Politechnika Warszawska

MSI

8. Planowanie działań

Włodzimierz Kasprzak

Układ

1. Agent realizujący cel – przeszukiwanie, planowanie
2. Klasyczne planowanie
3. Plany częściowe
4. Przykład: tworzenie planu częściowo uporządkowanego
5. Graf wspierający planowanie
6. Strategia „Graphplan”

Pytania

1. Agent realizujący cel

Agent realizujący cel reprezentuje problem w postaci:

- celu, przestrzeni stanów i wykonywanych akcji.

Dwa główne rodzaje agentów realizujących cel:

1) w wyniku przeszukiwania przestrzeni stanów znajdujemy sekwencję akcji wiodącą od stanu początkowego do stanu końcowego, w którym spełniony jest cel;
2) poprzez konstruowanie **planu** dla zadanego **celu** (czyli **sekwencji akcji**) w przestrzeni stanów i jego późniejsze wykonywanie, co prowadzi agenta do zadanego celu.

Proces tworzenia planu można realizować jako przeszukiwanie przestrzeni (częściowych) planów.

Planowanie

Planowanie ma na celu „usprawnienie” przeszukiwania: „**otwarcie**” reprezentacji stanów, celów i akcji.

Algorytmy planujące korzystają z formalnych języków, zwykle L1R lub podzbiorów języka L1R, do **opisu stanów, celów i akcji**.

Stany i **cel** są reprezentowane poprzez zbiory formuł, które są spełnialne.

1) Akcje są charakteryzowane poprzez logiczne opisy warunków wykonania i efektów akcji. Pozwala to na określenie bezpośrednich związków pomiędzy stanami a akcjami i na pominięcie bezzasadnych akcji w danym stanie.
2) Akcje są dodawane do planu i wykonywane w częściowo dowolnej kolejności wtedy, gdy są potrzebne, a nie stanowią sztywnej sekwencji.
3) Problem jest często formułowany jako koniunkcja pod-problemów. Możemy wtedy wyróżnić podcele i wyznaczać osobne plany dla osiągnięcia kolejnych podcelów.

Prosty agent planujący

funkcja PROSTY-AGENT-PLANUJĄCY(*percepcja*); **wynik**: *akcja*;
{ **Dane statyczne (trwałe):** *KB* – baza wiedzy (zawiera opisy akcji);
p – plan, początkowo = *BrakPlanu*;
t – licznik czasu, początkowo = 0;
Lokalne zmienne: *cel* – aktualny cel; *stan* – aktualny stan;
TELL(KB, UTWÓRZ-FORMUŁĘ-PERCEPCJI(*percepcja*, *t*));
stan := STAN(*KB*, *t*);
if ((*p* = = *BrakPlanu*) **then** {
cel := ASK(*KB*, UTWÓRZ-FORMUŁĘ-ZAPYTANIA(*t*));
p := IDEALNY-PLANER(*stan*, *cel*, *KB*); }
if ((*p* = = *BrakPlanu*) **or** (*p* = = ∅)) **then** *akcja* := *BrakAkcji*;
else { *akcja* := PIERWSZY(*p*); p := RESZTA(p); }
TELL(*KB*, UTWÓRZ-FORMUŁĘ-AKCJI(*akcja*, *t*);
t := *t*+1;
return *akcja*;
}

Przestrzeń stanów problemu a przestrzeń planów

Reprezentacja **stanu problemu** – jest to struktura danych potrzebnych dla *generacji następnika, funkcji oceny* i *warunku stopu.*

- Stan ma zwykle charakter **„czarnej skrzynki”.**
- Reprezentacja **akcji** – procedury, które generują następne stany problemu.
- Reprezentacja celu – zwykle pośrednia informacja o celu, wyrażona poprzez warunek stopu i funkcję oceny kosztów resztkowych (heurystykę); taka reprezentacja nie daje agentowi żadnych wskazówek o sensie akcji w aktualnym stanie.

Reprezentacja **planu** – jest to sekwencja operatorów w przestrzeni planów, która (a) **rozszerza** plan początkowy (prosty, niepełny plan) do końcowego; lub (b) **modyfikuje pełny**, ale błędny plan.

- W **przestrzeni planów** mamy znacznie mniejszą wariantowość i dowolność kolejności operatorów dla planu niż akcji dla stanu problemu.

Przykład: przeszukiwanie stanów zawiedzie

Prosty problem (cel): „*kup karton mleka i kiść bananów i bezprzewodową wiertarkę oraz wróć do domu*”.

Poszukiwanie w przestrzeni stanów: generacja wszystkich możliwych stanów w celu znalezienia stanu końcowego.

Heurystyka nie pozwala eliminować zbędnych akcji a jedynie wybierać stany.

Agent powinien rozpatrzyć pewne sekwencje akcji zanim podejmie decyzję.

Bardzo wysoki współczynnik rozgałęzienia drzewa

Planowanie w języku logiki

Planowanie wyrażone w rachunku sytuacyjnym (podzbiór L1R).

- Stan początkowy reprezentowany jest jako koniunkcja predykatów zwykle działających na stałych. Np.

$$Być(\text{Dom}) \wedge \neg Mieć(\text{Mleko}) \wedge \neg Mieć(\text{Wiertarka}) \ldots$$

- Cele (stany końcowe) reprezentowane są jako koniunkcje predykatów i mogą one zawierać zmienne. Np.

$$\exists x\ At(x) \wedge Sprzedaje(x, \text{Mleko})$$

Zmienne w formule celu są kwantyfikowane egzystencjalnie – jest to formuła zapytania.

- Operatory są formułami o postaci *aksjomatu następnika stanu* i są powiązane z akcjami (np. predykatem *Rezultat*). Np. $\forall a,s\ Mieć(\text{Mleko}, Rezultat(a,s)) \Leftrightarrow$

$$[(a=Kupić(\text{Mleko}) \wedge At(\text{Supermarkiet}, s) \vee$$
$$(Mieć(\text{Mleko}, \text{s}) \wedge \text{a} \neq Stracić(\text{Mleko}))]$$

Wnioskowanie logiczne a planowanie

- W praktyce procedura wnioskowania logicznego nie jest stosowana do tworzenia planu działania agenta. Zajmuje się tym odpowiednia procedura **planująca**, która jest bardziej efektywna od ogólnej procedury wnioskowania.
- Istnieje różnica pomiędzy *formułą celu* podaną do procedury planującej a *zapytaniem* kierowanym do procedury wnioskującej.

  Pierwsza pyta o sekwencję akcji, która spowoduje, że formuła celu będzie spełniona.

  Druga sprawdza, czy formuła zapytania jest spełniona, jeśli prawdziwe są formuły w bazie wiedzy.

2. Klasyczne planowanie

- „Klasyczna” budowa planu odwołuje się do języka STRIPS (*Stanford Research Institute Problem Solver*), będącego ograniczoną formą rachunku sytuacyjnego.
- Stany są reprezentowane jako koniunkcje literałów - predykatów zawierających jedynie stałe.
- Cele są również koniunkcjami predykatów, ale obok stałych mogą też zawierać zmienne, dla których zakłada się kwantyfikację egzystencjalną.
- Operator składa się z 3 części:
  1. Warunek – koniunkcja **pozytywnych** predykatów atomowych warunkująca wykonalność operatora.
  2. Akcja – jej wykonanie przez agenta przeprowadzi środowisko do nowej sytuacji.
  3. Następnik – efekt operatora opisany za pomocą koniunkcji predykatów atomowych (pozytywnych lub zanegowanych).

Reprezentacja operatora w STRIPS

Operator przekształca sytuację opisaną w jej warunku w sytuację opisaną w jej następniku i podaje związaną akcję.

Pełny opis stanu nie jest konieczny – dla poznania stanu w aktualnej sytuacji wystarczy śledzić zmiany zachodzące od sytuacji początkowej.

Brak jest jawnych indeksów sytuacji.

Być(tu), Droga(tu, tam)
Pójść(tam)
Być(tam), ¬Być(tu)

- Warunek operatora (PRECOND)
- Akcja (ACTION)
- Następnik (EFFECT)

$Op($ ACTION: $Pójść(tam)$,
PRECOND: $Być(tu) \wedge Droga(tu, tam)$,
EFFECT: $Być(tam) \wedge \neg Być(tu))$

Operatory w STRIPS

- Operator jest stosowalny w stanie *S*, jeżeli warunek operatora jest spełniony w tym stanie.
- Operator zawierający symbole zmiennych reprezentuje rodzinę akcji. Wykonany może być tylko w pełni konkretny operator (po podstawieniu wartości pod wszystkie zmienne). Procedura planująca musi zapewnić dostępność wartości dla zmiennych w danym stanie.
- W stanie powstałym dzięki zastosowaniu operatora spełnione są:
  - wszystkie predykaty zawarte w następniku operatora,
  - wszystkie predykaty zawarte w warunku operatora, które nie zostały zanegowane przez następnik operatora.
- **„Goal" (cel)** jest reprezentowany za pomocą operatora bez „następnika", a **„Init" (start)** jest operatorem bez „warunku".
- Operator **„Goal"** nie posiada zmiennych.

