Sztuczna inteligencja : Budowanie algorytmów sterowania do wyszukiwania w przestrzeni stanów

(VI / XVI)

ĆWICZENIA

1.

a. Napisz algorytm sprawdzania członków, aby rekurencyjnie określać, czy dany element należy do listy.

b. Napisz algorytm zliczający liczbę elementów na liście.

c. Napisz algorytm zliczający liczbę atomów na liście.

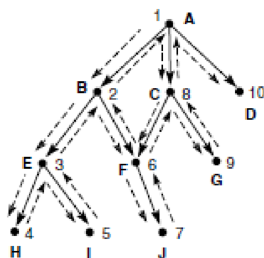
(Różnica między atomami a pierwiastkami polega na tym, że element może sam być listą).

2. Napisz algorytm rekurencyjny (przy użyciu list otwartych i zamkniętych), aby zaimplementować wyszukiwanie wszerz. Czy rekurencja pozwala pominąć otwartą listę podczas wdrażania pierwszego wyszukiwania? Wyjaśnić.

3. Śledź wykonanie rekurencyjnego algorytmu wyszukiwania w pierwszej kolejności (wersja, która nie korzysta z otwartej listy) w przestrzeni stanu na rysunku

Initialize: SL = [A]; NSL = [A]; DE = [ ]; CS = A;

AFTER ITERATION	CS	SL	NSL	DE
0	A	[A]	[A]	[ ]
1	B	[B A]	[B C D A]	[ ]
2	E	[E B A]	[E F B C D A]	[ ]
3	H	[H E B A]	[H I E F B C D A]	[ ]
4	I	[I E B A]	[I E F B C D A]	[H]
5	F	[F B A]	[F B C D A]	[E I H]
6	J	[J F B A]	[J F B C D A]	[E I H]
7	C	[C A]	[C D A]	[B F J E I H]
8	G	[G C A]	[G C D A]	[B F J E I H]



4. W starożytnej hinduskiej ceremonii parzenia herbaty bierze udział trzech uczestników: starszy, sługa i dziecko. Cztery zadania, które wykonują: karmienie ogniem, serwowanie ciastek, nalewanie herbaty

i czytanie poezji; ta kolejność odzwierciedla malejące znaczenie zadań. Na początku ceremonii dziecko wykonuje wszystkie cztery zadania. Są one przekazywane pojedynczo służącemu i starszemu, aż pod koniec ceremonii starszy wykona wszystkie cztery zadania. Nikt nie może podjąć mniej ważnego zadania niż te, które już wykonują. Wygeneruj sekwencję ruchów, aby przenieść wszystkie zadania z dziecka na starszego. Napisz algorytm rekurencyjny, aby wykonać sekwencję ruchu.

5. Korzystając z definicji ruchu i ścieżki dla trasy rycerza, prześledź wykonanie wzorca_wyszukiwania na celach:

a. ścieżka (1,9).

b. ścieżka (1,5).

c. ścieżka (7,6).

Podczas próby wykonania predykatów ruchu, wyszukiwanie jest często zapętlone. Porozmawiaj o wykrywaniu pętli i nawracaniu w tej sytuacji.

6. Napisz definicję pseudokodu dla pierwszej wersji wzorca_wyszukiwania. Omów efektywność czasową i przestrzenną tego algorytmu.

7. Używając reguły z przykładu 6.2.3 jako modelu, napisz osiem reguł ruchu wymaganych dla pełnej wersji 8 × 8 trasy rycerza.

8. Korzystając ze stanów celu i początkowego z rysunku 6.3, ręcznie uruchom rozwiązanie systemu produkcyjnego 8-puzzle:

a. W sposób zorientowany na cel.

b. W sposób oparty na danych.

9. Rozważ problem doradcy finansowego omówiony w rozdziałach 2, 3 i 4. Używanie rachunku predykatu jako języka reprezentacji:

a. Napisz problem wprost jako system produkcyjny.

b. Wygeneruj przestrzeń stanu i etapy pamięci roboczej dla rozwiązania opartego na danych z

przykładu w rozdziale 3.

10. Powtórz problem 9.b, aby uzyskać rozwiązanie ukierunkowane na cel.

11. W sekcji 6.2.3 przedstawiono ogólne strategie rozwiązywania konfliktów załamania, aktualności i specyficzności. Zaproponuj i uzasadnij jeszcze dwie takie strategie.

12. Zaproponuj dwie aplikacje odpowiednie do rozwiązania z wykorzystaniem architektury tablicowej. Krótko scharakteryzuj organizację tablicy i źródeł wiedzy dla każdego wdrożenia.

Wprowadzenie

Do tego momentu przedstawiono rozwiązywanie problemów jako przeszukiwanie zestawu sytuacji lub stanów problemowych. Przedstawiono rachunek predykatów jako medium do opisu stanów problemu i wnioskowania dźwiękowego jako metodę tworzenia nowych stanów. Wprowadzono wykresy do reprezentowania i łączenia sytuacji problemowych. Algorytmy cofania, a także algorytmy wyszukiwania w pierwszej kolejności i w pierwszej kolejności mogą eksplorować te wykresy. W Części

4 przedstawiono algorytmy wyszukiwania heurystycznego. W rozdziale 5, biorąc pod uwagę probabilistyczne stany świata, wnioskowanie stochastyczne wykorzystano do wytworzenia nowych stanów. Podsumowując, wcześniejsza część:

1. Rozwiązanie problemu jako ścieżkę na wykresie od stanu początkowego do celu.

2. Wykorzystano wyszukiwanie do systematycznego testowania alternatywnych ścieżek do celów.

3. Zastosowano backtracking lub jakiś inny mechanizm, aby umożliwić algorytmom odzyskanie ze ścieżek, które nie znalazły celu.

4. Używane listy do prowadzenia wyraźnych rejestrów rozważanych stanów.

a. Otwarta lista pozwala algorytmowi zbadać niesprawdzone stany, jeśli to konieczne.

b. Zamknięta lista odwiedzonych stanów umożliwia algorytmowi wdrożenie wykrywania pętli i uniknięcie powtarzania bezowocnych ścieżek.

5. Zaimplementowano otwartą listę jako stos dla pierwszego wyszukiwania w głębokości, kolejkę dla pierwszego wyszukiwania i kolejkę priorytetową dla najlepszego wyszukiwania.

W tej sekcji wprowadzimy dalsze techniki budowania algorytmów wyszukiwania. Wyszukiwanie rekurencyjne implementuje wyszukiwanie głębokości, szerokości i najlepsze pierwsze w bardziej zwięzły i naturalny sposób. Ponadto rekurencja jest wzbogacona o ujednolicenie w celu przeszukiwania przestrzeni stanu generowanej przez twierdzenia rachunku predykatu. Ten algorytm wyszukiwania ukierunkowanego jest podstawą PROLOGU i kilka systemów eksperckich W dalszej sekcji wprowadzamy systemy produkcyjne, ogólną architekturę rozwiązywania problemów kierowanych wzorcami, która była szeroko stosowana do modelowania człowieka rozwiązywanie problemów, , a także w innych aplikacjach AI, w tym systemach eksperckich. Przedstawiamy kolejną architekturę sterowania rozwiązywaniem problemów AI, tablicę.

Wyszukiwanie na podstawie rekurencji (opcjonalnie)

Wyszukiwanie rekurencyjne

W matematyce definicja rekurencyjna wykorzystuje termin zdefiniowany jako część własnej definicji. W informatyce rekurencja służy do definiowania i analizowania zarówno struktur danych, jak i procedur. Procedura rekurencyjna składa się z:

1. Krok rekurencyjny: procedura wzywa się do powtórzenia sekwencji działań.

2. Warunek zakończenia, który zatrzymuje powtarzanie się procedury w nieskończoność (rekurencyjna wersja nieskończonej pętli).

Oba te komponenty są niezbędne i pojawiają się we wszystkich rekurencyjnych definicjach i algorytmach. Rekurencja to naturalny konstrukt kontrolny dla struktur danych, które mają regularną strukturę i nie mają określonego rozmiaru, takich jak listy, drzewa i wykresy, i jest szczególnie odpowiedni do wyszukiwania w przestrzeni stanów. Bezpośrednie tłumaczenie algorytmu przeszukiwania w pierwszej kolejności na formę rekurencyjną ilustruje równoważność rekurencji i iteracji. Algorytm wykorzystuje zmienne globalne zamknięte i otwarte do utrzymywania list stanów. Wyszukiwanie według szerokości i najlepszego wyniku można zaprojektować praktycznie z tym samym algorytmem, tzn. Zachowując zamkniętą jako globalną strukturę danych i implementując opcję otwartą jako kolejkę lub kolejkę priorytetową, a nie jako stos (stos kompilacji staje się kolejką kompilacji lub zbuduj kolejkę priorytetową):

```
unction depthsearch; % open & closed global
begin
if open is empty
then return FAIL;
current_state := the first element of open;
if current_state is a goal state
then return SUCCESS
else
begin
open := the tail of open;
closed := closed with current_state added;
for each child of current_state
if not on closed or open % build stack
then add the child to the front of open
end;
depthsearch % recur
end.
```

Wyszukiwanie w pierwszej kolejności, tak jak właśnie przedstawione, nie wykorzystuje pełnej mocy rekurencji. Możliwe jest dalsze uproszczenie procedury poprzez użycie samej rekurencji (zamiast jawnej otwartej listy) do organizowania stanów i ścieżek w przestrzeni stanów. W tej wersji algorytmu globalna zamknięta lista jest używana do wykrywania duplikatów stanów i zapobiegania pętlom, a otwarta lista jest uwikłana w rekordy aktywacji środowiska rekurencyjnego. Ponieważ nie można już jawnie manipulować otwartą listą, wyszukiwanie w pierwszej kolejności i wyszukiwanie w pierwszej kolejności nie są już naturalnymi rozszerzeniami następującego algorytmu:

```
function depthsearch (current_state); % closed is global
begin
if current_state is a goal
then return SUCCESS;
add current_state to closed;
while current_state has unexamined children
begin
child := next unexamined child;
if child not member of closed
```

```
then if depthsearch(child) = SUCCESS

then return SUCCESS

end;

return FAIL % search exhausted

end
```

Algorytm ten nie generuje wszystkich potomków stanu i umieszcza je na otwartej liście, ale generuje stany potomne pojedynczo i rekurencyjnie przeszukuje potomków każdego dziecka przed wygenerowaniem jego rodzeństwa. Zauważ, że algorytm zakłada zamówienie do operatorów generujących stan. Podczas rekurencyjnego przeszukiwania stanu potomnego, jeśli jakiś potomek tego stanu jest celem, wywołanie rekurencyjne zwraca sukces, a algorytm ignoruje rodzeństwo.

Jeśli wywołanie rekurencyjne w stanie potomnym nie znajdzie celu, generowane jest następne rodzeństwo i przeszukiwane są wszystkie jego potomki. W ten sposób algorytm przeszukuje cały wykres w pierwszej kolejności głębokości. Czytelnik powinien zweryfikować, czy faktycznie przeszukuje wykres w tej samej kolejności, co algorytm przeszukiwania z głębokością pierwszą .

