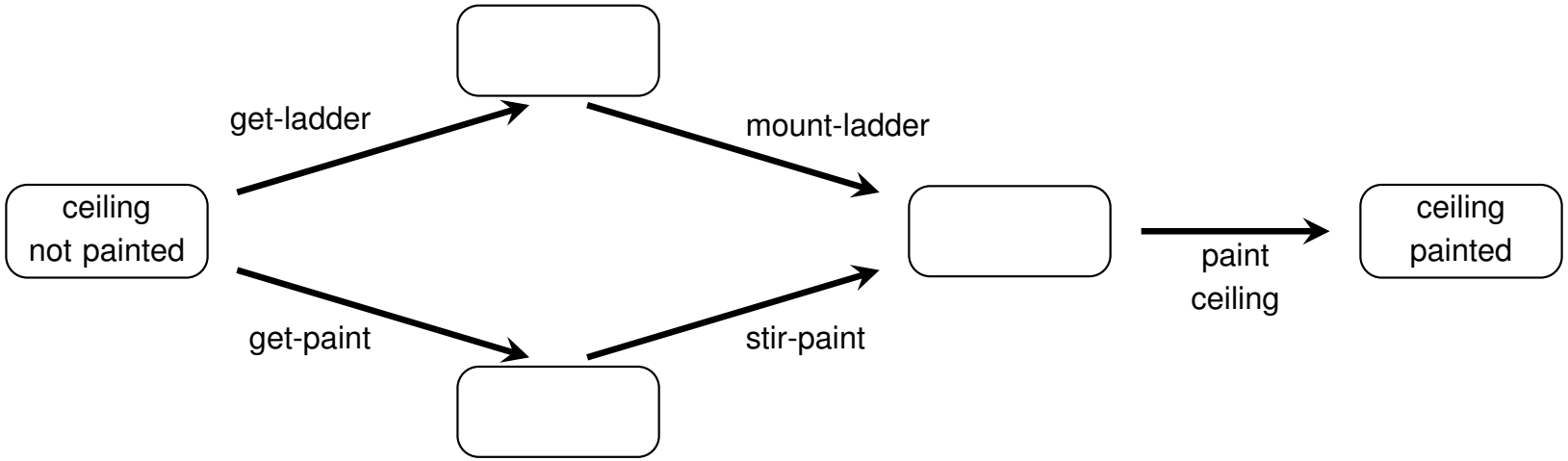
SID – Wykład 9
Planowanie

Dominik Ślęzak

Wydział Matematyki, Informatyki i Mechaniki UW
slezak@mimuw.edu.pl

Problem planowania



Język STRIPS: stany

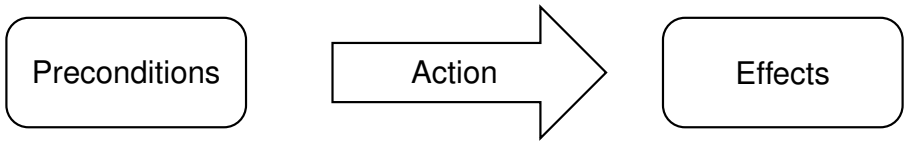
Reprezentują chwilowe stany opisywanego świata,
każdy stan opisany jest jako koniunkcja literałów relacyjnych,
literały są zawsze ustalone i bez wyrażeń funkcyjnych,
w opisie występują tylko literały pozytywne,
o pozostałych zakłada się, że są nieprawdziwe

Np. $At(Spare, Trunk) \wedge At(Flat, Axle)$

Niedopuszczalne: ~~$At(x, y)$~~, ~~$At(Father(Fred), Sydney)$~~

Język STRIPS: akcje

Reprezentują czynności zmieniające stan opisywanego świata, każda akcja ma warunki wstępne, które muszą być spełnione przed zastosowaniem akcji, oraz efekty, które zachodzą po wykonaniu akcji.



Warunki wstępne:

koniunkcja pozytywnych literałów relacyjnych, może zawierać zmienne i stałe, ale bez wyrażeń funkcyjnych

Efekty:

koniunkcja negatywnych i pozytywnych literałów relacyjnych, może zawierać zmienne z war. wstępnych i stałe, ale bez wyrażeń funkcyjnych

$At(x, Trunk)$ | $Remove(x, Trunk)$ | $\neg At(x, Trunk)$, $At(x, Ground)$

Język STRIPS: postawienie problemu

Stan początkowy

Wybrany stan reprezentujący warunki początkowe zadanego problemu

Cel

Zbiór faktów, tzn. pozytywnych literałów relacyjnych z ustalonymi wartościami (tzn. bez zmiennych i wyrażeń funkcyjnych)

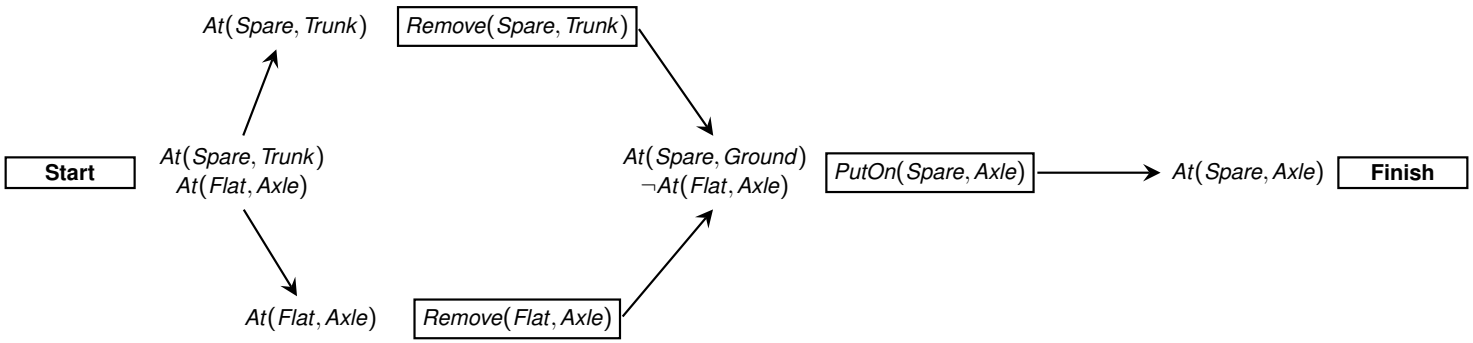
Uwaga: cel nie jest stanem — wiele stanów może spełniać warunki docelowe

Plan jako rozwiązanie problemu

Plan

Częściowo uporządkowany zbiór akcji przekształcający stan początkowy w stan spełniający warunki docelowe

Akcje w planie są ukonkretnione, tzn. zmienne występujące jako parametry akcji mają przypisane konkretne wartości.