Przykład planu w STRIPS

Problem transportu przesyłek lotniczych pomiędzy dwoma lotniskami może być następująco reprezentowany w STRIPS:

- Init(W(C1,WAW) $\wedge$ W(C2, FRA) $\wedge$ W(P1, WAW) $\wedge$ W(P2, FRA) $\wedge$
  Cargo(C1) $\wedge$ Cargo(C2) $\wedge$ Samolot(P1) $\wedge$ Samolot(P2) $\wedge$
  Lotnisko(WAW) $\wedge$ Lotnisko(FRA))
- Goal(W(C1,FRA) W(C2, WAW))
- Action(Załadunek(c, p, a),
  Precond: W(c,a) $\wedge$ W(p,a) $\wedge$ Cargo(c) $\wedge$ Samolot(p) $\wedge$ Lotnisko(a)
  Effect: $\neg$W(c,a) $\wedge$ W(c,p))
- Action(Rozładunek(c, p, a),
  Precond: W(c,p) $\wedge$ W(p,a) $\wedge$ Cargo(c) $\wedge$ Samolot(p) $\wedge$ Lotnisko(a)
  Effect: W(c,a) $\wedge$ $\neg$W(c,p))
- Action(Lot(p, od, do),
  Precond: W(p,od) $\wedge$ Samolot(p) $\wedge$ Lotnisko(od) $\wedge$ Lotnisko(do)
  Effect: $\neg$W(p,od) $\wedge$ W(p,do))

Przykład w STRIPS (2)

Rozwiązaniem podanego problemu może być następujący plan:

[Załadunek(C1; P1; WAW);

Lot(P1; WAW; FRA);

Załadunek(C2; P2; FRA);

Lot(P2; FRA; WAW)] .

Uwaga:

Wadą reprezentacji powyższego problemu w STRIPS jest to, że generowany może być plan, który zezwala na przelot samolotu z lotniska do tego samego lotniska docelowego.

ADL

- Po pojawieniu się STRIPS w kolejnych latach okazało się, że jego siła wyrazu nie wystarczy dla opisu wielu praktycznych dziedzin. Upowszechniła się odmiana tego języka nazwana „Action Description Language” (**ADL).**
- Przykład. W języku ADL akcję *Lot* reprezentujemy następująco:

Action(Lot(p:Samolot, od:Lotnisko, do:Lotnisko),

Precond: $W(p,od) \wedge (od \neq do)$

Effect: $\neg W(p,od) \wedge W(p,do)$)

W warunku operatora występuje dodatkowo wyrażenie $(od \neq do)$, które wyklucza planowanie wykonania lotu z dowolnego lotniska na to samo lotnisko. Tego nie można było wyrazić w STRIPS.

Porównanie STRIPS i ADL

STRIPS	ADL
Tylko pozytywne literały w warunkach, np. $W(\text{dom}) \wedge Ma(\text{mleko})$	Pozytywne i zanegowane literały w warunkach: np. $\neg W(\text{dom}) \wedge \neg Ma(\text{mleko})$
Założenie „zamkniętego świata”: brakujące literały są niespełnione (False).	Założenie „otwartego świata”: spełnianie brakujących literałów nie jest znane.
$(P \wedge \neg Q)$ oznacza: *dołącz P i usuń Q*	$(P \wedge \neg Q)$ oznacza: *dołącz P i $\neg Q$, oraz usuń $\neg P$ i Q*
Tylko bazowe literały (predykatowe) w operatorze „cel”. Np. $W(\text{P1},\text{WAW}) \wedge W(\text{P2},\text{WAW})$	Kwantyfikowane zmienne w operatorze „cel”. Np. $\exists x\ At(\text{P1},x) \wedge At(\text{P2},x)$
Cele są koniunkcyjnej postaci. Np. $W(\text{dom}) \wedge Ma(\text{mleko})$	Cele są koniunkcyjne i alternatywne. Np. $W(\text{dom}) \wedge (Ma(\text{mleko}) \vee Ma(\text{banany}))$
Brak wbudowanej semantyki dla równości (=). Brak typów.	Wbudowany predykat równości (np. x=y). Zmienne są określonych typów. Np. $(x: \text{Miasto})$

Logiki deskrypcyjne

„Description logic” to rodzina języków reprezentacji wiedzy - rozszerzenie logiki predykatów – stanowiąca formalną podstawę dla systemów ontologicznych (np. KL-ONE, DAML) i „Semantic Web” (OWL, OWL2).

Nie zakłada się **unikalności nazw** ani „**zamkniętego świata**”:

- Dwa pojęcia o różnych nazwach mogą być równoważne.
- Brak wiedzy o zachodzeniu faktu nie oznacza zachodzenie negacji faktu.

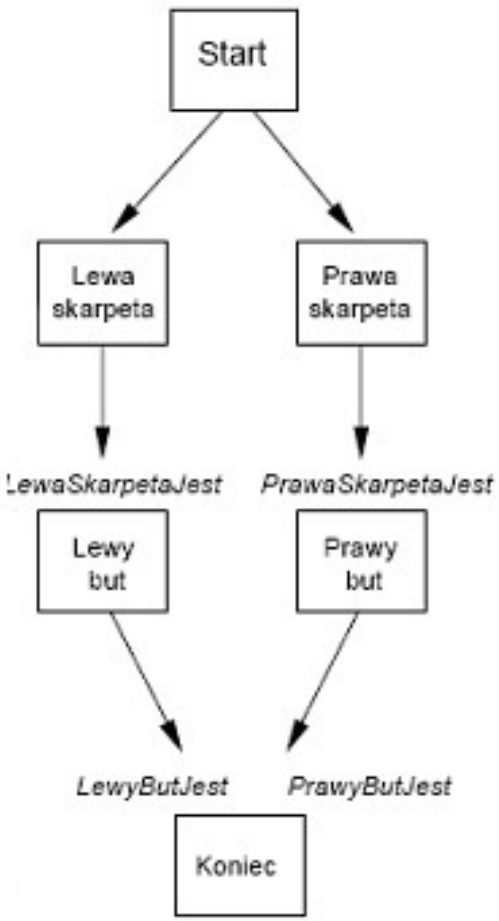
Logika predykatów	**DL**
Unarny predykat – klasa	Concept (pojęcie, „koncept”)
Predykat	Rola
Obiekt	Individual (instancja)
Wyrażenie (formuła)	Aksjomat
Formuła ze zmiennymi (zależności pomiędzy pojęciami)	TBox (terminological box)
Formuła bez zmiennych (zależności pomiędzy obiektami a klasami)	Abox (assertional box)

3. Przestrzeń planów częściowych

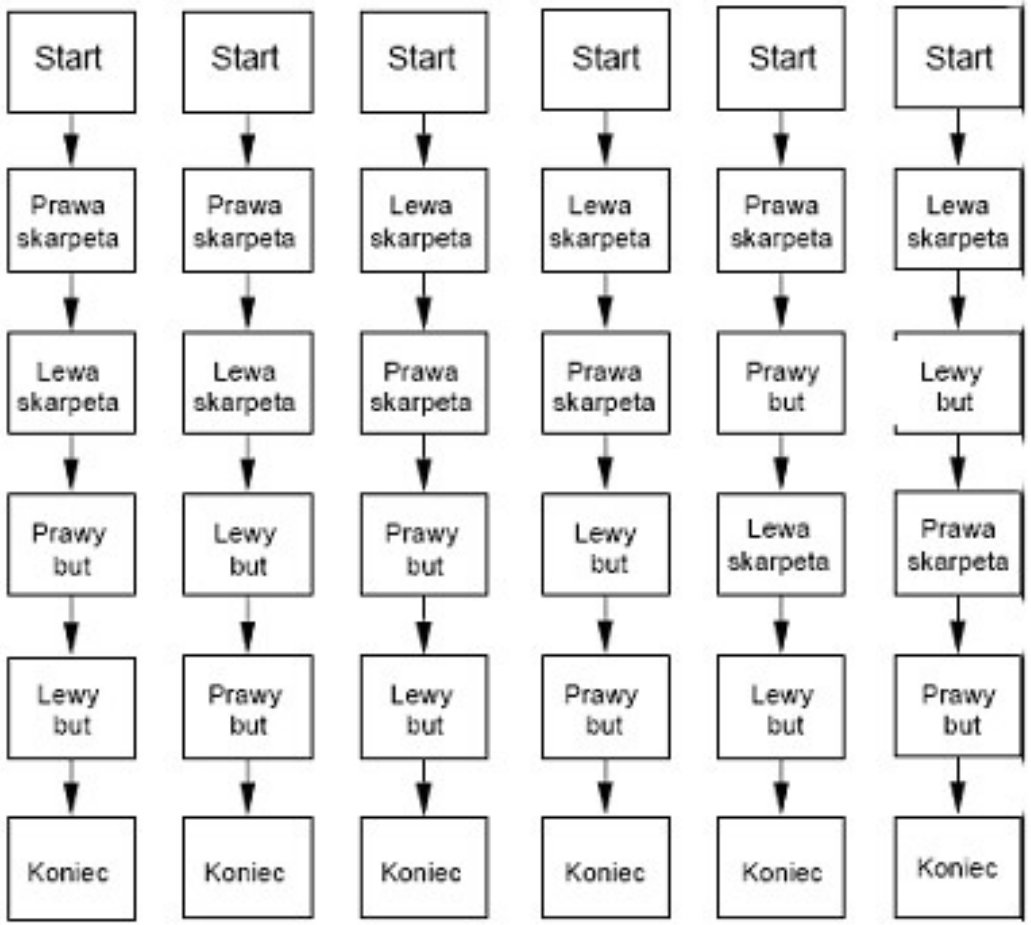
- Zamiast prowadzić poszukiwania w przestrzeni stanów, poszukiwania będą prowadzone w **przestrzeni planów.**
- Rozpoczynamy z planem początkowym (bardzo zgrubnym) i generujemy kolejne (coraz dokładniejsze) plany **częściowe**, dopóki nie znajdziemy pełnego planu, stanowiącego rozwiązanie naszego problemu.
- Kolejność kroków planu (zastosowanych operatorów) musi być taka, aby były spełnione warunki wstępne (poprzedniki) dla każdej z nich.
- Plan częściowo uporządkowany: niektóre kroki są uporządkowane względem siebie, a niektóre - nie.
- Plan w pełni uporządkowany: stanowi sekwencję kroków.
- Linearyzacja planu *P*: plan w pełni uporządkowany uzyskany z *P* po dodaniu w nim ograniczeń porządkujących.