Pominięcie jawnej otwartej listy jest możliwe poprzez rekurencję. Mechanizmy, za pomocą których język programowania realizuje rekurencję, obejmują osobny rekord aktywacyjny (Aho i Ullman 1977) dla każdego wywołania rekurencyjnego. Każdy rekord aktywacji przechwytuje zmienne lokalne i stan wykonania każdego wywołania procedury. Gdy procedura jest wywoływana rekurencyjnie z nowym stanem, nowy rekord aktywacyjny przechowuje jego parametry (stan), dowolne zmienne lokalne i bieżący stan wykonania. W algorytmie wyszukiwania rekurencyjnego serie stanów na bieżącej ścieżce są zapisywane w sekwencji rekordów aktywacyjnych wywołań rekurencyjnych. Zapis każdego połączenia wskazuje również ostatnią operację użytą do wygenerowania stanu potomnego; pozwala to na wygenerowanie następnego rodzeństwa w razie potrzeby. Cofanie następuje, gdy wszyscy potomkowie stanu nie uwzględniają celu, co powoduje niepowodzenie wywołania rekurencyjnego. To zwraca błąd procedury rozwijania stanu nadrzędnego, który następnie generuje i powtarza się przy następnym rodzeństwie. W tej sytuacji wewnętrzne mechanizmy rekurencji działają na otwartej liście stosowanej w iteracyjnej wersji algorytmu. Rekursywna implementacja pozwala programiście ograniczyć jego punkt widzenia do jednego stanu i jego dzieci, bez konieczności jawnego utrzymywania otwartej listy stanów. Zdolność rekurencji do wyrażania globalnych koncepcji w formie zamkniętej jest głównym źródłem jej siły. Jak pokazują te dwa algorytmy, wyszukiwanie w przestrzeni stanów jest z natury procesem rekurencyjnym. Aby znaleźć ścieżkę od bieżącego stanu do celu, przejdź do stanu potomnego i powtórz. Jeśli ten stan dziecka nie prowadzi do celu, spróbuj po kolei rodzeństwa. Rekurencja dzieli duży i trudny problem (przeszukiwanie całej przestrzeni) na mniejsze, prostsze części (generuje dzieci z jednego stanu) i stosuje tę strategię (rekurencyjnie) do każdego z nich. Proces ten trwa do momentu wykrycia stanu celu lub wyczerpania przestrzeni. W następnej sekcji to rekurencyjne podejście do rozwiązywania problemów zostało rozszerzone na kontroler logicznego rozwiązywania problemów, który wykorzystuje unifikację i wnioskowanie do generowania i przeszukiwania przestrzeni relacji logicznych. Algorytm obsługuje wiele celów, a także łańcuchy wstecz od celu do lokalu.

Przykład wyszukiwania rekurencyjnego: Uzasadnienie oparte na wzorcach

Stosujemy wyszukiwanie rekurencyjne w przestrzeni logicznych wniosków; wynikiem jest ogólna procedura wyszukiwania specyfikacji problemów opartych na rachunku predykatu. Załóżmy, że chcemy napisać algorytm, który określa, czy wyrażenie rachunku predykatu jest logiczną konsekwencją

pewnego zestawu twierdzeń. Sugeruje to wyszukiwanie ukierunkowane z początkowym zapytaniem tworzącym cel i modus ponens definiujący przejścia między stanami. Biorąc pod uwagę cel (taki jak p (a)), algorytm używa unifikacji, aby wybrać implikacje, których wnioski pasują do celu (np. Q (X) → p (X)). Ponieważ algorytm traktuje implikacje jako potencjalne reguły rozwiązywania zapytania, często są one po prostu nazywane regułami. Po ujednoliceniu celu z wnioskiem o implikacji (lub regule) i zastosowaniu wynikających z tego podstawień w regule, założenie reguły staje się nowym celem (q (a)). To się nazywa subgoal. Algorytm następnie powtarza się w podzadaniu. Jeśli podskala jest zgodna z faktem w bazie wiedzy, wyszukiwanie kończy się. Szereg wnioskowania, które doprowadziły od pierwotnego celu do podanych faktów, potwierdzają prawdziwość pierwotnego celu.

function pattern_search (current_goal);

begin

if current_goal is a member of closed % test for loops

then return FAIL

else add current_goal to closed;

while there remain in data base unifying facts or rules do

begin

case

current_goal unifies with a fact:

return SUCCESS;

current_goal is a conjunction (p

∧ ...):

begin

for each conjunct do

call pattern_search on conjunct;

if pattern_search succeeds for all conjuncts

then return SUCCESS

else return FAIL

end;

current_goal unifies with rule conclusion (p in q

→ p):

begin

apply goal unifying substitutions to premise (q);

call pattern_search on premise;

if pattern_search succeeds

then return SUCCESS

else return FAIL

end;

end; % end case

end;

return FAIL

end.

W funkcji wzorzec_wyszukiwania wyszukiwanie jest wykonywane przez zmodyfikowaną wersję algorytmu wyszukiwania rekurencyjnego, który korzysta z unifikacji, aby określić, kiedy dwa wyrażenia pasują do siebie, i modus ponens w celu wygenerowania potomków stanów. Bieżący fokus wyszukiwania jest reprezentowany przez zmienną current_goal. Jeśli current_goal pasuje do faktu, algorytm zwraca sukces. W przeciwnym razie algorytm próbuje dopasować wartość bieżąca do wniosku z jakąś regułą, rekurencyjnie próbując rozwiązać przesłankę. Jeśli wartość bieżąca nie pasuje do żadnego z podanych twierdzeń, algorytm zwraca błąd. Algorytm ten obsługuje również cele łączone. Dla uproszczenia algorytm nie rozwiązuje problemu utrzymania spójności między podstawieniami zmiennych wytwarzanymi przez unifikację. Jest to ważne przy rozwiązywaniu spójnych zapytań ze wspólnymi zmiennymi (jak w p (X) ∧ q (X)). Obie koniunkcje się powiodły, ale muszą również odnieść sukces z tym samym ujednoliconym wiązaniem dla X. Główną zaletą stosowania ogólnych metod, takich jak unifikacja i modus ponens do generowania stanów, jest to, że wynikowy algorytm może przeszukiwać dowolną przestrzeń wnioskowania logicznego, w której specyfika problem opisano za pomocą twierdzeń rachunku predykatu. Mamy zatem sposób na oddzielenie wiedzy na temat rozwiązywania problemów od jej kontroli i wdrażania na komputerze. pattern_search stanowi nasz pierwszy przykład oddzielenia wiedzy problemowej od kontroli wyszukiwania. Chociaż początkowa wersja wzorca_wyszukiwania określa zachowanie algorytm wyszukiwania dla wyrażeń rachunku predykatu, należy jeszcze zająć się kilkoma subtelnościami. Obejmują one kolejność, z jaką algorytm próbuje alternatywnych dopasowań oraz poprawne działanie pełnego zestawu operatorów logicznych (∧, ∨ i ¬). Logika jest deklaratywna i bez ustalonej strategii wyszukiwania: określa przestrzeń możliwych wnioskowań, ale nie mówi rozwiązującemu problemu, jak zrobić użyteczne. Aby uzasadnić za pomocą rachunku predykatu, potrzebujemy systemu kontroli, który systematycznie przeszukuje przestrzeń, unikając bezsensownych ścieżek i pętli. Algorytm kontroli, taki jak wzorzec_wyszukiwania, musi wypróbować alternatywne dopasowania w pewnej kolejności. Znając tę kolejność, projektant programu może kontrolować wyszukiwanie poprzez prawidłowe porządkowanie reguł w bazie wiedzy. Prostym sposobem zdefiniowania takiego porządku jest wymaganie od algorytmu wypróbowania reguł i faktów w kolejności, w jakiej pojawiają się w bazie wiedzy. Drugim zagadnieniem jest istnienie logicznych łączników w przesłankach reguł: np. Implikacje postaci „p ← q ∧ r ”lub„ p ← q ∨ (r ∧ s). ” Jak zostanie przypomniane z dyskusji i / lub wykresów, operator indicates wskazuje, że oba wyrażenia muszą być prawdziwe, aby cała przesłanka była prawdziwa. Ponadto spójniki wyrażenia należy rozwiązać za pomocą spójnych wiązań zmiennych. Zatem, aby rozwiązać p (X) ∧ q (X), nie wystarczy rozwiązać p (X) z podstawieniem {a / X} i q (X) z podstawieniem {b / X}. Oba muszą zostać rozwiązane przy użyciu tego samego jednoznacznego wiązania dla X. An lub operator, z drugiej strony, i wskazuje, że jedno z wyrażeń musi zostać uznane za prawdziwe. Algorytm wyszukiwania musi to uwzględnić. Ostatnim dodatkiem do algorytmu jest umiejętność rozwiązywania celów z logiczną negacją (¬). pattern_search radzi sobie z zanegowanymi celami, rozwiązując operand z ¬. Jeśli ten podzadanie powiedzie się, wówczas wyszukiwanie wzorzec_z powodzeniem nie powiedzie się. Jeśli operand się nie

powiedzie, wówczas attern_search zwraca pusty zestaw podstawień, co wskazuje na sukces. Zauważ, że nawet jeśli podzadanie może zawierać zmienne, wynik rozwiązania jego negacji może nie zawierać żadnych podstawień. Jest tak, ponieważ ¬ może odnieść sukces tylko wtedy, gdy operand zawiedzie; dlatego nie może zwrócić żadnych powiązań dla operandu.

Na koniec algorytm nie powinien zwracać sukcesu, ale powinien zwracać powiązania związane z rozwiązaniem. Pełna wersja wzorca_wyszukiwania, który zwraca zestaw ujednoliceń, który spełnia każde podzadanie, to:

```
function pattern_search(current_goal);

begin

if current_goal is a member of closed % test for loops

then return FAIL

else add current_goal to closed;

while there remain unifying facts or rules do

begin

case

current_goal unifies with a fact:

return unifying substitutions;

current_goal is negated (¬ p):

begin

call pattern_search on p;

if pattern_search returns FAIL

then return {}; % negation is true

else return FAIL;

end;

current_goal is a conjunction (p ∧…):

begin

for each conjunct do

begin

call pattern_search on conjunct;

if pattern_search returns FAIL

then return FAIL;

else apply substitutions to other conjuncts;

end;
```

```
if pattern_search returns SUCCESS for all conjuncts
then return composition of unifications;
else return FAIL;
end;
current_goal is a disjunction (p ∨ …):
begin
repeat for each disjunct
call pattern_search on disjunct
until no more disjuncts or SUCCESS;
if pattern_search returns SUCCESS
then return substitutions
else return FAIL;
end;
current_goal unifies with rule conclusion (p in p ← q):
begin
apply goal unifying substitutions to premise (q);
call pattern_search on premise;
if pattern_search returns SUCCESS
then return composition of p and q substitutions
else return FAIL;
end;
end; %end case
end %end while
return FAIL
end.
```

Ten algorytm wzorca wyszukiwania dla przeszukiwania przestrzeni reguł i faktów rachunku predykatów stanowi podstawę Prologu (gdzie stosowana jest forma predykatów z klauzuli Horn, Rozdział 14.3) oraz w wielu powłokach systemu ekspertów ukierunkowanych na cel (Rozdział 8). Alternatywna struktura sterowania dla wyszukiwania ukierunkowanego na wzorzec jest zapewniona przez system produkcyjny, omówiony w następnym rozdziale.

Systemy produkcyjne

Definicja i historia

System produkcyjny jest modelem obliczeniowym, który okazał się szczególnie ważny w sztucznej inteligencji, zarówno przy wdrażaniu algorytmów wyszukiwania, jak i modelowaniu rozwiązywania ludzkich problemów. System produkcyjny zapewnia ukierunkowane na wzorce sterowanie procesem rozwiązywania problemów i składa się z zestawu reguł produkcji, pamięci roboczej i cyklu kontroli rozpoznawania.