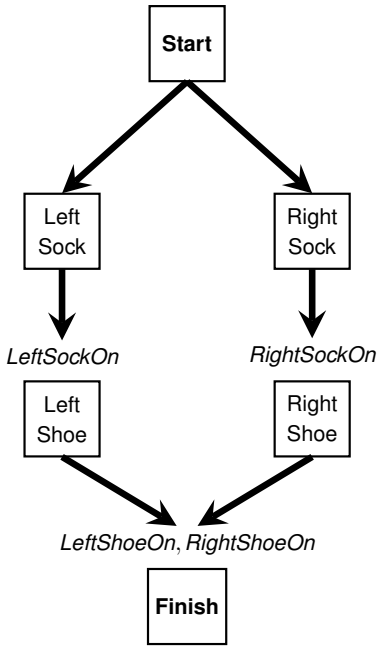
Po zastosowaniu akcji nowy stan powstaje poprzez dodanie pozytywnych oraz usunięcie negatywnych efektów akcji ze stanu poprzedzającego tę akcję.

Poprawność planu polega na tym, że dla każdej akcji stan poprzedzający tę akcję spełnia ukonkretnione warunki wstępne tej akcji.

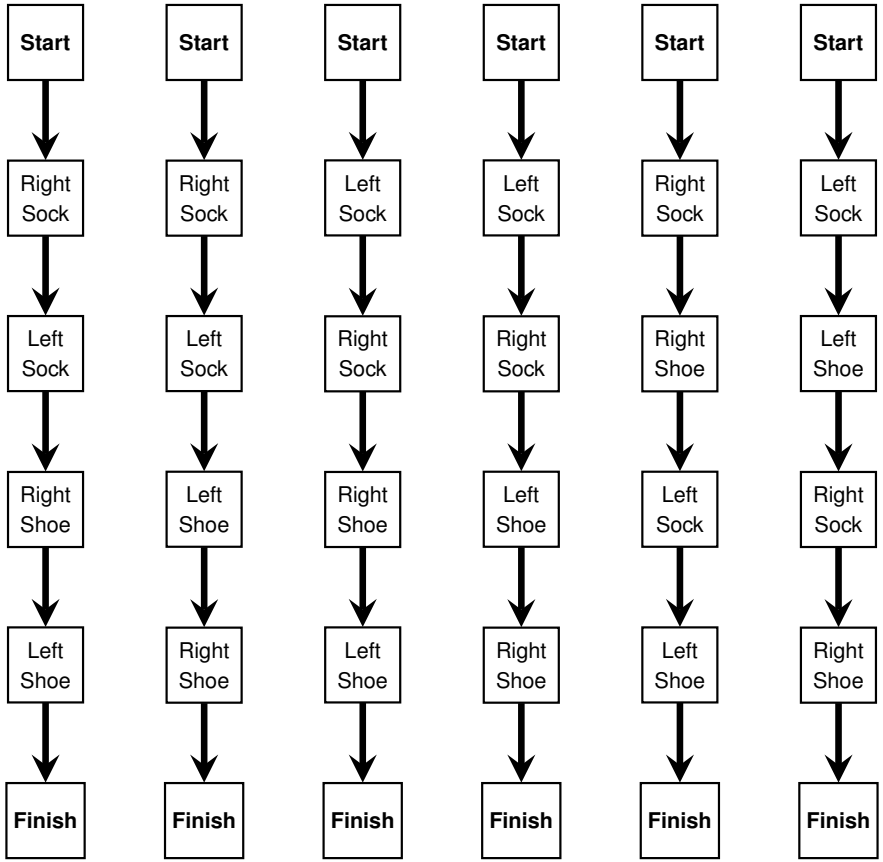
Plan jako rozwiązanie problemu

W praktyce poszukuje się liniowo uporządkowanych planów.

Partial-Order Plan:



Total-Order Plans:



Redukcja przestrzeni planów

Przestrzeń możliwych planów jest zazwyczaj bardzo duża, dlatego stosuje się heurystyczne metody redukcji przestrzeni stanów:

Linearyzacja

pomija plany, w których akcje prowadzące do różnych podcelów występują naprzemiennie

Ochrona

pomija plany, w których występuje akcja eliminująca jeden z wcześniej osiągniętych podcelów

linearyzacja i ochrona mogą spowodować wykluczenie poprawnych planów, w szczególności mogą wykluczyć wszystkie poprawne plany i uniemożliwić znalezienie rozwiązania

Metody poszukiwania poprawnych planów

- Przeszukiwanie wprzód
  - — ForwardChaining
- Przeszukiwanie wstecz
  - — PopPlan
  - — GraphPlan
- Sprowadzenie do problemu spełnialności
  - — SatPlan

ForwardChaining

Poszukuje planu z liniowo uporządkowanymi akcjami zaczynając od stanu początkowego i dodając akcje w przód. Algorytm cofa się jeśli nie może zastosować już żadnej akcji w bieżącym stanie lub bieżący stan wystąpił już wcześniej.

