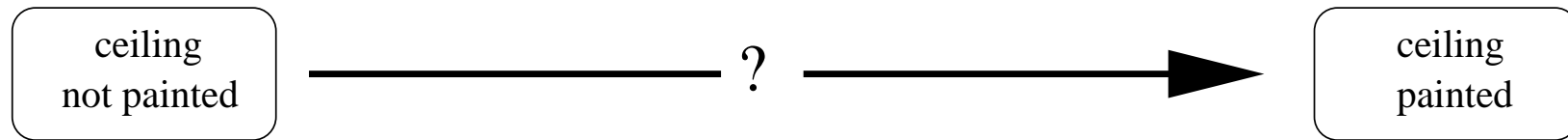


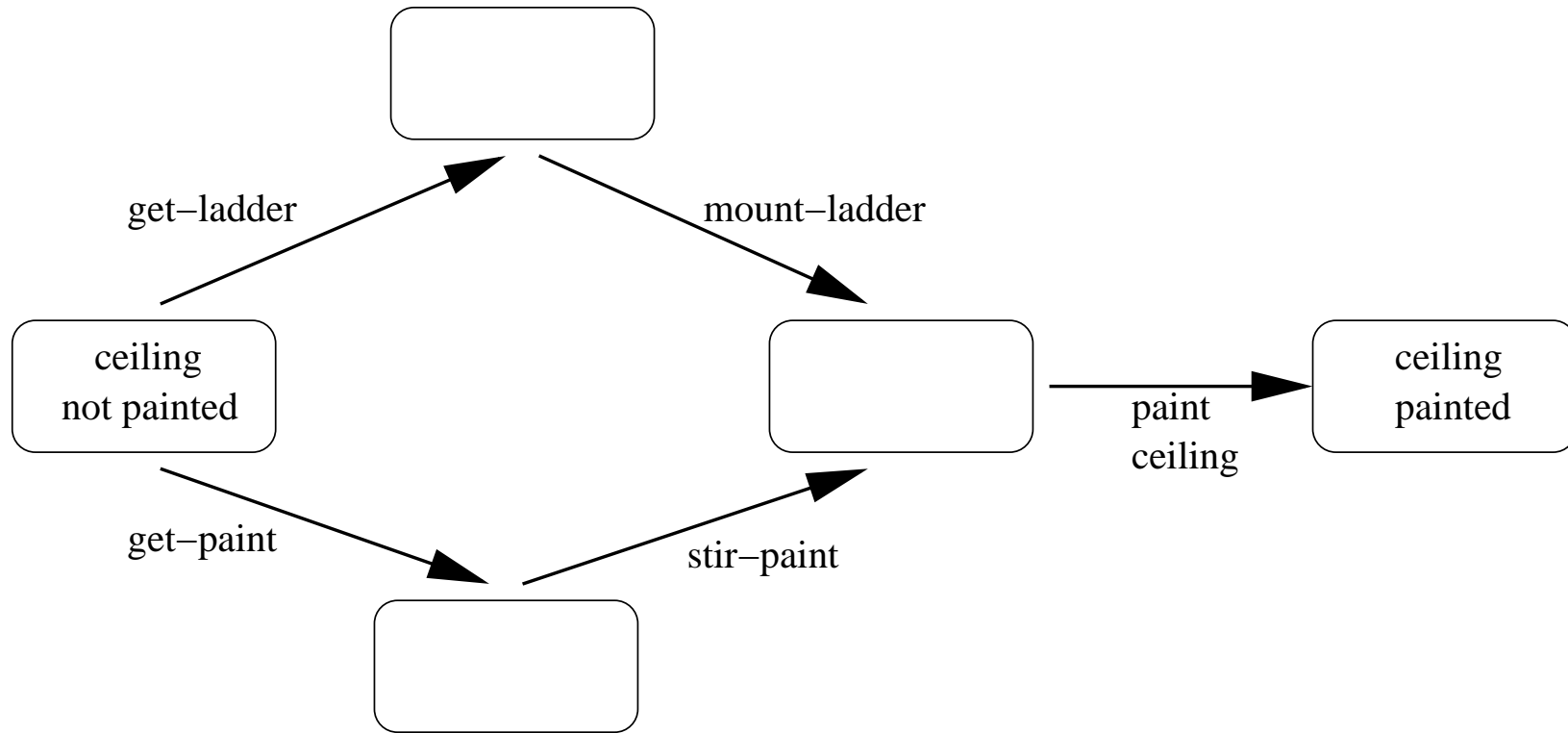
SZTUCZNA INTELIGENCJA I SYSTEMY DORADCZE