Przykład: plan częściowo uporządkowany i w pełni uporządkowany

Częściowo uporządkowany



W pełni uporządkowany



Reprezentacja planu

Formalna reprezentacja planu:

- Zbiór kroków S, czyli zbiór zastosowanych operatorów.
- Zbiór ograniczeń porządkujących kroki planu, PORZĄDEK(*plan*), zapisanych w postaci $S_i < S_j$ (tzn. S_i poprzedza S_j, ale niekoniecznie bezpośrednio).
- Zbiór ograniczeń narzuconych na zmienne, o postaci $v=x$, gdzie v jest zmienną, a x - stałą lub inną zmienną, PRZYPISANIA(*plan*).
- Zbiór związków przyczynowo-skutkowych, ZWIĄZKI(*plan*), o postaci $S_i \text{->}^c S_j$; krok S_i tworzy warunek c dla S_j ; taki związek określa powód stosowania kroku S_i .

Własności planów częściowo uporządkowanych

- Może posiadać wiele linearyzacji (każda jest równie dobra, gdyż interesuje nas dowolna sekwencja akcji prowadząca do celu).
- Może być wykonany w trybie równoległym.
- Może zostać wykorzystany przez hierarchiczny program planujący jako pod-plan.

Kompletny plan

- Kompletny plan to plan, w którym każdy warunek każdego jego kroku jest spełniony:

  Dla każdego warunku (c, S_j) zachodzi:

  – *Istnieje krok* $S_i < S_j$

  taki, że $c \in \text{EFEKTY}(S_i)$

  i $\neg \exists S_k : \neg c \in \text{EFEKTY}(S_k)$, gdzie $(S_i < S_k < S_j)$

- Czyli dla każdego dowolnego warunku dowolnego kroku w planie istnieje inny krok powodujący spełnienie tego warunku i nie istnieje jakikolwiek krok pomiędzy tymi dwoma krokami, który kasowałby ten warunek.

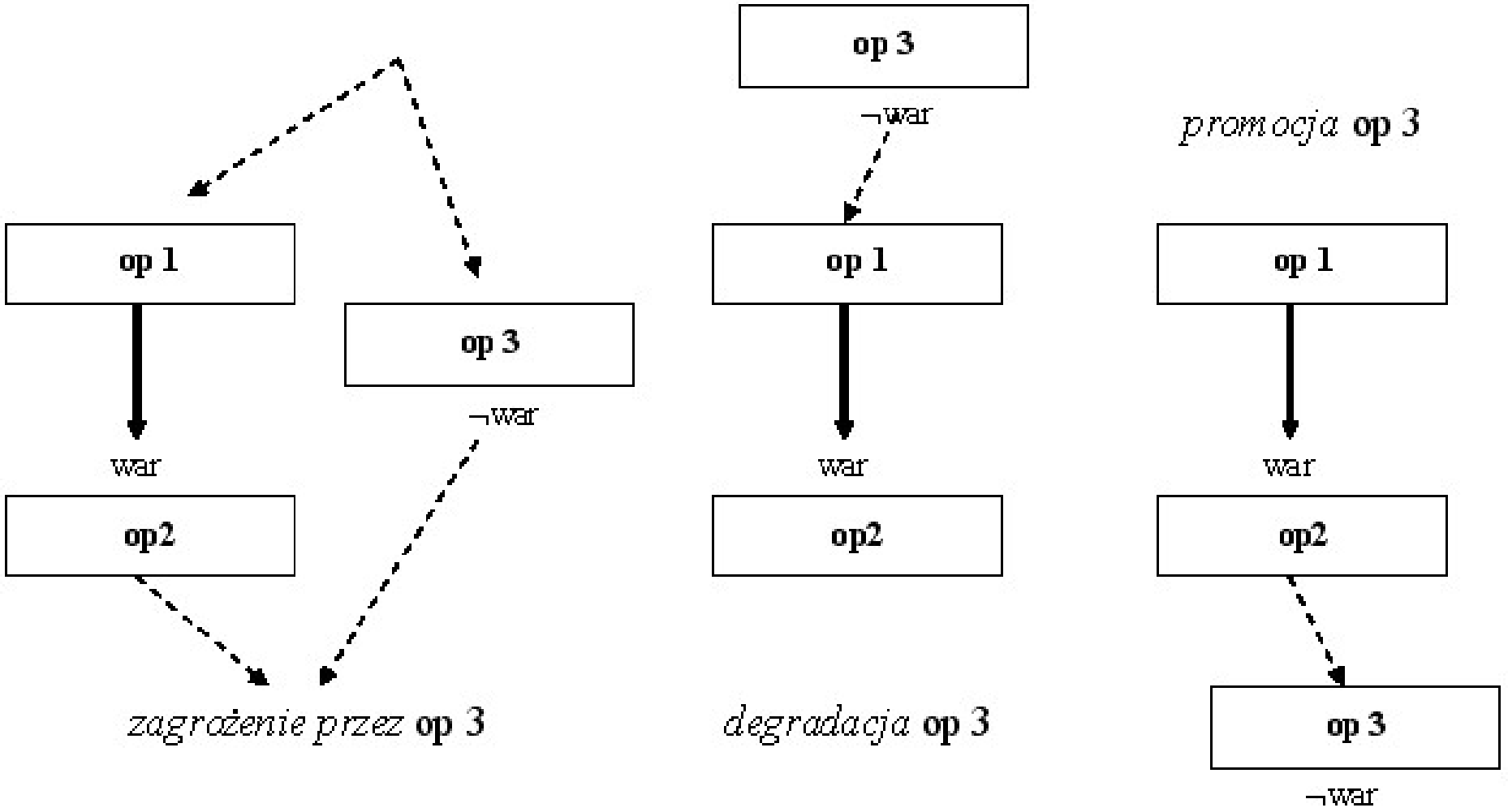
Tworzenie planu częściowo uporządkowanego

1. Tworzymy plan początkowy zawierający jedynie akcje Init (start) i Goal (cel)
2. W każdym kolejnym kroku albo dodajemy nową operację albo wykorzystujemy już obecną (poprzez dodanie połączenia przyczynowo-skutkowego), aby zapewnić spełnienie jednego z warunków wstępnych którejś z operacji już utworzonego planu.
3. Powtarzamy krok 2, aż uzyskamy plan końcowy (→ koniec) albo operację niewykonalną → wtedy :

3.a. próbujemy zmienić kolejność operacji (degradacja, promocja), lub

3.b. wykonujemy nawrót (w ramach decyzji podjętych w kroku 2) i zmieniamy warunki wykonania operacji.