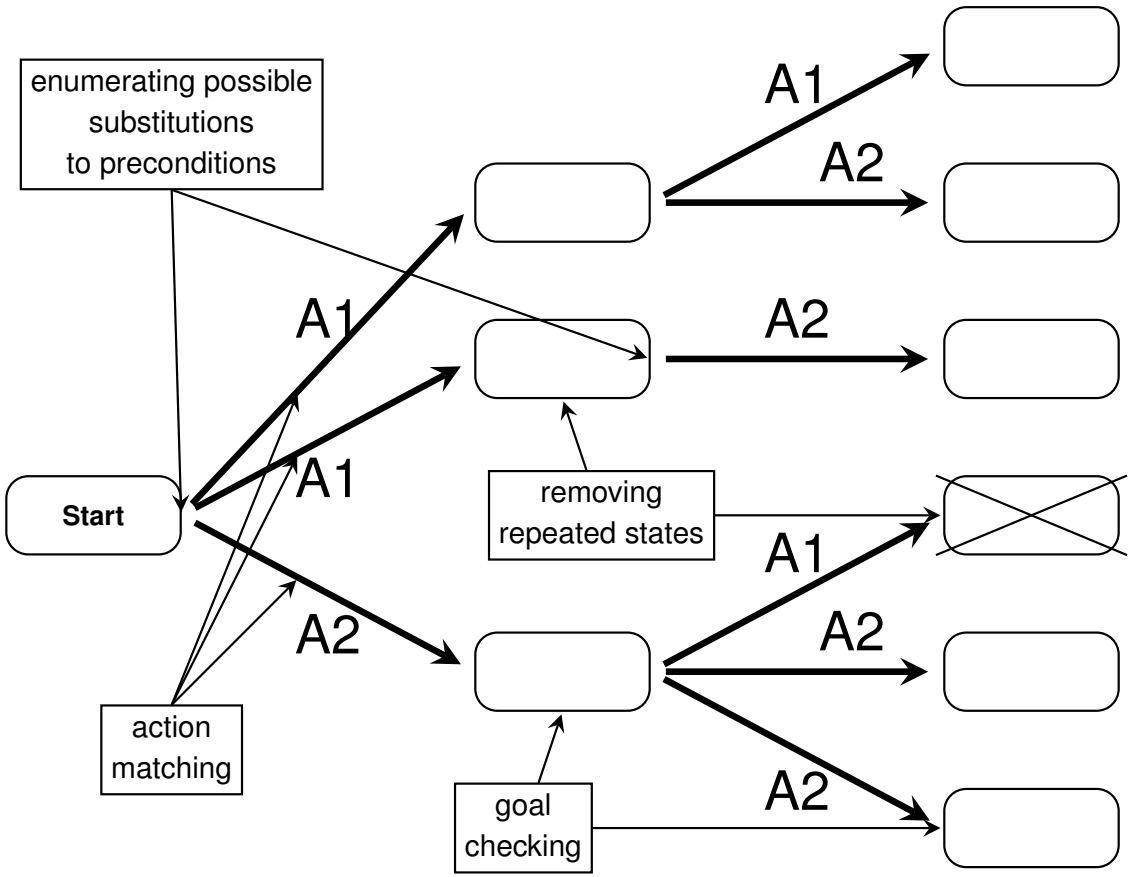
♢ Przeglądanie możliwych podstawień do warunków wstępnych akcji

Najpierw można dopasowywać fakty z warunków wstępnych akcji i ze stanu, a potem przeglądać pasujące podstawienia.

♢ Usuwanie powtarzających się stanów

Stany wyliczone w poprzednich krokach algorytmu mogą być pamiętane w strukturze danych pozwalającej na szybkie wyszukiwanie.

ForwardChaining: algorytm



Połączenia przyczynowe i więzy porządkujące

Połączenia przyczynowe

Połączenia w planie prowadzące od efektu jednej akcji do warunku wstępnego innej akcji.

Więzy porządkujące

Więzy między akcjami w planie wskazujące, że jedna akcja występuje przed inną akcją.

