Планирование с помощью пропозициональной логики и планирование действий в реальном мире

Планирование с помощью пропозициональной логики

Данный подход основан на проверке выполнимости логического высказывания, модель которого выглядит примерно так:

Начальное состояние & Все возможные описания действий & Цель

Такое высказывание должно содержать пропозициональные символы, соответствующие каждому возможному проявлению действия. Модель, в которой выполняется это высказывание, должна присваивать значение true действиям, являющимся частью правильного плана, и false – другим действиям. Если задача планирования неразрешима, то данное высказывание будет невыполнимым.

Рассмотрим задачу воздушной транспортировки. В начальном состоянии (время 0) самолет P1 находится в аэропорту SFO, а самолет P2 – в аэропорту JFK:

$at(P1,SFO)^0$ & $at(P2,JFK)^0$ & $\neg at(P1,JFK)^0$ & $\neg at(P2,SFO)^0$

Задача состоит в выработке плана действий для достижения состояния, когда самолет P1 находится в аэропорту JFK, а самолет P2 – в аэропорту SFO.

Пропозициональная запись **аксиом состояния-преемника** выглядит следующим образом:

$at(P1,JFK)^1 \Leftrightarrow (at(P1,JFK)^0 \ \& \ \neg Fly(P1,JFK,SFO)^0)$ or $(at(P1,SFO)^0 \ \& \ Fly(P1,SFO,JFK)^0)$

$at(P2,SFO)^1 \Leftrightarrow (at(P2,SFO)^0 \ \& \ \neg Fly(P2,SFO,JFK)^0)$ or $(at(P2,JFK)^0 \ \& \ Fly(P2,JFK,SFO)^0)$

Здесь, например, символ $Fly(P1, SFO, JFK)^0$ является истинным, если самолет P1 вылетает из аэропорта SFO в аэропорт JFK во время 0.

Чтобы предотвратить выработку планов с недопустимыми действиями, необходимо добавить аксиомы предусловий, которые указывают, что для совершения любого действия требуется, чтобы были выполнены его предусловия. В нашем примере необходимо добавить следующие аксиомы предусловий:

$at(P1,JFK)^0 \Rightarrow Fly(P1,JFK,SFO)^0$

$at(P2,SFO)^0 \Rightarrow Fly(P2,SFO,JFK)^0$

Предположим, что цель истинна в начальном состоянии, во время T=0. То есть проверяем целевое утверждение:

$at(P1,JFK)^0 \ \& \ at(P2,SFO)^0$

Если попытка окажется неудачной, то повторим ее снова во время T=1, то есть проверим выполнимость высказывания

Начальное состояние & Аксиомы состояния-преемника & $Цель^1$

и т.д. до тех пор, пока не достигнем осуществимого плана с минимальной длиной. Можно наложить верхний предел T_{max} для безусловного завершения этого алгоритма. В нашем случае $T_{max}= 1$. После нахождения модели выполнимого высказывания план извлекается путем поиска тех пропозициональных символов, которые относятся к действиям, и которым в модели присвоено значение true. Решением данной задачи является следующий план:

$Fly(P1,SFO,JFK)^0, \ Fly(P2,JFK,SFO)^0$

Добавим еще один аэропорт LAX. Рассмотренные аксиомы состояния-преемника и аксиомы предусловия разрешают самолету вылететь в два аэропорта назначения одновременно! Необходимо ввести дополнительные аксиомы для устранения таких фиктивных решений, которые называются **аксиомами частичного исключения действий**, предотвращающие

одновременное выполнение несовместимых действий. Для рассматриваемой задачи это аксиомы:

$\neg Fly(P1,SFO,JFK)^0$ & $Fly(P1,SFO,LAX)^0$

$\neg Fly(P2,JFK,SFO)^0$ & $Fly(P2,JFK,LAX)^0$

Вместо аксиом исключения можно использовать **аксиомы ограничения состояния**, утверждающие, что ни один объект не может находиться в двух местах одновременно:

для любых p, $x,y,t \quad x \neq y \Rightarrow (at(p,x)^t$ & $at(p,y)^t)$

В пропозициональной логике необходимо будет записать все базовые экземпляры каждого ограничения состояния.