DEFINICJA

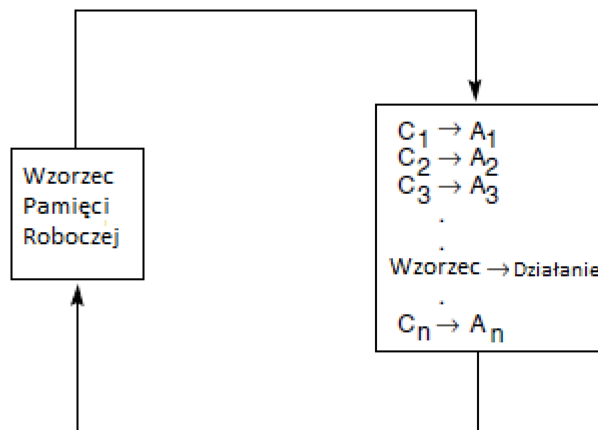
SYSTEM PRODUKCYJNY

System produkcji jest zdefiniowany przez:

1. Zestaw reguł produkcji. Są to często po prostu produkcje. Produkcja jest parą warunek-działanie i definiuje pojedynczy fragment wiedzy na temat rozwiązywania problemów. Warunek stanowi część reguły, która określa, kiedy reguła może być zastosowana do wystąpienia problemu. Część czynności określa powiązane kroki rozwiązywania problemów.

2. Pamięć robocza zawiera opis aktualnego stanu świata w procesie wnioskowania. Opis ten jest wzorcem dopasowanym do warunku części produkcji, aby wybrać odpowiednie działania rozwiązywania problemów. Gdy element warunku reguły jest dopasowany do zawartości pamięci roboczej, wówczas można wykonać działania powiązane z tym warunkiem. Działania reguł produkcji są specjalnie zaprojektowane w celu zmiany zawartości pamięci roboczej.

3. Cykl rozpoznawania-działania. Struktura sterowania dla systemu produkcyjnego jest prosta: pamięć robocza jest inicjowana początkowym opisem problemu. Obecny stan rozwiązywania problemów jest utrzymywany jako zestaw wzorców w pamięci roboczej. Wzory te są dopasowane do warunków reguł produkcji; tworzy to podzbiór reguł produkcji, zwany zestawem konfliktów, którego warunki są zgodne z wzorcami w pamięci roboczej. Mówi się, że produkcje w zestawie konfliktów są włączone. Następnie wybiera się jedną z produkcji w zestawie konfliktów (rozwiązywanie konfliktów) i uruchamia produkcję. Aby uruchomić regułę, wykonuje się jej działanie, zmieniając zawartość pamięci roboczej. Po uruchomieniu wybranej produkcji cykl sterowania powtarza się ze zmodyfikowaną pamięcią roboczą. Proces kończy się, gdy zawartość pamięci roboczej nie spełnia warunków żadnej reguły. Rozwiązywanie konfliktów wybiera regułę z zestawu konfliktów do strzelania. Strategie rozwiązywania konfliktów mogą być proste, takie jak wybranie pierwszej reguły, której stan odpowiada stanowi świata, lub może obejmować złożoną heurystykę wyboru reguł. Jest to ważny sposób, w jaki system produkcyjny pozwala na dodanie kontroli heurystycznej do algorytmu wyszukiwania. Model czystego systemu produkcyjnego nie ma mechanizmu odzyskiwania z ślepych zaułków podczas poszukiwań; po prostu trwa, dopóki nie zostaną włączone żadne produkcje i nie zatrzyma się. Większość praktycznych wdrożeń systemów produkcyjnych pozwala na powrót do poprzedniego stanu pamięci roboczej w takich sytuacjach. Schemat systemu produkcyjnego przedstawiono na rysunku



Bardzo prosty przykład wykonania systemu produkcyjnego pokazano na rysunku.

Iteracja #	Pamięć robocza	Zbiór konfliktu	Reguła Wyzwolona
0	cbaca	1, 2, 3	1
1	cabca	2	2
2	acbca	2, 3	2
3	acbac	1, 3	1
4	acabc	2	2
5	aacbc	3	3
6	aabcc	Ø	Halt

Jest to program systemu produkcyjnego do sortowania łańcucha złożonego z liter a, b i c. W tym przykładzie produkcja jest włączona, jeśli jej stan odpowiada części ciągu w pamięci roboczej. Po uruchomieniu reguły podciąg pasujący do warunku reguły jest zastępowany łańcuchem po prawej stronie reguły. Systemy produkcyjne to ogólny model obliczeń, który można zaprogramować do robienia wszystkiego, co można zrobić na komputerze. Ich prawdziwą siłą jest jednak architektura systemów opartych na wiedzy. Pomysł projektowania opartego na produkcji dla komputerów powstał pierwotnie z pism Posta, który zaproponował model reguł produkcji jako formalną teorię obliczeń. Głównym konstruktem tej teorii był zestaw reguł przepisywania ciągów znaków na wiele sposobów podobnych do reguł parsowania w przykładzie 3.3.6. Jest również ściśle związany z podejściem przyjętym przez algorytmy Markowa (Markov 1954) i, podobnie jak one, ma moc równoważną maszynie Turinga. Interesujące zastosowanie reguł produkcji do modelowania ludzkiego poznania znajduje się w pracach Newella i Simona w Carnegie Institute of Technology (obecnie Carnegie Mellon University) w latach 60. i 70. XX wieku. Opracowane przez nich programy, w tym ogólne rozwiązywanie problemów, są w dużej mierze odpowiedzialne za znaczenie systemów produkcyjnych w AI. W tych badaniach ludzi monitorowano w różnych czynnościach rozwiązywania problemów, takich jak rozwiązywanie problemów w logice predykatów i granie w gry takie jak szachy. Protokół (wzorce zachowań, w tym słowne opisy procesu rozwiązywania problemów, ruchy gałek ocznych itp.) Osób rozwiązujących problemy zostały zapisane i rozbite na podstawowe elementy. Składniki te zostały uznane za podstawowe elementy wiedzy na temat rozwiązywania problemów u ludzi i zostały utworzone jako przeszukanie wykresu (zwanego wykresem zachowań problemowych). Następnie zastosowano system produkcyjny do wdrożenia earch tego wykresu. Reguły produkcji reprezentowały

zestaw umiejętności rozwiązywania problemów przez człowieka. Obecne skupienie uwagi było reprezentowane jako obecny stan świata. Podczas uruchamiania systemu produkcyjnego „uwaga” lub „bieżąca koncentracja” rozwiązania problemu pasowałaby do reguły produkcyjnej, która zmieniałaby stan „uwagi” w celu dopasowania do innej zakodowanej w produkcji umiejętności i tak dalej. Należy zauważyć, że w tej pracy Newell i Simon wykorzystali system produkcyjny nie tylko jako narzędzie do wdrażania wyszukiwania grafów, ale także jako rzeczywisty model postępowania człowieka w rozwiązywaniu problemów. Produkcje odpowiadały umiejętności rozwiązywania problemów w długotrwałej pamięci człowieka. Podobnie jak umiejętności w zakresie pamięci długoterminowej, produkcje te nie ulegają zmianie w wyniku działania systemu; są przywoływane przez „wzorzec” konkretnej instancji problemu, a nowe umiejętności mogą być dodawane bez konieczności „przekodowywania” wcześniejszej wiedzy. Pamięć robocza systemu produkcyjnego odpowiada pamięci krótkotrwałej lub bieżącej koncentracji uwagi u człowieka i opisuje aktualny etap rozwiązania problemu. Zawartość pamięci roboczej zasadniczo nie jest zachowywana po rozwiązaniu problemu. Te początki technologii systemu produkcji są dalej opisane w Human Problem Solving przez Newella i Simona oraz w Luger. Newell, Simon i inni nadal używają reguł produkcji do modelowania różnicy między nowicjuszami a ekspertami w takich obszarach, jak rozwiązywanie problemów ze słowami algebry i problemów fizyki. Systemy produkcyjne stanowią również podstawę do nauki uczenia się zarówno na ludziach, jak i na komputerach. ACT i SOAR bazują na tej tradycji. Systemy produkcyjne zapewniają model kodowania ludzkiej wiedzy specjalistycznej w formie reguł i projektowania algorytmów wyszukiwania opartych na wzorach, zadań, które są kluczowe w projektowaniu systemu eksperckiego opartego na regułach. W systemach eksperckich niekoniecznie zakłada się, że system produkcyjny faktycznie modeluje ludzkie zachowanie podczas rozwiązywania problemów; jednak aspekty systemów produkcyjnych, które czynią je użytecznymi jako potencjalny model rozwiązywania problemów człowieka (modułowość reguł, rozdział wiedzy i kontroli, rozdział pamięci roboczej i wiedzy na temat rozwiązywania problemów), czynią je idealnym narzędziem do projektowania i budowania ekspertów systemy. Ważna rodzina języków AI pochodzi bezpośrednio z badań języka systemu produkcyjnego w Carnegie Mellon. To są języki OPS; OPS oznacza oficjalny system produkcji. Chociaż ich początki polegają na modelowaniu rozwiązywania problemów ludzkich, języki te okazały się bardzo skuteczne w programowaniu systemów eksperckich i innych aplikacji AI. OPS5 był językiem implementacyjnym dla konfiguratora VAX XCON i innych wczesnych systemów eksperckich opracowanych w Digital Equipment Corporation. Tłumacze OPS są szeroko dostępni na komputery PC i stacje robocze. CLIPS, zaimplementowany w języku programowania C, jest szeroko stosowaną, obiektową wersją systemu produkcyjnego zbudowanego przez NASA. JESS, system produkcyjny wdrożony w Javie, został stworzony przez Sandia National Laboratories. W następnej sekcji podajemy przykłady wykorzystania systemu produkcyjnego do rozwiązywania różnych problemów związanych z wyszukiwaniem.