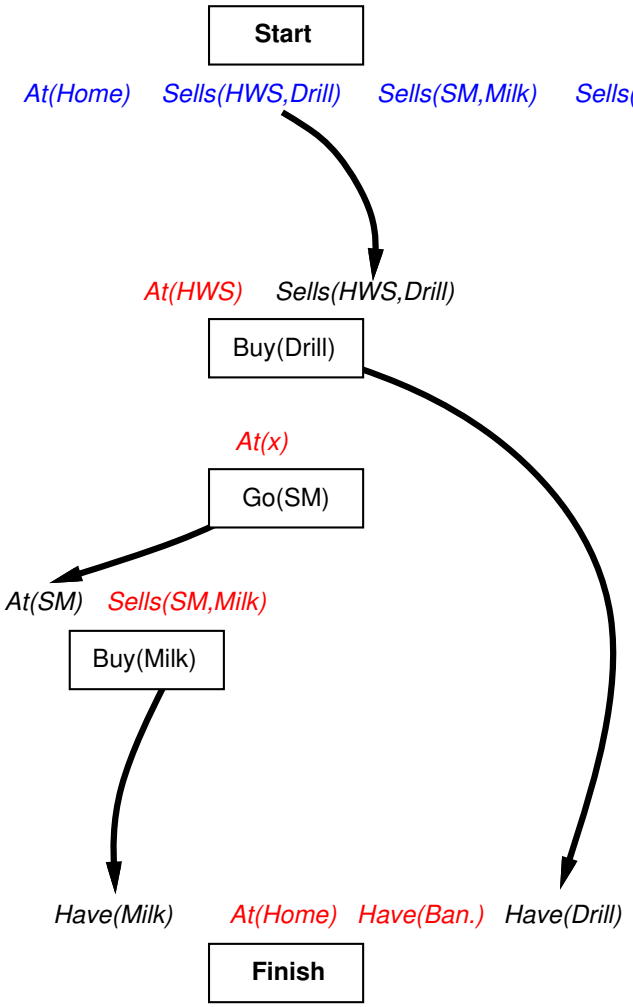
Warunek otwarty

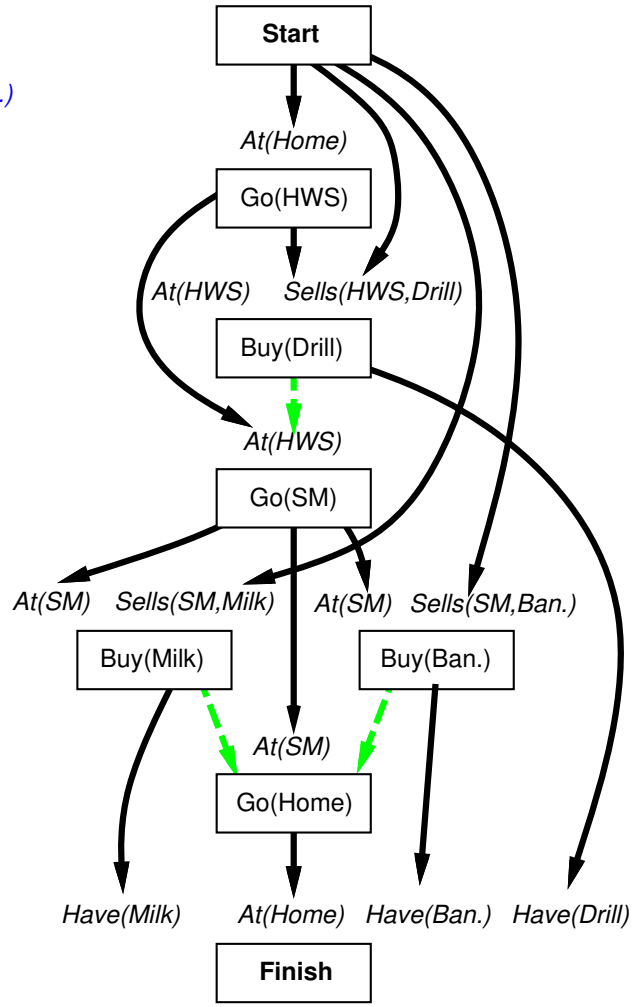
Warunek wstępny jednej z akcji nie posiadający połączenia przyczynowego prowadzącego od efektu innej akcji.

Plan jest kompletny $\Longleftrightarrow$ kiedy każdy warunek wstępny jest osiągnięty.

Warunek wstępny jest osiągnięty $\Longleftrightarrow$ jeśli jest efektem wcześniejszej akcji i żadna akcja przeszkadzająca nie eliminuje tego efektu.

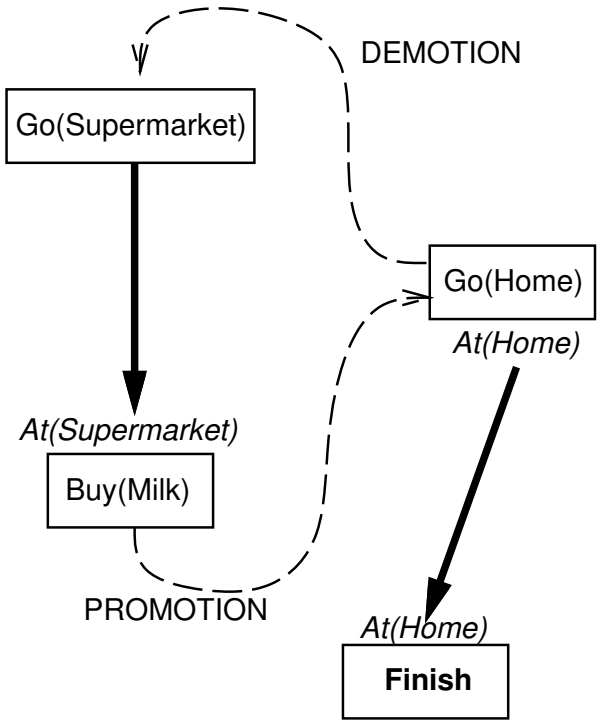
Połączenia przyczynowe i więzy porządkujące





Kloberowanie oraz promocja/democja

Akcja kloberująca jest potencjalnie przeszkadzającą akcją, która eliminuje warunek uzyskany z powiązania przyczynowego, np. $Go(Home)$ kloberuje $At(Supermarket)$:



Democja: dodanie więzu porządkującego
$Go(Home) \rightarrow Go(Supermarket)$

Promocja: dodanie więzu porządkującego
$Buy(Milk) \rightarrow Go(Home)$

PopPlan: algorytm

Pomysł: Stopniowo kompletuje plan zaczynając od pustego planu poprzez dodawanie połączeń przyczynowych, akcji i więzów porządkujących.
Cofa, gdy otwarty warunek jest nieosiągalny lub konflikt jest nierozstrzygalny.

function POP(*initial*, *goal*, *operators*) **returns** plan
  plan ← MAKE-MINIMAL-PLAN(*initial*, *goal*)
  loop
    if SOLUTION?(*plan*) **then**
      return *plan*
    end if
    $S_{need}, c \leftarrow$ SELECT-SUBGOAL(*plan*)
    CHOOSE-OPERATOR(*plan*, *operators*, S_{need}, c)
    RESOLVE-THREATS(*plan*)
  end loop
end function

---

function SELECT-SUBGOAL(*plan*) **returns** S_{need}, c
  pick a plan step S_{need} from STEPS(*plan*) with a precondition c that has not been achieved
  return S_{need}, c
end function

