的顏色。於初始狀態我們有兩罐油漆，但是油漆和家具的顏色是未知的。只有桌子剛開始的時候是在代理人的視野之內：

$$Init(Object(Table) \land Object(Chair) \land Can(C_1) \land Can(C_2) \land InView(Table))$$

$$Goal(Color(Chair, c) \land Color(Table, c))$$

有兩個動作：將蓋子從油漆罐移走並使用被打開之漆罐內的油漆來漆物件。行動模式很簡單，但有一個例外：我們現在允許前提與效果包含沒有出現在動作變數表內的變數。也就是說，$Paint(x, can)$ 沒有提到代表罐裡油漆顏色的變數 c。在完全可觀察的情況，這是不允許的——我們必須命名動作 $Paint(x, can, c)$。但是於局部可觀察的情況，我們可能或可能不知道罐中是什麼顏色。(該變數 c 被全稱量化，正如於一個行動模式中所有其他變數)。

$Action(RemoveLid(can),$
PRECOND: $Can(can)$
EFFECT: $Open(can))$
$Action(Paint(x, can),$
PRECOND: $Object(x) \land Can(can) \land Color(can, c) \land Open(can)$
EFFECT: $Color(x, c))$

想要求解一個局部可觀察的問題，代理人必須推理它執行規劃時候它將得到的察覺。當它實際地動作時，察覺會由代理人的感測器提供，但是當它正在規劃時，它會需要一個感測器模型。於第 4 章，這個模型是經由一個函數，PERCEPT(s)，所給定。至於規劃，我們以一個新的模式類型，**察覺模式**，來擴增 PDDL：

Percept(*Color*(*x*, *c*),
PRECOND：*Object*(*x*) ∧ *InView*(*x*)
Percept(*Color*(*can*, *c*),
PRECOND：*Can*(*can*) ∧ *InView*(*can*)∧*Open*(*can*)

第一個模式說的是當一個物件出現在眼前的時候，代理人可察覺物件的顏色(也就是說，對於物件 *x*，代理人會知道所有 *c* 的 *Color*(*x*, *c*)的眞假值)。第二個模式說的是如果一個開了口的罐子出現在眼前，那麼代理人會察覺罐內的漆的顏色。因爲於這個世界中沒有外來的事件，物件的顏色仍然是相同的，即使它沒有被察覺出來，直到代理人執行一個動作去更改物件的顏色。當然，代理人需要一個會讓物件(一次一個)出現在眼前的動作：

Action(*LookAt*(*x*),
PRECOND: *InView*(*y*) ∧ (*x* ≠ *y*)
EFFECT: *InView*(*x*) ∧ ¬*InView*(*y*))

對於完全可觀察的環境，我們會有一個對每個流都沒有前提的察覺公理(知覺公理)。一個無感測器的代理人，在另一方面，沒有任何察覺公理。請注意即使無感測器代理人也能夠求解塗漆問題。解是能夠打開任何一罐油漆並把它用於椅子和桌子上，這樣**強制**它們成爲同一種顏色(即使代理人不知道是什麼顏色)。

具有感測器的應變規劃代理人能產生一個較好的規劃。首先，注視桌子與椅子以獲知它們的顏色； 如果它們顏色已經相同，則規劃就完成了。如果不相同，注視漆罐；如果罐上的漆色與某件家具的的顏色是相同的，那麼漆他件家具。否則，用任何一種顏色來漆兩件傢俱。

最後，線上規劃代理人剛開始可能製造出一個分支較少的應變規劃—或許不考慮任何罐子與家具顏色完全不匹配的可能性——當它們出現時，應付該問題的方法會是重規劃。它也能處理其行動描述的不正確性。然而應變規劃器直接假設動作的效果是必然是成功——油漆椅子就是這樣的工作——重規劃代理人會檢查結果並做額外的規劃，以修補任何意想不到的失敗，如一片未漆到的部份或是原顏色外顯。

在眞實世界中，代理人使用這些方法的組合。車輛製造商出售備胎與氣囊，這些是應變規劃一部分的實際體現，旨在處理刺穿或碰撞等意外事件。在另一方面，大多數的車輛司機從不考慮有這些可能性；當問題出現它們的反應有如重規劃代理人。通常，代理人僅僅會去規劃那些有重要的後果且發生機會不可輕忽的意外事件。因此，一個打算來趟橫越撒哈拉沙漠之旅的汽車駕駛員應該會對汽車拋錨做出仔細的應變規劃，然而去趟超市則較不需預先規劃。我們接著詳細地看看三個方法。

11.3.1 無感測規劃

第 4.4.1 節介紹過在信度狀態空間中搜尋的基本構想，以尋找無感測器問題的解答。轉換無感測器規劃問題爲信度狀態規劃問題的運作方式與第 4.4.1 節所做的方式相同；主要差別是底層的實體轉移模型被表示成一群行動模式且信度狀態被表示成一個邏輯公式而非逐一列舉的狀態集。爲求簡化，我們假設底層的規劃問題是確定的。

無感測器油漆問題的初始信度狀態能夠不考慮 *InView* 流，因爲代理人沒有感測器。而且，我們視爲已知的不變事實 $Object(Table) \wedge Object(Chair) \wedge Can(C_1) \wedge Can(C_2)$，因爲於每一個信度狀態這些都成立。代理人不知道罐子或是物件的顏色，或罐子是否已被打開的還是仍然是緊閉的，但是它確實知道物件與罐子有顏色：$\forall x\ \exists c\ Color(x, c)$。特殊化推論(*Skolemizing*)之後，(見第 9.5 節)，我們獲得初始信度狀態：

$$b_0 = Color(x, C(x))$$

於經典規劃，其中已設定**封閉世界假設**，我們會假設任何沒有於狀態提及的流都爲假，但是於無感測器(與局部可觀察的)規劃，我們必須轉向到**開放世界假設**，於此，同時存在正和負流的狀態，如果流沒有出現，則它的值是未知的。因此，信度狀態正好對應到一組滿足該公式之可能的世界集。給定這個初始信度狀態，以下動作序列是一個解答：

$$[RemoveLid(Can_1), Paint(Chair, Can_1), Paint(Table, Can_1)]$$

我們現在展示如何透過動作序列來發展信度狀態，以顯示滿足目標的最後信度狀態。

首先，注意於一個已知的信度狀態 b，代理人可以考慮任何前提被 b 滿足的動作。(其他動作無法被使用，因爲轉移模型並沒有定義前提未被滿足之動作的效果)。根據式(4.4)，用於更新於一個確定世界中已給定之適用動作 a 的信度狀態 b 的一般公式如下：

$$b' = RESULT(b, a) = \{s' : s' = \text{RESULT}_P(s, a) \text{ 及 } s \in b\}$$

其中 RESULT_P 定義實體轉移模型。暫時，我們假設初始信度狀態必然是個文字的連言，也就是說，一個 1-CNF 公式。爲了建構新的信度狀態 b'，我們必須考慮當動作 a 作用後，b 中每個實體狀態 s 的文字 ℓ 會發生什麼事。對於在 b 中眞假值已經知道的文字，於 b'的眞假值是從目前的值以及動作的增加表與刪除表計算而來(舉例來說，如果 ℓ 是在動作的刪除表之內，那麼$\neg\ell$ 被加進 b')。對於那些在 b 中眞假值是未知的文字該如何處理呢？有 3 點不同：

1. 如果動作加入 ℓ，則 ℓ 會轉換成 b'無論它的初始值爲何。
2. 如果動作刪除 ℓ，則 ℓ 於 b'中會爲假無論它的初始值爲何。
3. 如果動作不影響 ℓ，則 ℓ 將停留在它的初始值(是未知的)且將不會出現於 b'。