Rozwiązywanie zagrożeń



Funkcja PCzU (tworzenie planu częściowo uporządkowanego)

```
function AGENT-PCzU(Init, Goal, operatory) : returns plan
{
    plan := UTWÓRZ-PLAN-MINIMALNY(Init, Goal);
    while(true)
    {
        if (KOMPLETNY(plan) == true) then return plan;
        [krok_wym, c] := WYBIERZ-PODCEL(plan);
        WYBIERZ-OPERATOR(plan, operatory, krok_wym, c);
        krok_zagrożenie := ROZWIĄŻ-ZAGROŻENIA(plan);
        if (krok_zagrożenie ≠ ∅)  NAWRÓT-NAPRAW(krok_zagrożenie);
    }
}
```

Podfunkcje PCzU (2)

function WYBIERZ-PODCEL(*plan*) : **returns** [$krok_{wym}$, *c*]
{ wybierz krok planu ($krok_{wym}$) ze zbioru KROKI(*plan*) o warunku wstępnym *c*, który nie został jeszcze spełniony;
return [$krok_{wym}$, *c*];
}
procedure WYBIERZ-OPERATOR(*plan*, *operatory*, $krok_{wym}$, *c*)
{ wybierz $krok_{dod}$ ze zbioru *operatory* lub KROKI(*plan*), taki, że jego efektem jest *c*;
if (brak $krok_{dod}$) **then return**;
Dodaj związek przyczynowy ($krok_{dod} \rightarrow^{c} krok_{wym}$) do ZWIĄZKI(*plan*);
Dodaj porządek ($krok_{dod} < krok_{wym}$) do PORZĄDEK(*plan*);
if ($krok_{dod}$ jest nowo dodanym krokiem ze zbiory *operatory*) **then**
{ dodaj $krok_{dod}$ do KROKI(*plan*);
dodaj (*Start* < $krok_{dod}$ < *Finish*) do PORZĄDEK(*plan*);
}
}

Podfunkcje PCzU (3)

function ROZWIĄŻ-ZAGROŻENIA(*plan*) : **returns** $krok_{\text{zagrożenie}}$
{
  for (każdy $krok_{\text{zadrożenie}}$, który zagraża związkowi ($krok_A \rightarrow^c krok_B$) ze zbioru ZWIĄZKI(*plan*) **do**
  {
    Wybierz {
      Degradacja: dodaj $krok_{\text{zagrożenie}} < krok_A$ do PORZĄDEK(*plan*);
        lub
      Promocja: dodaj $\text{krok}_B < krok_{\text{zagrożenie}}$ do PORZĄDEK(*plan*);
    }
    if (ZGODNY(*plan*) == **false**) **then return** $krok_{\text{zagrożenie}}$;
  }
  return ∅ ;
}

4. Przykład: plan częściowo-uporządkowany

Powróćmy do naszego problemu, którego rozwiązaniem (celem działania) jest posiadanie przez agenta: mleka, bananów i wiertarki oraz przebywanie w domu.

Zakładamy, że akcja **„Pójść”** pozwala przemieścić agenta pomiędzy dowolnymi dwoma miejscami.

Akcja **„Kupuje”** – zakup produktu, bez precyzowania ceny czy ilości.

OBI oznacza market budowlany, **SM** – oznacza „*supermarket*”.

Init – akcja inicjalizacji – daje stan początkowy, w którym agent jest w domu a produkty są w sklepach.

Goal – akcja kończąca (zatrzymanie) – daje stan docelowy, w którym agent posiada produkty i jest w domu.

Przykład: kroki planu (operatory)

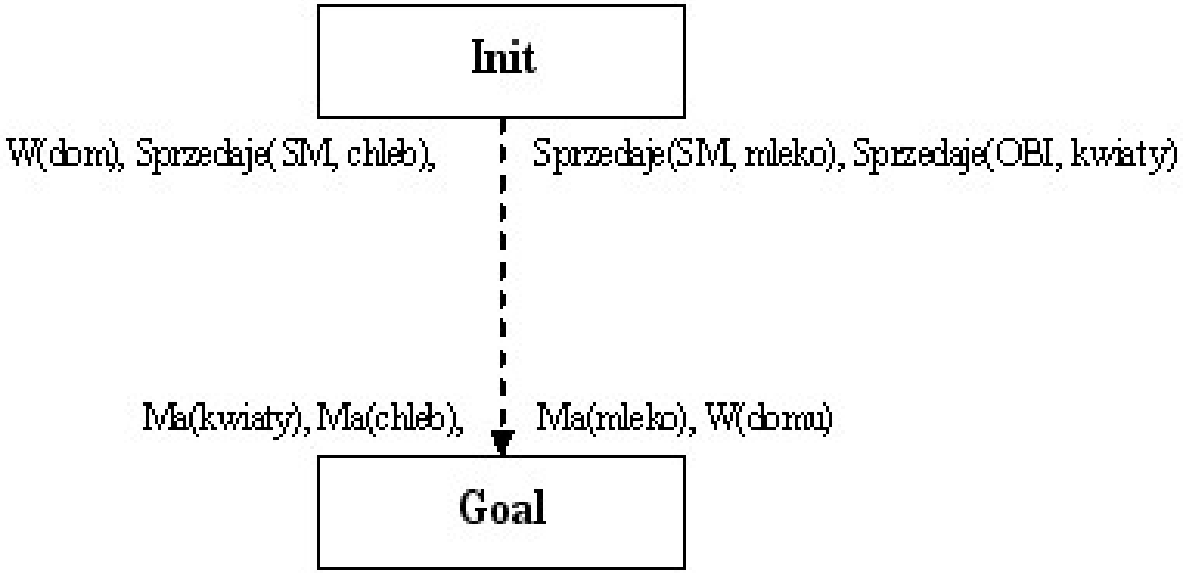
Op(ACTION: Init,
EFFECT: W(dom) $\wedge$ Sprzedaje(OBI, kwiaty) $\wedge$
Sprzedaje(SM, mleko) $\wedge$ Sprzedaje(SM, chleb)).

Op(ACTION: Goal,
PRECOND: Ma(kwiaty) $\wedge$ Ma(mleko) $\wedge$ Ma(chleb) $\wedge$
W(dom))

Op(ACTION: Pójść(tam),
PRECOND: W(tu),
EFFECT: W(tam) $\wedge \neg$ W(tu)).

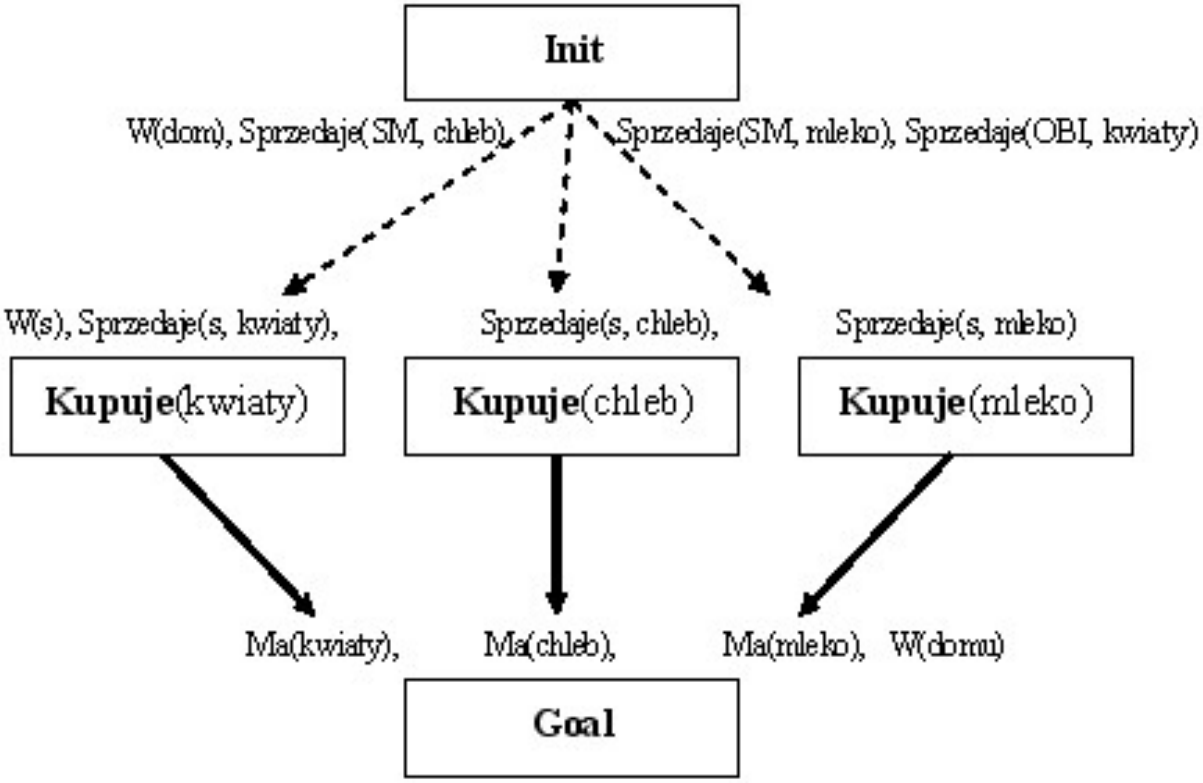
Op(ACTION: Kupuje(x),
PRECOND: W(sklep) $\wedge$ Sprzedaje(sklep,x),
EFFECT: Ma(x)).

