第22章 规 划

22.1 STRIPS规划系统

22.1.1 描述状态和目标

解决框架问题的一个比较好的办法是混合状态空间和状态演算方法。这种混合涉及到把描述一组世界状态的谓词演算公式想像成一种“状态”——就像在第7章讨论基于特征的状态空间时所做的一样。例如，在积木世界中，*B*在*A*上，*A*在*C*上及*C*在地面上（见图22-1）的世界状态可以描述为：

```
On(B, A)
On(A, C)
On(C, Fl)
Clear(B)
Clear(Fl)
```

这些公式描述了一组世界状态，因为它们被所有的那些状态满足，在这些状态中由公式意指的关系都是真的。如果有其他的积木，积木有其他的属性，如颜色，那么正被描述的状态将包括所有的那些东西，在它们中有这些其他的积木，积木有所有的可能颜色等等。

在本章描述的第一种规划方法中，把描述一组世界状态的公式作为一个能够用相应的agent动作改变的数据结构。将描述一些对这些数据结构的空间进行搜索的方法，以便发现一个结构，它描述了一组满足一个目标的世界状态。在一个对这些数据结构上的状态空间进行的搜索中，数据结构是规划过程的状态。为了区分规划状态和世界状态，一般把前者称为状态描述。

为了避免各种技术困难——在此我们不必考虑它们，把动作前后的状态描述限制为基本的文字合取（或集）。对这种限制允许一个例外，即状态描述能包含所有状态中的任何公式真值——像`On(x, y) (y=Fl)∨ ¬Clear(y)`。把目标约束为文字合取（可能存在的量化）。下面把（ *x*1, *x*2,...,*xn*）γ (*x*1, *x*2,..., *xn*) 形式的目标合式公式写成简单的γ (*x*1, *x*2, ..., *xn*)——假定存在所有变量的量化。尽管在多种方法中都有这些约束，但几个有趣的规划问题能被形成，并适当地求解。

给定一个目标合式公式γ，我们的搜索方法企图发现一个动作序例，它产生一个由某些状态描述s，如s ⊨γ 描述的世界状态，那么我们就说那个状态描述满足那个目标。对受限状态和目标合式公式 s ⊨γ，当存在一个s 替换时，这样γ s 就是一个基本的文字合取，它的每一个文字都出现在s 中。通过试图合一γ 中的第一个文字和s 中的一个文字，不管能否建立 s ⊨γ，都把合一置换应用到 γ 中的其他文字并继续对γ 中的所有文字进行（这个过程与被PROLOG解释器使

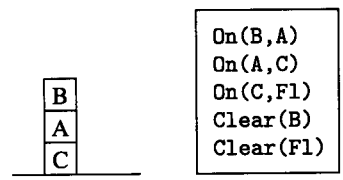


图22-1 一个状态描述

用的相同，PROLOG 解释器检查一个子句体是否与事实统一）。

我们可以用从开始到目标的向前搜索，或者用从目标回到开始的向后搜索来对一个状态描述空间进行搜索。有些作者把使用向前搜索的规划方法称为进步规划（*progression planning*），把使用向后搜索的方法叫回归规划(*regression planning*)。下面先讲向前搜索。

22.1.2 向前搜索方法

为了在状态描述空间上进行向前搜索，我们将需要与动作对应的算子，它把一个动作前状态描述改变成一个动作后状态描述。当它产生一个状态描述 s，使得对目标合式公式 g，有 $s \models g$，则搜索成功完成。我们的算子是基于一个叫做 STRIPS的系统[Fikes & Nilsson 1971, Fikes,Hart & Nilsson 1972]。一个STRIPS算子由三部分构成：

1) 集合*PC*，称为算子的前提条件（*preconditions*）的基本文字。相应于一个算子的动作只有*PC*中的所有文字都在动作前状态描述中时才能被执行。

2) 集合*D*，称为删除列表(*delete list*)的基本文字。

3) 集合*A*，称为加入列表(*add list*)的基本文字。

为了生成动作后状态描述，我们首先从动作前状态描述中删除 *D*中的所有文字，加入*A*中的所有文字。在*D*中没有提到的所有文字延续至动作后状态描述中。这种延续被叫做 STRIPS假设，它是解决框架问题的一种方法。

STRIPS算子常常用与动作模式相符的模式定义。一个称为 STRIPS规则的算子模式有自由变量，它是那些应于实际算子规则的基例。下面是STRIPS规则的一个例子，它带有自由变量x、y和z：

```
move(x, y, z)
PC: On(x, y)∧Clear(x)∧Clear(z)
D: Clear(z) On(x, y)
A: On(x, z), Clear(y), Clear(Fl)
```

(把公式Clear(Fl)显式地包括在加入列表中，因为如果 z是Fl，Clear (Fl)将被删除，而我们希望它总是真的)。这个STRIPS规则的一个实例（把积木 B从A上移到地面）显示在图 22-2中。

注意，一个STRIPS规则本身的前提条件是在一个叫做文字合取的目标形式中。这不是一个巧合且将被利用。当所解释的是一个目标合式公式时， *PC*中的自由变量被假定是存在量化的。一个STRIPS规则的实例能被应用到一个状态描述 s —— 如果*PC*（被作为一个目标）的一个基例被状态描述满足。如前面所讲，这样一个实例能用合一发现；反过来， *PC*中的每个文字在状态描述中有一个文字 —— 把合一置换应用到*PC*中的其余文字。然后通过把结果置换应用到STRIPS规则的所有元素，从而获得可用的算子实例。规则必须用这样一种方式书写，在这种方式中，应用这样一个置换总是产生规则的一个基例。即，在 *D*或*A*中不可有任何不在*PC*中出现的自由变量。图22-2是一个STRIPS算子应用的示例。

在向前搜索方法中，通过应用 STRIPS规则实例直到产生一个满足目标合式公式的状态描述，生成一个新的状态描述。在图22-3中，显示了由向前搜索生成搜索图的一部分。由于一个状态描述中的大部分文字不会被 STRIPS算子改变，通过只跟踪变化的部分，在向前搜索的实

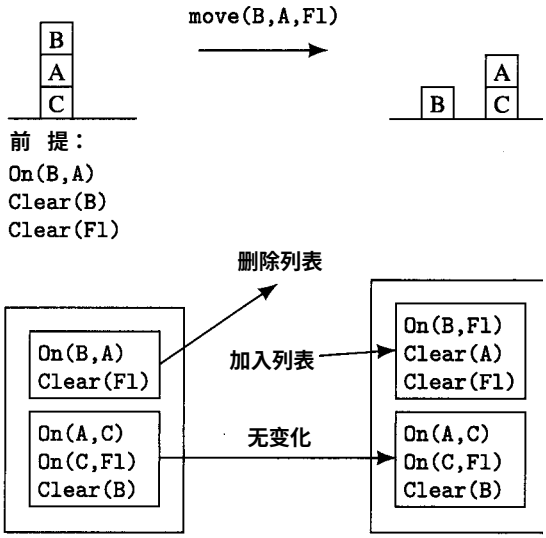


图22-2 一个STRIPS算子

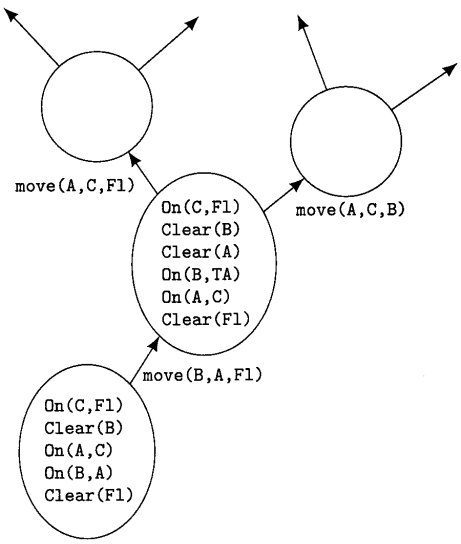


图22-3 向前搜索

能使用A* 搜索）。注意，和状态演算不同，我们不必证明动作留下了一些没有改变的关系，STRIPS假设已经考虑了这些问题。

没有关于应用哪个规则到哪个实例的启发性，向前搜索在那些状态描述和规则数量巨大的实际应用中是不现实的。使向前搜索更有效的一个方法是在搜索空间中确定岛，把到达目标状态描述的搜索路径集中地朝向岛。在下面的小节中，描述了一个标识和利用岛的方法。

22.1.3 递归STRIPS

当一个目标条件包含一个文字合取项时，像这里假定的一样，我们能通过一个分治启发式方法集中向前搜索。首先，获得一个合取项，然后另一个，等等继续下去。一个合取项被满足的状态描述表示搜索空间中的一个岛。我们先朝着那个岛前进，应用算子产生一个对应合取项被满足的状态描述，然后继续朝着另一个岛前进，等等。当然，可能会在获得随后的一个合取项时不能获得前面的合取项。