因此，我們看到 b'的計算幾乎與於可觀察的情況是一般無貳的，於該情況是透過式(10.1)來制定：

$$b' = \text{RESULT}(b, a) = (b - \text{DEL}(a)) \bigcup \text{ADD}(a)$$

我們無法常使用集合語意，因爲：(1) 我們必須確認 b'不包含 ℓ 與$\neg\ell$，且(2) 原子可能包含無範圍限制的變數。但是它仍然是以 b 做起頭來計算 $\text{RESULT}(b, a)$，設定任何出現於 $\text{DEL}(a)$的原子爲假，且設定任何出現於 $\text{ADD}(a)$的原子爲眞的情況。舉例來說，如果我們施加 $RemoveLid(Can_1)$到初始信度狀態 b_0，我們得到

$$b_1 = Color(x, C(x)) \wedge Open(Can_1)$$

當我們施加動作 $Paint(Chair, Can_1)$，前提 $Color(Can_1, c)$可透過已知的文字 $Color(x, C(x))$滿足綁定 $\{x/Can_1, c/C(Can_1)\}$而新的信度狀態是

$$b_2 = Color(x, C(x)) \wedge Open(Can_1) \wedge Color(Chair, C(Can_1))$$

最後，我們施加動作 $Paint(Table, Can_1)$得到

$$b_3 = Color(x, C(x)) \wedge Open(Can_1) \wedge Color(Chair, C(Can_1)) \wedge Color(Table, C(Can_1))$$

最後的信度狀態滿足目標，$Color(Table, c) \wedge Color(Chair, c)$，變數 c 勢必爲 $C(Can_1)$。

前面更新規則的分析已經顯示一個非常重要的事實：以文字連言來定義之信度狀態的族裔於 PDDL 行動模式下定義之更新是封閉的。也就是說，如果信度狀態開始時爲一個文字的連言，那麼任何更新所得的也會是文字的連言。這意指在一個有 n 個流的世界，任何的信度狀態能夠被表示成一個大小爲 $O(n)$的連言。想到這是個有 2^n 個狀態的世界時，這樣的結果是非常令人寬心。它說我們能夠精簡的表示出所有我們亟需的 2^n 個狀態的子集合。而且，檢查信度狀態是先前訪問過之信度狀態的子集合或是超集合的過程也很容易，至少於命題的例子是如此。

這令人愉快的畫面有個美中不足之處是，它只適用於對於所有前提已被滿足的狀態有相同的效果的行動模式。也就是這個性質使得 1-CNF 信度狀態表示得以保存下來。一旦效果依賴於狀態，相依性介入到各流之間而 1-CNF 性質就會遺失了。考慮，舉例來說，定義於第 3.2.1 節的簡單眞空吸塵器世界。令機器人位置之流爲 *AtL* 與 *AtR* 而方格的狀態爲 *CleanL* 與 *CleanR*。根據問題的定義，*Suck* 動作並沒有前提——它一定能被完成。困難的是它的效果依賴於機器人的位置：當機器人是 *AtL*，結果是 *CleanL*，但是當它是 *AtR*，結果是 *CleanR*。對於如是的動作，我們的行動模式將需要新的東西：一個**有條件效果**。這些效果的句法是「**when** *condition*: *effect*」，其中 *condition* 是一個與目前狀態做比較的條件公式，而 *effect* 則是一個描述結果狀態的公式。對於眞空世界，我們有

Action(*Suck*,

EFFECT: **when** *AtL*: *CleanL* ∧ **when** *AtR*: *CleanR*)

當施加於初始信度狀態 *True*，得到的信度狀態是$(AtL \wedge CleanL) \vee (AtR \wedge CleanR)$，這個不在 1-CNF 之中(這個轉移能夠由圖 4.14 看出)。通常，有條件效果能夠於一個信度狀態中的流之間引起任意的相依性，於最壞情況下，會導致信度狀態的規模呈現指數量級的成長。

很重要的是了解前提與有條件效果之間的差別。所有的條件被滿足的有條件效果會將它們的效果發揮出來以製造出結果狀態； 如果沒有一個被滿足，則結果的狀態不變。在另一方面，如果前提未被滿足，則動作不可用的且結果狀態是未未定義的。從無感測器規劃的觀點，有條件效果會比不可用動作爲好。舉例來說，我們能夠將 *Suck* 分解爲兩個無條件效果的動作如下：

Action(*SuckL*,

PRECOND: *AtL*; EFFECT: *CleanL*)

Action(*SuckR*,

PRECOND: *AtR*; *EFFECT*: *CleanR*)

現在我們僅有無條件的模式，所以所有的信度狀態仍然是於 1-CNF；不幸的是，我們無法決定於初始信度狀態下 *SuckL* 與 *SuckR* 的可用性。

那麼，問題似乎無法避免的會牽涉到 wiggly 的信度狀態，如同我們考慮 wumpus 世界狀態估計時所碰過的問題(見圖 7.21)。推薦的解答是使用正確的信度狀態的**保守近似值**；舉例來說，信度狀態仍然能夠是 1-CNF，只要它所包含的文字之眞假值能夠被判斷出來且將所有的其他文字視爲未知的。雖然這方法聽起來似乎不錯，在於它絕不會產生一個錯誤的規劃，它是不完整的，因爲當必須涉及文字之間的互動時，它或許就沒辦法找出問題的解答。給個平凡的例子，如果目標是機器人要在乾淨的方格，那麼[*Suck*]是一個解答，但是一個沒有感測器而堅持處於 1-CNF 信度狀態的代理人就找不到它。

或許一個更好的解答是尋找個簡單的動作序列，儘可能的保持信度狀態。舉例來說，在無感測器的眞空吸塵器世界，動作序列[*Right*, *Suck*, *Left*, *Suck*]產生如下的信度狀態序列：

$b_0 = True$
$b_1 = AtR$
$b_2 = AtR \wedge CleanR$
$b_3 = AtL \wedge CleanR$
$b_4 = AtL \wedge CleanR \wedge CleanL$

亦即，代理人能於保持 1-CNF 信度狀態下解出該問題，即使一些的序列(例如，那些以 *Suck* 開頭者)偏離 1-CNF。一般的教訓是不要悖離人性：我們總是執行少數的動作(核對時間，輕拍我們的口袋確保我們還懷著車鑰匙，當我們在一座城市中行走時會看路標)消除不確定性並讓信度狀態容易管理。

對於無法管理地扭曲信度狀態的問題有另一個非常的不同的方法：根本就不要想去計算它們。假設初始的信度狀態是 b_0，且我們會想要知道從動作序列$[a_1, ..., a_m]$所得到的信度狀態。不直接地計算它，而是將它表示爲「b_0 則$[a_1, ..., a_m]$」。這是偷懶的做法但是信度狀態的表示並不含糊，且它是非常的簡明的——$O(n + m)$其中 n 是初始信度狀態的大小(假設是於 1-CNF)而 m 是動作序列的最大長度。不過，做爲信度狀態表示，它有一個缺點：判斷該目標是否被滿足了或是一個動作是否能用，可能需要大量的計算。

該計算能夠當成一個蘊涵測試來實作：如果 A_m 代表一群定義動作 $a_1, ..., a_m$ 發生所需的後續狀態公理——一如於第 10.4.1 節對 SATPLAN 所做的解釋——而 G_m 斷言於數個步驟之後目標爲眞，如果 $b_0 \wedge A_m \models G_m$，也就是說，如果 $b_0 \wedge A_m \wedge \neg G_m$ 未被滿足，那麼該規劃就達成目標。給予一個現代的 SAT 求解器，有可能比計算整個信度狀態更快完成這件事。舉例來說，如果在它的增加表中並沒有一個序列中的動作有特別的目標流，求解器會立即發現這件事。如果關信度狀態的部分結果——舉例來說，流已經知道爲眞或假——被快取起來以簡化隨後的計算也是有幫助的。

無感測器規劃謎題的最後一片拼圖是個啓發式函數，用來引導搜尋。啓發式函數的意義與經典規劃者是相同的：從已知的信度狀態達成目標之代價的估計(或許可採納的)。有了信度狀態，我們有一個額外的事實：求解任何一個信度狀態的子集合一定比求解信度狀態簡單：