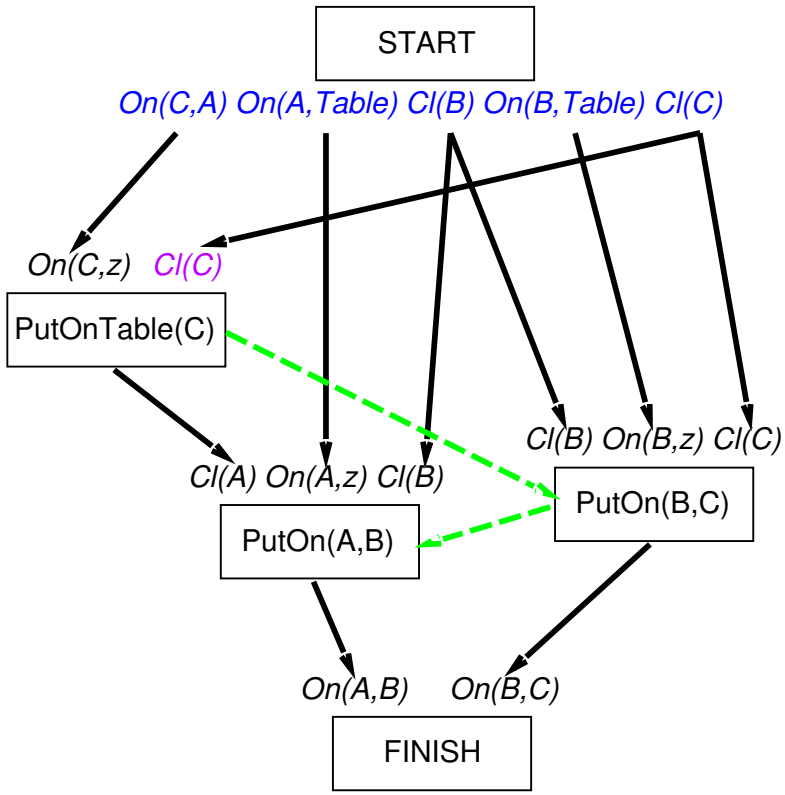
PopPlan: algorytm

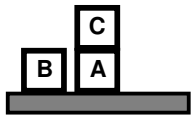
function CHOOSE-OPERATOR(*plan*, *operators*, S_{need}, c)
  choose a step S_{add} from *operators* or STEPS(*plan*) that has c as an effect
  if there is no such step **then**
    fail
  end if
  add the causal link $S_{add} \xrightarrow{c} S_{need}$ to LINKS(*plan*)
  add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS(*plan*)
  if S_{add} is a newly added step from *operators* **then**
    add S_{add} to STEPS(*plan*)
    add $Start \prec S_{add} \prec Finish$ to ORDERINGS(*plan*)
  end if
end function

---

function RESOLVE-THREATS(*plan*)
  for each S_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS(*plan*) **do**
    choose either
      Demotion: Add $S_{threat} \prec S_i$ to ORDERINGS(*plan*)
      Promotion: Add $S_j \prec S_{threat}$ to ORDERINGS(*plan*)
    if not CONSISTENT(*plan*) **then**
      fail
    end if
  end for
end function

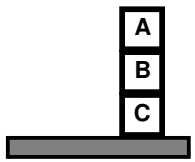
PopPlan: przykład





PutOn(A,B) clobbers Cl(B) => order after PutOn(B,C)

PutOn(B,C) clobbers Cl(C) => order after PutOnTable(C)



PopPlan: własności

Niedeterminizm:

- wybór akcji S_{add} do powiązania przyczynowego z S_{need}
- wybór democji lub promocji dla akcji kloberującej (wybór S_{need} może być ustalony: każdy warunek wstępny musi zostać osiągnięty w końcowym planie)

PopPlan jest pełny i systematyczny (wyklucza powtórzenia).

Można efektywnie przyspieszyć wybierając właściwą heurystykę rozstrzygania konfliktów na podstawie opisu problemu.

Szczególnie efektywny dla problemów z wieloma luźno powiązanymi podcelami.

Graf planowania

Graf planowania składa się z poziomów:

- poziom S_0 odpowiada stanowi początkowemu z opisu zadania
- kolejne poziomy odpowiadają kolejnym krokom czasu

Każdy poziom zawiera:

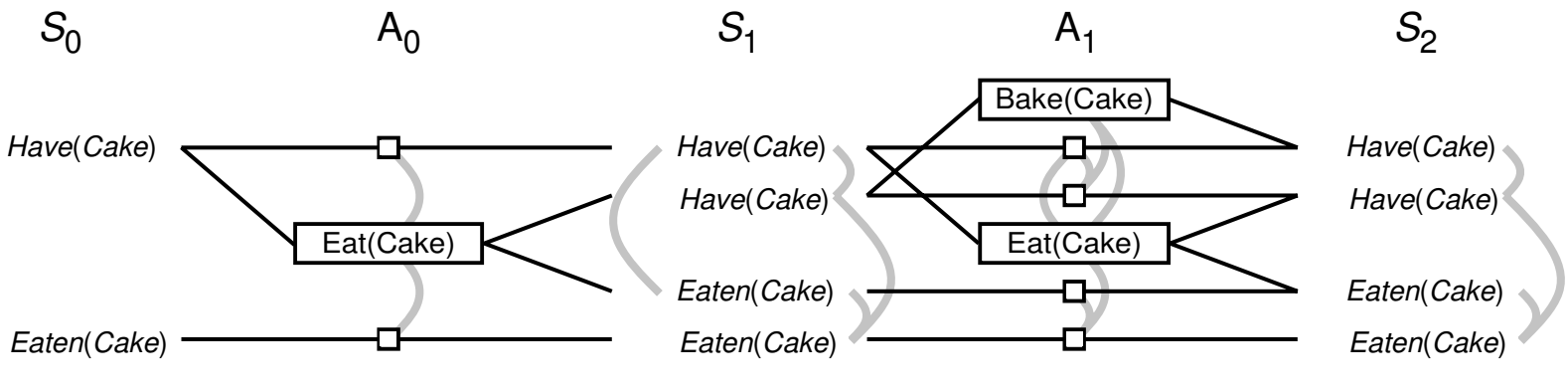
- zbiór literałów: reprezentuje te, które mogą być spełnione w danym kroku
- zbiór akcji: reprezentuje akcje, które mogą być użyte w danym kroku; oprócz akcji z opisu problemu, dla każdego literału dopuszczalna jest akcja zachowująca stan tego literału

Poziomy są rozwijane do momentu, kiedy dwa kolejne poziomy są identyczne (tzn. każdy następny poziom byłby identyczny z poprzednim).