AI历史的早期开发了一个叫做通用问题求解程序（ *General Problem Solver*，GPS）[Newell, Shaw, & Simon 1959, Newell & Simon 1963]系统，它利用了这种方式下的搜索空间岛。到达这些岛的算子由一个叫做“中间结局分析（*means-ends analysis*）”的过程确定。当应用到使用STRIPS规则的系统时，这个技术涉及到选择目标条件中合取项之一的一个实例和选择到达目标的一个STRIPS规则的一个实例。接下来，我们为应用那个规则的前提条件建立一个子目标，用确定孤岛的同样方法递归地朝着那个目标前进直到它被满足；然后，应用算子建立孤岛状态描述并继续标识另一个孤岛，等等。这个过程是我们称做的 STRIPS递归程序的基础。下面是STRIPS如何求解一个合取目标公式γ：

STRIPS（γ）：这个过程使用包含一个基本文字集合的一个全局数据结构 δ。这个数据结构开始时设置为初始状态描述，并在过程进行中或之前改变。

1) repeat STRIPS的主要循环是迭代和继续直到产生一个满足目标 γ 的状态描述。第9步

的终止测试产生一个置换σ（可能是空的），以便γ σ 的一些合取项（可能没有）出现在第8步中。在执行测试中可能有几个试验置换，因此测试是一个可能的回溯点。

2) g←γ σ 的一个元素，以使 $S_1 \neq g$。这是另一个选择，因此也是一个回溯点。在“中间结局分析”方法中，g被作为一个“差别”，它必须被“缩小”以到达目标。

3) f ← 一个STRIPS规则，它的加入列表包含一个文字λ，它把g和mgu s统一起来。由于可能有几个这样的规则，这是另一个回溯点。f是一个与减少差别“有关”的算子。

4) f ' ← fs，使用置换s的f实例。注意，f ' 不必是一个基例，因此它的前提条件可以包含变量。

5) p ← f ' 的前提条件公式（用置换s实例化）。

6) STRIPS（p），一个递归调用以产生一个满足子目标的状态描述。这个调用通常会改变 S。

7) f ' ← 可以在8中应用的 f ' 的一个基例。

8) S←把f″应用到 S 的结果。注意 S 总是包含一个基本文字合取。

9) until$d \models \gamma$

注意，在第9步，我们总是相对整个目标γ进行测试。如果γ的一个合取项在到达另一个合取项的过程中没有到达，那么在过程终止前第一个将必须被重新到达，或者回溯该过程，以寻找一条能到达第一个合取项的一条路径。

虽然在算法描述中没有明确提及，但构成一个计划的目标算子序列能从那个算法的成功执行记录中提取出来。它们依次对应agent能执行的动作，或者是“发射导弹似地（*ballistically*）”（在适当的情况下），或者在一个感知/计划/动作循环中。

下面的例子演示了算法是如何工作的。假如开始有图 21-1中描述的状态，我们想获得目标 On (A, Fl) ∧ On(B, Fl)∧On (C, B)。第9步的目标测试没有发现任何被初始状态描述满足的合取项。假定STRIPS选择On（A，Fl）作为g，规则实例move(A, x, Fl)在它的加入列表中有On(A,Fl)，因此我们递归地调用 STRIPS获得规则的前提条件 Clear (A) ∧Clear (Fl) ∧ On(A, x)。递归调用中第9步的测试产生置换C/x，它留下了除满足初始状态的Clear(A)以外的所有合取项。选择这个文字，并选择规则实例 move（y，A，u）以获取该文字。

再次递归调用STRIPS以获得规则的前提条件Clear (y) ∧ Clear (u) ∧ On(y, A)。这个第二次递归调用中的第 9步测试产生置换B/y和Fl/u，它使前提条件中的每个文字成为初始状态中出现的文字。因此，我们能退出第二个递归调用，并应用一个算子 move (B, A, Fl)，这把初始状态改变成：

```
On(B, Fl)
On(A, C)
On(C, Fl)
Clear(A)
Clear(B)
Clear(Fl)
```

现在，回到第一次递归调用，再次执行第9步的测试（即Clear (A) ∧ Clear (Fl) ∧On (A, x)）。这个测试被我们用置换C/x改变的状态描述所满足，因此退出第一个递归调用应用算子move (A, C, Fl)，产生第三个状态描述：

On(B, Fl)
On(A, Fl)
On(C, Fl)
Clear(A)
Clear(B)
Clear(C)
Clear(Fl)

现在再次执行主程序中的第9步测试。在初始状态中惟一没有出现在新状态描述中的合取项是On(C, B)。再次从主程序递归，我们注意到move(C, Fl, B)的前提条件已被满足，因此应用它生成一个满足主目标的状态描述，过程终止。

22.1.4 带有运行时条件的计划

如果在计划执行期间能通过知觉处理精炼那些状态描述，我们就能让状态描述中允许的各种公式稍稍一般化。考虑在一个状态描述公式中允许文字析取意味着什么。为了清楚地说明，假定在一个状态描述中有合式公式 On(B, A) ∨On(B, C)。这个合式公式的预期含意是*B*在*A*上，或者*B*在*C*上，但是系统不知道当时正在产生哪个计划。一些 STRIPS算子对这样一个状态描述的应用及这些算子的结果可能并不依赖于 On(B, A)被满足还是On(B, C)被满足。对这些算子而言，在状态描述中有一个析取式是无所谓的，析取式将仅传给动作后的状态描述。

但是有的算子可能把On(B, A)或On(B, C)作为一个前提条件。为了执行和这种算子对应的动作，系统必须知道在执行动作时（即在运行时）哪一个析取项被满足。也许知觉过程能做出这个决定。如果这样，一个运行时条件能在这样一个算子的应用点被插入那个计划中。在构造随后的运行时条件计划中，计划过程分割成与满足算子前提条件的析取项一样多的分枝，每个分枝保持状态描述和算子的一个独立线路。在例子中，一个分枝将假定当执行随后的动作时，On(B, A)是（或将是）世界的真值，另一个将假定 On(B, C)是（或将是）世界的真值，这些分枝中的每一个被称为一个上下文。在制定计划时，系统可能不知道在上下文分离时哪个状态描述描述了世界的真正状态，但它们中肯定有一个是。对每个上下文构造可选的完成计划，然后，在运行时当系统碰到分裂成两个或更多上下文时，知觉过程决定哪个析取项是真的，执行过程就顺着合适的路径向前推进。在复杂问题中，可能有几个分枝上下文。

有时一个知觉过程能够决定在需要信息时哪个析取项是真的，这种情况仅作为前面动作的结果发生。更典型地讲，系统必须通过使用与信息收集动作相应的算子进行计划以获得那个信息。在例子中，为了决定 On(B, A)与On(B, C)哪一个是真的，系统可能必须执行能读出积木上字母的动作。这种动作能用 STRIPS算子描述，算子的结果包含一个正式指定“知道哪一个”的方法。然后运行时条件让这些动作知道哪一个是前提条件。

22.1.5 Sussman异常

考虑图22-4中显示的将STRIPS规则递归地应用到积木世界的问题。目标条件是 On(A, B)

∧On(B, C)，STRIPS必须首先获得一个合取项。假定它选择了On(A, B)，并把C从A上移到地板上，然后将A移到B上。但在获得另一个合取项On(B, C)时，它将取消On(A, B)且必须再次到达它。假定它首先选择了On(B, C)。它把B移到C上，但在获得另一个合取项 On(A, B)时，它将取消On(B, C)且必须重新到达它。在一个计划中没有一个可选结果有最少数量的算子。这个特殊的积木问题由Sussman提出 [Sussman 1975]，之后被叫做Sussman异常。

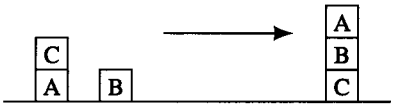


图22-4 Sussman异常

这个问题中递归STRIPS的困难是它用一种深度优先方式，每次仅仅集中在一个合取项上。如果向前搜索使用一个广度优先策略，它将发现最短计划。但是如已指出的那样，在实际问题中广度优先向前搜索在计算上是不可行的。

一种可选方案是尝试一个广度优先的向后搜索，也许从计算上讲，它比向前搜索效率更高，因为目标合式公式中的合取项比初始状态描述中的基本文字要少一些。下面将讨论向后搜索。

22.1.6 向后搜索方法

我们能用STRIPS规则从目标状态向后搜索。为此，必须通过 STRIPS规则倒退目标合式公式，产生子目标合式公式。把公式 γ通过一个STRIPS规则α倒退到最弱公式γ′，以便如果γ′在应用一个α实例前被一个状态描述满足（且γ′满足α的那个实例的前提条件），那么γ被应用α的那个实例后的状态描述所满足（如果 $\phi_2 \mid = \phi_1$，则公式ϕ_1比ϕ_2弱。因此P∨Q比p弱，P比P∧Q弱）。