如果 $b_1 \subseteq b_2$，則 $h^*(b_1) \leq h^*(b_2)$

因此，任何爲子集合所計算之可採納啓發式，該信度狀態本身是可採納的。最明顯的候選者是單值子集合，也就是說，個別實體的狀態。我們能夠隨機拿取信度狀態 b 中任何一群狀態 $s_1, ..., s_N$，運用第 10 章中任一個可採納啓發式 h，再回來。

$$H(b) = \max\{h(s_1), ..., h(s_N)\}$$

當作啓發式估計用以求解 b。我們也能夠直接使用 a 規劃圖於 b 本身：如果它是文字的連言(1-CNF)，只要設定那些文字爲圖形的初始狀態層。如果 b 不是在 1-CNF，那麼也有可能找出一起蘊涵 b 的一組文字。舉例來說，如果 b 是在選言正規形式(DNF)，DNF 公式是文字的連言，它蘊涵 b 且能夠形成規劃圖初始層。一如先前，我們能夠拿出由每組文字所得到的最大啓發式。我們也能夠使用非可採納啓發式，如忽略刪除表啓發式(10.2.3 節)，實務上效果似乎不錯。

11.3.2 應變規劃

我們於第 4 章看過應變規劃——根據察覺產生有條件的分支規劃——是適合部分可觀察性，非決定性，或是兩者兼具的環境。對於稍早所給具有知覺公理的局部可觀察的油漆問題，一個可能的應變解答如下：

[*LookAt*(*Table*), *LookAt*(*Chair*),

if *Color*(*Table*, *c*)∧*Color*(*Chair*, *c*) **then** *NoOp*

else [*RemoveLid*(*Can*$_1$), *LookAt*(*Can*$_1$), *RemoveLid*(*Can*$_2$), *LookAt*(*Can*$_2$),

if *Color*(*Table*, *c*)∧*Color*(*can*, *c*) **then** *Paint*(*Chair*, *can*)

else if *Color*(*Chair*, *c*)∧*Color*(*can*, *c*) **then** *Paint*(*Table*, *can*)

else [*Paint*(*Chair*, *Can*$_1$), *Paint*(*Table*, *Can*$_1$)]]]

於這規劃中的變數應該被視爲被存在性性地量化了；第二行說的是如果存在一些的顏色 c 也就是說桌子與椅子的顏色，則代理人不必做任何事情來達成目標。當執行這規劃，應變規劃代理人能夠維持它的信度狀態如一個邏輯的公式，並且藉由決定信度狀態蘊涵條件公式或是它的否定來評估每個分支狀態。(端視應變規劃演算法如何確認代理人永遠不會陷於條件公式的眞假值爲未知的信度狀態)。注意於一階條件下，該公式可以用一個以上的方式來滿足；舉例來說，如果兩個罐子的顏色與桌子是相同的，條件 *Color*(*Table*, *c*)∧*Color*(*can*, *c*)可以被{*can*/*Can*$_1$}與{*can*/*Can*$_2$}所滿足。那樣的話，代理人能夠選取有令人滿意的代換運用於其餘的規劃。

如第 4.4.2 節所示，在一個動作之後計算新的信度狀態且隨後的察覺於兩個階段內完成。第一個階段計算動作之後的信度狀態，一如爲無感測器代理人所做的：

$$\hat{b} = (b - \text{DEL}(a)) \cup \text{ADD}(a)$$

其中，一如之前，我們已經假設信度狀態以一個文字的連言來表示。第二個階段有些難對付。假設察覺文字 $p_1,...,p_k$ 被收到。你可能會想我們只須要將這些加入到信度狀態；事實上，我們也能夠推斷用於感測的前提被滿足。現下，若察覺 p 正好有一個察覺公理，*Percept*(p, PRECOND：c)，其中 c 是一個文字的連言，那麼那些文字能夠隨 p 一起扔入信度狀態。在另一方面，如果 p 有多於一個察覺公理，根據所預測的信度狀態 $\hat{b}$，它的前提或許會成立，則我們必須加進前提的選言。顯而易見，這將信度狀態取離 1-CNF，並帶來與有條件效果相同的複雜情況，具有大致相同的解答類型。

給予一個計算精確或者近似信度狀態的機制，我們能夠以於第 4.4 節對信度狀態所用的 AND-OR 前向搜尋的延伸來製造出應變規劃。具有不確定性效果的動作——它是只使用行動模式 EFFECT 中的選言來定義——能夠接受於信度狀態更新計算上的細微變更，而不會更動搜尋演算法[2]。對於啓發式函數，許多對無感測器規劃方法的建議也適用於局部可觀察的，不確定性的情況。

11.3.3 線上重規劃

想像看著於一個車輛工廠的點焊機器人。當每台車輛通過作業線，機器人迅速，準確的動作會一再重複。儘管技術上給人深刻印象，機器人看不出有些許的智慧，因爲運動是一個固定的、預先設計好的順序；在任何有意義的感知中，機器人顯而易見不 「知道它正做什麼」。現在假設正當機器人準備點焊的時候，一個沒固定好的門掉出車外。機器人迅速將它的焊接致動器換爲夾鉗，拿起車門，檢查是否有任何抓痕，重新裝上車輛，發電子郵件給廠區作業主管人員，換回焊接致動器，並且恢復它的作業。突然之間，機器人的行爲好像有目的而非機械性的方法；我們假設它不是肇因於一個龐大，預先計算好的應變規劃，而是從一個線上重規劃過程——這意指機器人必須知道它試圖做的是什麼事情。

重規劃預先假設**執行監控**的形式以判斷是否需要一個新的規劃。有這麼一個需要是當應變規劃代理人厭倦爲每一個小意外事故做規劃，如天空是否會落到它的頭上[3]。一些部份建構出的應變規劃的分支能夠直說 *Replan*；如果這樣的分支於執行期間能到達的話，代理人轉回到規劃模式。一如我們先前提及過的，關於有多少問題要提前求解以及有多少留待重規劃等方面的決定，是一個牽涉到有不同代價的可能事件與發生的可能性之間的妥協。沒人想要讓他們的車輛在撒哈拉沙漠的中間時拋錨，而只有那時才想到要有足夠的水。

如果代理人的世界模型是錯誤的，重規劃便有其需要。一個動作模型，可能有**前提漏失**——舉例來說，代理人可能不知道，打開漆罐的蓋子通常需要一個螺絲起子；模型可能有**效果漏失**——舉例來說，漆一個物件時也可能一併漆到地板；或是模型可能有個**狀態變數漏失**——舉例來說，前面所給的模型並沒有註記罐內的漆量多少、它的動作會如何改變漆量、或是需要漆量不是空的等等。模型也可是缺乏對於**外來事件**的條款，例如有人打翻漆罐。外來的事件也包括目標的變更，如增加桌子與椅子不能被漆成黑色的需求。如果代理人倚賴絕對正確性的模型，但沒有監視與重規劃的能力，則它的動作很可能是極其脆弱的。

線上代理人有個如何小心地去監視環境的選擇。我們區分三個層級：

- **動作監控**：執行動作之前，代理人先確認所有的前提仍然成立
- **規劃監控**：執行動作之前，代理人確認其餘的規劃仍然成功。
- **目標監控**：之前執行動作之前，代理人檢查看看是否有一個更好的目標集是它能夠努力達成的。

圖 11.9 看到一個監控動作的模式。代理人同時追蹤其原始規劃、*whole plan* 以及部分被該規劃註記爲還沒有執行的規劃。執行規劃的前幾個步驟之後，代理人預期會處於狀態 *E*。但是代理人觀察它實際處於狀態 *O*。然後，它必須找出在原始規劃的一些能夠讓它退回去的點 *P* 來修復該規劃(可能 *P* 是目標狀態 *G*)。代理人試著最小化全部的規劃代價：該修復部份(從 *O* 至 *P*)加上繼續(從 *P* 至 *G*)。

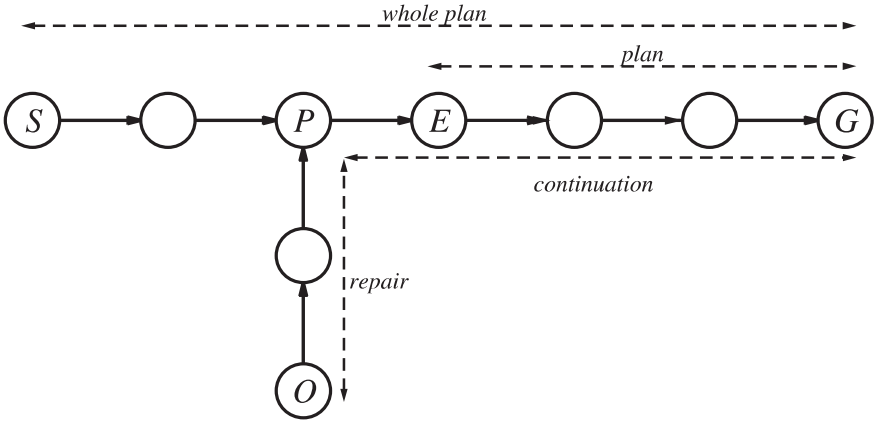

圖 11.9　在執行前，規劃器提出一個此處稱為 *whole plan* 的從 *S* 到 *G* 的規劃。代理人執行該規劃的步驟直到它預期處於狀態 *E*，但是觀察到它實際是處於 *O*。代理人然後重規劃以求最小修復並繼續前進到達 *G*