PLANOWANIE

Problem planowania



Problem planowania



Jezyk STRIPS: stany

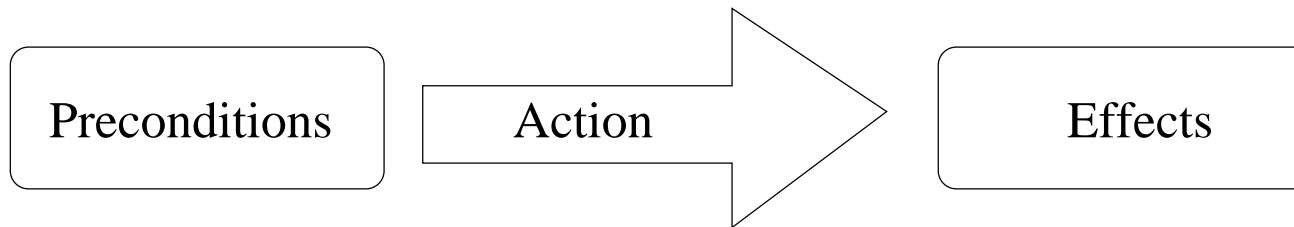
Reprezentują chwilowe stany opisywanego świata,
każdy stan opisany jest jako koniunkcja literałów relacyjnych,
literały są zawsze **ustalone** i **bez wyrażeń funkcyjnych**,
w opisie występują tylko literały pozytywne,
o pozostałych zakłada się, że są nieprawdziwe

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

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

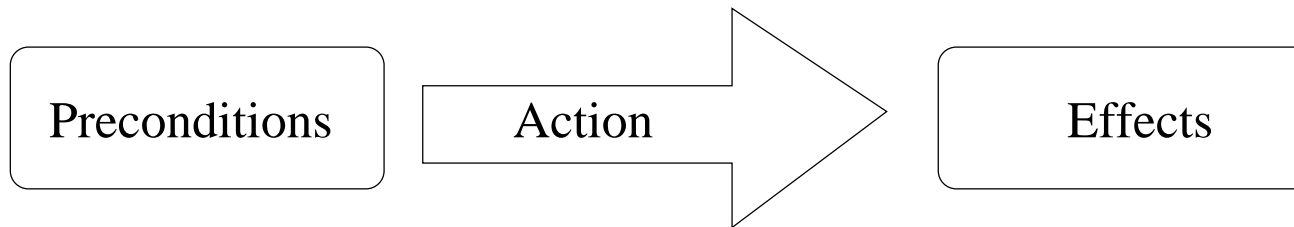
Jezyk STRIPS: akcje

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



Jezyk 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 wyr. funkcyjnych

$At(x, Trunk)$

$Remove(x, Trunk)$

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

Jezyk 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)

Start

At(Spare,Trunk)
At(Flat,Axle)

At(Spare,Axle)

Finish

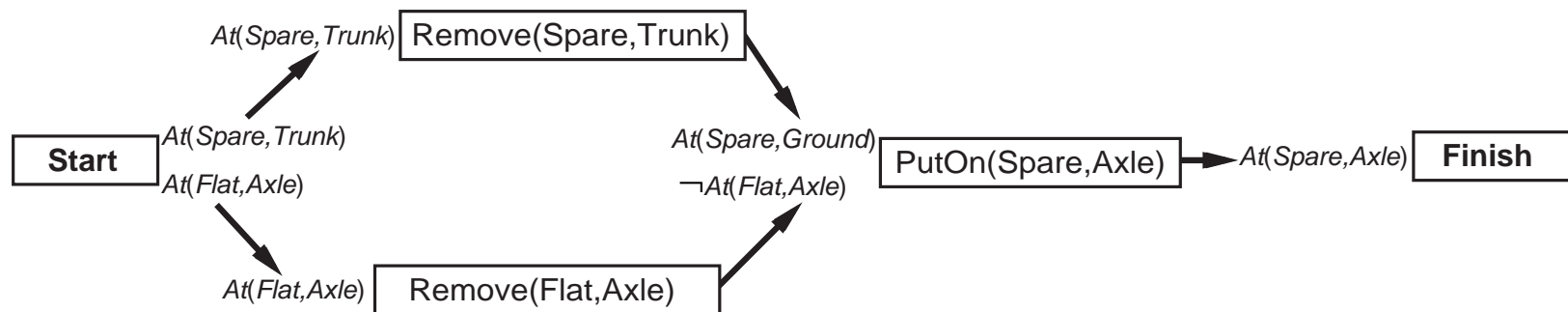
Uwaga: cel nie jest stanem — wiele stanów może spełniać warunki docelowe