Przykłady systemów produkcyjnych

PRZYKŁAD: ZMIENIONO 8-PUZZLE

Przestrzeń wyszukiwania wygenerowana przez 8-puzzle, , jest zarówno wystarczająco złożona, aby była interesująca, jak i wystarczająco mała, aby była możliwa do przeszukiwania, dlatego często jest używana do eksploracji różnych strategii wyszukiwania, takich jak wyszukiwanie od pierwszej do głębokości i od pierwszego do drugiego , a także strategie heurystyczne. Prezentujemy teraz rozwiązanie systemu produkcyjnego. Przypomnijmy, że zyskujemy ogólność, myśląc raczej o „przesunięciu pustego miejsca”, a nie o przeniesieniu numerowanego kafelka. Legalne ruchy są zdefiniowane przez produkcje poniżej:

Stan startowy :

2	8	3
1	6	4
7		5

Stan docelowy :

1	2	3
8		4
7	6	5

Zestaw produkcyjny:

stan celu w pamięci roboczej -> postój

puste miejsce nie znajduje się na lewej krawędzi -> przesuń puste miejsce w lewo

puste miejsce nie znajduje się na górnej krawędzi ->przesuń puste miejsce w górę

puste miejsce nie znajduje się na prawej krawędzi -> przesuń puste miejsce w prawo

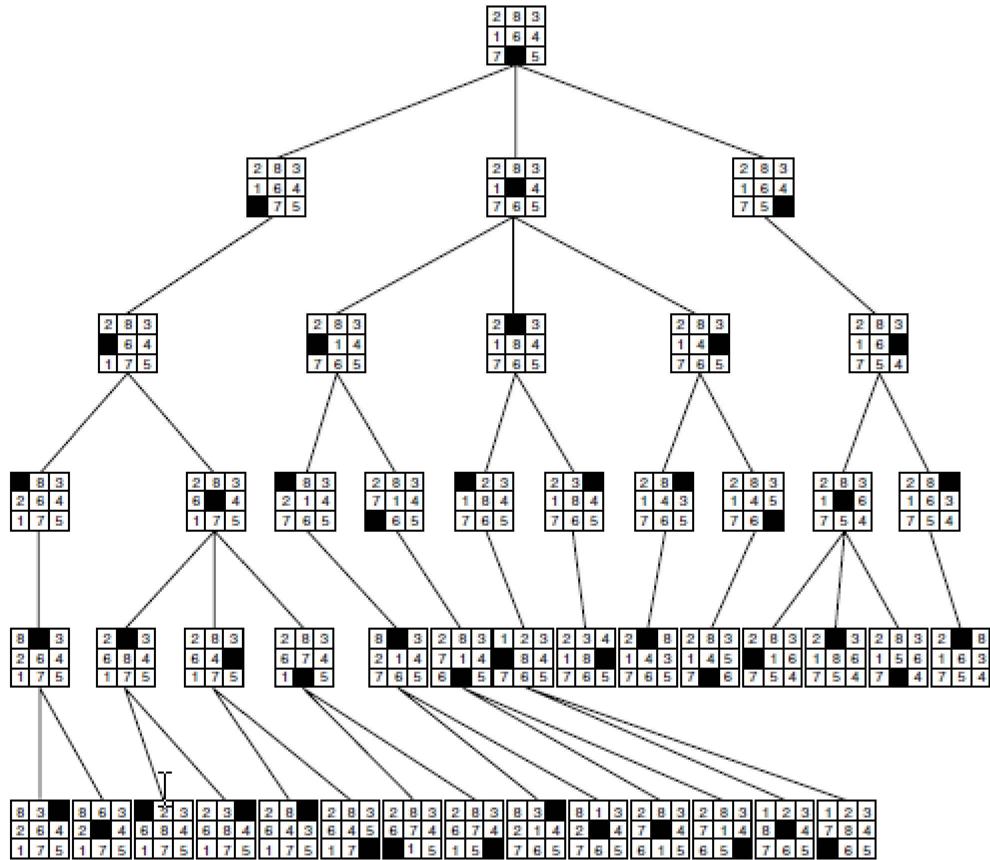
puste miejsce nie znajduje się na dole -> przesuń puste miejsce w dół

Pamięć robocza to obecny stan płyty i stan celu

System kontroli:

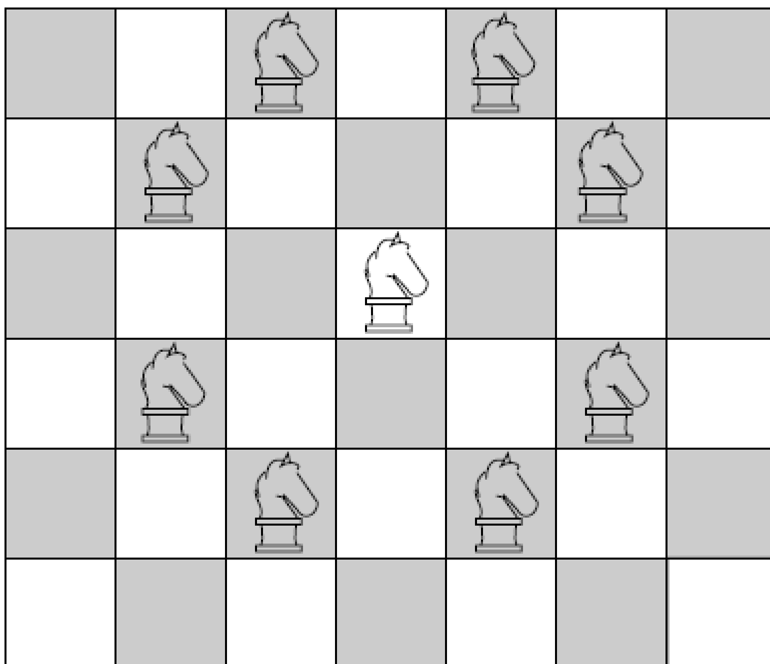
1. Wypróbuj każdą produkcję w kolejności.
2. Nie zezwalaj na pętle.
3. Zatrzymaj się, gdy cel zostanie znaleziony

Oczywiście wszystkie cztery te ruchy mają zastosowanie tylko wtedy, gdy puste miejsce znajduje się na środku; gdy jest w jednym z rogów, możliwe są tylko dwa ruchy. Jeśli określono teraz stan początkowy i stan celu dla 8 puzzli, możliwe jest, aby system produkcyjny rozliczał przestrzeń wyszukiwania problemu. Rzeczywista implementacja tego problemu może reprezentować każdą konfigurację planszy z predykatem „stanowym” z dziewięcioma parametrami (dla dziewięciu możliwych lokalizacji ośmiu płytek i pustego pola); reguły można zapisać jako implikacje, których założenie wykonuje wymaganą kontrolę stanu. Alternatywnie, dla stanów tablic można zastosować tablice lub struktury list. Przykład zaczerpnięty z Nilssona (1980) dotyczący przestrzeni poszukiwanej w celu znalezienia rozwiązania problemu podanego na rysunku znajduje się na rysunku poniżej. Ponieważ ta ścieżka rozwiązania może być bardzo głęboka, jeśli nie jest ograniczona, do wyszukiwania dodano granicę głębokości. (Prostym sposobem na dodanie granicy głębokości jest śledzenie długości / głębokości bieżącej ścieżki i wymuszenie cofania, jeśli granica ta zostanie przekroczona.) W rozwiązaniu zastosowano granicę głębokości 5 rysunku poniej. Zauważ, że liczba możliwych stanów pamięci roboczej rośnie wykładniczo wraz z głębokością wyszukiwania.



PRZYKŁAD: PROBLEM Z WYCIECZKĄ Rycerza

W grze w szachy rycerz może poruszać się o dwa pola w poziomie lub w pionie, a następnie o jedno pole w kierunku prostopadłym, o ile nie zejdzie z planszy. Istnieje więc co najwyżej osiem możliwych ruchów, które rycerz może wykonać.



Zgodnie z tradycyjną definicją problem trasy rycerza polega na znalezieniu serii legalnych ruchów, w których rycerz ląduje dokładnie na każdym polu szachownicy. Problem ten stanowił podstawę opracowania i prezentacji algorytmów wyszukiwania. Przykład, którego używamy w tym rozdziale, to uproszczona wersja problemu rycerza. Pyta, czy istnieje seria legalnych ruchów, które przeniosą rycerza z jednego pola na drugie na szachownicy o zmniejszonych rozmiarach (3 × 3). Rysunek 6.6 pokazuje szachownicę 3 × 3 z każdym kwadratem oznaczonym liczbami całkowitymi od 1 do 9. Ten schemat znakowania jest stosowany zamiast bardziej ogólnego podejścia do nadawania każdej spacji numeru wiersza i kolumny w celu dalszego uproszczenia przykładu. Z powodu zmniejszonej wielkości problemu po prostu wyliczamy ruchy alternatywne, zamiast opracowywać ogólny operator ruchów. Legalne ruchy na planszy są następnie opisywane w rachunku predykatów przy użyciu predykatu zwanego ruchem, którego parametrami są początek i koniec kwadratu legalnego ruchu. Na przykład ruch (1,8) przenosi rycerza z lewego górnego rogu na środek dolnego rzędu. Predykaty na rysunku poniżej wyliczają wszystkie możliwe ruchy szachownicy 3 × 3.

move(1,8)	move(6,1)
move(1,6)	move(6,7)
move(2,9)	move(7,2)
move(2,7)	move(7,6)
move(3,4)	move(8,3)
move(3,8)	move(8,1)
move(4,9)	move(9,2)
move(4,3)	move(9,4)

1	2	3
4	5	6
7	8	9

Problem trasy rycerskiej 3 × 3 można rozwiązać za pomocą systemu produkcyjnego. Każdy ruch może być reprezentowany jako reguła, której warunkiem jest lokalizacja rycerza na danym polu i którego działanie przenosi rycerza na inne pole. Przedstawiono szesnaście produkcji poniżej przedstawiają wszystkie możliwe ruchy rycerza.

ZASADA# WARUNEK DZIAŁANIE

1 rycerz na kwadracie 1 → przenieś rycerza na kwadrat 8

2 rycerz na kwadracie 1 → przenieś rycerza na kwadrat 6

3 rycerz na kwadracie 2 → przenieś rycerza na kwadrat 9

4 rycerz na kwadracie 2 → przenieś rycerza na kwadrat 7

5 rycerz na kwadracie 3 → przenieś rycerza na kwadrat 4

6 rycerzy na kwadracie 3 → przenieś rycerza na kwadrat 8

7 rycerz na kwadracie 4 → przenieś rycerza na kwadrat 9

8 rycerzy na kwadracie 4 → przenieś rycerza na kwadrat 3

9 rycerz na kwadracie 6 → przenieś rycerza na kwadrat 1

10 rycerzy na polu 6 → przenieś rycerza na pole 7

11 rycerz na kwadracie 7 → przenieś rycerza na kwadrat 2

12 rycerz na kwadracie 7 → przenieś rycerza na kwadrat 6

13 rycerz na kwadracie 8 → przenieś rycerza na kwadrat 3

14 rycerz na kwadracie 8 → przenieś rycerza na kwadrat 1

15 rycerz na kwadracie 9 → przenieś rycerza na kwadrat 2

16 rycerz na kwadracie 9 → przenieś rycerza na kwadrat 4

Następnie określamy procedurę rekurencyjną w celu zaimplementowania algorytmu sterowania dla systemu produkcyjnego. Ponieważ ścieżka (X, X) ujednolici się tylko z predykatami, takimi jak ścieżka (3,3) lub ścieżka (5,5), określa pożądany warunek zakończenia. Jeśli ścieżka (X, X) nie powiedzie się, sprawdzamy reguły produkcji dla możliwego następnego stanu, a następnie powtarzamy się. Ogólna definicja ścieżki rekurencyjnej jest następnie podawana w dwóch predykatowych formułach rachunku różniczkowego:

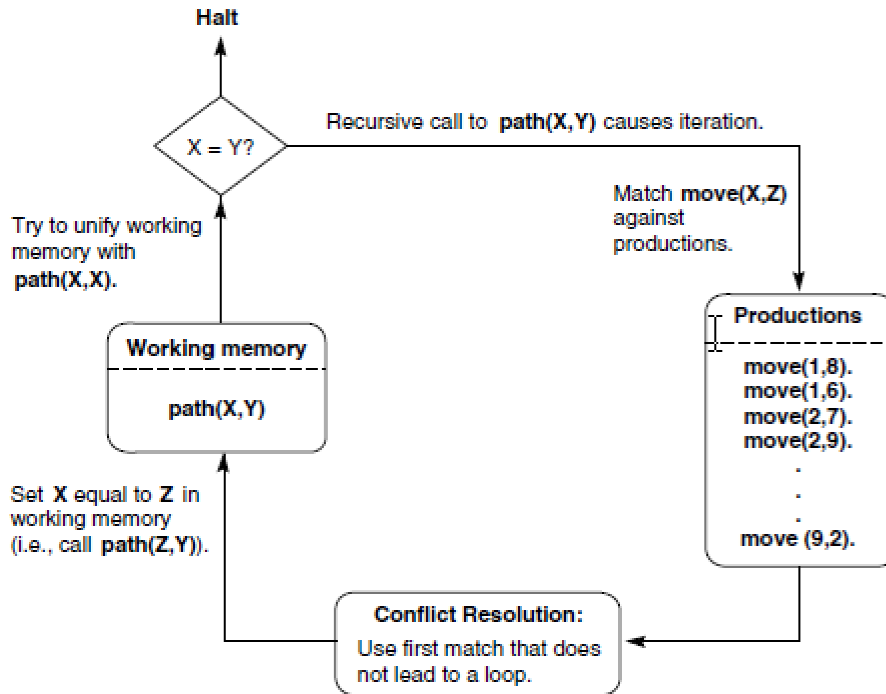
Path Ścieżka X (X, X)

∀ X, Y [ścieżka (X, Y) ← ∃ Z [ruch (X, Z) ∧ ścieżka (Z, Y)

Pamięć robocza, parametry predykatu ścieżki rekurencyjnej, zawiera zarówno bieżący stan płytki, jak i stan celu. System kontroli stosuje reguły, dopóki obecny stan nie zrówna się z celem, a następnie się zatrzyma. Prosty schemat rozwiązywania konfliktów uruchomiłby pierwszą regułę, która nie spowodowała zapętlenia wyszukiwania. Ponieważ wyszukiwanie może prowadzić do ślepych zaułków (z których każdy możliwy ruch prowadzi do poprzednio odwiedzonego stanu, a tym samym pętli), reżim kontroli musi również umożliwiać powrót do poprzedniego stanu; wykonanie tego systemu produkcyjnego, który określa, czy istnieje ścieżka od kwadratu 1 do kwadratu 2, pokazano na rysunku

Iteration #	Working memory		Conflict set (rule #'s)	Fire rule
	Current square	Goal square		
0	1	2	1, 2	1
1	8	2	13, 14	13
2	3	2	5, 6	5
3	4	2	7, 8	7
4	9	2	15, 16	15
5	2	2		Halt

Ta charakterystyka definicji ścieżki jako systemu produkcyjnego jest podana na rysunku .



Systemy produkcyjne są w stanie generować nieskończone pętle podczas przeszukiwania wykresu przestrzeni stanów. Pętle te są szczególnie trudne do wykrycia w systemie produkcyjnym, ponieważ reguły mogą być uruchamiane w dowolnej kolejności. Oznacza to, że podczas wykonywania systemu może pojawić się zapętlenie, ale nie można go łatwo znaleźć na podstawie kontroli składni zestawu reguł. Na przykład przy regułach „ruchu" problemu trasy rycerza uporządkowanych jak w tabeli 1 i strategii rozwiązywania konfliktu wyboru pierwszego dopasowania, ruch wzorca (2, X) pasowałby do ruchu (2,9), wskazując przejdź do kwadratu 9. Podczas następnej iteracji ruch wzoru (9, X) będzie pasował do ruchu (9,2), przenosząc wyszukiwanie z powrotem na kwadrat 2, powodując pętlę. Aby zapobiec zapętleniu, wzorzec_wyszukiwania sprawdził globalną listę (zamkniętych) odwiedzonych stanów. Rzeczywista strategia rozwiązywania konfliktów była zatem: wybierz pierwszy pasujący ruch, który prowadzi do niezapowiedzianego stanu. W systemie produkcyjnym właściwym miejscem do rejestrowania danych specyficznych dla przypadku, takich jak lista wcześniej odwiedzonych stanów, nie jest globalna zamknięta lista, ale sama pamięć robocza. assert służy do umieszczania „znacznika" w pamięci roboczej, aby wskazać, kiedy stan został odwiedzony. Ten znacznik jest reprezentowany jako jednoargumentowy predykat, był (X), który jako argument przyjmuje kwadrat na planszy. been (X) jest dodawany do pamięci roboczej, gdy nowy stan X jest odwiedzany. Rozwiązanie konfliktu może wtedy wymagać, aby wcześniej (Z) nie mógł znajdować się w pamięci roboczej, zanim ruch (X, Z) będzie mógł zostać uruchomiony. W przypadku określonej wartości Z można to sprawdzić, dopasowując wzorzec do pamięci roboczej. Zmodyfikowany kontroler ścieżki rekurencyjnej dla systemu produkcyjnego to:

Path Ścieżka X (X, X)

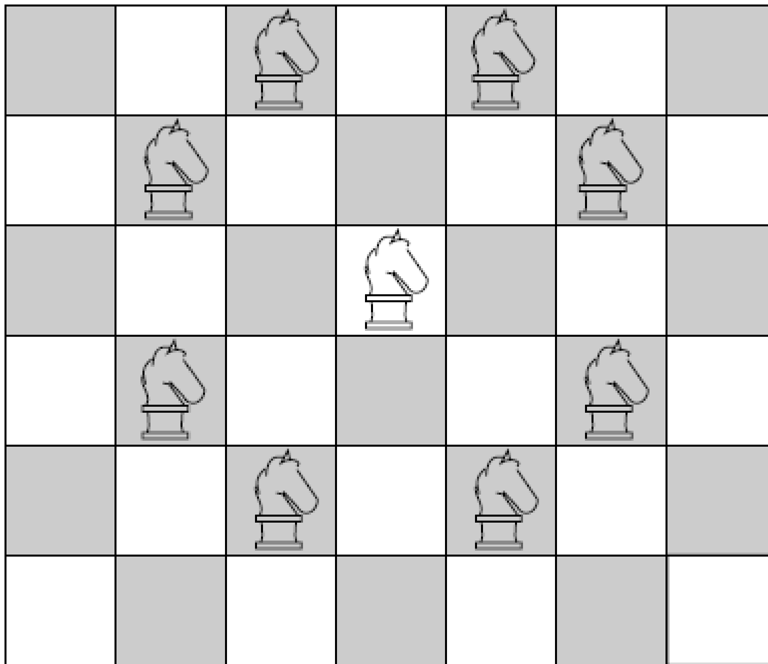
∀ X, Y [ścieżka (X, Y) ← ∃ Z [ruch (X, Z) ∧ ¬ (były (Z)) ∧ potwierdzają (były (Z)) ∧ ścieżka (Z, Y)]]

W tej definicji ruch (X, Z) kończy się pierwszym meczem z predykatem ruchu. Wiąże to wartość z Z. Jeśli wcześniej (Z) pasuje do wpisu w pamięci roboczej, ¬ (był (Z)) spowoduje awarię (tj. Będzie fałszem). wzorzec_wyszukiwania następnie cofa się i próbuje innego dopasowania dla ruchu (X, Z). Jeśli kwadratowy Z jest nowym stanem, wyszukiwanie będzie kontynuowane, z uwzględnieniem (Z) w pamięci roboczej, aby zapobiec przyszłym pętlom. Faktyczne uruchomienie produkcji ma miejsce, gdy

algorytm ścieżki się powtarza. Zatem obecność wcześniejszych predykatów w pamięci roboczej implementuje wykrywanie pętli w tym systemie produkcyjnym.

PRZYKŁAD [6.2.3]: WYCIECZKA PEŁNEGO RYCERZA

Możemy uogólnić rozwiązanie wycieczki rycerza do pełnej szachownicy 8 × 8. Ponieważ nie ma sensu wyliczanie ruchów dla tak złożonego problemu, zastępujemy 16 faktów ruchów zestawem 8 reguł, aby wygenerować legalne ruchy rycerza. Te ruchy (produkcje) odpowiadają 8 możliwym sposobom ruchu rycerza.



Jeśli zindeksujemy szachownicę według numerów wierszy i kolumn, możemy zdefiniować regułę produkcji dla przesunięcia rycerza w dół o dwa pola i prawo o jedno pole:

WARUNEK: bieżący wiersz 6 ∧ bieżąca kolumna 7

DZIAŁANIE: nowy wiersz = bieżący wiersz + 2 ∧ nowa kolumna = bieżąca kolumna + 1

Jeśli użyjemy rachunku predykatu do przedstawienia produkcji, kwadrat kwadratu może być zdefiniowany przez kwadrat predykatu (R, C), reprezentujący rząd R i kolumnę C. Powyższą zasadę można przepisać w rachunku predykatów jako:

przesuń (kwadrat (wiersz, kolumna), kwadrat (Newrow, Newcolumn)) ←

less_than_or_equals (Row, 6) ∧

równa się (Newrow, plus (Row, 2)) ∧

less_than_or_equals (kolumna, 7) ∧

równa się (Newcolumn, plus (Column, 1))

plus to funkcja dodawania; less_than_or_equals i equals mają oczywiste interpretacje arytmetyczne. Można zaprojektować siedem dodatkowych reguł, które obliczą pozostałe możliwe ruchy. Te zasady zastępują fakty dotyczące przenoszenia w wersji problemu 3 × 3. Definicja ścieżki z przykładu rycerza 3 × 3 oferuje również pętlę sterowania dla tego problemu. Jak widzieliśmy, kiedy opisy rachunku

predykatu są interpretowane proceduralnie, na przykład za pomocą algorytmu wzorzec_wyszukiwania, subtelne zmiany są wprowadzane do semantyki rachunku predykatu. Jedną z takich zmian jest sekwencyjny sposób rozwiązywania celów. Narzuca to uporządkowanie lub semantykę proceduralną interpretacji wyrażeń rachunku predykatu. Kolejną zmianą jest wprowadzenie predykatów meta-logicznych, takich jak aser, które wskazują działania wykraczające poza interpretację wartości prawdowych wyrażeń rachunku predykatu.

PRZYKŁAD: DORADCA FINANSOWY JAKO SYSTEM PRODUKCJI

Wcześniej opracowaliśmy małego doradcę finansowego, używając rachunku predykatów do reprezentowania wiedzy finansowej i wyszukiwania grafów w celu dokonania odpowiednich wniosków podczas konsultacji. System produkcyjny stanowi naturalny nośnik do jego wdrożenia. Implikacje logicznego opisu tworzą produkcje. Informacje dotyczące konkretnego przypadku (wynagrodzenie danej osoby, osoby pozostające na utrzymaniu itp.) Są ładowane do pamięci roboczej. Reguły są włączane, gdy ich przesłanki są spełnione. Reguła jest wybierana z tego zestawu konfliktów i uruchamiana, dodając swoje zakończenie do pamięci roboczej. Trwa to do momentu dodania wszystkich możliwych wniosków najwyższego poziomu do pamięci roboczej. Rzeczywiście, wiele „powłok” systemu eksperckiego, w tym JESS i CLIPS, to systemy produkcyjne z dodatkowymi funkcjami do obsługi interfejsu użytkownika, obsługi niepewności w uzasadnieniu, edycji wiedzy wykonanie bazy i śledzenia

[6.2.3]

Kontrola wyszukiwania w systemach produkcyjnych

Model systemu produkcyjnego oferuje szereg możliwości dodania kontroli heurystycznej do algorytmu wyszukiwania. Należą do nich wybór strategii wyszukiwania opartych na danych lub celu, struktura samych reguł oraz wybór różnych opcji rozwiązywania konfliktów.