現在讓我們回到獲得顏色匹配的椅子和桌子的範例問題。假設代理人提出了這規劃：

[*LookAt*(*Table*), *LookAt*(*Chair*),
if *Color*(*Table*, *c*) ∧ *Color*(*Chair*, *c*) **then** *NoOp*
  else [*RemoveLid*(Can_1), *LookAt*(Can_1),
    if *Color*(*Table*, *c*) ∧ *Color*(Can_1, *c*) **then** *Paint*(*Chair*, Can_1)
    else REPLAN]]

現在代理人準備好執行規劃。假設代理人觀察到桌子和漆罐是白色的和椅子是黑色的。它然後執行 *Paint*(*Chair*, Can_1)。在這一點上經典規劃器會宣布勝利；規劃已執行。但是一個線上執行監控代理人必須檢查其餘空規劃的前提——桌子與椅子有相同的顏色。假設代理人察覺它們並沒有相同的顏色——事實上，椅子現在是雜灰色，黑色漆被透出來。然後代理人需要計算出 *whole plan* 中的一個目標位置和一個達到那裡的修補行動序列。代理人注意到當前狀態跟 *Paint*(*Chair*, Can_1)行動前的前提是一樣的，所以代理人爲 *repair* 選擇空序列，並讓它的 *plan* 與剛剛嘗試過的[*Paint*]序列一樣。把這個新的序列放在適當位置，執行監控繼續進行，而且 *Paint* 行動再次被嘗試。這個行爲會循環往復，直到桌子被感知到已經被完全漆過了。但是應該注意到的是，迴圈是由規劃-執行-重規劃的過程建立的，而不是由規劃中的一個明確的迴圈建立的。也注意原始規劃並不需要涵蓋每一個意外事件。如果代理人達到標示爲 REPLAN 的那個步驟，那麼它能夠產生一個新的規劃(也許會與 Can_2 有關)。

行動監控是執行監控的一個十分簡單的方法，但是它有時導致不太智慧的行動。例如，假定沒有黑色或白色油漆，且代理人構造一個規劃來透過將桌子和椅子兩者漆成紅色而解決油漆問題。假設只有足夠的紅色漆供椅子所用。隨著動作監控，代理人將如期動作並將椅子漆成紅色，然後注意到漆沒有了且無法漆桌子，此時它會重規劃一個修復——或許是椅子和桌子都漆成綠色。規劃-監控代理人能夠發現失敗，只要目前的狀態會使得其餘的規劃變得無用時。因此，它不會浪費時間去將椅子漆成紅色。**規劃監控**透過檢驗整個剩餘規劃成功的前提——也就是，規劃中每一步的前提，除了那些被剩餘規劃中另一個步驟獲得的前提之外——來實現這個目的。規劃監控儘早截斷註定失敗的規劃的執行，而不是繼續執行直到失敗確實出現[4]。規劃監控也允許**意外發現**——偶然的成功——偶然的成功。如果有人一起過來，在代理人把椅子漆成紅色的同時，把桌子漆成紅色，那麼最後規劃前提得到滿足(已經獲得目標)，代理人能夠早早回家。

修改一個規劃演算法使得在規劃中的每個動作都註明出所需的前提是很簡單的，由此發揮動作監控的作用。發揮規劃監控則稍微複雜些。偏序與規劃圖規劃器有個優勢是它們已經建立好包含規劃監控所需之關係的架構。以必要的註記來擴增狀態-空間規劃器是能夠藉著小心地登記規劃回歸時的目標流來達成。

現在我們已描述了監控和重規劃的一種方法，我們需要問「它可行嗎？」這是一個令人驚異的棘手問題。如果我們的意思，「我們是否能夠保證代理人一定會達成目標？」，那麼答案是否定的，因爲代理人能夠無意中到達一個無法修復的死路。例如，眞空吸塵器代理人也許具有本身的缺陷模型，且不知道它的電池可能會被用完。一旦它們做了，它無法修復任何規劃。如果我們排除死路——假設存在一個規劃，可以從環境的任何狀態達到目標——並且假設環境眞的是不確定的，如是的規劃在任何給定的執行嘗試下總會有一些成功的機會，在這個意義下，則代理人最終會達到目標。

麻煩發生於當一個動作是實際上不是非確定性的，而是相當依賴於一些代理人不知道的前提。舉例來說，有時候一個漆罐可以是空空的，所以該漆罐沒有任何效果。不管重試幾次都不會改變這種個事實[5]。一個解是從可能的修補規劃集中隨機選擇，而不是每次都試同一個。在這情況下，打開另一罐漆罐的修復規劃能夠或許發揮作用。更好方法是**學習**一個更好的模型。每一個失敗的預言，都是學習的機會；一個代理人應該能夠修改其模型的世界，以符合其所察覺者。從那時起，重規劃器將能夠從問題的根本來修復，而非倚賴運氣來選取一個好的修復。這種類型的學習將在第 18 及 19 章中描述。

11.4 多代理人規劃

迄今，我們已經假設認爲只有一個代理人在感測、規劃並採取動作。當環境中有數個代理人，每個代理人面臨一個**多代理人規劃問題**，於此在其他代理人的幫助或妨礙之下，它努力達成自己的目標。

介於純粹的單代理人與眞正多代理人情況之間是問題範圍的寬窄，展示了單調性代理人的不同分解程度。一個具有數個能夠同時運行之作用器的代理人——舉例來說，一個在同一個時間能夠打字和講話的人——必須做**多重作用器規劃**，以便在應付作用器之間的正負互動的同時管理每個作用器。當作用器實際地被拆解成各自獨立的單元——如在一間工廠一群輸送機器人——多重作用器規劃變成**多體規劃**。多體問題仍然是一個「標準」的單一代理人問題，只要每個個體相關的感測器收集的資料能夠被匯集——無論集中或分佈在數個個體——形成一個常見的世界狀態估計，然後通知執行整體規劃，在這情況下，數個個體變成單一個體。當即時通訊的限制使得這變得不可能時，我們有個有時稱之爲**去集中化規劃**問題，或可能是用詞不當，因爲規劃階段還是集中的但是執行階段至少部份地被分散了。在這情況下，爲每個個體架構的子規劃可能必須包括與其他個體明確的溝通動作。舉例來說，數個偵察機器人，巡察的範圍很廣泛，可能常常會在無線電能互相聯繫的範圍之外，應該在通訊可及之時分享它們的調查結果。

當一個單一個實體在做規劃，其實只有一個目標，是所有個體需要分享。當個體是不同代理人做它們自己的規劃時，它們仍然可以分享相同的目標；舉例來說，兩個由人類網球選手組成的雙打隊伍分享贏得比賽的目標。即使有了共享的目標，不過，多體和多代理人情況是非常的不同。在一個多體機器人的雙打隊伍，一個單一規劃會指明哪個個體應上球場，哪個個體應擊球。另一方面，於一個多代理人雙打隊伍中，每個代理人決定該做什麼；並沒有**協調**的方法，雙方代理人可決定涵蓋同一部分的場地，而每個代理人可能將球留給其他代理人去打擊。

一個多代理人問題的最清楚的狀況，當然，是當代理人有不同目標的時候。在網球裡，兩個敵隊的目標是直接衝突的，而出現第 5 章的零和狀況。觀眾能夠被看成是代理人，如果他們的支持或藐視是一個重要的因素且會受到球員行爲的影響；否則，他們被看成是一個自然的面向——就像天氣——被假設對球員的意圖是無關緊要的[6]。