如果用一个基本文字合取式开始，且回溯仅通过STRIPS算子（STRIPS规则的基例），那么倒退计算是直接的。在这种情况下，所有的倒退将也是基本文字的合取式。在图 22-5中，显示了一个例子，它用该方法从一个目标描述向后搜索。此处的问题是从状态 B在A上、A在C上、C在地面上，到达状态A在B上、B在C上、C在地面上。在这个问题中，通过得到一个目标合取项的任何算子倒退目标On(C, Fl) ∧ On(B, C) ∧ On(A, B)。假设通过move(A, Fl, B)倒退产生一个子目标合式公式，算子获得一个合取项 On(A, B)，因此On(A, B)不必在子目标中，但是不在目标描述中的任何算子的前提条件必须在子目标中。在这种情况中，有 Clear(B)、Clear(A)和On(A, Fl)。在目标合式公式中的另两个合取项On(C, Fl) 和 On(B, C)既没被算子加入也没被破坏，因此它们仅仅通过算子传递到子目标中，如图 22-5中所示。另一种向后搜索方法是通过move（B，Fl，C）倒退目标合式公式。向后搜索继续进行（也许使用某个类似 A*的方法）直到产生一个被当前状态描述满足的子目标。

如果一组文字通过一个删除其中有这些文字之一的算子倒退，注意会发生什么。如果一个文字λ被一个算子删除，则该算子没有任何方法可以获得文字 λ！因此，任何包含λ的合取式通过一个删除了λ的算子进行倒退都是F，当然，搜索没有必要从一个包含 F的状态描述（这样一个状态是不可能到达的）向后跟踪。

通过一个没有完全实例化的规则倒退一个目标常常是明智的——也就是说，通过一个仍然包含着一些模式变量的规则。在图 22-5中，我们考虑用move（A，Fl，B）作为到达On(A, B)的一个方法。为什么决定从地面移走 A呢？的确，我们必须从某个地方移动它，但是为什么确定要从地面移动它呢？也许有更好的计划将它从其他某个地方移动。延迟这个决定的一种方法

是把“从……地方”留给移动算子暂时不指定。这样做是所谓的最小承诺规划（*least commitment Planning*）的一个实例。通过部分实例化的算子 move(A, x, B) 倒退目标，留下“从……地方”不去指定。在图 22-6中，给出了这个倒退的结果。除了现在的一个前提条件 On(A, x)有一个变量外，这个结果更像早期得到的结果。如果这个文字与一个状态描述中的一个基本文字相一致，它就能被满足，可以解释为一个子目标。通过一个包含变量的子目标空间的向后搜索必须允许附加的算子对变量进行实例化——用其他变量或者用常量。一个启发式搜索将延迟实例化直到包含它们的文字能与初始状态描述中的文字合一。

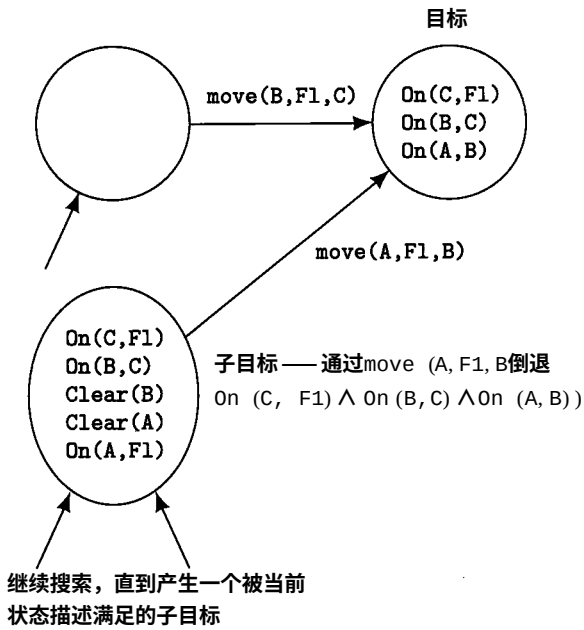


图22-5 通过一个STRIPS算子倒退一个合取式

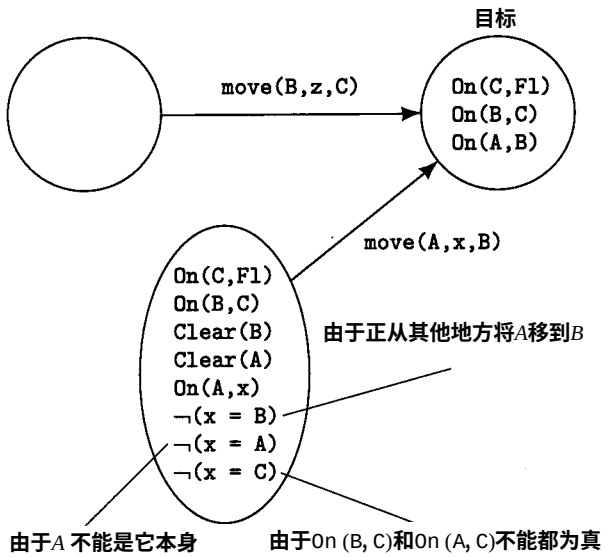


图22-6 通过一个包含一个变量的STRIPS规则倒退一个合取式

有时可以用各种推理过程来约束变量。在图22-6所示的例子中，On(A, x)中的变量x不能等于A，因为它是我们正在移动的积木*A*。x也不能是B，因为我们正把*A*从另外的地方移向*B*。x也不能是C，但推理就有点微妙了。如果x是C，我们把On(A, C)作为子目标一个的合取项。但是，目标中的On(B, C)不能倒退子目标的On(B, C)，因为*B*和*A*不能同时在*C*上（注意，用在递归调用STRIPS中的子目标不同于由倒退产生的子目标！）。

倒退目标的全过程和通过没有完全实例化的 STRIPS规则产生子目标是相当复杂的——在[Nilsson 1980 pp.288以后]中解释得更详细一些。图22-7是一个向后搜索的例子。在这个例子中，变量通过推理被实例化为Fl，因为它们的约束排除了惟一可能的选择实例。当产生一个满足初始状态描述的子目标时搜索终止。连接主目标和满足初始状态的子目标的序列能被读出来并被实例化为一个获得目标的计划算子，该计划由执行搜索中发现的实例化生成。[Nilsson 1980, pp 292 ~ 296]给出了一个更复杂的例子。

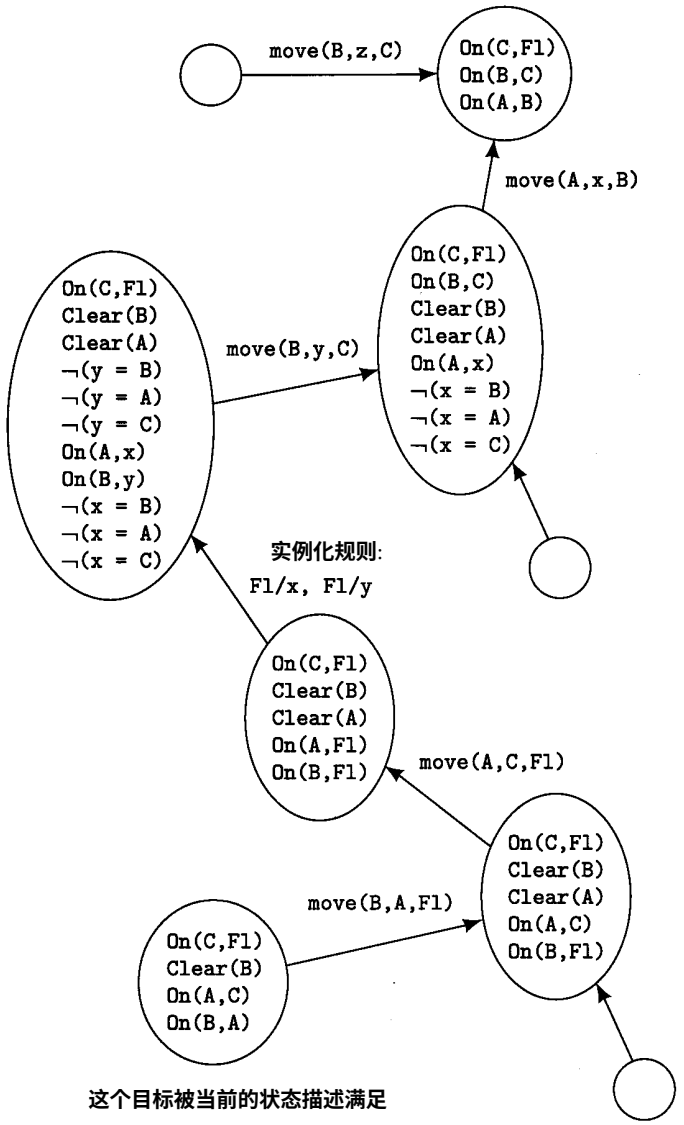


图22-7 向后搜索

使用倒退方法的向后搜索，虽然可能比向前搜索更有效，但由于出现变量而复杂化了。在小的积木问题中，变量能被实例化而不会有太多的困难，但是在很大的、没有使用指定领域启发的问题中，该方法在计算上是否可行是值得怀疑的。