Kontrola poprzez wybór strategii wyszukiwania opartej na danych lub celu Wyszukiwanie oparte na danych rozpoczyna się od opisu problemu (takiego jak zbiór logicznych aksjomatów, objawów choroby lub zbioru danych wymagających interpretacji) i wyciąga z nich nową wiedzę. Odbywa się to poprzez zastosowanie reguł wnioskowania, legalnych ruchów w grze lub innych operacji generujących stan do bieżącego opisu świata i dodanie wyników do opisu problemu. Proces ten trwa do momentu osiągnięcia celu. Ten opis wnioskowania opartego na danych podkreśla jego ścisłe dopasowanie do modelu obliczeniowego systemu produkcyjnego. „Bieżący stan świata” (dane, które albo uznano za prawdziwe, albo wydedukowano jako prawdziwe przy wcześniejszym użyciu reguł produkcji) są umieszczane w pamięci roboczej. Cykl rozpoznawania-działania dopasowuje następnie aktualny stan do (uporządkowanego) zestawu produkcji. Kiedy te dane pasują (są zunifikowane) do warunków jednej z reguł produkcji, działanie produkcji dodaje (modyfikując pamięć roboczą) nową informację do aktualnego stanu świata. Wszystkie produkcje mają postać WARUNEK → DZIAŁANIE. Kiedy WARUNEK pasuje do niektórych elementów pamięci roboczej, wykonywana jest AKCJA. Jeśli reguły produkcji są sformułowane jako logiczne implikacje, a AKCJA dodaje twierdzenia do pamięci roboczej, wówczas odpalenie reguły może odpowiadać zastosowaniu modus ponens reguły reguły wnioskowania. To tworzy nowy stan wykresu. Rycina 6.9 przedstawia proste wyszukiwanie oparte na danych na zestawie produkcji wyrażonych jako implikacje rachunku zdań.

Production set:

1. p ∧ q → goal
2. r ∧ s → p
3. w ∧ r → q
4. t ∧ u → q
5. v → s
6. start → v ∧ r ∧ q

Trace of execution:

Iteration #	Working memory	Conflict set	Rule fired
0	start	6	6
1	start, v, r, q	6, 5	5
2	start, v, r, q, s	6, 5, 2	2
3	start, v, r, q, s, p	6, 5, 2, 1	1
4	start, v, r, q, s, p, goal	6, 5, 2, 1	halt

Space searched by execution:

Strategia rozwiązywania konfliktów polega na wybraniu włączonej reguły, która została uruchomiona co najmniej niedawno (lub wcale); w przypadku powiązań wybierana jest pierwsza zasada. Realizacja zostaje zatrzymana, gdy cel zostanie osiągnięty. Na rysunku przedstawiono również sekwencję odpalenia reguł i etapy pamięci roboczej w wykonaniu wraz z wykresem przeszukiwanej przestrzeni. Do tego momentu potraktowaliśmy systemy produkcyjne w sposób oparty na danych; mogą jednak być również wykorzystywane do wyszukiwania ukierunkowanego na cel. Jak zdefiniowano wcześniej, wyszukiwanie goaldriven rozpoczyna się od celu i działa wstecz do faktów problemu, aby go osiągnąć. Aby zaimplementować to w systemie produkcyjnym, cel jest umieszczany w pamięci roboczej i dopasowywany do AKCJI reguł produkcji. Te AKCJE są dopasowane (na przykład przez ujednolicenie), podobnie jak WARUNKI produkcji zostały dopasowane w rozumowaniu opartym na danych. Wszystkie reguły produkcji, których wnioski (AKCJE) są zgodne z celem, stanowią zbiór konfliktów. Gdy AKCJA reguły jest dopasowana, WARUNKI są dodawane do pamięci roboczej i stają się nowymi podzbiorami (stanami) wyszukiwania. Nowe stany są następnie dopasowywane do AKCJI innych reguł produkcji. Proces ten trwa do momentu znalezienia faktów, zwykle w początkowym opisie problemu lub, jak to często bywa w systemach eksperckich, poprzez bezpośrednie poproszenie użytkownika o określone informacje. Wyszukiwanie kończy się, gdy okaże się, że WARUNKI obsługujące produkcje wystrzelone w ten sposób wstecz, które prowadzą do celu, są prawdziwe. WARUNKI i łańcuch ostrzeżeń prowadzących do celu stanowią dowód jego prawdy poprzez kolejne wnioski, takie jak modus ponens. Zobacz rysunek, gdzie znajduje się przykład wnioskowania ukierunkowanego na cel na tym samym zestawie produkcji, co na rycinie powyżej.

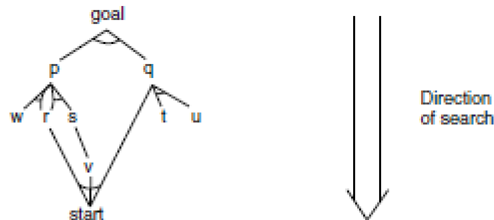
Production set:

1. p ∧ q → goal
2. r ∧ s → p
3. w ∧ r → p
4. t ∧ u → q
5. v → s
6. start → v ∧ r ∧ q

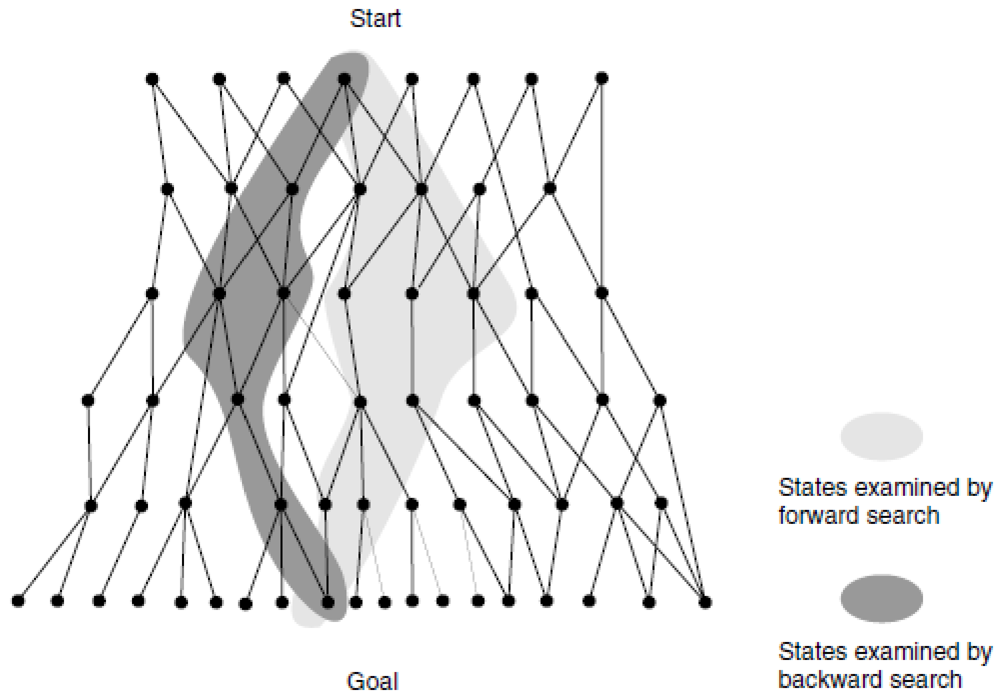
Trace of execution:

Iteration #	Working memory	Conflict set	Rule fired
0	goal	1	1
1	goal, p, q	1, 2, 3, 4	2
2	goal, p, q, r, s	1, 2, 3, 4, 5	3
3	goal, p, q, r, s, w	1, 2, 3, 4, 5	4
4	goal, p, q, r, s, w, t, u	1, 2, 3, 4, 5	5
5	goal, p, q, r, s, w, t, u, v	1, 2, 3, 4, 5, 6	6
6	goal, p, q, r, s, w, t, u, v, start	1, 2, 3, 4, 5, 6	halt

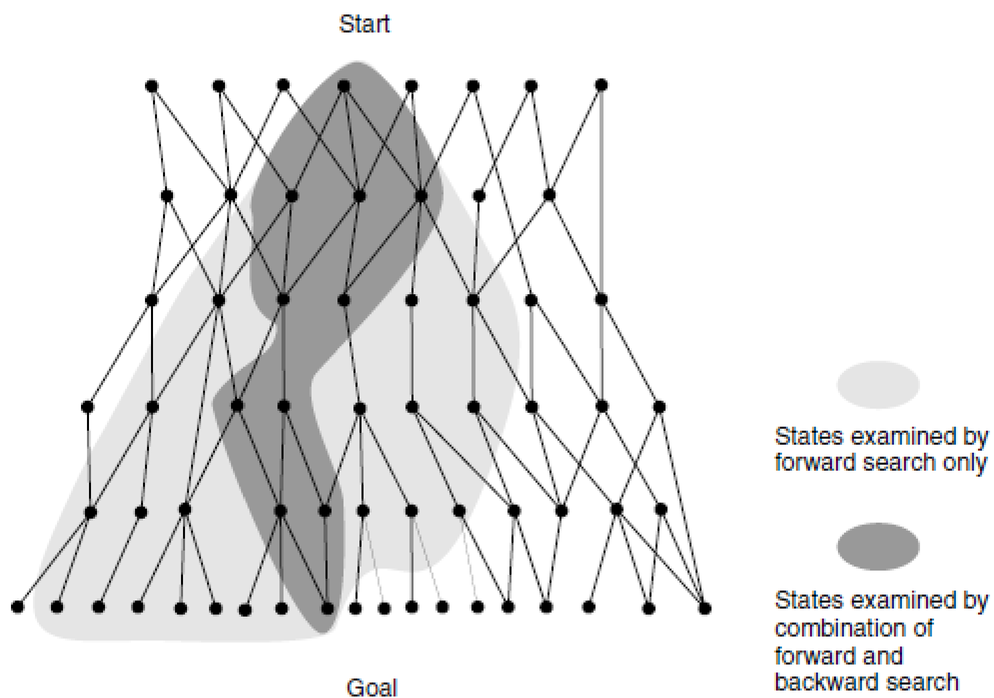
Space searched by execution:



Pamiętaj, że wyszukiwanie zorientowane na cel uruchamia inną serię produkcji i szuka innej przestrzeni niż wersja oparta na danych. Jak pokazuje ta dyskusja, system produkcyjny oferuje naturalną charakterystykę wyszukiwania ukierunkowanego na cel i danych. Reguły produkcji to zakodowany zestaw wniosków („wiedza" w systemie eksperckim opartym na regułach) do zmiany stanu na wykresie. Kiedy aktualny stan świata (zbiór prawdziwych stwierdzeń opisujących świat) odpowiada WARUNKOM reguł produkcji i to dopasowanie powoduje, że część AKCJA reguły tworzy kolejny (prawdziwy) deskryptor dla świata, jest to określane jako wyszukiwanie oparte na danych. Alternatywnie, gdy cel jest dopasowany do części AKCJA reguł w zestawie reguł produkcji, a ich WARUNKI są następnie ustawiane jako podzadania, które mają być „prawdziwe" (przez dopasowanie AKCJI reguł w następnym cyklu system produkcji), wynikiem jest ukierunkowane na rozwiązywanie problemów. Ponieważ zestaw reguł może być wykonywany w sposób oparty na danych lub celu, możemy porównywać i kontrastować skuteczność każdego podejścia w kontrolowaniu wyszukiwania. Złożoność poszukiwania którejkolwiek ze strategii jest mierzona takimi pojęciami, jak czynnik rozgałęziający lub penetracja. Te miary złożoności wyszukiwania mogą zapewnić oszacowanie kosztów zarówno dla wersji rozwiązania problemów, jak i danych, a tym samym pomóc w wyborze najbardziej skutecznej strategii. Możemy również stosować kombinacje strategii. Na przykład możemy wyszukiwać w kierunku do przodu, aż liczba stanów stanie się duża, a następnie przełączyć się na wyszukiwanie ukierunkowane na cel, aby użyć możliwych podzadań do wyboru spośród alternatywnych stanów. Niebezpieczeństwo w tej sytuacji polega na tym, że w przypadku wyszukiwania heurystycznego lub najlepszego wyszukiwania (rozdział 4) części faktycznie przeszukiwanych wykresów mogą się wzajemnie „ominąć" i ostatecznie wymagać więcej wyszukiwania niż prostszego podejścia, jak pokazano na rysunku



Jednak gdy rozgałęzienie przestrzeni jest stałe i stosowane jest wyczerpujące wyszukiwanie, łączona strategia wyszukiwania może drastycznie zmniejszyć ilość przeszukiwanego miejsca, jak pokazano na rysunku.