最後，某些系統是一個集中式和多代理人規劃的混合體。舉例來說，一個宅配公司可以離線地規劃卡車的路線且每天規劃，但是留下某些面向予司機與飛行員爲自主決定，由其個別地根據交通和天候的情況做出反應。此外，藉由獎勵的付出(薪水和紅利)讓公司及其員工的目標達成一致，至相當的程度——一個眞的是多代理人系統的明確徵兆。

與多代理人規劃有關的議題能夠被粗略分成爲兩組。第一組，涵蓋在第 11.4.1 節，是與表示有關的議題，並規劃數個同步動作；這些議題出現於所有的設定，從多作用器到多代理人規劃。第二組，涵蓋在第 11.4.2 節，牽涉到眞正多代理人出現之合作、協調和競爭的議題。

11.4.1 規劃數個同步動作

眼下，我們將會以相同方式對待多作用器、多個體、多代理人等等設定，將它們統稱爲**多行為者**設定(multiactor seetings)，將作用器、個體和代理人統稱爲**行為者**(actor)。這一節的目標是針對多行爲者設定研究出如何定義轉移模型，正確的規劃以及有效率的規劃演算法。一個正確的規劃是一個如果被行爲者所執行就會達成目標的規劃(在眞正的多代理人設定，當然，代理人可能不會同意執行任何特殊的規劃，但是至少它們會知道什麼樣的規劃會是有效的，如果它們眞的同意執行它們)。爲求簡化，我們假設完美的**同步**：每個動作佔用相同的時間量且在聯合規劃上每個點的動作都同步。

我們以轉移模型開始；對於確定的情況，這是函數 RESULT(*s*, *a*)。於單一代理人設定，該動作或許有 *b* 個不同的選擇；*b* 能夠非常的大，特別是對於有許多可作用於其上的物件的一階表示，但是儘管如此，行動模式 *s* 提供一個簡明潔的表示。於有 *n* 個行爲者的多行爲者設定，單一動作 *a* 被替換成一個**聯合動作**$\langle a_1,..., a_n\rangle$，其中 a_i 是第 *i* 個行爲者所採取的動作。立即地，我們看到兩個問題：首先，我們必須對 b^n 個不同的聯合動作描述轉移模型；其次，我們有一個聯合規劃之分支因子爲 b^n 的問題。

將行爲者放在一起成爲一個有鉅大分支因子的多行爲者系統，多行爲者規劃的主要研究重點已經與行爲者儘可能的脫鉤，使得問題的複雜性與 *n* 是線性量級而非指數量級的成長程度。如果行爲者彼此之間沒有互動——舉例來說，*n* 個行爲者每個人玩單人球戲——則我們只需要求解 *n* 個不同的問題。如果行爲者**非緊密耦合**在一起，我們是否能夠取得某些優於指數量級的改進？這是，當然，是許多的 AI 領域的一個核心問題。於 CSP 範圍內我們已明確地看過它，其中「近似樹形」的限制圖可得到有效率的解答方法(見 6.5 節)，以及於無交集的模式資料庫(3.6.3 節)範圍和附加啓發式規劃(10.2.3 節)。

$$
\begin{array}{l}
Actors(A, B) \\
Init(At(A, LeftBaseline) \ \wedge \ At(B, RightNet) \ \wedge \\
\qquad Approaching(Ball, RightBaseline)) \ \wedge \ Partner(A, B) \ \wedge \ Partner(B, A) \\
Goal(Returned(Ball) \ \wedge \ (At(a, RightNet) \ \vee \ At(a, LeftNet)) \\
Action(Hit(actor, Ball), \\
\qquad \text{PRECOND:} Approaching(Ball, loc) \ \wedge \ At(actor, loc) \\
\qquad \text{EFFECT:} Returned(Ball)) \\
Action(Go(actor, to), \\
\qquad \text{PRECOND:} At(actor, loc) \ \wedge \ to \ \neq \ loc, \\
\qquad \text{EFFECT:} At(actor, to) \ \wedge \ \neg \ At(actor, loc))
\end{array}
$$

圖 11.10　網球雙打問題。兩個代理人 *A* 及 *B* 正在一起打球，而且可以是下面 4 個位置之一：*LeftBaseline*，*RightBaseline*，*LeftNet* 及 *RightNet*。僅當打者在合適位置時才能把球打回去。請注意，每個行動包含當作引數的行為者

非緊密耦合問題的標準方法是假裝該問題是完全地脫鉤，然後安排互動。對於轉移模型，這意指寫出宛如行爲者各自獨立地行動的行動模式。讓我們看看這個方式如何運用在雙人網球問題。讓我們假設在遊戲中的某一點，它們有聯合目標，將擊給它們的球打回去並確保它們中至少有一個防守網前。第一個多行爲者定義可能看起來像圖 11.10。以這個定義，很容易看出下述**聯合規劃**能發揮效用：

PLAN 1:

A : [*Go*(*A*, *RightBaseline*), *Hit*(*A*, *Ball*)]

B : [*NoOp*(*B*), *NoOp*(*B*)]

不過，當規劃中兩名代理人同一時間擊中球時，問題就出現了。於眞實的世界，這不會發揮作用，但是 *Hit* 行動模式說的是球會被成功地回傳。技術上，困難之處在於前提限制了一個動作能夠被成功執行的狀態，而不限制可能弄得一團亂的其他動作。我們藉由將行動模式擴增一個新的功能來解決這個困難：一份明白列出必須或不必須同時執行的**同步動作清單**。例如，*Hit* 行動可以被描述如下：

Action(*Hit*(*a*, *Ball*),
CONCURRENT: $b \neq a \Rightarrow \neg Hit(b, Ball)$
PRECOND: $Approaching(Ball, loc) \wedge At(a, loc)$
EFFECT: $Returned(Ball)$)

換句話說，僅僅如果在同一個時間在另一個代理人沒有其他 *Hit* 動作出現，*Hit* 動作才有它指明的效果(於 SATPLAN 的方式，這會被一個部份**動作排斥公理**所處理)。對於某些動作，僅僅當另一個動作同時發生時所想要的效果才會被達成。舉例來說，需要兩個代理人才能把一個裝滿飲料的冷飲箱帶到網球場：

Action(*Carry*(*a*, *cooler*, *here*, *there*),
CONCURRENT: $b \neq a \wedge Carry(b, cooler, here, there)$
PRECOND: $At(a, here) \wedge At(cooler, here) \wedge Cooler(cooler)$
EFFECT: $At(a, there) \wedge At(cooler, there) \wedge \neg At(a, here) \wedge \neg At(cooler, here)$)

以這些類型的行動模式，第 10 章所描述過的任何一個規劃演算法僅需細微的修改就能用於產出多行爲者規劃。在子規劃之間耦合是鬆散的範圍下——也就是說，在規劃搜尋的過程中，很少會有同時性限制來參上一腳——我們會期望於單一代理人規劃情況下所推導的各式各樣的啓發式，在多行爲者方面也可以一樣的有效。我們可以用上兩章的調整方法——HTN、部分可觀察性、條件、執行監控和重規劃——來擴展這個方法，不過這超出了本書的範圍。

11.4.2 數個代理人的規劃：合作和協調

現在讓我們考慮眞的多代理人設定，於此每個代理人製作它自己的規劃。開始前，讓我們假設目標與知識庫是共享的。你可能會想這縮減爲多體情況——每個代理人僅僅計算該聯合解答並執行解答中它自己的部份。可惜，於「該聯合解答」中的「該」會引人誤解。對於我們的雙打比賽隊伍來說，存在一個以上的聯合解答：

PLAN2：

A : [$Go(A, LeftNet)$, $NoOp(A)$]

B : [$Go(B, RightBaseline)$, $Hit(B, Ball)$]

如果兩個代理人能夠同意於規劃 1 或者規劃 2，目標將會達成。如果 *A* 選擇規劃 2，*B* 選擇規劃 1，那麼沒有人會把球打回去。相反，如果 *A* 選擇 1 並且 *B* 選擇 2，那時它們都會去擊球。代理人可能察覺到這個，但是它們該如何協調以確保它們都同意於該規劃呢？