Graf planowania: przykład

Stan początkowy: $Have(Cake)$
Cel: $Have(Cake) \wedge Eaten(Cake)$
Akcje:

- $Eat(Cake)$
  warunki wstępne: $Have(Cake)$
  efekty: $\neg Have(Cake) \wedge Eaten(Cake)$
- $Bake(Cake)$
  warunki wstępne: $\neg Have(Cake)$
  efekty: $Have(Cake)$



Relacja wzajemnej wyłączności (mutex)

Relacja mutex zachodzi pomiędzy dwoma akcjami, jeśli występują:

- albo sprzeczne efekty akcji
- albo interferencja, tzn. efekt jednej z akcji jest negacją warunku wstępnego drugiej akcji
- albo akcje mają sprzeczne warunki wstępne

Relacja mutex zachodzi pomiędzy dwoma literałami jeśli jeden jest negacją drugiego lub jeśli każda para akcji dająca te dwa literały jest w relacji mutex.

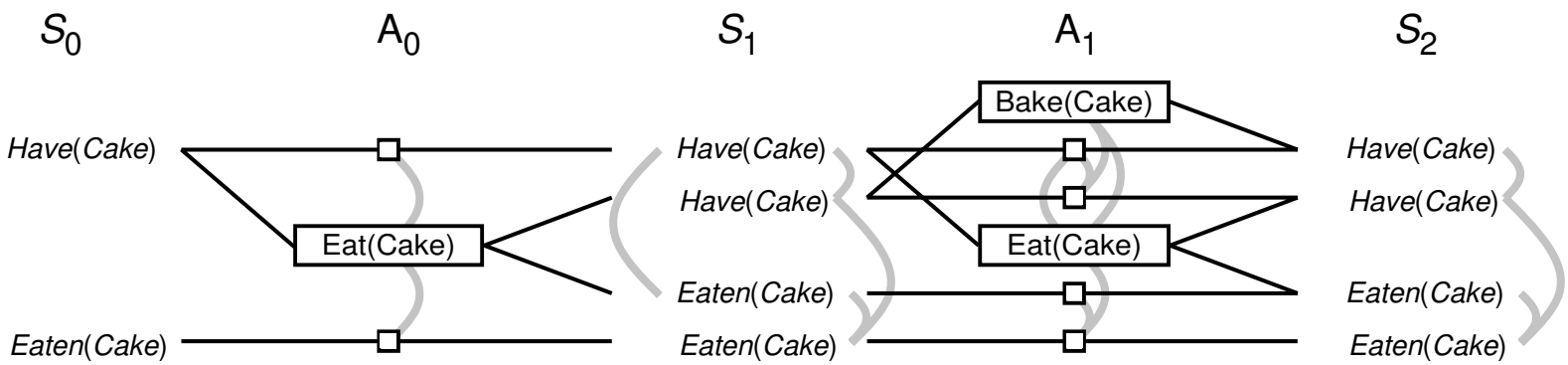
Relacja mutex: przykład

Akcje *Eat*(*Cake*) i akcja zachowująca *Have*(*Cake*) są sprzeczne, bo mają sprzeczny efekt *Have*(*Cake*).

Akcje *Eat*(*Cake*) i akcja zachowująca *Have*(*Cake*) są sprzeczne, bo efekt akcji *Eat*(*Cake*) jest zaprzeczeniem warunku wstępnego akcji zachowującej *Have*(*Cake*).

Akcje *Eat*(*Cake*) i *Bake*(*Cake*) są sprzeczne, bo mają sprzeczny warunek wstępny *Have*(*Cake*).

Literały *Have*(*Cake*) i *Eaten*(*Cake*) są w relacji mutex, bo wymagają użycia wykluczających się akcji *Eat*(*Cake*) i akcji zachowania *Have*(*Cake*).



GraphPlan: algorytm

```
function GRAPHPLAN(problem) returns solution or failure
    graph ← INITIAL-PLANNING-GRAPH(problem)
    goals ← GOALS[problem]
    loop
        if goals all not-mutex in last level of graph then
            solution ← EXTRACT-SOLUTION(graph, goals, LENGTH(graph))
            if solution ≠ failure then return solution
            else
                if NO-SOLUTION-POSSIBLE(graph) then return failure
                end if
            end if
        end if
        graph ← EXPAND-GRAPH(graph, problem)
    end loop
end function
```

Funkcja INITIAL-PLANNING-GRAPH generuje literały na poziomie S_0 grafu planowania czyli literały ze stanu początkowego.

W każdym kroku pętli funkcja EXPAND-GRAPH dodaje akcje z bieżącego poziomu i literały z następnego poziomu grafu planowania.

GraphPlan: szukanie rozwiązania

Funkcja EXTRACT-SOLUTION próbuje znaleźć plan na podstawie dotychczas wygenerowanego grafu planowania. Cofa się od ostatniego poziomu grafu przekształcając bieżący zbiór niespełnionych celów w następujący sposób:

- początkowy zbiór celów to wszystkie cele z zadania na ostatnim bieżącym poziomie S_n
- dla celów na poziomie S_i algorytm wybiera podzbiór akcji z poziomu A_{i-1} tak, żeby efekty tych akcji pokrywały zbiór celów na poziomie S_i i żadne dwie nie wykluczały się wzajemnie,
  funkcja zwraca porażkę, jeśli takiego zbioru akcji nie da się wybrać
- warunki wstępne akcji wybranych na poziomie A_{i-1} stają się bieżącymi celami na poziomie S_{i-1}
- szukanie rozwiązania kończy się sukcesem jeśli na poziomie S_0 bieżące cele będą podzbiorem faktów w stanie początkowym

GraphPlan: wybór celów i akcji

Koszt poziomowy literału: najniższy poziom w grafie planowania, na którym dany literał występuje po raz pierwszy.