Kontrola wyszukiwania poprzez strukturę reguł

Struktura reguł w systemie produkcyjnym, w tym rozróżnienie między warunkiem a działaniem oraz kolejnością, w jakiej warunki są wypróbowywane, określa sposób przeszukiwania przestrzeni.

Wprowadzając rachunek predykatów jako język reprezentacyjny, podkreśliliśmy deklaratywny charakter jego semantyki. Oznacza to, że wyrażenia rachunku predykatu po prostu definiują prawdziwe relacje w dziedzinie problemów i nie twierdzą o ich kolejności interpretacji. Zatem indywidualną regułą może być $\forall X\ (foo\ (X) \land goo\ (X) \rightarrow moo\ (X))$. Zgodnie z regułami rachunku predykatów alternatywną formą tej samej reguły jest $\forall X\ (foo\ (X) \rightarrow moo\ (X) \lor \neg\ goo\ (X))$. Równoważność tych dwóch klauzul pozostawia się jako ćwiczenie. Chociaż te formuły są logicznie równoważne, nie prowadzą do takich samych rezultatów, gdy są interpretowane jako produkcje, ponieważ system produkcyjny narzuca porządek w dopasowywaniu i uruchamianiu reguł. Tak więc określona forma reguł określa łatwość (lub możliwość) dopasowania reguły do wystąpienia problemu. Jest to wynikiem różnic w sposobie, w jaki system produkcyjny interpretuje reguły. System produkcji narzuca semantykę proceduralną na język deklaratywny używany do tworzenia reguł. Ponieważ system produkcyjny wypróbowuje każdą z reguł w określonej kolejności, programista może kontrolować wyszukiwanie w strukturze i kolejności reguł w zestawie produkcyjnym. Chociaż logicznie równoważne, $\forall X\ (foo\ (X) \land goo\ (X) \rightarrow moo\ (X))$ i $\forall X\ (foo\ (X) \rightarrow moo\ (X) \lor \neg\ goo\ (X))$ nie zachowują się tak samo wdrożenie wyszukiwania. Ludzcy eksperci kodują kluczowe heurystyki w ramach swoich zasad ekspertyzy. Kolejność przesłania koduje często krytyczne informacje proceduralne w celu rozwiązania problemu. Ważne jest, aby zachować tę formę przy tworzeniu programu, który „rozwiązuje problemy jak ekspert". Kiedy mechanik mówi: „Jeśli silnik się nie przekręci, a światła się nie zapalą, sprawdź akumulator", sugeruje określoną sekwencję działań. Informacje te nie są przechwytywane przez logicznie równoważne stwierdzenie „silnik się obraca lub zapala się lampka lub sprawdź akumulator". Ta forma reguł ma kluczowe znaczenie w kontrolowaniu wyszukiwania, dzięki czemu system zachowuje się logicznie, a porządek uruchamiania reguł jest bardziej zrozumiały.

Kontrola wyszukiwania poprzez rozwiązywanie konfliktów

Chociaż systemy produkcyjne (jak wszystkie architektury systemów opartych na wiedzy) pozwalają na zakodowanie heurystyki w treści wiedzy o samych regułach, oferują inne możliwości kontroli heurystycznej poprzez rozwiązywanie konfliktów. Chociaż najprostszą taką strategią jest wybranie pierwszej reguły pasującej do zawartości pamięci roboczej, każda strategia może potencjalnie zostać zastosowana do rozwiązywania konfliktów. Na przykład strategie rozwiązywania konfliktów obsługiwane przez OPS5 (Brownston i in. 1985) obejmują:

1. Refrakcja. Załamanie określa, że po uruchomieniu reguły reguła może nie zostać ponownie uruchomiona, dopóki elementy pamięci roboczej odpowiadające jej warunkom nie zostaną zmodyfikowane. To zniechęca do zapętlania.

2. Niedawność. Strategia aktualności preferuje reguły, których warunki są zgodne z wzorcami ostatnio dodanymi do pamięci roboczej. Skupia to wyszukiwanie na jednej linii rozumowania.

3. Specyfika. Strategia ta zakłada, że należy zastosować bardziej szczegółową zasadę rozwiązywania problemów, a nie bardziej ogólną. Jedna reguła jest bardziej szczegółowa niż inna, jeśli ma więcej warunków, co oznacza, że będzie pasować do mniejszej liczby wzorców pamięci roboczej.

Zalety systemów produkcyjnych dla AI

Jak pokazano w poprzednich przykładach, system produkcyjny oferuje ogólne ramy wdrażania wyszukiwania. Ze względu na swoją prostotę, modyfikowalność i elastyczność w stosowaniu wiedzy na temat rozwiązywania problemów, system produkcji okazał się ważnym narzędzie do budowy systemów eksperckich i innych aplikacji AI. Główne zalety systemów produkcji sztucznej inteligencji obejmują:

Rozdział wiedzy i kontroli. System produkcyjny to elegancki model rozdziału wiedzy i kontroli w programie komputerowym. Kontrolę zapewnia cykl rozpoznawania-działania w pętli systemu produkcyjnego, a wiedza na temat rozwiązywania problemów jest zakodowana w samych regułach. Zalety tego rozdzielenia obejmują łatwość modyfikacji bazy wiedzy bez konieczności zmiany kodu do sterowania programem i, przeciwnie, możliwość zmiany kodu do sterowania programem bez zmiany zestawu reguł produkcji.

Naturalne mapowanie na wyszukiwanie w przestrzeni stanu. Komponenty systemu produkcyjnego mapują się naturalnie w konstrukcje przeszukiwania przestrzeni stanów. Kolejne stany pamięci roboczej tworzą węzły wykresu przestrzeni stanów. Reguły produkcji to zbiór możliwych przejść między stanami, przy czym rozwiązywanie konfliktów implementuje wybór gałęzi w przestrzeni stanu. Reguły te upraszczają implementację, debugowanie i dokumentację algorytmów wyszukiwania.

Modułowość reguł produkcji. Ważnym aspektem modelu systemu produkcyjnego jest brak jakichkolwiek interakcji składniowych między regułami produkcji. Reguły mogą wpływać na odpalanie innych reguł tylko poprzez zmianę wzorca w pamięci roboczej; nie mogą „dzwonić” kolejną regułą bezpośrednio, tak jakby to była podprogram innych reguł produkcji. Zakres zmiennych tych reguł jest ograniczony do jednostki reguły. Ta niezależność składniowa wspiera stopniowy rozwój systemów eksperckich poprzez sukcesywne dodawanie, usuwanie lub zmienianie wiedzy (zasad) systemu.

Sterowanie według wzorca. Problemy rozwiązywane przez programy AI często wymagają szczególnej elastyczności w wykonywaniu programu. Celowi temu służy fakt, że reguły w systemie produkcyjnym mogą strzelać w dowolnej kolejności. Opisy problemu, które składają się na bieżący stan świata, determinują zestaw konfliktów, aw konsekwencji konkretną ścieżkę wyszukiwania i rozwiązanie, ani nie mogą ustawiać wartości zmiennych w innych regułach produkcji. Zakres zmiennych tych reguł jest ograniczony do reguły indywidualnej. Ta niezależność składniowa wspiera stopniowy rozwój systemów eksperckich poprzez sukcesywne dodawanie, usuwanie lub zmienianie wiedzy (zasad) systemu.

Sterowanie według wzorca. Problemy rozwiązywane przez programy AI często wymagają szczególnej elastyczności w wykonywaniu programu. Celowi temu służy fakt, że reguły w systemie produkcyjnym mogą strzelać w dowolnej kolejności. Opisy problemu, które składają się na bieżący stan świata, determinują zestaw konfliktów, aw konsekwencji konkretną ścieżkę wyszukiwania i rozwiązanie

Możliwości heurystycznej kontroli wyszukiwania. (W poprzedniej części opisano kilka technik kontroli heurystycznej).

Śledzenie i objaśnienie. Modułowość reguł i iteracyjny charakter ich wykonywania ułatwiają śledzenie wykonania systemu produkcyjnego. Na każdym etapie cyklu rozpoznawania czynności można wyświetlić wybraną regułę. Ponieważ każda reguła odpowiada pojedynczemu „fragmentowi” wiedzy związanej z rozwiązywaniem problemów, treść reguły może zapewnić sensowne wyjaśnienie bieżącego stanu i działania systemu. Ponadto łańcuch reguł zastosowanych w procesie rozwiązania odzwierciedla zarówno ścieżkę na wykresie, jak i „linię rozumowania” eksperta. Natomiast pojedynczy wiersz kodu lub procedury w tradycyjny język aplikacji, taki jak Pascal, FORTRAN lub Java, jest praktycznie bez znaczenia.

Niezależność językowa. Model sterowania systemem produkcyjnym jest niezależny od reprezentacji wybranej dla reguł i pamięci roboczej, o ile reprezentacja ta obsługuje dopasowanie wzorca. Opisaliśmy reguły produkcji jako implikacje rachunku predykatu postaci $A \Rightarrow B$, gdzie prawda A i modus ponens reguły reguły wnioskowania pozwalają nam dojść do wniosku B. Chociaż istnieje wiele korzyści z używania logiki jako podstawy reprezentacji wiedzy i źródła reguł wnioskowania dźwiękowego, produkcja model systemu może być używany z innymi reprezentacjami, np. CLIPS i JESS.