在广度优先向前或向后搜索中，在任何特殊顺序中，没有承诺保证到达目标合取项。Sussman异常是一种问题的一个例子，在这种问题中，最好把解决每个合取项需要的步骤进行交叉。延迟提交最终规划中的步骤顺序是最小承诺规划的另一个实例，它能通过在一个规划空间而不是公式空间中进行搜索而获得最佳。在“计划空间”中，一个计划的步骤可以部分有序。在下一部分讨论这个叫做部分有序规划(*partial-order planning*，*POP*)的计划策略（部分有序计划有时也叫做非线性计划）。

22.2 计划空间和部分有序规划

图22-8中给出了两种不同的计划产生方法，当在一个公式空间中搜索（向前）时，STRIPS规则被应用到公式集合以产生它们的后继，这样一直进行，直到产生一个满足目标公式的状态描述。但是在计划空间中搜索时，后继算子不是 STRIPS规则，而是能把不完全的、未实例化的或其他不合适的计划转化为表达力更强的计划的算子，等等，直到产生一个能把初始状态描述转换为一个满足目标条件的状态描述的执行计划。在计划空间搜索中的算子能通过各种方式把计划转换为其他的计划。这些包括： (a)给计划加一些步骤；（b）对已在计划中的步骤重新排序；（c）把一个部分有序的计划改变成一个完全有序的计划；（d）把一个计划模式（带有未实例化的变量）改变成那个模式的某个实例。图 22-9给出了这些计划转换算子的一些例子。

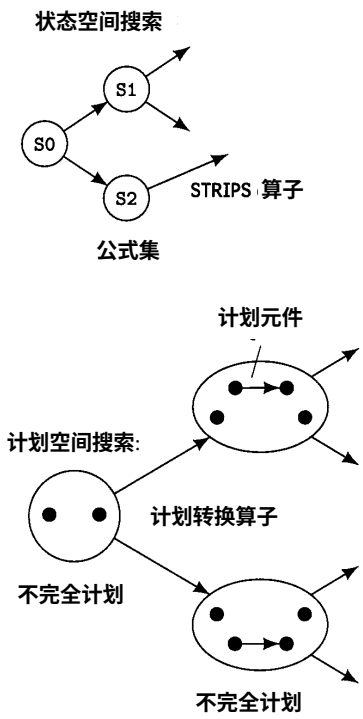


图22-8 状态空间与计划空间搜索

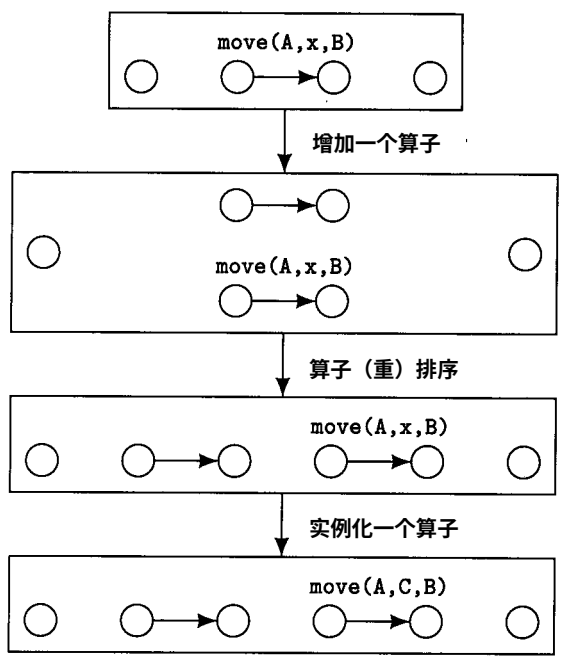


图22-9 一些计划转换算子

让我们说明一下应用到 Sussman异常（图 22-4）的计划空间搜索方法。描述基于[McAllester & Rosenblitt 1991]系统的非线性计划（SNLP）技术和[Tate 1977]的NONLIN。一个计划的基本构成是STRIPS规则，它们被有两类节点的图结构表示：随圆型节点用 STRIPS规则的名字标识，长方形节点用那些规则的前提条件和结果标识。这样一个图结构显示在图 22-10中。注意，图顶部的长方形是结果。底部的长方形是前提条件。

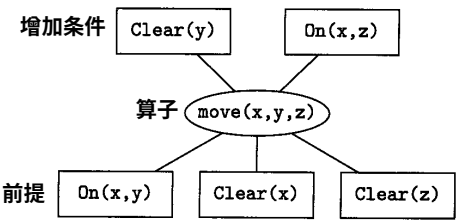


图22-10 一个STRIPS规则的图形表示

我们通过假想的、称为 finish和start的规则图表示目标合式公式和初始状态的文字。规则finish用总目标作为它的前提条件，它的结果是 nil。算子start用初始状态描述中的所有文字作为它的结果；它的前提条件是 T。在图22-11中示例了 Sussman异常的这两个图。

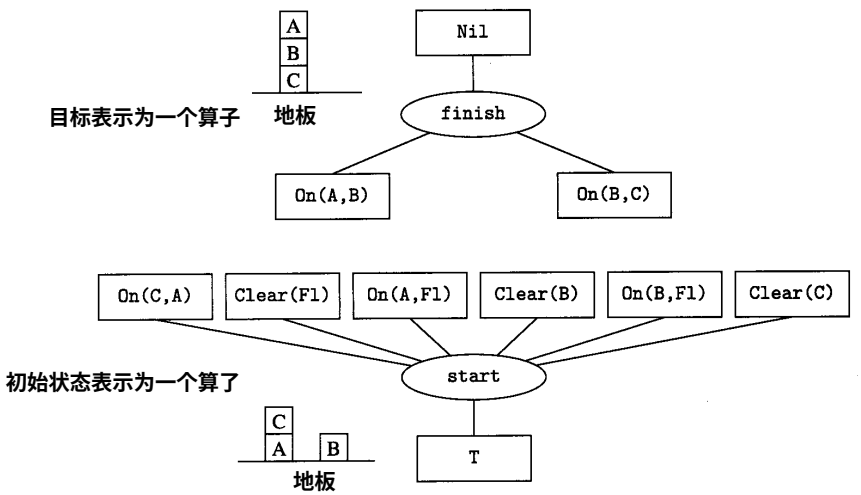


图22-11 finish 和start 规则的图形表示

初始计划结构（在我们应用任何计划转换算子之前）仅包括图 22-10所示的（没有连接的）start和finish规则。当然它是相当不完全的（将定义它对一个后面要完成的计划的意义）。

现在一个计划的搜索通过把一个计划转换算子应用到初始计划结构开始，最小承诺计划者用各种方法选择计划转换算子。在这个阶段可能被选择的一个转换通过增加一个规则以到达目标的一个合取项来扩展初始计划结构。假定我们决定通过增加规则实例 move(A, y, B)以到达On(A, B)。把这个规则的图结构加到初始计划结构中，由于已决定这个规则的加入会到达 On(A, B)，我们把这个规则的结果框与finish规则相应的前提条件框相连接，其结果如图22-12所示。注意我们还没有承诺A从那个位置移走。

在这个阶段可能有几个计划转换。我们能把 y实例化为Fl，并把它放在一个在实例化的move算子前提条件On(A, Fl)和在初始状态为真的On(A, Fl)之间的连接中。我们也能放进一个在两个Clear (B)之间的连接中，或者设法建立 move(A, y, B)的Clear (A)前提条件，这个条件能通过插入规则move (u, A, v)（移动A上的某个东西（u的一个实例）到另外某个地方（v的一个实例）建立。然后把u实例化为C，v为Fl，并建立与初始状态中的公式相对应的连接。这一系列计划变更步骤的结果将产生图22-13中的计划结构。