Funkcja EXTRACT-SOLUTION używa następującej heurystyki wyboru celów i akcji na każdym poziomie:

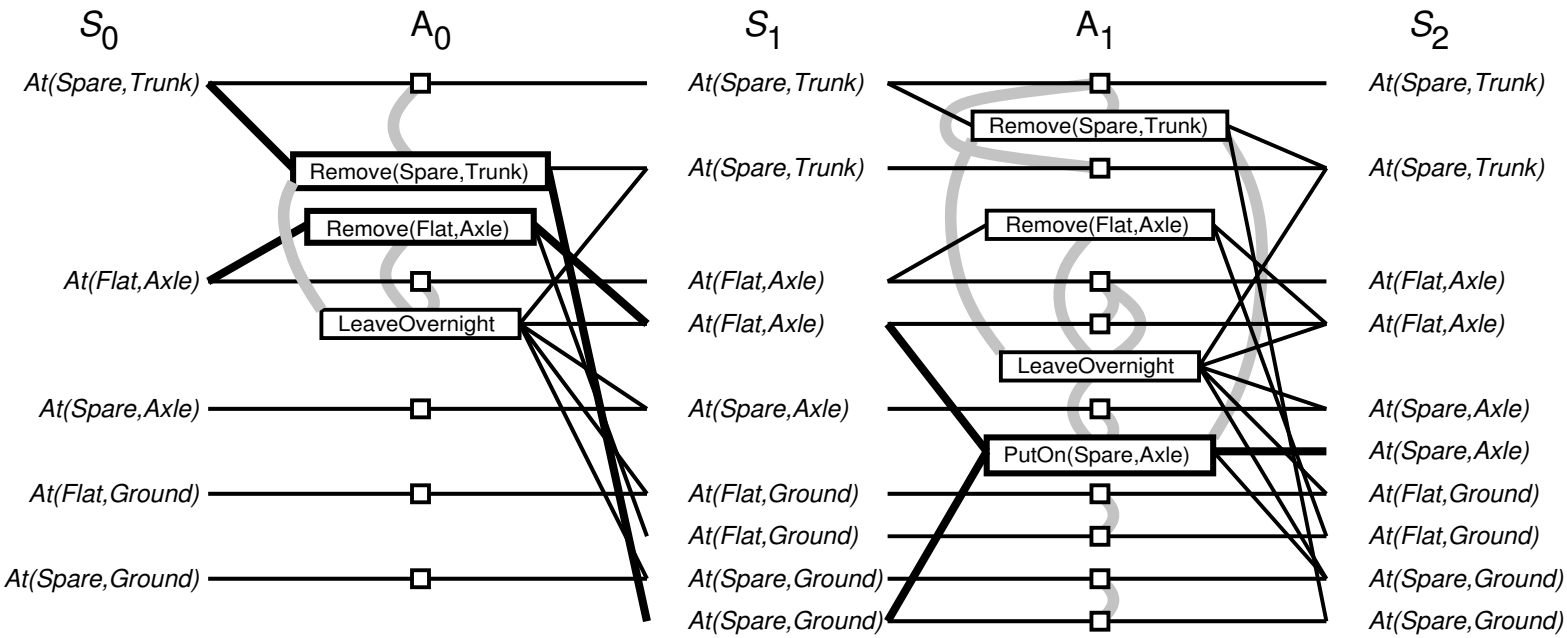
- cele wybierane są w kolejności od najwyższego do najniższego kosztu poziomowego
- do osiągnięcia danego celu wybierana jest akcja z najmniejszą sumą (lub maksimum) kosztów poziomowych warunków wstępnych akcji

Twierdzenie

> Dla każdego zadania planowania można efektywnie wyznaczyć skończony poziom grafu planowania, od którego funkcja EXTRACT-SOLUTION znajdzie rozwiązanie, jeśli ono istnieje.