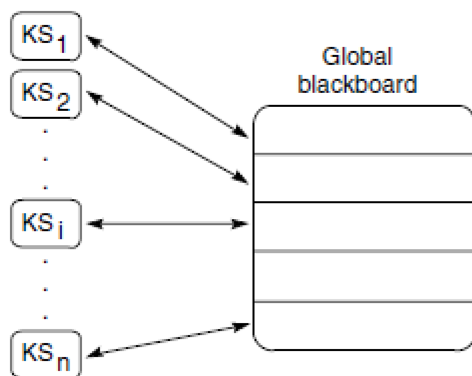
Chociaż rachunek predykatów oferuje przewagę logicznego wnioskowania, wiele problemów wymaga rozumowania, które nie jest logiczne. Zamiast tego dotyczą rozumowania probabilistycznego, użycia niepewnych dowodów i domyślnych założeń. Późniejsze rozdziały (7, 8 i 9) omawiają alternatywne reguły wnioskowania, które zapewniają te możliwości. Niezależnie od rodzaju zastosowanych reguł wnioskowania system produkcyjny zapewnia pojazd do przeszukiwania przestrzeni stanu

Możliwy model rozwiązywania problemów u ludzi. Modelowanie rozwiązywania problemów ludzkich było jednym z pierwszych zastosowań systemów produkcyjnych, patrz Newell i Simon (1972). Są one nadal stosowane jako model wydajności człowieka w wielu obszarach badań kognitywistyki (rozdział 16). Wyszukiwanie ukierunkowane na wzorzec daje nam możliwość zbadania przestrzeni wnioskowania logicznego w rachunku predykatów. Wiele problemów opiera się na tej technice, wykorzystując rachunek predykatów do modelowania określonych aspektów świata, takich jak czas i zmiana. W następnej sekcji systemy tablicowe są przedstawione jako odmiana metodologii systemu produkcyjnego, w której określone reguły zadań produkcji są łączone w źródła wiedzy i współpracują przy rozwiązywaniu problemów poprzez komunikację poprzez globalną pamięć roboczą lub tablicę.

Architektura tablicy do rozwiązywania problemów

Tablica jest ostatnim mechanizmem kontrolnym przedstawionym w tym rozdziale. Kiedy chcemy badać stany w przestrzeni wnioskowania w bardzo deterministyczny sposób, systemy produkcyjne zapewniają dużą elastyczność, umożliwiając nam reprezentowanie wielu częściowych rozwiązań jednocześnie w pamięci roboczej i wybieranie następnego stanu poprzez rozwiązywanie konfliktów. Tablice rozszerzają systemy produkcyjne, pozwalając nam organizować pamięć produkcyjną w osobne moduły, z których każdy odpowiada innemu podzbiorowi produkcji . Tablice integrują te oddzielne zestawy reguł produkcji i koordynują działania wielu agentów rozwiązywania problemów, zwanych czasami źródłami wiedzy, w ramach jednej globalnej struktury - tablicy. Wiele problemów wymaga koordynacji wielu różnych rodzajów agentów. Na przykład, program do rozumienia mowy może najpierw zmanipulować wypowiedź przedstawioną w postaci cyfrowej fali. W trakcie procesu dekodowania musi znaleźć słowa w tej wypowiedzi, uformować je w frazy i zdania, a na końcu stworzyć semantyczną reprezentację znaczenia wypowiedzi.

Powiązany problem występuje, gdy wiele procesów musi współpracować, aby rozwiązać jeden problem. Przykładem tego jest problem zespolenia czujnika. Załóżmy, że mamy sieć czujników, z których każdy jest monitorowany przez osobny proces. Załóżmy również, że procesy mogą się komunikować i że właściwa interpretacja danych każdego czujnika zależy od danych otrzymanych przez inne czujniki w sieci. Problem ten powstaje w sytuacjach tak różnorodnych, jak śledzenie samolotów w wielu lokalizacjach radarowych po łączenie odczytów wielu czujników w procesie produkcyjnym. Architektura tablicowa jest modelem kontroli zastosowanym do tych i innych problemów wymagających koordynacji wielu procesów lub źródeł wiedzy. Tablica jest centralną globalną bazą danych do komunikacji z niezależnymi źródłami wiedzy asynchronicznej, koncentrującymi się na powiązanych aspektach konkretnego problemu. Rysunek 6.13 przedstawia schemat projektu tablicy.



Na rysunku każde źródło wiedzy KS_i pobiera dane z tablicy, przetwarza dane i zwraca wyniki do tablicy, aby mogły zostać wykorzystane przez inne źródła wiedzy. Każdy KSi jest niezależny, ponieważ jest oddzielnym procesem działającym zgodnie z nim własne specyfikacje, a gdy stosowany jest system wieloprocesorowy lub wieloprocesorowy, jest on niezależny od innych procesów przetwarzania w problemie. Jest to system asynchroniczny, w którym każdy KSi rozpoczyna działanie, gdy tylko znajdzie odpowiednie dane wejściowe umieszczone na tablicy. Po zakończeniu przetwarzania publikuje wyniki na tablicy i oczekuje na nowe odpowiednie dane wejściowe. Podejście tablicowe do organizowania dużego programu zostało po raz pierwszy przedstawione w badaniu HEARSAY-II. HEARSAY-II był programem do rozumienia mowy; został pierwotnie zaprojektowany jako interfejs do bibliotecznej bazy danych artykułów informatycznych. Użytkownik biblioteki zwracałby się do komputera w mowie po angielsku, pytając: „Czy są to Feigenbaum i Feldman?" a komputer odpowiedziałby na pytanie informacjami z bazy danych biblioteki. Zrozumienie wymaga zintegrowania wielu różnych procesów, z których wszystkie wymagają bardzo różnej wiedzy i algorytmów, z których wszystkie mogą być wykładniczo złożone. Przetwarzanie sygnałów; rozpoznawanie fonemów, sylab i słów; parsowanie składniowe; oraz analiza semantyczna wzajemnie się ograniczają w interpretacji mowy. Architektura tablicy umożliwiła HEARSAY-II koordynację kilku różnych źródeł wiedzy wymaganych do tego złożonego zadania. Tablica jest zwykle zorganizowana w dwóch wymiarach. W HEARSAY-II tymi wymiarami były czas, w którym powstał akt mowy i poziom analizy wypowiedzi. Każdy poziom analizy przetwarzany był przez inną klasę źródeł wiedzy. Te poziomy analizy to:

KS_1 Kształt fali sygnału akustycznego.

KS_2 Fonemy lub możliwe segmenty dźwiękowe sygnału akustycznego.

KS_3 Sylaby, które fonemy mogłyby wytworzyć.

KS_4 Możliwe słowa analizowane przez jeden KS.

KS_5 Możliwe słowa analizowane przez drugi KS (zwykle biorąc pod uwagę słowa z różnych części danych).

KS_6 KS, aby spróbować wygenerować możliwe sekwencje słów.

KS_7 KS, który umieszcza sekwencje słów w możliwych frazach.

Możemy wizualizować te procesy jako elementy na rysunku 6.13. Podczas przetwarzania mowy mówionej fala sygnału mówionego jest wprowadzana na najniższym poziomie. Źródła wiedzy na temat przetwarzania tego wpisu są włączone i zamieszczają swoje interpretacje na tablicy, aby je zebrać w odpowiednim procesie. Ze względu na dwuznaczności języka mówionego na każdym poziomie tablicy może występować wiele konkurencyjnych hipotez. Źródła wiedzy na wyższych poziomach próbują

ujednoznacznić te konkurencyjne hipotezy. Analiza HEARSAY-II nie powinna być postrzegana jako po prostu jeden niższy poziom generujący dane, które wyższe poziomy mogą następnie analizować. To jest o wiele bardziej skomplikowane. Jeśli KS na jednym poziomie nie może przetworzyć (zrozumieć) przesłanych do niego danych, KS może poprosić KS, który wysłał mu dane, o powrót do kolejnej próby lub postawienie kolejnej hipotezy na temat danych. Co więcej, różne KS mogą jednocześnie pracować nad różnymi częściami wypowiedzi. Wszystkie procesy, jak wspomniano wcześniej, są asynchroniczne i sterowane danymi; działają, gdy mają dane wejściowe, kontynuują działanie do momentu zakończenia zadania, a następnie publikują wyniki i czekają na następne zadanie.

Jeden z KS, zwany harmonogramem, obsługuje komunikację „zużycie danych po wyniku" między KS. Ten harmonogram ma oceny wyników każdej KS i jest w stanie dostarczyć, za pomocą kolejki priorytetowej, pewien kierunek w rozwiązywaniu problemów. Jeśli żaden KS nie jest aktywny, program planujący stwierdza, że zadanie zostało zakończone i wyłącza się. Gdy program HEARSAY miał bazę około 1000 słów, działał całkiem dobrze, choć nieco wolno. Gdy baza danych została dodatkowo rozszerzona, dane dla źródeł wiedzy stały się bardziej złożone, niż mogły sobie z tym poradzić. HEARSAY-III jest uogólnieniem podejścia przyjętego przez HEARSAY-II. Wymiar czasowy HEARSAY-II nie jest już potrzebny, ale zachowano wiele KS dla poziomów analizy. Tablica HEARSAY-III ma na celu interakcję z ogólnym systemem relacyjnych baz danych. Rzeczywiście, HEARSAY-III to powłoka do projektowania systemów eksperckich. Ważną zmianą w HEARSAY-III było rozdzielenie programu planującego KS (jak opisano powyżej dla HEARSAY-II) i uczynienie go oddzielnym kontrolerem tablicy dla pierwszej (lub domeny problemowej) tablicy. Ta druga tablica pozwala rozbić proces planowania, podobnie jak dziedzina problemu, na osobne KS zajmujące się różnymi aspektami procedury rozwiązania (na przykład, kiedy i jak zastosować wiedzę w tej dziedzinie). Druga tablica może zatem porównać i zrównoważyć różne rozwiązania dla każdego problemu (Nii i Aiello 1979, Nii 1986a, 1986b). Alternatywny model tablicy zachowuje ważne części bazy wiedzy w tablicy, zamiast rozprowadzać je między źródłami wiedzy

Epilog

Omówiliśmy wdrażanie strategii wyszukiwania . Przedstawiono rekurencję jako ważne narzędzie do programowania wyszukiwania wykresów, implementujące algorytmy głębokości pierwszej i wstecznej w formie rekurencyjnej. Wyszukiwanie ukierunkowane na wzorce z regułami unifikacji i wnioskowania, , upraszcza implementację wyszukiwania w przestrzeni logicznych wniosków. Pokazano system produkcyjny jako naturalną architekturę do modelowania rozwiązywania problemów i implementacji algorytmów wyszukiwania. Sekcja zakończyła się przykładami implementacji systemu produkcyjnego wyszukiwania opartego na danych i celu. W rzeczywistości system produkcyjny zawsze był ważnym paradygmatem programowania AI, poczynając od pracy Newella i Simona oraz ich kolegów z Carnegie Mellon University. System produkcyjny był także ważną architekturą wspierającą badania w dziedzinie kognitywistyki. Odniesienia do budowania systemów produkcyjnych, zwłaszcza języków OPS, obejmują programowanie systemów ekspertowych w OPS5 autorstwa Lee Brownstona i in. oraz Pattern Directed Inference Systems autorstwa Donalda Watermana i Fredericka Hayes-Rotha. Aby zapoznać się z nowoczesnymi systemami produkcyjnymi w języku C i Java, przejdź do stron internetowych dla CLIPS i JESS. Wczesna praca w modelach tablicowych została opisana w badaniach HEARSAY-II. Późniejsze zmiany w tablicach opisano w pracy HEARSAY-III oraz Blackboard Systems, pod redakcją Roberta Engelmore'a i Tony'ego Morgana. Badania systemów produkcji, planowania i architektury tablic pozostają aktywną częścią sztucznej inteligencji. Zalecamy, aby zainteresowany czytelnik zapoznał się z najnowszymi obradami Konferencji Amerykańskiego Stowarzyszenia dla Sztucznej Inteligencji i Międzynarodowej Wspólnej Konferencji na temat Sztucznej Inteligencji. Morgan Kaufmann i AAAI Press publikują materiały z konferencji, a także zbiory lektur na tematy AI