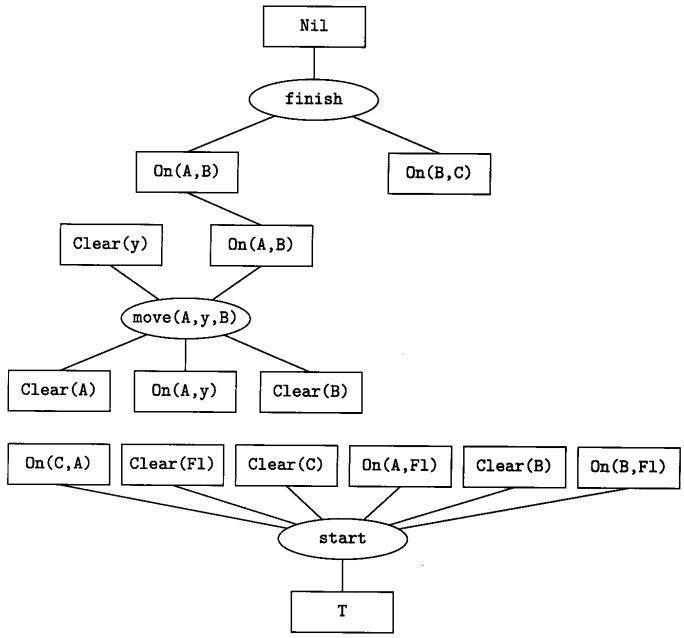


图22-12 下一个计划结构

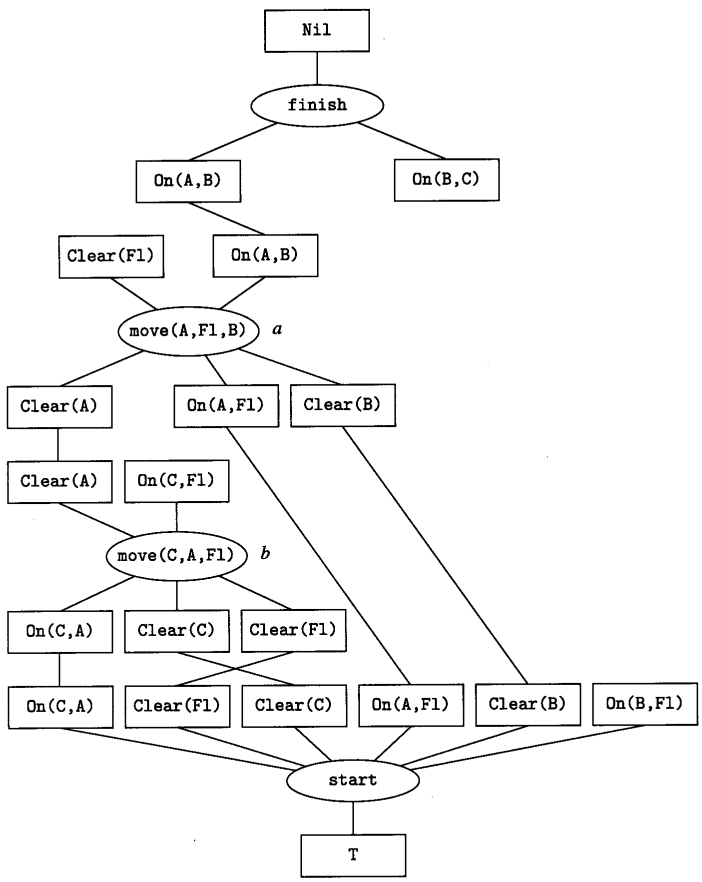


图22-13 一个接下来的计划结构

在这个阶段，我们看到两个 move算子的前提条件在算子将被应用时得到满足。在表示不完全计划的图结构中有那两个 move算子的一个隐含排序。在接下来关于计划过程的描述中，会引用这两个move算子，因此，在图22-13中用*a*和*b*标识它们。用符号*b*<*a* , < 代表“在……之前”，明确表达了*b*必须出现在*a*的前面。注意两个长方形框On(C, Fl)和Clear(Fl)，它们是move算子的“产品”，但没有被随后的算子“消耗掉”。只要它们和后面的计划中必须为真的条件不矛盾，它们就不会妨碍计划。

假定下面我们考虑如何到达另一个主要目标合取项On(B, C)。这个条件能用规则实例move(B，z，C）实现，move(B, z, C)是指把*B*从某个位置移到*C*。因此我们能把这一步加到计划结构中，然后假定把z实例化为Fl，以便我们能连接到初始条件。产生的计划结构如图22-14所示。

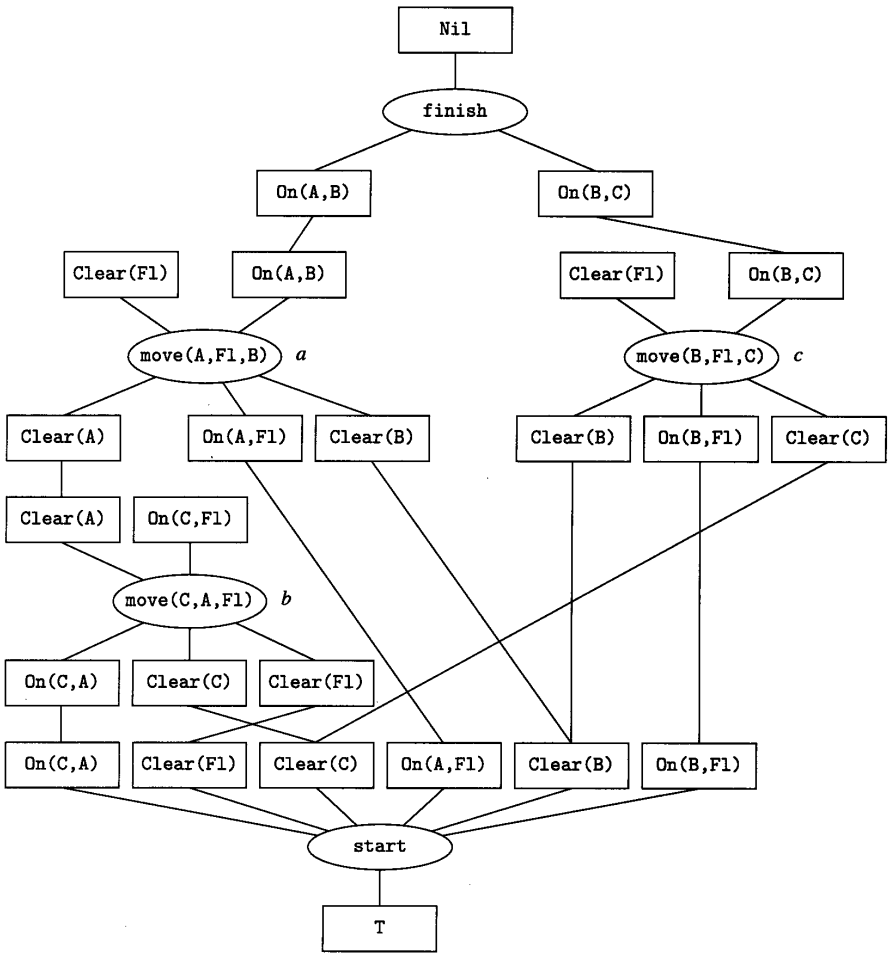


图22-14 计划结构的一个后期阶段

现在我们有一个涉及到算子*a*、*b*和*c*的计划结构，它仅仅是部分有序的。到目前为止，我们对*c*在计划步骤序列中的什么地方发生没有做出任何承诺。在部分有序计划中，一个算子如果在错误的时间被执行了，它可以取消另一个算子需要的前提条件，我们需要考虑由上述事实引起的问题。这种可能性在计划结构图中用威胁弧（*threat arc*）表示。考虑图22-15中的图结构。那些粗灰色弧线是威胁弧。威胁弧从算子（椭圆形的）节点画向那些前提条件（长方形的）

节点，它们是（a）在那个算子的删除列表上；（b）不是那个算子节点的后继。因此，在图22-15中，算子move（A，Fl，B）删除了move(B, Fl, C)的一个前提条件Clear(B)。在这种意义上，move(A, Fl, B)威胁move (B, Fl, C)，因为如果前者被首先执行，我们将不能执行后者。威胁弧必须通过给算子排序加上一些约束才能被消除。一个计划直到我们能发现一个一致的、能抛弃所有威胁弧的排序约束时才能完成。在例子中，这样的一组一致排序约束是（ $c< a, b <c$）。当然，图结构本身暗示了约束（ $b< a$） —— 它和其他的排序约束一致。因此，图22-15中的计划结构产生了总的排序：（$b< c< a$）。因为所有的威胁弧被消除了，这个计划也就完成了，所有被算子（包括finish算子）要求的条件在算子被应用时得到满足。最终的计划是{move(C, A, Fl), move (B, Fl, C), move (A, Fl, B)}。