選擇之一是在展開聯合活動之前採納一項**公約**。公約是於選擇聯合規劃上的任何限制。舉例來說，公約「堅守你這一邊的場地」會排除規劃 1，導致雙打夥伴選擇規劃 2。在路上的司機面臨不彼此碰撞的問題；這可以採用在大多數國家的公約「靠右行駛」而被(部分的)解出，相反的，「靠左行駛」同樣的合用，只要在環境中的所有代理人都同意。將相似的考慮應用到人類語言的發展上，其中重要的事情不是每個個體該說哪種語言，而是所有個體說同一種語言的事實。當公約是廣泛使用時，稱爲**社會法律**。

在缺乏可應用的公約時，代理人可以使用**通訊**來獲得一個可行聯合規劃的常識。例如，一個網球雙打比賽者可能喊「我的！」或「你的！」以指示一個偏好的聯合規劃。我們在第二十二章中更深入地討論通訊機制，其中我們觀察到涉及口頭交換的通訊不是必要的。例如，一個比賽者可以簡單地透過執行規劃的第一部分來對另一個同伴傳達一個偏好聯合規劃。在我們的網球問題中，如果代理人 A 去網前，那麼代理人 B 不得不後退到底線來擊球，因爲規劃 2 是唯一的以 A 去網前開始的聯合規劃。這個協調的方法，有時稱爲**規劃識別**，在一個單個行動(或短的行動序列)足以無歧義地確定一個聯合規劃時是可行的。注意通訊於競爭性代理人與作用於合作性代理人一樣的適用。

公約也會透過進化的過程出現。舉例來說，吃種子的收穫蟻是從較低等的胡蜂進化過來的群居動物。螞蟻的殖民地執行非常詳細的聯合規劃而沒有任何集中化的控制——女王的工作是生育，不做集中化規劃——全賴每隻螞蟻非常有限的計算、通訊和記憶能力(Gordon，2000，2007)。殖民地有許多的角色，包括內部工作人員、巡邏者和工蜂。每隻螞蟻根據它觀察到的當地條件來選擇一個角色來執行。舉例來說，蜂遠離蜂巢，搜尋種子，當它們找到一顆，立即將其帶回。因此，工蜂回巢的速率是今天食物夠用程度的近似值。當回巢率很高，其他螞蟻會放棄它們目前的動作，併承擔起清道夫的角色。螞蟻似乎有一個重要角色公約——覓食是最重要的——螞蟻會輕易地切換到更重要的角色，但是不會換到較不重要的。有一些學習機制：一個殖民地在長達幾十年的生活中學會作出更審愼與更成功的動作，即使各個螞蟻只活大約一年。

最後一個合作性多代理人行爲的例子出現在一群鳥的行爲。我們能夠獲得鳥群的合理模擬，如果每隻鳥代理人(有時候稱之爲 **boid**)觀察最靠近它的鳥的位置，然後選擇會最大化這三個分量加權總和的飛行方向與加速度：

1. **凝聚性**：越接近鄰鳥的平均位置得正分。
2. **分離性**：過於接近任何一隻鄰鳥得負分。
3. **列隊性**：越接近鄰鳥的平均飛行方向得正分。

如果所有的 boids 執行這個策略，鳥群飛行的**突現行為**會有如虛似剛體(pseudorigid)的結構，鳥群密度大約保持一定，不會隨時間沖散，且有時會出現突然俯衝動作。你能夠看看圖 11.11(a)中的靜止圖片並與(b)中的實際鳥群做比較。如同昆蟲，不需要每個代理人都擁有以其他代理人的行動爲模型的聯合規劃。

大多數困難的多代理人問題牽涉到同時與自己的隊伍的成員合作以及與敵方隊伍的成員競爭，所有的人都沒有受到集中化控制。我們於遊戲中看到這種現象，如圖 11.11 (c) 所展示的機器人足球或是 NERO 遊戲，於此兩個軟體代理人隊伍競相奪取控制塔。迄今，於這些種類的環境中有效率的規劃方法——舉例來說，利用鬆散耦合的特性——還處於它們的強褓期。

(a) (b) (c)

圖 11.11 (a) **一群模擬的鳥，使用** Reynold **的** boidsmodel。**圖片承蒙** Giuseppe Randazzo，novastructura.net **提供**。(b) **一群實際的八哥鳥群。圖片承蒙** Eduardo(pastaboy sleeps on flickr)**提供**。(c) NERO **遊戲中，兩個競爭的代理人隊伍試圖奪取塔。圖片承蒙** Risto Miikkulainen **提供**

11.5 總結

本章處理了眞實世界的規劃和行動的一些複雜因素。要點爲：

- 許多行動消耗**資源**，諸如錢、汽油或原材料。把這些資源看作池中的數值度量是方便的，勝過試著去推理，例如說，關於世界上每個單個的硬幣和鈔票。行動能夠產生和消耗資源，通常在嘗試進一步調整之前對滿足資源限制的偏序規劃進行檢驗是便宜和有效的。
- 時間是一種最重要的資源。它能被專門的排程演算法或者與規劃整合在一起的排程安排來處理。
- **分層式任務網路**(HTN)規劃可以讓代理人自領域設計者以**高層次動作**(HLAs)的形式取得建議，這些高層次的動作能夠透過更低層次動作順序以不同的方式被實作出來。HLA 的效果夠以**天使語意**予以定義，讓可證明正確高層次規劃被推導出來，而不用考慮更低層次的實作。HTN 方法能夠製作出許多的眞實世界應用所需的大型規劃。
- 標準規劃演算法假設有完備的和正確的資訊、確定性的和完全可觀察的環境。許多領域違反這個假設。
- **應變規劃**允許代理人在執行期間感覺世界以決定遵循規劃的哪個分支。在某些情況下，**無感測**或**一致性規劃**可以被用來構造一個不需要感知就能工作的規劃。無感測規劃和條件規劃都能透過信度狀態空間的搜尋進行構造。有效率的表示或是計算**信度狀態**是個關鍵的問題。
- 一個**線上規劃代理人**在需要時使用執行監控並結合修復，以便從起因於非確定性的動作、外來的事件或是錯誤的環境模型等等意想不到的情境中恢復。
- 當環境中有其他合作、競爭或協調的代理人時，**多代理人**規劃是必要的。聯合規劃能夠被建構出來，但是如果兩個代理人同意該執行哪一個聯合規劃時，就必須以某種形式的協調予以擴增。

- 本章延伸經典規劃以涵蓋不確定的環境(於此動作的結果不確定)，但是這還不是規劃的最後介紹文字。第 17 章描述隨機性環境(於此動作的結果具有機率性)的技術：馬爾可夫決策過程，局部可觀察的**馬爾可夫決策過程**，以及賽局理論。於第 21 章我們展示強化學習(reinforcement learninmg)可以讓一個代理人從過去的成功與失敗經驗中學習如何行爲。

參考文獻與歷史的註釋 BIBLIOGRAPHICAL AND HISTORICAL NOTES

DEVISER(Vere，1983)第一個處理了連續時間規劃。規劃中的時間表示法由 Allen(1984)及 Dean 等人(1990)在 FORBIN 系統中提出。NONLIN+(Tate 和 Whiter，1984)和 SIPE(Wilkins，1998，1990)能夠爲不同規劃步驟分配有限資源進行推理。O-規劃(Bell 和 Tate，1985)，一個 HTN 規劃器，具有對時間和資源限制的統一、通用的表示。除了正文中提到的日立公司的應用之外，O-規劃已經被應用於普萊斯·沃特豪斯公司(Price Waterhouse)的軟體採購規劃和美洲虎汽車的後輪軸裝配規劃中。

SAPA(Do 和 Kambhampati，2001)和 T4(Haslum 和 Geffner，2001)這兩個規劃器都使用複雜精密的啓發式的前向狀態空間搜尋來處理考慮持續時間和資源的行動。一個可選方案是使用非常有表現能力的行動語言，但需要人類寫下領域特定的啓發式來指導它們，ASPEN(Fukunaga 等人，1997)，HSTS(Jonsson 等人，2000)和 IXTET(Ghallab 和 Laruelle，1994)就是這麼做的。