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



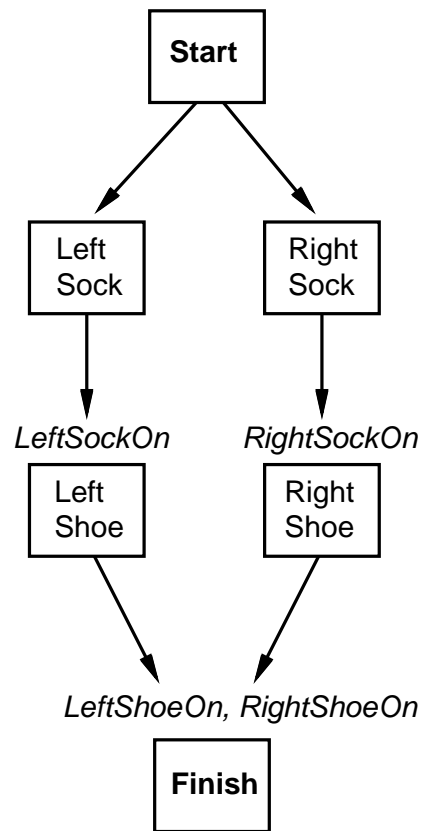
Poprawność planu polega na tym, że dla każdej akcji stan poprzedzający tą akcją spełnia ukonkretnione warunki wstępne tej akcji

Po zastosowaniu akcji nowy stan powstaje poprzez dodanie pozytywnych oraz usunięcie negatywnych efektów akcji ze stanu poprzedzającego tą akcją

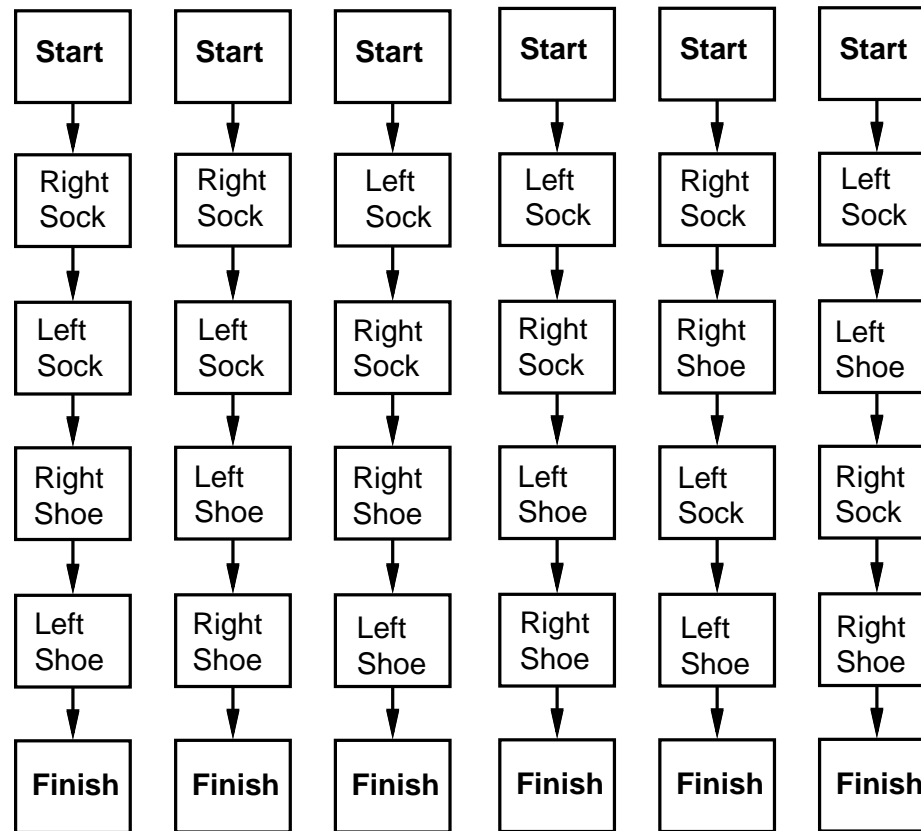
Plan jako rozwiązanie problemu

W praktyce poszukuje się liniowo uporządkowanych planów

Partial-Order Plan:



Total-Order Plans:



Redukcja przestrzeni planow

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

Uwaga: linearyzacja i ochrona są w praktyce skuteczne, ale mogą spowodować wykluczenie poprawnych planów, w szczególności mogą wykluczyć wszystkie poprawne plany i uniemożliwić znalezienie rozwiązania

Metody poszukiwania poprawnych planow

- ◇ Przeszukiwanie wprzód
 - ForwardChaining

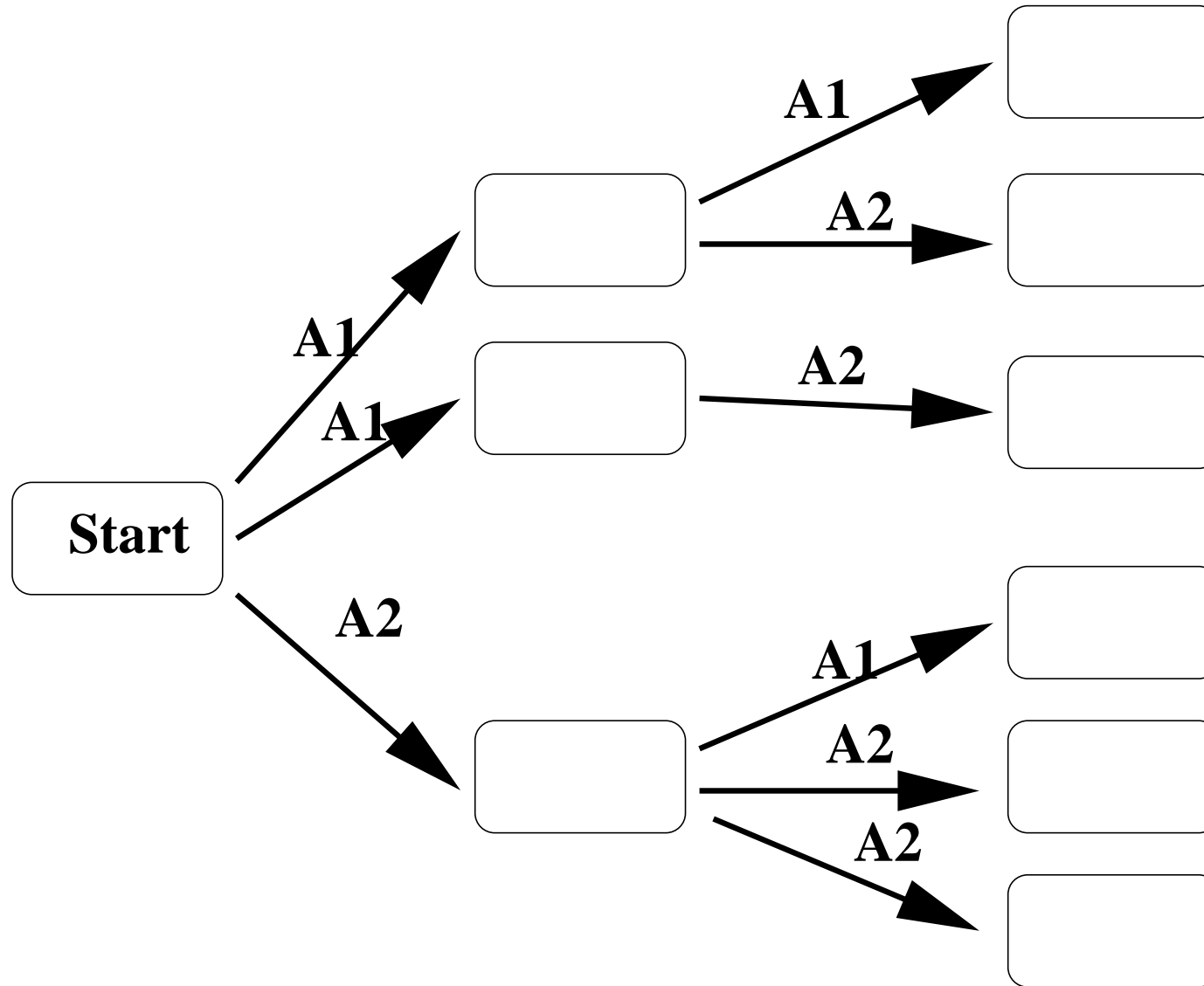
- ◇ Przeszukiwanie wstecz
 - PopPlan
 - GraphPlan