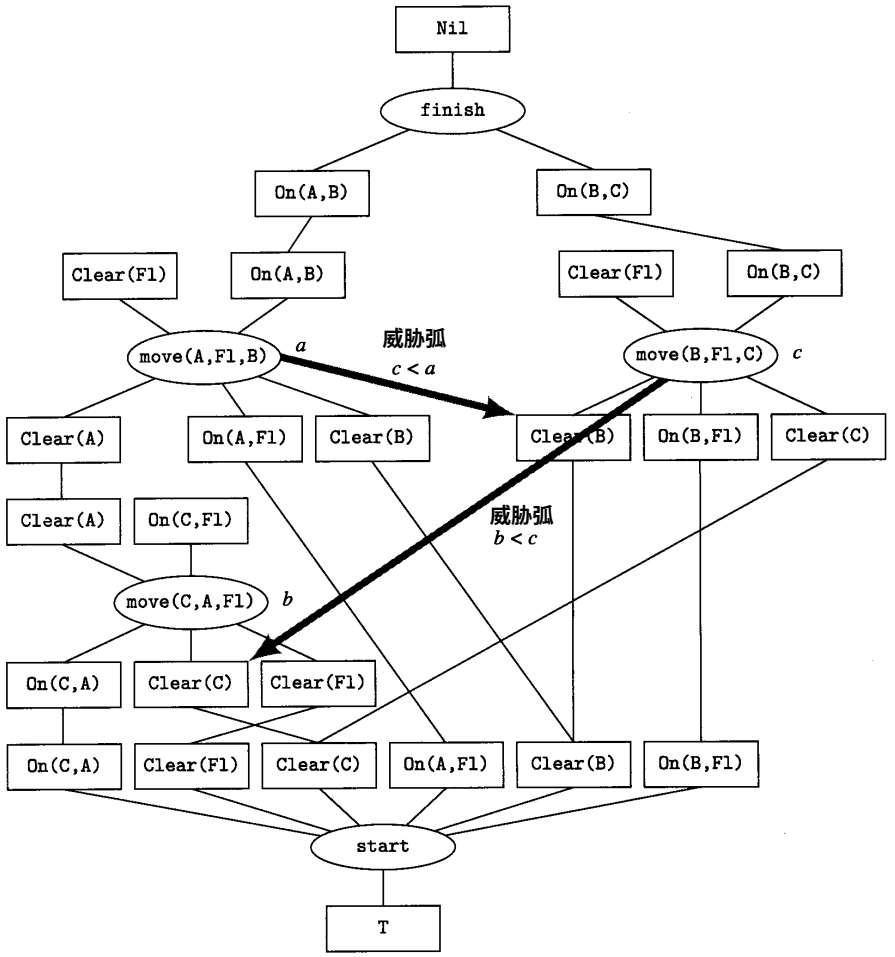


图22-15 插入威胁弧

22.3 层次规划

22.3.1 ABSTRIPS

第10章已提到过层次搜索过程。几个研究人员已经开发了基于 STRIPS规则的一些层次计

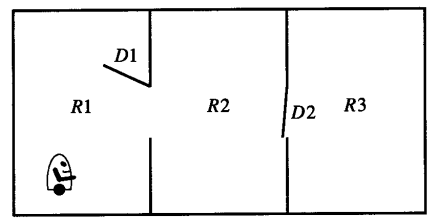
划者。一个是ABSTRIPS系统[Sacerdoti 1974]，它给一个STRIPS规则中每个前提条件的每个合取项分配关键度编号。到达一个合取项越容易（所有其他事情都相等），它的关键度编号越低。在ABSTRIPS中，规划在使用这些关键度编号的级别中进行。

1) 假定关键程度小于某个阈值的所有前提条件都已为真，生成一个基于该假设的计划。这里，除了最难的合取项，我们基本延迟了所有合取项的到达。

2) 把阈值减1，用第1步中产生的计划做向导，假定关键度比阈值低的前提条件都为真，生成一个计划。

3) 如此继续。

用一个例子解释ABSTRIPS的思想，这次用一个必须从一个房间移到另一个房间的机器人作例子。在图22-16中定义了初始状态、目标和要用的规则。规则goto（r1, d, r2）是机器人从房间*r*1，通过门*d*到达房间*r*2的动作模式的模型。规则open(d)打开门*d*，关键度编号用圆圈显示在这些规则的前提条件的相应文字上。特别地，假定打开一个门相对于通过一个门要容易一些，open(d)的关键度值是1。



初始状态：In(R1)∧Open(D1)∧Closed(D2)

目标：In(R3)

STRIPS 规则

goto(r1,d,r2)

② ① ②

PC: In(r1)∧Open(d)∧Connects(r1,d,r2)
D: In(r1)
A: In(r2)

open(d)

①

PC: Closed(d)
D: Closed(d)
A: Open(d)

图22-16 ABSTRIPS的一个规划问题

首先，我们构造一个抽象计划（用已经讨论的任何规划生成方法） —— 假定关键度为1的所有前提条件已被满足。在这个抽象级别到达In(R3)的计划是{goto (R1, D1, R2), goto (R2, D2, R3)}。接着，我们构造一个更详细的计划来首先到达抽象计划中第一个算子的前提条件，然后应用那个算子，再到达抽象计划中第二个算子的前提条件，这样一直进行下去。这些算子的前提条件能被看作为搜索空间中的孤岛——这些岛被第一级的抽象计划过程所发现。但现在，当我们到达这些前提条件时，将关键度阈值减1以要求我们也到达所有关键度为1的前提条件。结果是产生计划{goto (R1, D1, R2), open(D2), goto (R2, D2, R3)}。一般地讲，我们有多于两个的关键度值，规划过程将随着几个抽象级别下降。

ABSTRIPS使用的计划生成过程是所谓的长度优先（*length first*）。即在下降到一个更详细的级别以生成一个完全的计划之前，我们在每个抽象级别生成一个完整的计划。对长度优先计划生成有另一种选择。例如，我们能在顶级生成一个完整的计划标识搜索空间中的一个孤岛序列。这些孤岛将是各种第一级算子的前提条件。然后，我们能在下一级生成一个计划的第一部分，它仅仅完成到第一个孤岛，依此类推，直到我们在最低一级有了一个完全的计划，它到达了那个级别的第一个孤岛。假定在详细的最低级别，计划中的第一个算子和一个能在初始状态下执行的动作相对应。深度优先规划过程适合于使用感知 /计划/动作循环的情况。在执行第一个动作且感知到结果状态后，我们通过重新规划重复那个过程 —— 也许用前一个计划的一些更抽象级别作为指导。

层次规划是第9章描述的技术的一个实例，为了获得一个用在实际问题中的启发式函数值，首先要求解一个问题的简化版本。一个抽象规划过程的每一级能被看做下一级的一个简化版本（[Pearl 1984, pp.131~132]谈到了他的简化模型建议和ABSTRIPS之间的关系）。

22.3.2 层次规划和部分有序规划的组合

规划系统NOAH、SIPE和O-PLAN[Sacerdoti 1977, Wilkins 1988, Currie & Tate 1991]，组合了部分有序、计划空间规划和层次规划。除了已经提及的计划空间算子（实例化变量、加入STRIPS规则等等），这些系统还包括一些算子，可以把抽象计划清楚地表达为更低级的详细计划，这些清楚表达能被连接到前面考虑过的计划空间算子中。例如，用在积木问题中的 move 规则可能由更原始的规则pickup和putdown 序列构成，这些更低级的规则通常有更详细的前提条件，它们将要求插入其他更低级的规则以使计划在那个级别完成（见图 22-17）。这个过程将继续，直到所有的算子（或至少第一个算子）与能被执行的原始动作相对应。

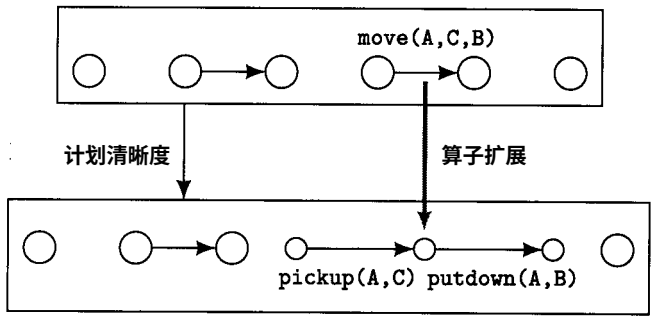


图22-17 清楚表达一个计划

22.4 学习计划

如同在第17章中用EBG来学习一个推理系统的新规则一样，也能用它学习包括一系列已经存在的STRIPS规则的新规则。然后用这些学习规则来建立更复杂的计划。当然，我们面临着前面提到过的相同应用问题。一个学到的规则不值得保存，除非它经常使用，节省规划的工作用起来不能是昂贵的。通过一个演示例子解释学习新规则的技术。