Przykład: plan początkowy



PORZĄDEK(*plan_początkowy*) = {Init < Goal }

Przykład: rozbudowa planu (1)

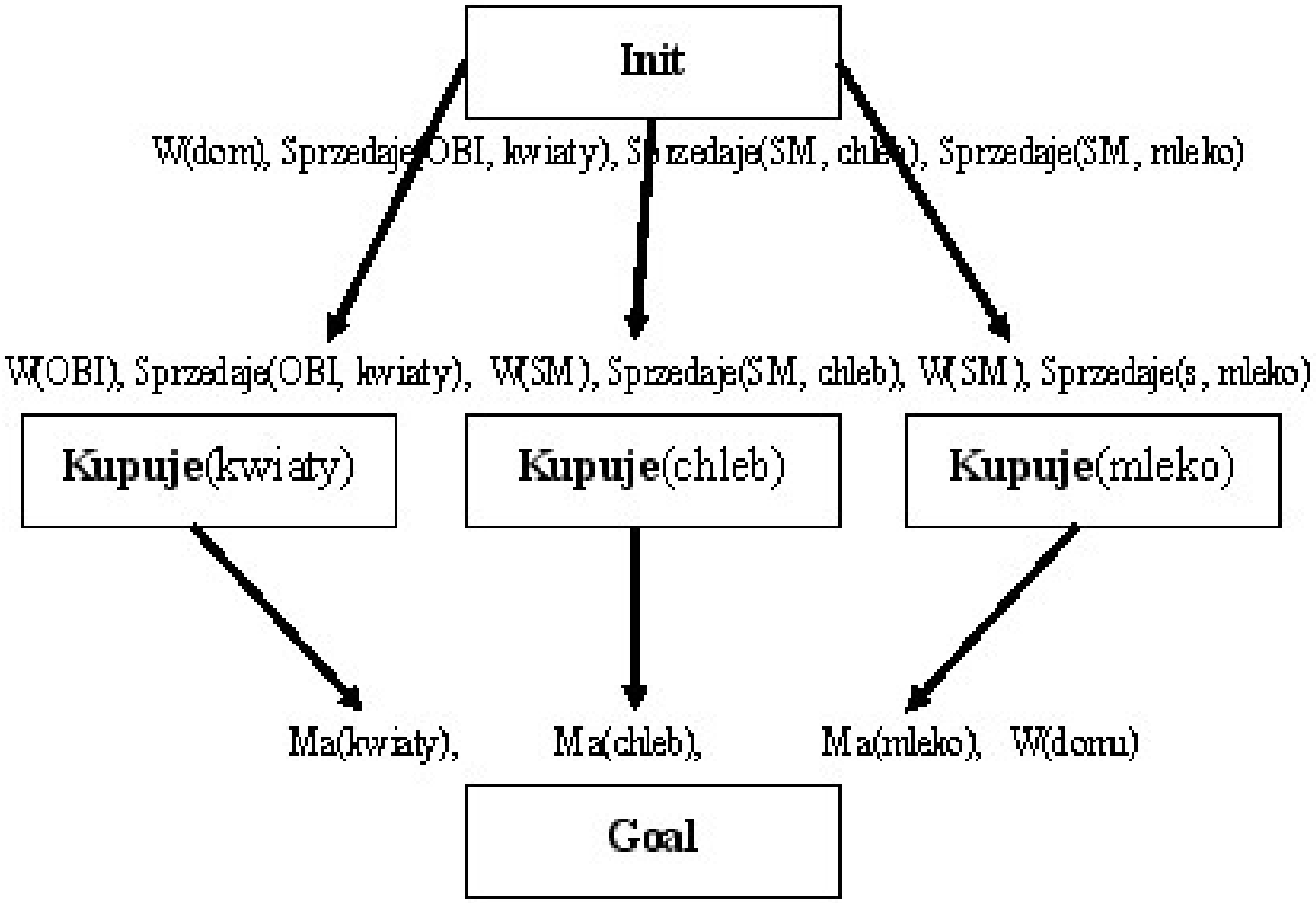


Plan po dodaniu trzech operacji Kupuje.

Pogrubione strzałki – związki przyczynowo-skutkowe

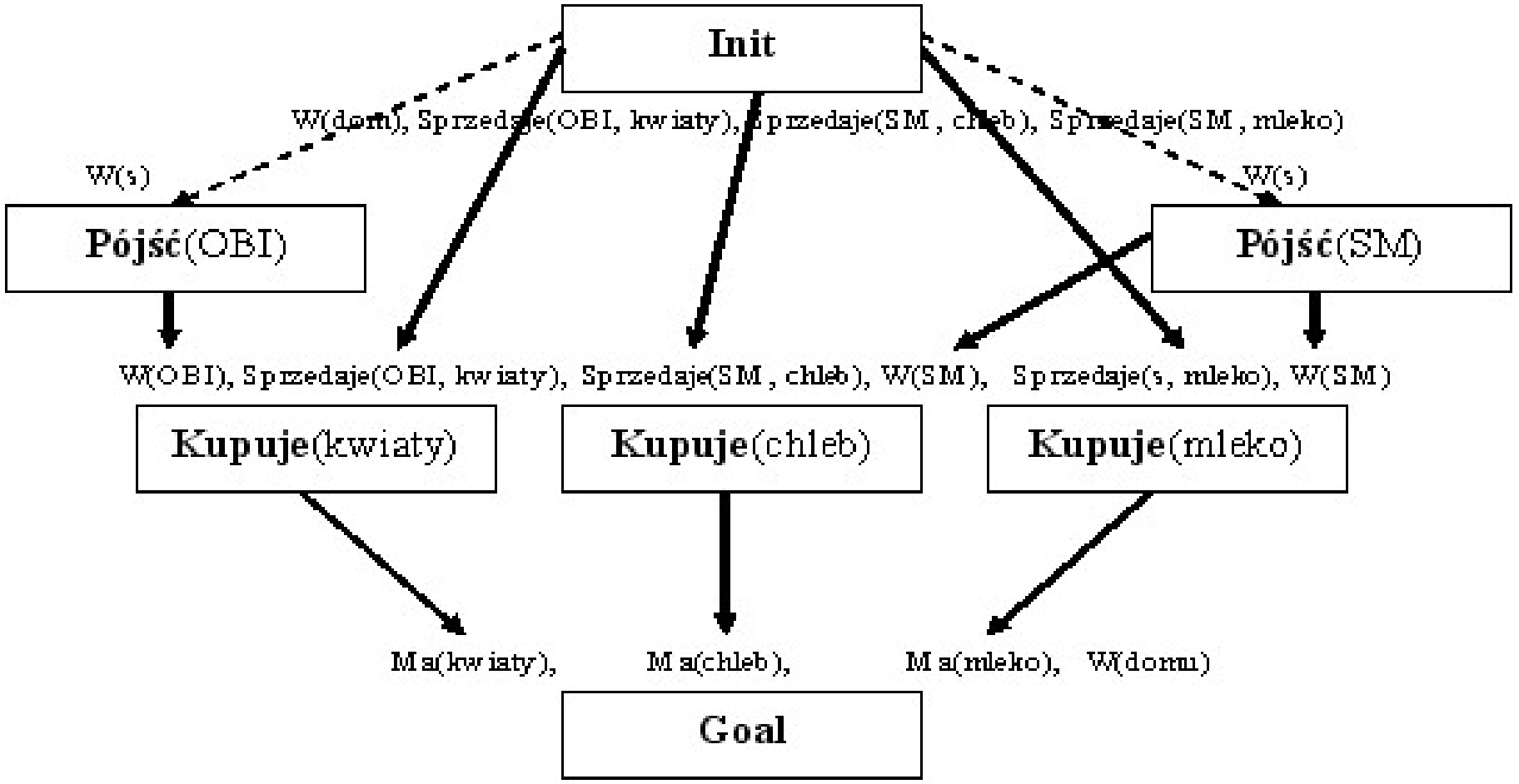
Cienkie strzałki – wskazują „porządek” czyli kolejność wykonywania operacji

Przykład: rozbudowa planu (2)



Plan po dodaniu związków przyczynowo-skutkowych dla kroków „Init – Kupuje”.