- ◇ Sprowadzenie do problemu spełnialności
 - SatPlan

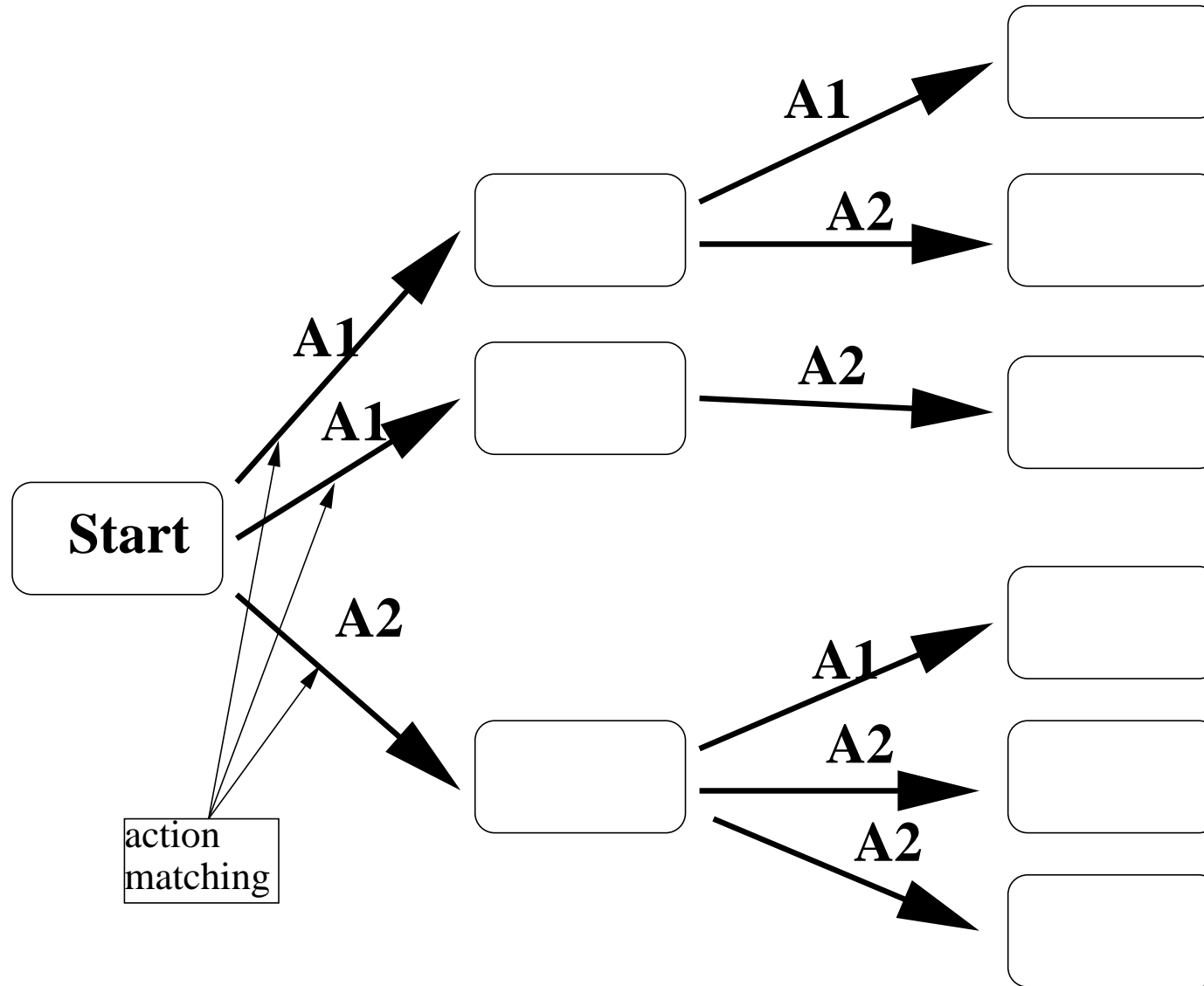
ForwardChaining

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

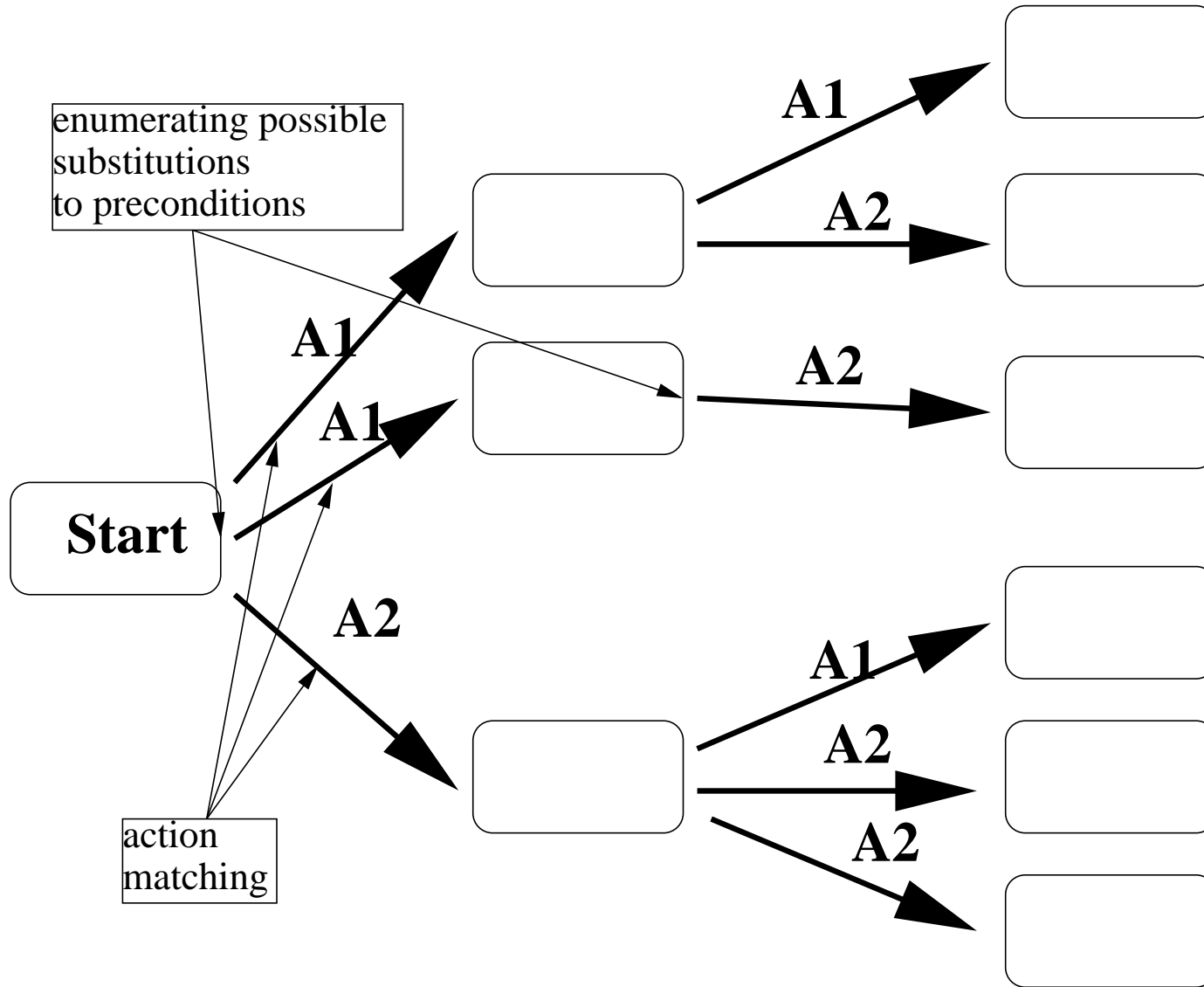