考虑改变图22-18中积木配置的问题。开始时，*A*在*B*上，*B*在*C*上，我们想让*A*和*B*都在地板上。到目前为止讨论的任何规划方法都将产生计划 {move(A, B, Fl), move(B, C, Fl)}。在一个新的STRIPS规则形式中，这个计划值得保存吗？如果我们的 agent经常必须解决这个指定的问题，它就值得。如果这个计划能被一般化，以便它从一个积木堆上把顶部的两个积木（不管它们的名字）移到相同的目标地方，那么这个计划会有更多的应用。由于计划的构造不依赖积木的名字，将会呈现出一个象EBG一样的过程能产生一个带有变量符号的计划模式，变量符号能用对象常量 A、B和C代替。但是由于这个两步计划依赖目标位置地板，故对象常量 Fl不能被一般化。

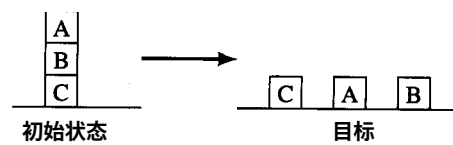


图22-18 分拆两个积木

计划一般化过程使用有关计划中算子的前提条件和结果的信息。表达这个信息的一个便利方法是用三角表形式[Fikes ,Hart & Nilsson 1972]。图22-19是一个拆积木问题的表。表的每一列用计划中的一个算子做标题——以这些算子在计划中出现的次序为序。就像在部分有序规划

中，很容易想像假想的start和finish算子。该表显示了每个算子的前提条件和每个算子的目的，每个算子下的单元包含一些由算子加入的文字（可能是重复的）。每个算子左边的单元包含那个算子的前提条件文字。我们用单元在表中的行位置 i和列位置j对它们进行索引，表顶部的行值为1,表左部的列值为1。特别地，单元（i, j）($i \geq j$)包含了由算子i加入的文字，算子i被用做算子$j+1$的前提条件，它也是在i和$j+1$之间幸存的算子应用。因此，start算子下面有一些单元，它们包含了出现在初始状态描述中的文字以及被随后的算子需要的或满足目标条件中的合取项的文字。finish算子在它的左边有一些单元，它们包含了满足目标条件的文字。

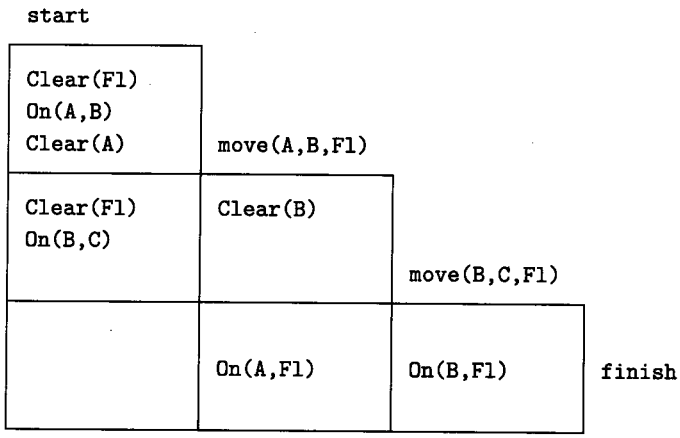


图22-19 拆积木的一个三角表

计划一般化过程的下一步是用变量符号代替所有的三角表中的对象常量。一个对象常量的每次出现被它自己的变量符号所代替。然后过程设法找到结果计划模式最一般的实例，以便所有规则的前提条件在应用一个规则实例时被满足。从第一个规则开始检查每个规则的前提条件，为了满足前提条件，还要做一些置换工作。图22-20给出了这个问题的结果三角表模式。

在三角表模式中，一般化的计划模式被称为一个 MACROPS（为宏算子），它能被用做一个STRIPS规则来建立更长的计划。第1列的文字合取式是MACROPS的前提条件，第3行的文字是加入列表中的文字（为了计算删除列表，需要分析每个规则的删除列表）。[Fikes, Hart & Nilsson 1972]中讨论了一般化过程本身和使用三角表监视计划的执行。

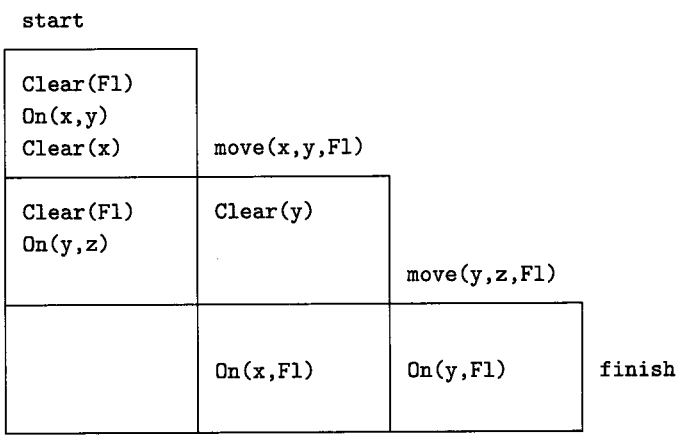


图22-20 拆积木的一个三角表模式

除了学习MACROPS，构造特定计划也能产生控制策略信息，这些信息可以减少将来的规划工作。例如，在上一个问题中，为试图获得 On(A, Fl)∧On (B, Fl)的合取式，首先获得合取项On(A, Fl)是重要的（首先获得On(B, Fl)可能要把*A*移到另一个积木的顶部，为了使*B*是空的，*A*必须被再次移到地面）。Minton的PRODIGY系统能学习这种使用 EBG的控制信息[Minton 1988]（一个基于部分评估的可选方法，见[Etzioni 1993]）。

另一类在规划中有用的学习是学习规则自身的结果。一个 agent可能有大量可用的动作，但它可能不知道这些动作的结果，或在什么条件下它们能被执行。有些在困难，但重要的学习动作模型问题中的结果已经被 [shen 1994,Gil 1992,Wang 1995,Benson 1997]得到。[Benson 1997]使用一个T-R（远程反应）程序表示用学到的算子构造的计划。如第 2章提到的，T-R表示法会给计划执行增加健壮性。

22.5 补充读物和讨论

[Lifschitz 1986]指出STRIPS的原始描述缺乏精确度和必要的约束以防止出现荒谬的公式。他讨论了STRIPS使用的语言约束，它们被用来避免这些问题的发生（在谈到公式时会提到这些约束）。[Pednault 1986，Pednault 1989]介绍了一个STRIPS版本，它既避免了Lifschitz提到的问题，又能用公式表示使用带有条件影响的规则的多 agent计划 。[Bylander 1994]证明了带有命题STRIPS算子的规划在最坏情况下是不可处理的（见 [Bylander 1993]的平均事件分析）。[Erol, Nau, & Subrahmanian 1992]给出一个STRIPS型规划系统复杂度的完整分析。

[Gupta & Nau 1992]证明了在世界状态和目标是由基原子合取式描述的积木世界问题中，找到一个最优（最短）计划，一般地讲是一个NP难题（然而，[Chapman 1989]指出一个简单的反应系统能快速地执行堆积木任务，如果它具有适当的感官谓词）。

[Waldinger 1975]介绍了通过STRIPS规则使用目标倒退，这个过程需要向后搜索。

[Blum & Furst 1995]把按照STRIPS规则形成的规划问题翻译成了能用路径发现方法求解的规划图结构。[Kautz & Selman 1996, Kautz, McAllester, & Selman 1996]提出了把规划图转化为能用WALKSAT求解的命题逻辑PSAT问题的方法。他们也提出了把规划问题直接写成 PSAT问题的新技术。见[Ernst , Millstein , & Weld 1997]。这项工作是非常重要的，因为它可以导致利用有效的随机性和爬山搜索技术的可升级规划方法。

Sacerdoti的NOAH系统[Sacerdoti 1975, Sacerdoti 1977]和Tate的INTERPLAN[Tate 1977]是最早的部分有序规划者。 Chapman的TWEAK系统[Chapman 1987]是第一个部分有序规划者的形式化，它介绍了几个重要的概念。他指出找到一个到达原子目标条件合取式的部分有序计划的问题是NP难题——在一定的合理公式表示下。

MCAllester和Rosenblitt的一个部分有序规划者的SNLP公式表示由[Soderland & Weld 1991]实现。UCPOP[Penberthy & Weld 1992]后来利用了这个实现。[Ephrati, Pollack, & Milshtein 1996]用一个A*型搜索去选择规划空间算子。[Minton, Bresina, & Drummond 1994]给出了一个部分有序和完全有序规划者的比较。对最小承诺和部分有序规划的一个综述见 [Weld 1994]。

[Christensen 1990, Knoblock 1990]开发的方法可以自动学习算子层次。[Tenenberg 1991]可以研究了层次规划者需要的抽象观念。[Erol, Hendler, & Nau 1994]是另一个层次的部分有序规划者。

[Dean & Wellman 1991]是一本将AI规划和时态推理方法与现代控制理论很好集成的书。从其他方向逼近中间是由[Ramadge & Wonham 1989]写的关于离散事件系统的著作。有关时态推