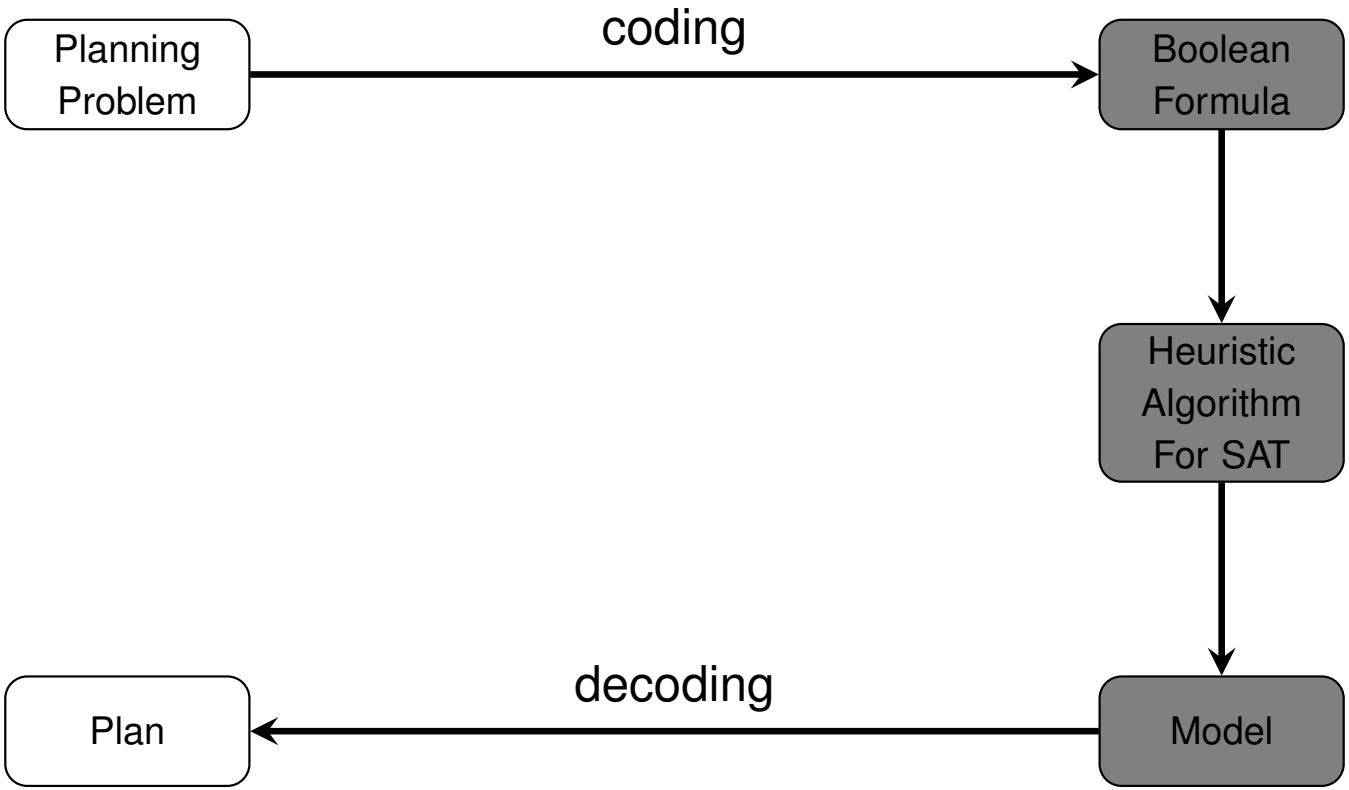
GraphPlan: przykład

Cel: $At(Spare, Axle)$



Powyżej zostały zaznaczone tylko niektóre relacje mutex istotne dla przykładu.

SATPlan


Planning Problem
coding
Boolean Formula
Heuristic Algorithm For SAT
Model
decoding
Plan

SATPlan: algorytm

```
function SATPLAN(problem, T_max) returns solution or failure
    inputs: problem - a planning problem
            T_max - an upper limit for plan length
    for T = 0 to T_max do
        cnf, mapping ← TRANSLATE-TO-SAT(problem, T)
        assignment ← SAT-SOLVER(cnf)
        if assignment is not null then
            return EXTRACT-SOLUTION(assignment, mapping)
        end if
    end for
    return failure
end function
```

Funkcja TRANSLATE-TO-SAT konwertuje problem planowania do formuły w postaci normalnej koniunkcyjnej.

SAT-SOLVER: DPLL lub WalkSAT

SATPlan – Przykład

Ograniczony problem planowania (P, n):

- P jest problemem planowania, a n jest liczbą naturalną
- każdy plan dla P o długości n jest rozwiązaniem dla (P, n)

Wykonuj poszukiwanie iteracyjnie pogłębiane dla $n = 0, 1, 2, \ldots$:

- Zakoduj (P, n) jako problem spełnialności zbioru formuł Φ
- Jeśli Φ jest spełnialny, to wyekstrahuj plan dla P z wartościowania spełniającego Φ

Język:

- robot r
- sąsiednie pokoje p_1, p_2
- akcja przemieszczenia robota

SATPlan – Przykład

Zakodowanie (P, n) dla $n = 1$

Stan początkowy: $\{poz(r, p_1)\}$

- Zakodowanie: $poz(r, p_1, 0) \wedge poz(r, p_2, 0)$
- Ignoruj: $\neg poz(r, r, 0), \neg poz(l_1, l_2, 0)$

Cel: $\{poz(r, p_2)\}$

- Zakodowanie: $poz(r, p_2, 1)$

SATPlan – Przykład

Zakodowanie (P, n) dla $n = 1$

Akcja: $poz(O, X) \xrightarrow{przemiesc(O,X,Y)} \neg poz(O, X), poz(O, Y)$

- Zakodowanie:
  $przemiesc(r, p_1, p_2, 0) \implies poz(r, p_1, 0) \wedge poz(r, p_2, 1) \wedge \neg poz(r, p_1, 1)$
  $przemiesc(r, p_2, p_1, 0) \implies poz(r, p_2, 0) \wedge poz(r, p_1, 1) \wedge \neg poz(r, p_2, 1)$
- Ignoruj:
  $przemiesc(r, p_1, p_1, 0) \implies poz(r, p_1, 0) \wedge poz(r, p_1, 1) \wedge \neg poz(r, p_1, 1)$
  $przemiesc(p_1, \_, \_, 0) \implies \ldots$
  $przemiesc(\_, r, \_, 0) \implies \ldots$

SATPlan – Przykład

Zakodowanie (P, n) dla $n = 1$

Wzajemne wykluczanie akcji:
$\neg przemiesc(r, p_1, p_2, 0) \vee przemiesc(r, p_2, p_1, 0)$

Aksjomaty tła (zmiany są wyłącznie wynikiem wykonania akcji):
$\neg poz(r, p_1, 0) \wedge poz(r, p_1, 1) \implies przemiesc(r, p_2, p_1, 0)$
$\neg poz(r, p_2, 0) \wedge poz(r, p_2, 1) \implies przemiesc(r, p_1, p_2, 0)$
$poz(r, p_1, 0) \wedge \neg poz(r, p_1, 1) \implies przemiesc(r, p_1, p_2, 0)$
$poz(r, p_2, 0) \wedge \neg poz(r, p_2, 1) \implies przemiesc(r, p_2, p_1, 0)$

SATPlan – Przykład

Ekstrakcja planu dla P

Załóżmy, że:

- (P, n) jest zakodowany przez zbiór formuł Φ
- Φ jest spełniony przez wartościowanie V

Wtedy i-ty krok planu dla (P, n) zawiera akcje a_i takie, że

$$V(a_i) = true$$

co oznacza, że została wybrana akcja a_i.

Dziękuję za uwagę!