許多混合規劃和排程安排系統已經被配備使用：ISIS(Fox 等人，1982；Fox，1990)已經用於威斯汀豪斯公司(我們 stinghouse)的加工車間排程，GARI(Descotte 和 Latombe，1985)規劃機械部分的加工和構造，FORBIN 被用於工廠控制，NONLIN+被用於海軍後勤規劃。我們選擇呈現規劃與排程爲兩個不同的問題；(Cushing 等人，2007)顯示在某些問題上這會造成不完整。在宇航領域的排程安排已經有了很長的歷史。T-SCHED(Drabble，1990)被用來排程 UOSAT-II 衛星的任務命令序列。OPTIMUM-AIV(Aarup 等人，1994)和規劃-ERS1(Fuchs 等人，1990)，兩者都基於 O-規劃，分別用於歐洲空間局的太空船裝配和觀測規劃。SPIKE(Johnston 和 Adorf，1992)用於 NASA(美國國家航空和宇宙航行局)的哈勃太空望遠鏡的觀測規劃，而太空梭著陸過程排程安排系統(Deale 等人，1994)進行多達 16000 名工人輪班的加工車間排程。當遠端代理人(Muscettola 等人，1998)，1999 年安裝在深空一號(Deep Space One)探測器上一起飛行時，它成爲第一個用於控制太空船的自主規劃器和排程器。空間應用已經驅動了資源分配演算法發展；請見 Laborie(2003)與 Muscettola(2002)。排程方面的文獻可見於一篇經典的調查文章(Lawler 等人，1993)，與新出版的書(Pinedo，2008)，以及一本編輯好的手冊(Blazewicz 等人，2007)。

STRIPS 程式中學習 **macrop**——由原始步驟序列組成的「巨集運算元」——的工具可以被認爲是第一個分層規劃機制(Fikes 等人，1972)。分層也被用於 LAWALY 系統(Siklossy 和 Dreussi，1973)中。ABSTRIPS 系統(Sacerdoti，1974)引入了**抽象分層**的想法，由此更高層的規劃被允許忽略行動的低層前提，以便得到可行規劃的一般結構。AustinTate 的博士學位論文(1975b)以及 Earl Sacerdoti(1977)的工作發展了現代形式的 HTN 規劃的基本想法。許多實用規劃器，包括 O-規劃和 SIPE，屬於 HTN 規劃器。Yang(1990)討論了使得 HTN 規劃有效率的行動特性。Erol，Hendler 和 Nau(1994，1996)提

出了一個完整的分層分解規劃器以及純 HTN 規劃器的複雜度結果的範圍。我們對於 HLA 所用的呈現方式與天使語意是取材自 Marthi 等人(2007，2008)。Kambhampati 等人(1998)已經提議一個分解只是另一個規劃調整的形式的方法，類似於非分層偏序規劃調整。

從 STRIPS 中大型運算子的研究工作開始，分層規劃的目標之一是以一般化規劃的形式複用以前的規劃經驗。第十九章中深入描述的**基於解釋的學習**技術已經在一些系統中用作對以前計算出的規劃進行一般化的手段，包括 SOAR(Laird 等人，1986)和 PRODIGY(車輛 bonell 等人，1989)。一個可選方法是用原始形式儲存以前計算出的規劃，然後透過對原始問題進行類推，複用它們以求解新的、類似的問題。這是被稱爲**基於案例的規劃**領域採用的方法(車輛 bonell，1983；Alterman，1988；Hammond，1989)。Kambhampati(1994)主張基於案例的規劃應該作爲一種調整規劃的形式來分析，並爲基於案例的偏序規劃提供了一個形式化的基礎。

早期規劃器缺乏條件和迴圈，但是一些規劃能夠使用強制來形成一致性規劃。司鐸 NOAH 使用強制解決「鑰匙和箱子」問題，這是規劃器對初始狀態知之甚少的規劃挑戰問題。Mason(1993)主張在機器人規劃中感覺總是能夠也應該能夠被省略，並描述了一個無感測器的規劃，它能根據一個傾斜行動序列把一件工具移到一個指定的位置，而不管初始位置在哪裡。

Goldman 與 Boddy(1996)引進**一致性規劃**這個名詞，請注意即使代理人有感測器，無感測器規劃還是常常有效的。第一個適度有效的一致性規劃器是 Smith 和 Weld(1998)的一致性圖規劃器或 CGP。Ferraris 和 Giunchiglia(2000)以及 Rintanen(1999)獨立開發了基於 SATPLAN 的一致性規劃器。Bonet 和 Geffner(2000)描述了一個基於信度狀態空間啓發式搜尋的一致性規劃器，利用了 20 世紀 60 年代首先爲部分可觀察馬爾可夫決策過程或稱 POMDP(參見第十七章)發展出來的想法。

目前，一致性規劃有三個主要方法。前兩個是於信度狀態空間之中使用啓發式搜尋：HSCP(Bertoli 等人，2001a)使用二元決策表(BDDs)來代表信度狀態，然而 Hoffmann 與 Brafman(2006)採用了一個使用 SAT 求解程式隨需的計算前提與目標測試的偷懶方法。第三種方式，主要受到 JussiRintanen(2007)支持，公式化整個無感測器規劃問題爲量化的布林公式(QBF)，並使用通用的 QBF 求解程式來求解。目前一致性規劃器較 CGP 快五個量級。2006 年國際規劃競賽於一致性規劃主題方面的贏家是 T0(Palacios 及 Geffner，2007)，它於信度狀態空間中使用啓發式搜尋的同時，藉由定義所推導出涵蓋條件性效果之文字，讓信度狀態的表示式保持簡單。Bryce 與 Kambhampati(2007)討論如何能夠一般化規劃圖以製造出好的一致性與應變規劃啓發式。

於文獻中，「條件性」與「應變」規劃這兩個用語之間一直有些混亂。依循 Majercik 與 Littman(2003)，我們使用「條件性」指的是一個會根據世界實際的狀態來決定不同效果的規劃(或動作)，而「應變」指的是一個代理人能夠根據感測的結果來選取不同動作的規劃。應變規劃問題在 McDermott(1978a)有影響力的文章《規劃與行動》(*Planning and Acting*)發表後受到了更多的關注。

本章描述的應變規劃方法是基於 Hoffmann 及 Brafman(2005)，並受影響於循環且或圖的有效搜尋演算法[由 Jimenez 和 Torras(2000)以及 Hansen 和 Zilberstein(2001)所開發]。Bertoli 等人(2001b)描述 MBP(模型型規劃器)，它使用二元決策表來做一致性與應變規劃。

回顧一下，現在有可能考察主要經典規劃演算法是如何通向涉及不確定性領域的擴展版本的。快速前向啓發式搜尋 through 狀態空間導致前向搜尋於信度空間(Bonet 與 Geffner，2000；Hoffmann 與 Brafman，2005)；SATPLAN 引領出隨機性 SATPLAN(Majercik 及 Littman，2003)以及量化二元邏輯規劃(Rintanen，2007)；偏序規劃引領出 UWL(Etzioni 等人，1992)與 CNLP(Peot 及 Smith，1992)；GRAPHPLAN 引領出感測器圖形規劃或稱 SGP(Weld 等人，1998)。

最早具有執行監控的線上規劃器是 PLANEX(Fikes 等人，1972)，它用 STRIPS 規劃器來控制機器人 Shakey。NASL 規劃器(McDermott，1978a)將規劃問題簡單地當作執行複雜行動的一個詳細描述來處理，這樣執行和規劃是完全統一的。SIPE(System for Interactive Planning and Execution monitoring，交互規劃與執行監控系統)(Wilkins，1988，1990)是系統地處理重規劃問題的一個規劃器。它已經應用於幾個領域的展示專案中，包括對航空母艦飛行甲板上的任務行動的規劃、一個澳大利亞啤酒廠的加工車間排程，以及多層建築的建造規劃(Kartam 及 Levitt，1990)。

於 1980 中期，由於對規劃系統執行時間的遲緩感到悲觀引領出稱之爲**反應式規劃**系統之反射型代理人的建議(Brooks，1986；Agre 與 Chapman，1987)。Pengi(Agre 和 Chapman，1987)能夠用結合了對當前目標及代理人內部狀態的「視覺化」表示的布林電路玩一個(完全可觀察的)視頻遊戲。「通用規劃」(Schoppers，1987，1989)是作爲反應式規劃的一種尋找表方法發展出來的，但是結果卻變成了對馬爾可夫決策過程中已長期使用的**策略**的想法再發現(參考第 17 章)。一個通用規則(或策略)包含從任何狀態到在該狀態中應該採用的行動的映對。Koenig(2001)以 Agent-Centered Search 爲名來調查線上規劃技術。