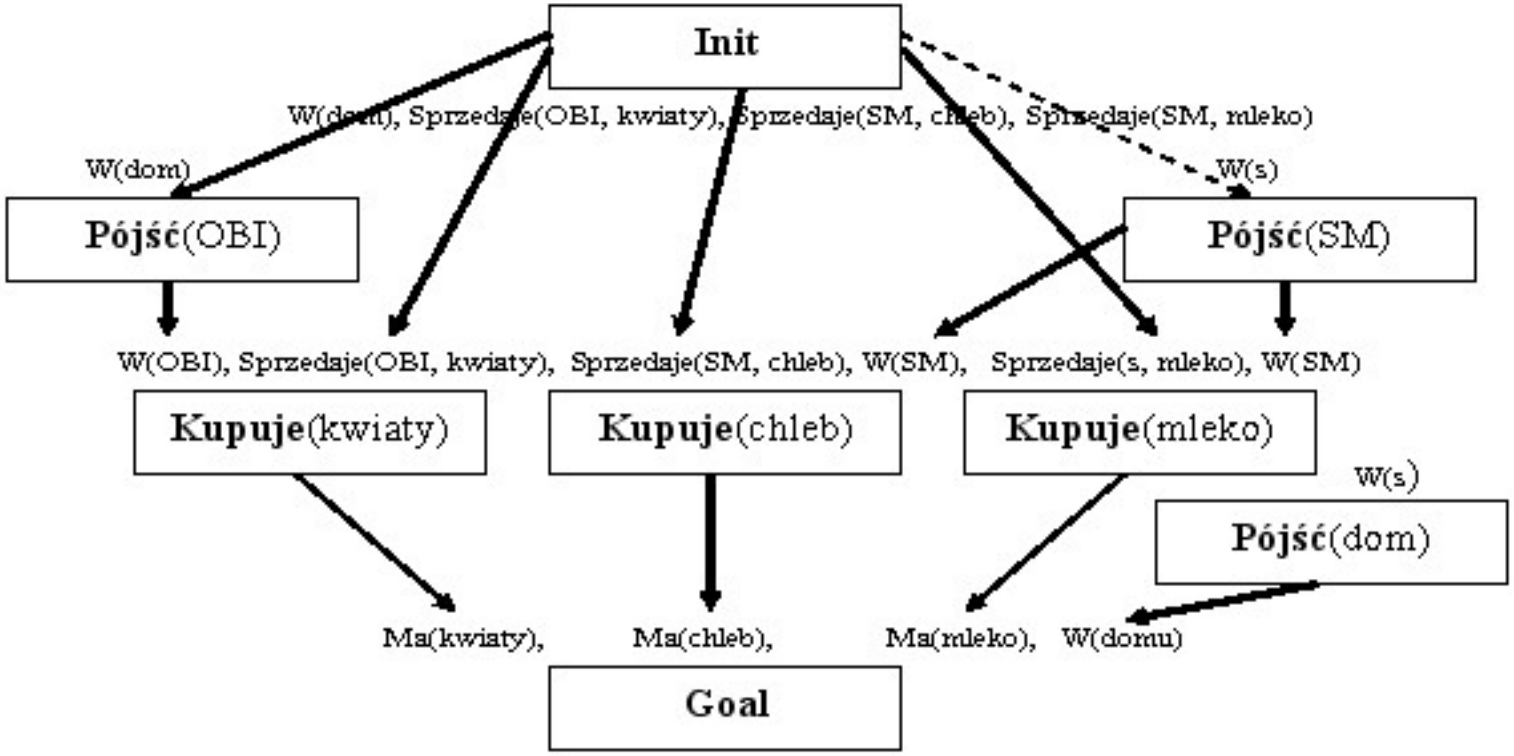
Przykład: rozbudowa planu (3)



Plan po spełnieniu warunków dla kroków „Kupuje”.

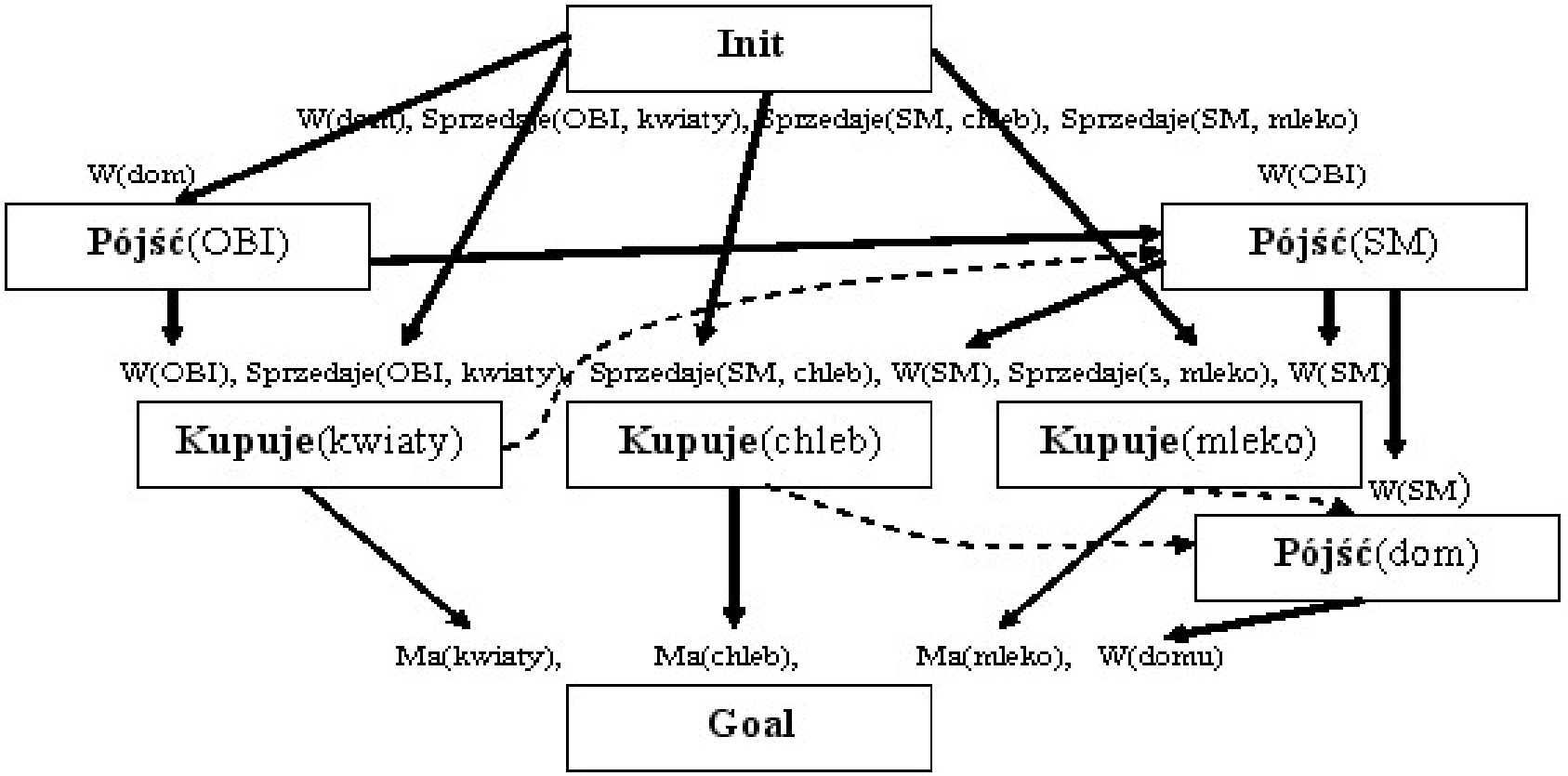
Przykład: rozbudowa planu (4)

Powstał kłopot (**zagrożenie**) związany z rozpoczynaniem z domu. Dodanie już pierwszego związku pomiędzy krokami Init – Pójść zmienia spełniony dotąd od początku warunek W(dom) dla operatora Goal. Zarówno próba degradacji (przed Init) jak i promocji (po Goal) zawiodą, więc **wykonywany jest nawrót** i zostanie dodany nowy krok Pójść(dom) dla spełnienia warunku W(dom) kroku Goal.



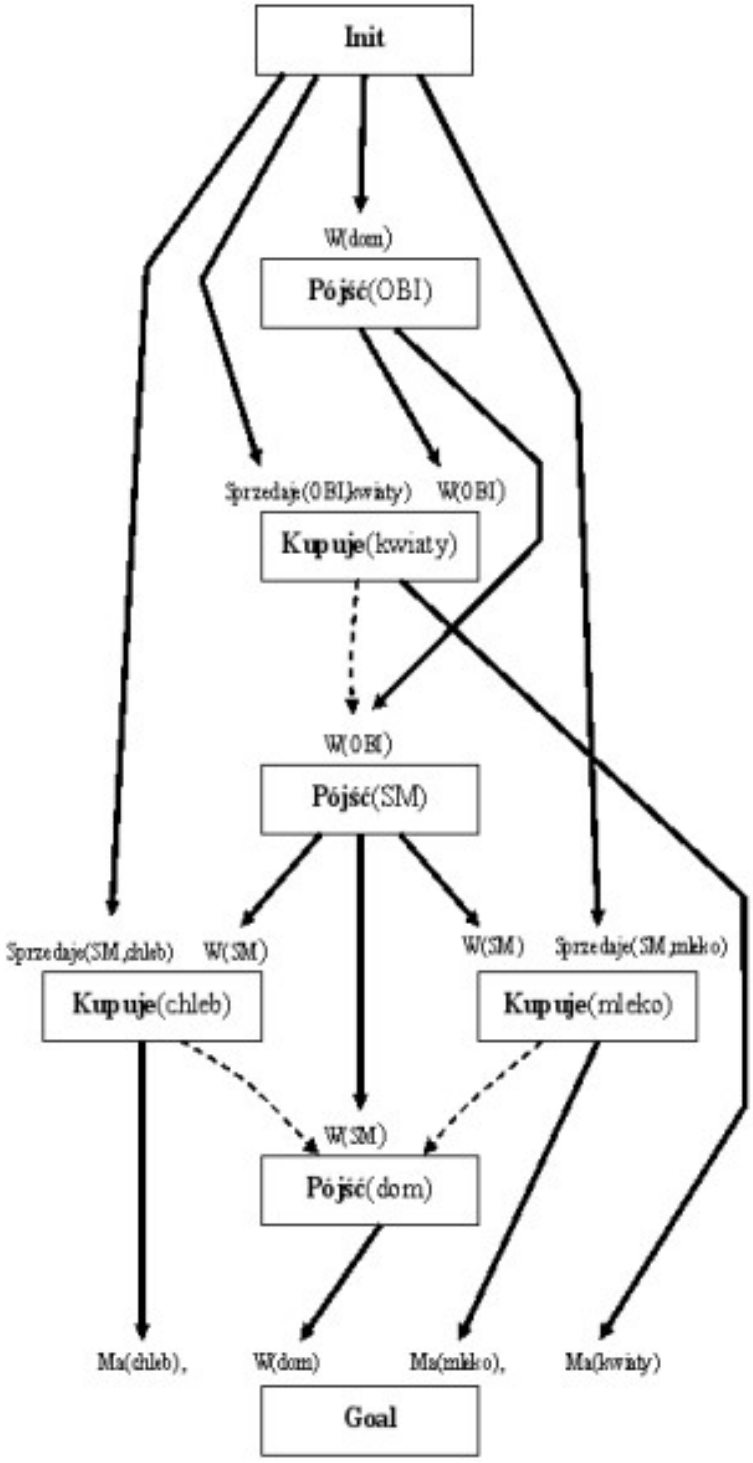
Przykład: plan pełny (5)

Można teraz dodać ponownie związek przyczynowy dla Init – Pójść(OBI). Nastąpi też próba spełnienia warunku dla operatora Pójść(dom). Załóżmy, że dodany zostanie związek przyczynowy z operatorem Pójść(SM). Po uwzględnieniu wymaganych kolejności wykonania operatorów otrzymamy plan końcowy.