Итак, планирование в форме доказательства выполнимости предусматривает поиск моделей для высказывания, включающего описание начального состояния, цели, аксиом состояния-преемника, аксиом предусловий, а также либо аксиом исключения действия, либо аксиом ограничения состояния. Можно показать, что эта совокупность аксиом является достаточной, в том смысле, что при их использовании больше не возникают какие-либо “фиктивные решения”.

Любая модель, в которой выполняется такое пропозициональное высказывание, будет представлять собой действительный план для первоначальной задачи, а любая линеаризация этого плана будет представлять собой допустимую последовательность действий, которая позволяет достичь цели.

Основным недостатком пропозиционального подхода к решению задач планирования являются размеры пропозициональной базы знаний. которая формируется в процессе решения задачи. Общее количество символов действий ограничено значением:

$T^*|act|^*|O|^P$,

где |act| - количество схем действий, |O|- количество объектов в проблемной области, P – максимальная арность (количество параметров) любой схемы действий. Количество выражений еще больше. Например, при 10 временных этапах, 12 самолетах и 30 аэропортах полная аксиома исключения действий состоит из 583 миллионов выражений.

Одним из способов преодоления указанного недостатка является процесс **расщепления символов**. Например, символ действия $fly(P1,SFO,JFK)^0$, арность которого равна 3, можно заменить тремя новыми символами:

$Fly_1(P1)^0$ – самолет P1 прилетел во время 0,

$Fly_2(SFO)^0$ – аэропортом отправления в этом полете был SFO,

$Fly_3(JFK)^0$ – аэропортом назначения в этом полете был JFK.

Теперь требуется только T*|act|*P*|O| символов.

Аналогичные сокращения допускаются в аксиомах предусловия и аксиомах исключения действий. Для описанного выше случая полная аксиома исключения действий сокращается с 583 миллионов выражений до 9360 выражений.

Планирование действий в реальном мире

В ряде реальных проблемных областей необходимо указание времени начала и окончания действий. Например, в проблемной области транспортировки грузов может потребоваться знать, когда именно прибывает самолет, перевозящий некоторый груз.

Для выполнения задач планирования производства требуется осуществление ряда работ, каждая из которых состоит из последовательности действий, имеющих определенную продолжительность и требующих определенных ресурсов. Требуется разработка расписания,

минимизирующего общее время, требуемое для выполнения всех работ, при соблюдении ограничений на ресурсы.

Рассмотрим упрощенную задачу сборки автомобиля, предусматривающую две работы: сборка автомобилей C1 и C2. Каждая работа состоит из трех действий: установка двигателя, установка колес и проверка результатов. Предположим, что двигатель должен устанавливаться в первую очередь, а проверка, безусловно, должна проводиться в последнюю очередь.

В тексте программы на языке CLIPS, приведенном ниже, используются следующие шаблоны для спецификации неупорядоченных фактов:

engine – спецификация факта начала установки двигателя **e** на шасси **c** с плановым временем **t**,

wheels – спецификация факта начала установки колес **w** на шасси **c** за время **t**,

chassis – спецификация факта использования шасси **c**,

enginein – спецификация факта окончания установки двигателя на шасси **c**,

wheelson – спецификация факта окончания установки колес на шасси **c**,

done – спецификация факта окончания поверки автомобиля, собранного на шасси **c**.