近年來多代理人規劃迅速進入大眾化，雖然它的確有很長的歷史。Konolige(1982)多代理人規劃的一階邏輯形式化方法，而 Pednault(1986)給出了一個 STRIPS 風格的描述。聯合意圖的概念(如果代理人準備執行一個聯合規劃則此概念是本質的)來自對通訊活動的研究工作(Cohen 和 Levesque，1990；Cohen 等人，1990)。Boutilier 與 Brafman(2001)展示如何讓偏序規劃適應多行爲者設定。Brafman 與 Domshlak(2008)想出一個多行爲者規劃演算法，它的複雜性的增長率與行爲者的數量呈現線性關係，只要耦合度(部份代理人之間的互動圖形之**樹的廣度**來量測)是有限度的。Petrik 與 Zilberstein(2009)展示一個方法根據雙線性程式設計勝過我們於本章所述的 cover-set 方法。

我們幾乎掠過了多代理人規劃協商研究工作的表面。Durfee 和 Lesser(1989)討論了如何透過協商在代理人中分攤任務。Kraus 等人(1991)描述了玩 Diplomacy(外交)遊戲的系統，這是一個需要協商、聯盟的形成與解散以及欺詐的棋盤遊戲。Stone(2000)顯示了代理人如何在機器人足球賽的競爭的、動態的和部分可觀察的環境中作爲隊友進行合作。有一篇近期文章，Stone(2003)分析兩個具有競爭性多代理人環境——RoboCup，一個機器人足球比賽，與 TAC，拍賣型交易代理人競賽——並發現我們目前理論上立論堅實之方法的計算互動性已經引申出許多以專門方法設計出的多代理人系統。

Marvin Minsky 引領風潮的「心智的社會」(Society of Mind)理論中，他(1986，2007)倡議人類心智是建構自代理人的合奏。Livnat 與 Pippenger(2006)證明，對於尋找最佳路徑的問題，並給予全部計算資源的限制，對於一個代理人最好的架構是子代理人的合奏，每個代理人努力最佳化它自己的目標，且所有的代理人都處於相互衝突狀態。

11.4.2 節的 boid 模型是基於 Reynolds(1987)，在 Batman Returns 一片中它應用於一群企鵝所表現的效果爲他贏得奧斯卡金像獎。Bryant 及 Miikkulainen(2007)描述了 NERO 遊戲及學習策略之方法。

多代理人系統方面的新書有 Weiss(2000a)，Young(2004)，Vlassis(2008)，與 Shoham 與 Leyton-Brown(2009)。自主代理人與多代理人系統(AAMAS)的年度會議。

❖ 習題 EXERCISES

11.1 迄今我們已經考慮過的目標都要求規劃器只能於一個單位時間內使得世界滿足目標。並非所有的目標都能夠以這個方式來表示：在地面上方懸掛一盞吊燈的目標，並沒有因你把它扔向空氣而達成。更嚴重地，你不希望太空船維生系統提供氧氣一天後隔日卻沒有了。當代理人的規劃會造成某個條件從一個已知的狀態之後持續地成立，一個維修目標即算達成。描述如何延伸這一章的正規化方法來支持維修目標。

11.2 你有好幾輛用之來運送一組包裹的貨車。每個包裹從格子地圖的某個位置出發，且有個目的地在某處。每輛卡車直接地以前進和轉彎等方式來控制。建構一個這個問題的高層次動作的階層。你的階層編碼進了哪些與解答有關的知識？

11.3 假設某個高層次動作恰有一個實作做爲原始動作序列。請給一個演算法用於計算它的前提與效果，完整的調整階層以及原始動作模式都已知。

11.4 假設一個高層次規劃的樂觀可到達集是一個目標集的超集合；關於規劃是否達成目標一事能歸納出什麼樣的結論？如果悲觀可到達集與目標集沒有交集該怎麼辦？請解釋。

11.5 寫出一個演算法，該法取一個初始狀態(由一組命題文字敘明)以及一個 HLA 序列(各由樂觀與悲觀可到達集的前提與天使規格所定義)並計算該序列之可到達集的樂觀與悲觀描述。

11.6 於圖 11.2 中我們已展示如何於一個排程問題中使用不同的 DURATION，USE，與 CONSUME 欄位來描述動作。現在假設我們想要以不確定性規劃來組合排程，它需要不確定性與條件效果。考慮三個欄位的各個欄位並解釋它們是否應該保持爲分開的欄位，或是它們應該變成動作的效果。爲三個欄位的每個欄位各舉一個例子。

11.7 標準程式語言中的某些操作可以作爲改變世界狀態的行動而建立模型。例如，指派值操作複製一個記憶體位置的內容，而列印操作改變輸出流的狀態。一個由這些操作組成的程式也可以被認爲是一個規劃，它的目標由程式的規格說明給出。因此，規劃演算法可以被用來構造一個實現給定規格的程式。

a. 寫出賦值運算子(將一變數的值指派給另一變數)的運算子模式。記住初始值將被涵蓋！

b. 說明規劃器如何使用物件建立來產生透過使用一個臨時變數交換兩變數值的規劃。

11.8 假設 *Flip* 動作一定會變更變數 *L* 的眞假值。展示如何藉由使用具有條件效果的行動模式來定義它的效果。證明，儘管使用有條件效果，一個 1-CNF 信度狀態表示經過 *Flip* 之後仍然是 1-CNF。

11.9 在積木世界中，爲了適當地保持 *Clear* 述詞，我們被迫引入了兩個行動 *Move* 和 *MoveToTable*。說明對於單個行動如何用條件效果表示這兩種情況。

11.10 有條件效果曾於眞空吸塵器世界的 *Suck* 動作上示範說明過——哪一個方格變成乾淨需視機器人所在的是哪一個方格而定。你能夠想出一組新的命題變數來定義眞空吸塵器世界的狀態，使得 *Suck* 有一個無條件的描述？用你的命題寫下 *Suck*，*Left* 和 *Right* 的描述，並證明它們足以描述世界的所有可能狀態。

11.11 找到一塊合適的髒地毯，沒有障礙物，用眞空吸塵器打掃它。盡可能正確地畫出眞空吸塵器採取的路徑。參考本章討論的規劃形式來解釋它。

11.12 對於上一道習題中的藥物治療問題，增加一個 *Test* 行動，當 *Disease* 爲眞以及任何情況下都有感知效果 *Known*(*CultureGrowth*)時，它條件效果 *CultureGrowth*。圖示一個解決問題並最小化使用 *Medicate* 行動的條件規劃。

本 章 註 腳

[1] HTN 規劃器常常可以讓調整變成偏序規劃，它們可以讓一個規劃中的兩個不同的 HLA 調整共享動作。我們不考慮這些重要的併發現象，以便有利於理解分層規劃之基本概念。

[2] 如果對於一個不確定性的問題循環性解答是必要的，AND-OR 搜尋必須被一般化成一個充滿循環的版本如 LAO* (Hansen 及 Zilberstein，2001)。

[3] 於 1954 年，美國阿拉巴馬州的 Hodges 女士的屋頂遭到隕石撞擊。於 1992 年，一塊 Mbale 隕石擊中小男孩的頭上；幸好，它墜落時撞到香蕉葉而減速(Jenniskens 等人，1994)。而於 2009 年，一個德國男孩聲稱被一顆豌豆大小的隕石打到手。這些事件都沒有出現重大傷害，表示為了對抗這樣的意外事件，而需要預先作規劃有時候是過於誇大的。

[4] 規劃監控意指，經過本書前面的篇幅之後，最後我們有了一個比蜣螂還聰明的代理人(見 2.2.2 節)。我們的代理人會注意糞球已經不在它的掌握中，並將重規劃以獲得另一個糞球來堵住它的洞。

[5] 規劃修補的無用迴圈正是黑足泥蜂所展示的行為(2.2.2 節)。

[6] 我們向英國居民道歉，那是個想到網球比賽就保證落雨的地方。