Przykład: plan ostateczny

Plan wykonywany jest „od góry do dołu”. Pogrubione strzałki wskazują zależności przyczynowo-skutkowe zapewniające spełnienie warunków dla poszczególnych akcji. Operatory na tym samym poziomie mogą być wykonane w dowolnej kolejności lub równolegle.



4. Graf dla planowania

Specjalna struktura danych, tzw. **graf dla planowania** (lub graf planera), może posłużyć do:

1. szacowania heurystyki dla przeszukiwania planów;
2. alternatywnie – do bezpośredniego wyznaczenia planu przez algorytm nazwany GRAPHPLAN-em.

Graf dla planowania:

• Sekwencja poziomów odpowiadająca przedziałom czasu – na przemian występują poziomy „stanów” (literały) (S_i) i „akcji” (A_i).

• Poziom S_0 zawiera literały spełnione w stanie początkowym.

• Akcja znajduje się w A_i wtedy, gdy jej warunek jest w S_i. Krawędzie prowadzą od warunków w S_i do efektów akcji w S_{i+1}.

• Krawędź pokazująca trwałość literału, łączy literał w S_i z tym samym literałem w S_{i+1}.

Graf dla planowania (2)

- Liczba poziomów w grafie planującym, po których uzyskany zostaje określony literał jest dobrym oszacowaniem (optymistycznym) tego, jak trudno jest uzyskać ten literał wychodząc ze stanu początkowego.
- Graf planujący wymaga literałów w postaci predykatowej – pozbawionych zmiennych. Ponieważ zarówno STRIPS jak i ADL dopuszczają stosowanie jedynie literałów podstawowych, tzn. występujące w nich termy nie zawierają symboli funkcji, takie reprezentacje mogą zostać przekształcone do postaci predykatowej.

Mutex

Wzajemna rozłączność - reprezentowana jest łukami typu „mutex” („mutual exclusion”):

1. Dwie akcje w A_i wzajemnie się wyłączają, jeśli jedna z nich (jej efekt) neguje warunek wykonania drugiej lub neguje efekt drugiej akcji.
2. Dwa literały w S_{i+1} połączone są łukiem „mutex” jeśli jeden jest negacją drugiego lub jeśli każda para generujących je akcji w A_i jest połączona łukiem „mutex”.
3. Akcja nie może wystąpić w A_i jeśli dowolna para jej warunków jest w S_i uznana za wykluczającą się (jest połączona łukiem „mutex”).
4. Dwie akcje w A_i wzajemnie się wyłączają, jeśli warunek jednej z nich wyklucza się z warunkiem drugiej.

Przykład

Dla problemu „mieć ciastko, zjeść ciastko i nadal je mieć” zdefiniujemy operatory:

OP(Init,
EFFECT:(*have(Cake)*))

OP(Goal,
PRECOND: (*have(Cake)* $\wedge$ *eaten(Cake)*))

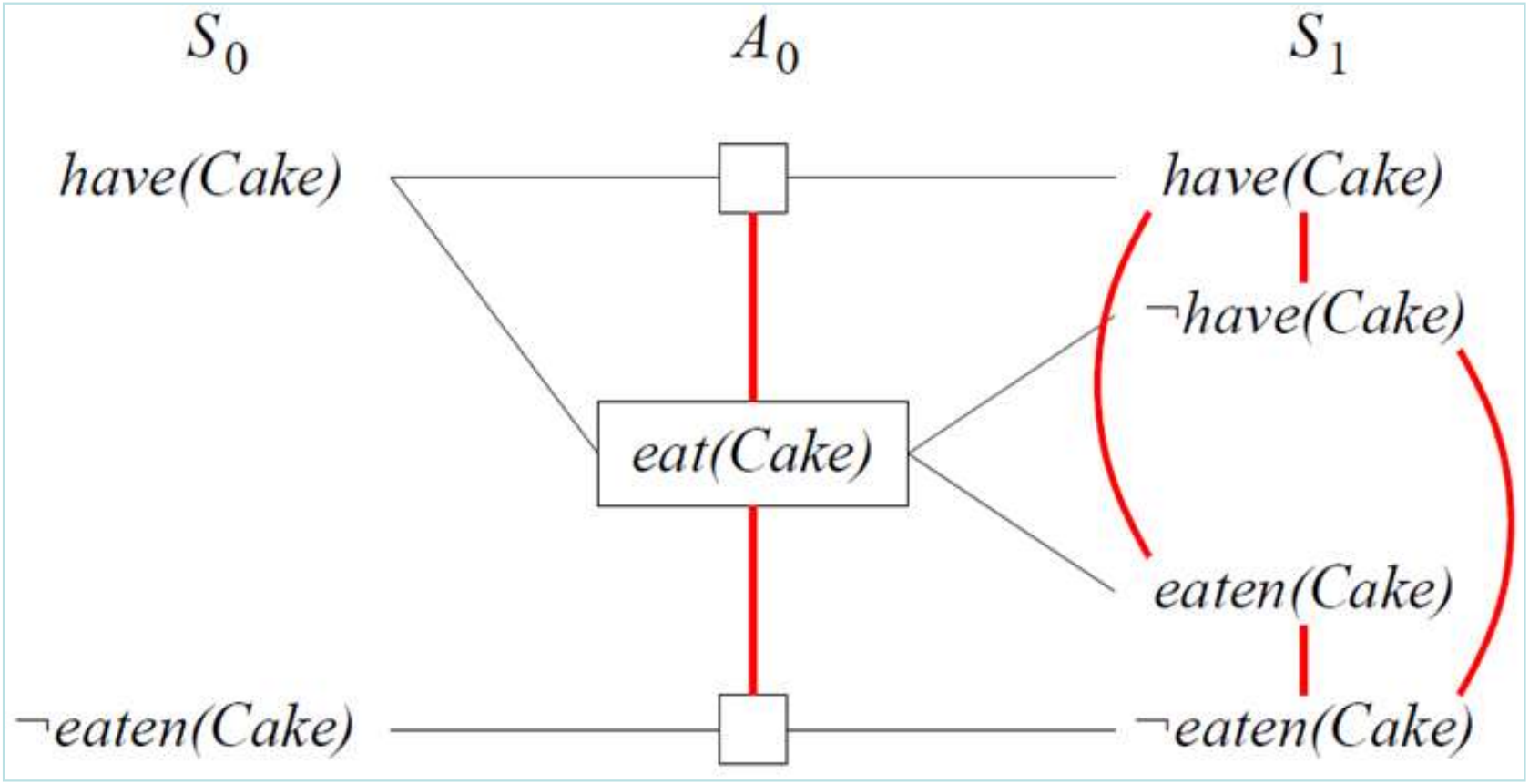
OP(Action(*eat(Cake)*),
PRECOND: *have(Cake)*
EFFECT: *eaten(Cake)* $\wedge$ $\neg$*have(Cake)*))

OP(Action(*bake(Cake)*),
PRECOND: $\neg$*have(Cake)*
EFFECT: *have(Cake)*))

Przykład (2)

Początkowy poziom akcji.