Правила addengine, addwheels и inspect описывают, соответственно, действия по установке двигателя, колес и проверке.

```
(deftemplate engine
    (slot e (type SYMBOL))
    (slot c (type SYMBOL))
    (slot t (type INTEGER))
)
(deftemplate wheels
```

```
    (slot w (type SYMBOL))
    (slot c (type SYMBOL))
    (slot t (type INTEGER))
)
(deftemplate chassis
    (slot c (type SYMBOL))
)
(deftemplate enginein
    (slot c (type SYMBOL))
)
(deftemplate wheelson
    (slot c (type SYMBOL))
)
(deftemplate done
    (slot c (type SYMBOL))
)
(deftemplate goal
    (slot current-task (type SYMBOL))
)


(deffacts init
    (engine (e E1)(c C1)(t 30))
    (engine (e E2)(c C2)(t 60))
    (wheels (w W1)(c C1)(t 30))
    (wheels (w W2)(c C2)(t 15))
    (chassis (c C1))
    (chassis (c C2))
)
(defrule init-system
```

```
    (initial-fact)
=>
    (assert (goal(current-task find)))
)


(defrule addengine
    (engine (e ?e)(c ?c)(t ?d))
    (chassis (c ?c))
    (not (enginein (c ?c)))
    (goal (current-task find))
=>
    (assert (enginein(c ?c)))
    (printout t crlf "addengine " ?e " " ?c " " ?d)
)
(defrule addwheels
    (enginein (c ?c))
    (wheels (w ?w)(c ?c)(t ?d))
    (chassis (c ?c))
    (goal (current-task find))
=>
    (assert (wheelson(c ?c)))
    (printout t crlf "addwheels " ?w " " ?c " " ?d)
)


(defrule inspect
    (enginein (c ?c))
    (wheelson(c ?c))
    (chassis (c ?c))
```

```
    (goal (current-task find))
=>
    (assert (done(c ?c)))
    (printout t crlf "inspect " ?c " 10")
)
(defrule finish-process
    (done (c C1))
    (done (c C2))
    ?goal-ptr <- (goal (current-task find))
=>
    (retract ?goal-ptr)
    (printout t crlf "Solution found" crlf)
)
```

Далее показан результат выполнения программы в среде CLIPS с использованием стратегии “в глубину” (`depth`). Так как действия по сборке обоих автомобилей лишь частично упорядочены, то есть действия, относящиеся к разным автомобилям, могут производиться параллельно, то полученный план можно изобразить так, как показано на рисунке 1. Продолжительность каждого действия обозначена в нижней части каждого прямоугольника, а значения самого раннего и самого позднего времени начала, показаны в верхней левой части.

```
addengine E1 C1 30
addwheels W1 C1 30
inspect C1 10
addengine E2 C2 60
addwheels W2 C2 15
inspect C2 10
Solution found
```

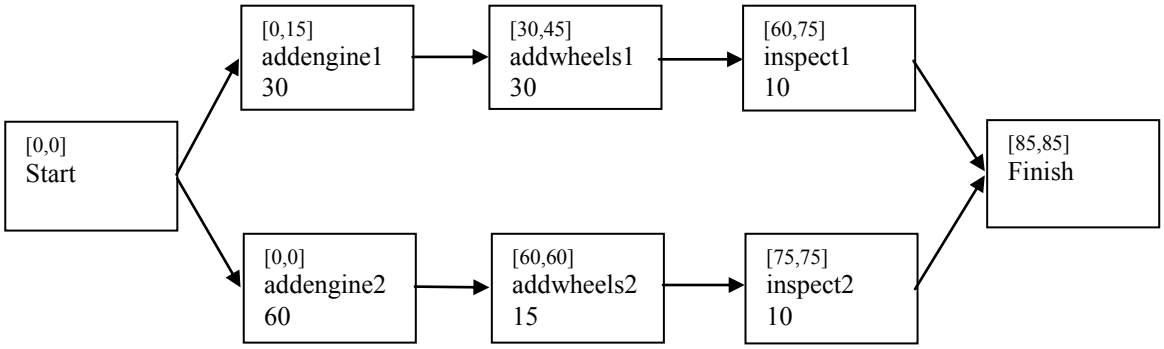

Рисунок 1 – План по сборке автомобилей

Для преобразования этой задачи в задачу составления расписания, необходимо определить, когда должно начаться и закончиться каждое действие, с учетом продолжительности действия, а также их упорядочения. Определив частичное упорядочение действий с учетом их продолжительности, можно применить метод критического пути для выяснения допустимых значений времени начала и окончания каждого действия.