理和它在规划中的作用的讨论，见[Allen, et al. 1990]。[Zweben & Fox 1994]介绍了应用于调度问题的各种规划和搜索方法。[Wilkins, et al. 1995]把规划方法扩展到包含不确定性的应用领域。

在[Allen, et al. 1990]中有很多有关规划的重要论文。[Minton 1993]中有关于学习和规划的论文。一本叫Practical Planning的书描述了SIPE系统和它的应用[Wilkins 1988]。

有关规划的论文定期出现在主要的AI期刊和会议论文集中。关于AI规划系统（AIPS）也有一个国际会议。规划和调度技术的一个最近应用的例子见[Tate 1996]。

习题

22.1 考虑为一个厨房清洁机器人设计一个计划的问题。

1) 写出一组可能要用的STRIPS型算子。当你描述这些算子时，做如下的考虑：
   （a）打扫炉子或冰箱会弄脏地板。
   （b）炉子必须在用铝箔盖住滴水的面板之前打扫。
   （c）打扫冰箱会产生垃场且会弄乱柜台。
   （d）清洗柜台或地板会把水槽弄脏。
2) 写出一个橱房的初始状态描述，开始时厨房中有一个脏的炉子、冰箱、柜台和地板（水槽是干净的，垃场已被清扫掉）。再写出一个目标状态描述。这时每件东西都被打扫过，没有任何废物。炉子的滴水面板已用铝箔盖住。

22.2 构思一个参数化的STRIPS规则move (x, y)，它给出了在8数码问题中一个从位置x到y的移动（空白方格）的前提条件和结果。

22.3 解释递归STRIPS规则如何解决Sussman异常。

22.4 解释基于STRIPS规则利用倒退的向后搜索如何解决Sussman异常。

22.5 建立汉诺塔问题的一个STRIPS公式表示（参见习题5.3、7.4和9.2）。

即：

1) 指定描述一个移动结果的STRIPS规则，并指定目标条件。
2) 写出描述初始状态的公理，显示如何应用这个规则以产生一个后继状态（不要忘了包括在所有的状态中都为真的公式，可能需要它们来证明前提条件和目标条件）。

22.6 一个可消耗的火星机器人要计划到离它的大本营20公里远的一个火星山脉旅游路线。（它不必返回到大本营）机器人加一箱燃料刚好能旅行10公里。它也能带一箱燃料作为货物。大本营叫B，山脉叫M，一个临时台架站点（离山和大本营各10公里）是S。有三箱燃料，$P1$、$P2$和$P3$在B，机器人开始时没有加燃料。机器人有4个动作：

1) goto (x, y), 其中x是初始位置（B、M或S之一），y是目标位置（B、M和S之一）。为了执行这个动作，机器人在它的燃料箱中必须有一箱燃料，一箱燃料只能在B和S与S和M之间移动。
2) pickup (u, x)，u是在位置x的一箱燃料，这个动作的结果是机器人将u做为运输的货物。
3) putdown (u, x)，u是存放在位置x的一箱燃料。为了执行putdown，机器人必须正携带着u。
4) refuel (u, x)，u是位置x的一箱燃料。它将被加进机器人的燃料箱中。（假定执行pickup、putdown和refuel动作不消耗任何燃料）。

用下面的谓词范式（带有明显的预期意思）：

Atrobot (x), 其中x是位置B、M或S之一。

At (r, x), 其中x是位置B、M或S之一，r是燃料箱之一即使机器人在x位置，机器人本身的一箱油也不认为是在x处。

Carrying (u)指出机器人正携带着燃料箱u（作为货物）。

Fueled 指出机器人的燃料箱中有一箱燃料（准备燃烧）。

Cango (x, y)给出了机器人在一箱燃料下的移动范围。它描述了机器人能从 x去y（如果它被加了燃料）。

我们想建立起这个问题，以便一个STRIPS型的问题求解者能生成一个计划以使机器人能到达山脉。

1) 初始状态描述和目标条件是什么？

2) 解决这个问题需要的算子和它们的描述（前提条件、删除列表和加入列表）是什么？为了使问题简单一些，在公式中不必包括像Place (x) 或Pellet (x) 这些谓词。

3) 给出一个STRIPS系统（从初始状态向目标状态前进）可能产生的上述问题的一个求解计划。用下面的格式写出你的方案：

在初始状态S0中的文字

第一个算子

在随后的状态S1中的文字

第二个算子

依次类推，直至达到一个满足目标的状态。

22.7 发明一个“图标化的”数据结构模式（不是谓词演算公式）来表字上述练习题中的火星机器人和燃料箱的可能状态。开始状态的数据结构是什么？

为了表明你明白如何构造和使用这个问题的算子，显示把算子 refuel(P1, B)和Pickup(P1, B)应用到开始节点产生的状态的数据结构。

22.8 假定我们的火星机器人有传感器，它使机器人能决定下面谓词的真假：

Atfuelpellet(S)

Atfuelpellet(B)

Fueled

Carrying

Atrobot(B)

Atrobot(S)

Atrobot(M)

它们有明显的预期含意。

机器人有如下的动作：

refuel 用一箱燃料给机器人加燃料（如果机器人的位置有一箱燃料）。

pickup 把一箱燃料做为货物加在机器人身上（如果机器人的位置有一箱燃料）。

putdown 在机器人的当前位置卸下一箱燃料。

goto(x) 使机器人到达位置x（如果机器人有燃料且x在10公里范围内）。

设计一个产生系统（第2章描述的类型），它将对所有传感输入的组合执行相应的

动作。

22.9 考虑使用一个部分有序规划系统（像本章描述的一样）来规划一个软件产品的开发和发布。假定在这个领域存在下面的4个STRIPS算子：

算 子	前提条件	加入列表	删除列表
optimize	HaveProgram	Optimized	BugFree
debug	HaveProgram	BugFree	
ship	HaveProgram BugFree Optimized HavePackaging	HappyCustomer	
designPackaging	HaveProgram	HavePackaging	

规划问题是从HaveProgram为真的一个状态到达HappyCustomer为真的一个状态。像我们知道的，部分有序规划方法将加入下面两个“虚构的”算子：

算 子	前提条件	加入列表	删除列表
start	T	HaveProgram	Nil
finish	HappyCustomer	Nil	HappyCustomer

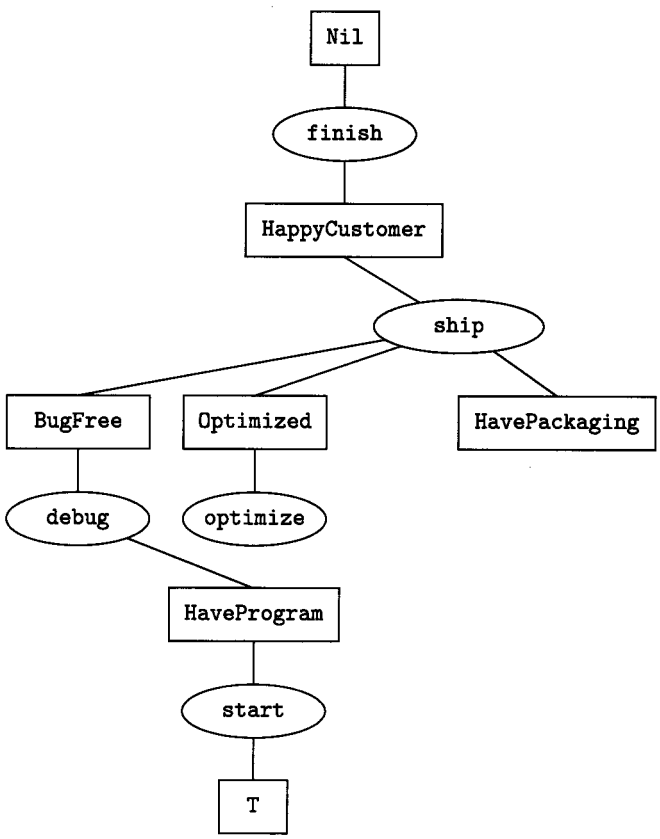


图22-21 习题22.9的部分计划

假定我们从图22-21中的部分规划开始。这个部分规划中当前没有包含任何排序约束，除了那些由部分规划的部分顺序暗示的约束。

1) 在这个部分规划中，如果有的话,现存的威胁弧是什么？

2) 列出这个部分规划中不被因果连接支持的任何算子的前提条件。

3) 出示一个部分有序规划过程可以产生的可能计划（即，用附加算子扩展显示的图，列出排除约束）。不必显示产生计划的所有步骤——显示完成的计划就行了。

22.10 描述如何用ABSTRIPS系统来产生一个“任何时间”的计划。简短评论一下为什么你希望随着计划所花时间的增多，计划的质量和可信度会改善？