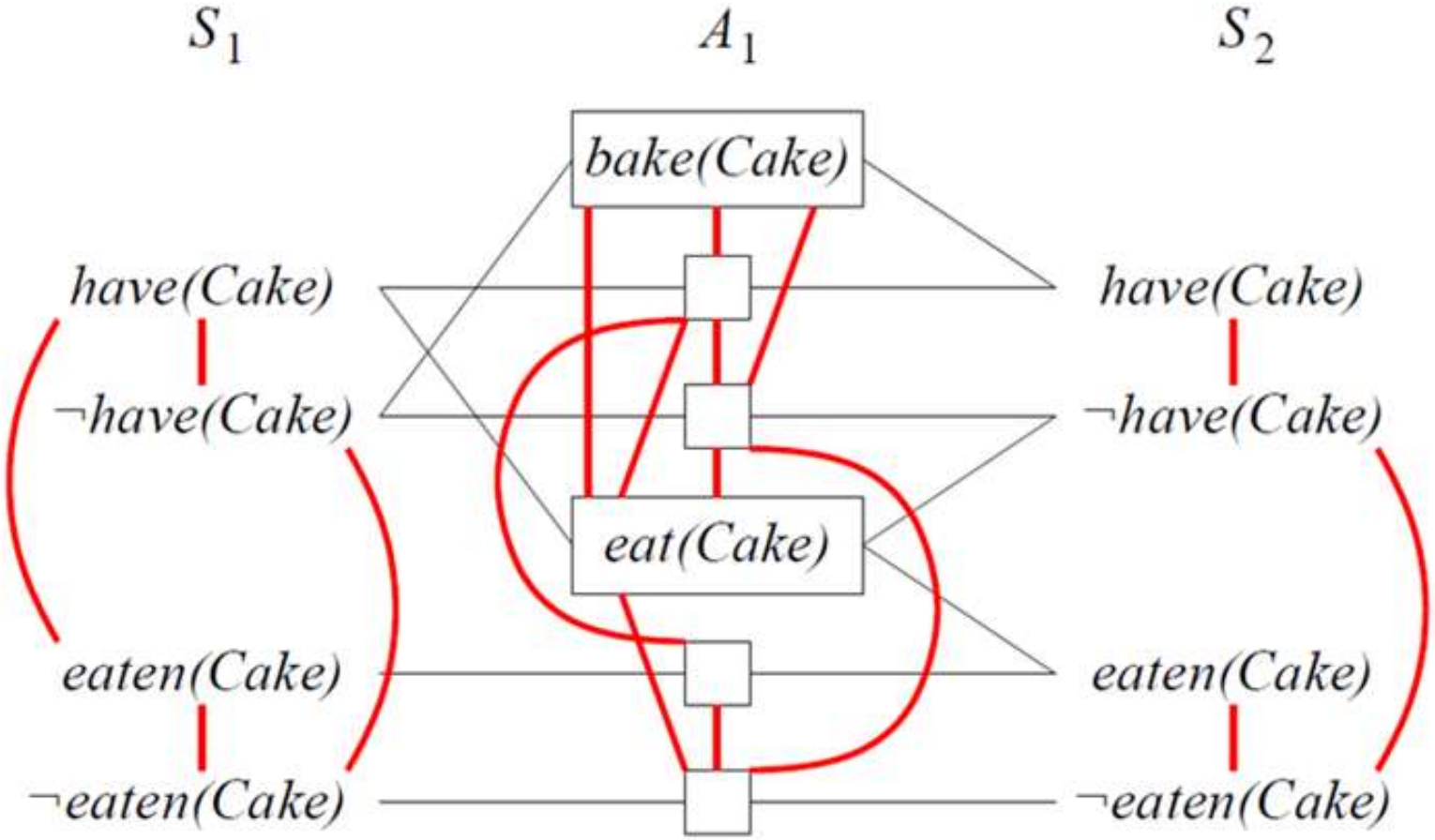
W S_1 literały $have(Cake)$ i $eaten(Cake)$ tworzą mutex, gdyż spełnianie $have(Cake)$ jest rozłączne z akcją $eat(Cake)$.



Przykład (3)

Kolejny poziom akcji.

Obie akcje tworzące literał $\neg have(Cake)$ tworzą mutex z akcją dającą $\neg eaten(Cake)$.



5. Algorytm „Graphplan”

- **Graphplan** pobiera problem planowania zdefiniowany w STRIPS i wyznacza sekwencję akcji prowadzącą z początkowego stanu problemu do docelowego stanu problemu.
- Graphplan operuje na grafie planującym, w którym:
  - węzły odpowiadają akcjom i literałom uporządkowanym w naprzemienne poziomy,
  - łuki są 3 rodzajów: od literału (warunku) do akcji, od akcji do literału (efektu), od literału do literału (gdy nie ulega zmianie),
- Pamiętane są listy wykluczających się literałów, czyli takich, które nie mogą być jednocześnie prawdziwe, a także listy wykluczających się akcji – nie mogą być wspólnie wykonywane na danym poziomie.
- Algorytm iteracyjnie rozszerza graf planujący (wstecz - „od tyłu do przodu), sprawdzając, czy nie ma rozwiązań o długości l-1 zanim zajmie się planami o długości l.

Graphplan (2)

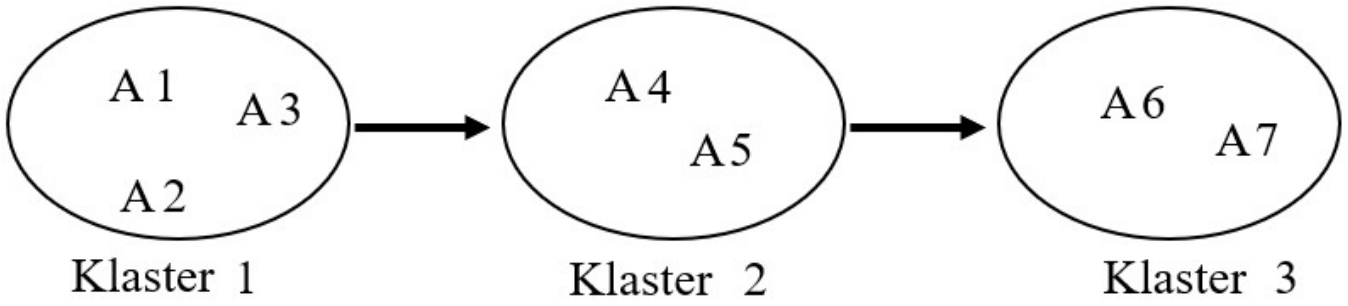
Algorytm „Graphplan”: wyznacza plan na podstawie grafu planującego analizując go od ostatniego poziomu wstecz. Iteracyjny sposób analizy odpowiada przeszukiwaniu wszerz pewnego drzewa decyzyjnego:

- Węzłem drzewa jest podzbiór zgodnych ze sobą akcji na jednym poziomie grafu, których efektem są wszystkie te literały, które uznane zostały za cele w poprzedniku węzła.
- Węzeł początkowy jest zbiorem celów na ostatnim poziomie S_n grafu planującego.
- Istnieje krawędź w drzewie decyzyjnym od węzła (=podzbiór w S_i) do węzła(=podzbiór w S_{i-1}), jeśli istnieje podzbiór zgodnych akcji w A_{i-1} , które łączą je jako efekty i warunki.
- Celem przeszukiwania jest osiągnięcie S_0.

Graphplan (3)

- Graphplan na przemian: a) podejmuje wykrycie rozwiązania („czy osiągnięty został stan początkowy problemu – w postaci spójnego zbioru literałów”?) i b) rozszerza graf o podzbiory akcji i warunki poprzedniego poziomu w grafie planującym (porusza się WSTECZ wzdłuż grafu planującego).
- W wyniku uzyskuje się plan, w którym akcje nie są ani w pełni uporządkowane ani częściowo uporządkowane (PCzU)
- Powstaje pośrednie rozwiązanie:
  - Plan składa się z sekwencji klastrów akcji,
  - W ramach klastra kolejność akcji jest dowolna,
  - Kolejność klastrów jest ustalona.

Np.



Funkcja „Graphplan”

```
function GRAPHPLAN(problem) returns solution lub failure
{  graph := GENERUJGRAFPOCZĄTKOWY(problem);
   goals := Cele(problem);  S0 := StanPoczątkowy(problem);
   loop do { if goals niepuste then do {
                    solution := WYZNACZROZWIĄZANIE(S0, graph, goals,
                    DŁUGOŚĆ(graph)); // jeśli S0 zawarty jest w goals
                    if solution ≠ ∅  then return solution; // osiągnięto stan
                    // początkowy, tzn. istnieje spójny zbiór w goals
                    }
              else if  BRAKROZWIĄZANIA(graph)  then return failure ;
              graph := ROZSZERZGRAF(graph, problem); // generuj wstecz
                                                     // warstwę akcji i stanu
              goals := {wszystkie zbiory spójne  (bez mutex) w najnowszej
                              warstwie stanu w graph};
            }
}
```

Zalety „Graphplan”-u

- Znaczna poprawa efektywności w porównaniu z planerem PCzU, dzięki:
  - wzajemnemu wykluczaniu, równoległemu planowaniu, przycinaniu nieosiągalnych zbiorów celów, nie ma potrzeby podstawiania pod zmienne podczas przeszukiwania.
- Dalsza zaleta: zakończenie pracy, gdy problem jest nierozwiązywalny:
  - wykrycie sytuacji, gdy dany poziom stanów w grafie planującym jest identyczny z kolejnym poziomem;
  - jeśli nie znaleziono planu przed powstaniem takiej sytuacji, to oznacza brak rozwiązania.

Pytania

1. Wyjaśnić problem planowania.
2. Czym są STRIPS i ADL?
3. Omówić podstawową strategię tworzenia planu częściowo-uporządkowanego.
4. Wyjaśnić istotę operacji „promocji” i „degradacji” oraz przyczyny ich stosowania podczas tworzenia planu.
5. Omówić istotę „grafu planującego” i jego zastosowanie w algorytmie „Graphplan”.