ForwardChaining: algorytm



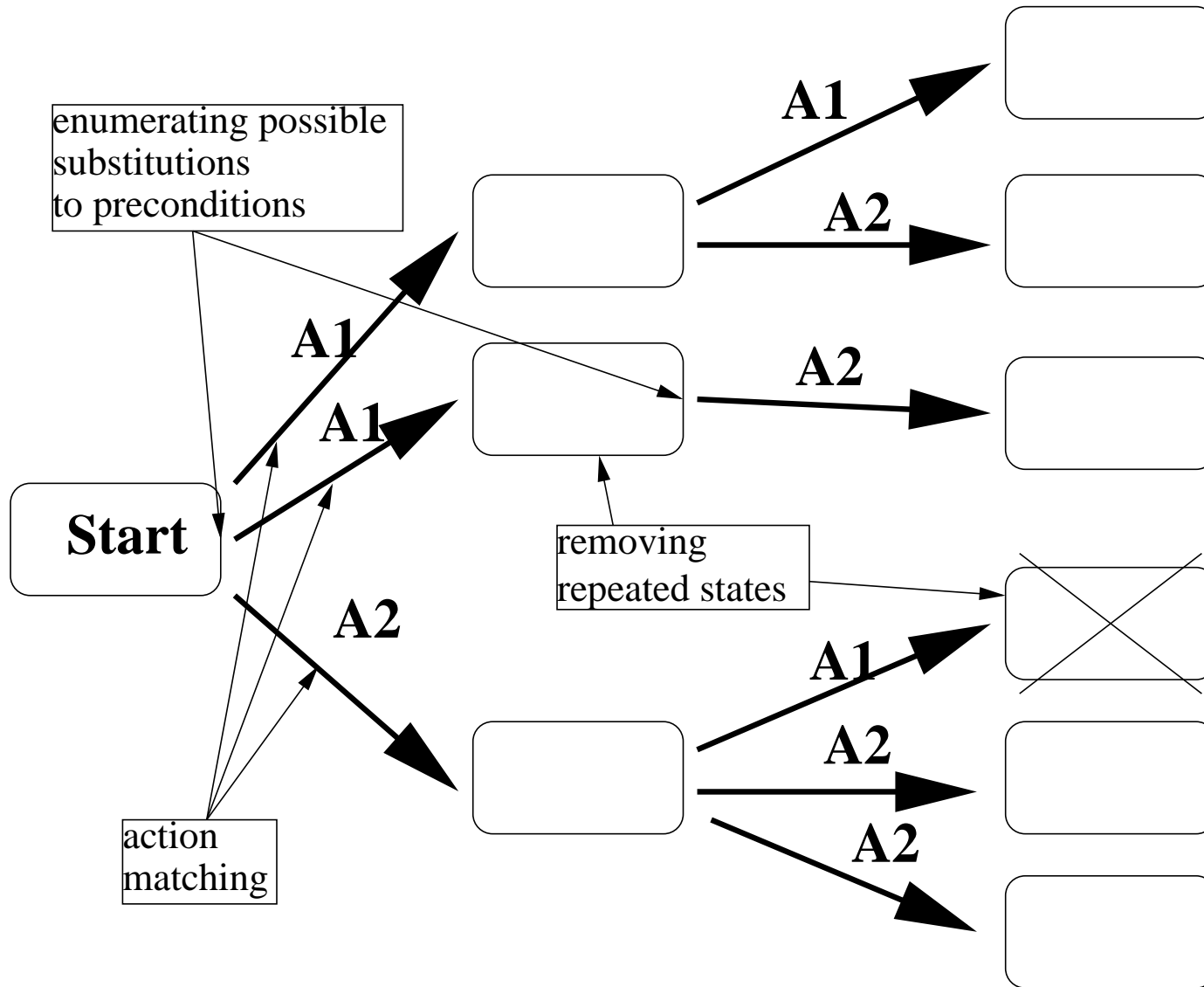
ForwardChaining: algorytm



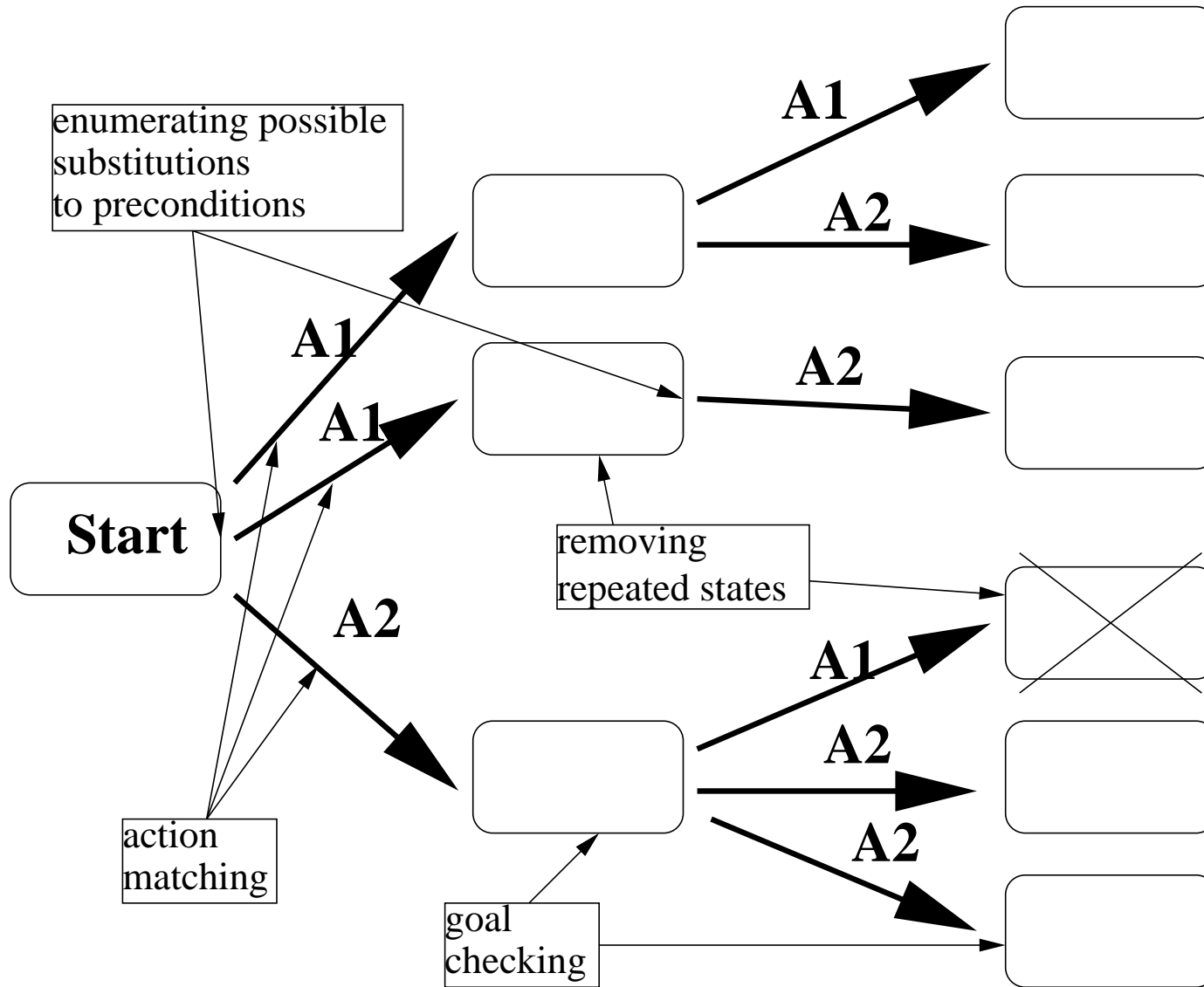
ForwardChaining: algorytm



ForwardChaining: algorytm



ForwardChaining: algorytm



ForwardChaining: efektywna implementacja

- ◇ 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

Polaczenia przyczynowe i wiezy porzadkujace

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** wtw kiedy każdy warunek wstępny jest osiągnięty

Warunek wstępny jest **osiągnięty** wtw jeśli jest efektem wcześniejszej akcji i żadna **akcja przeszkadzająca** nie eliminuje tego efektu

Polaczenia przyczynowe i wiezy porzadkujace

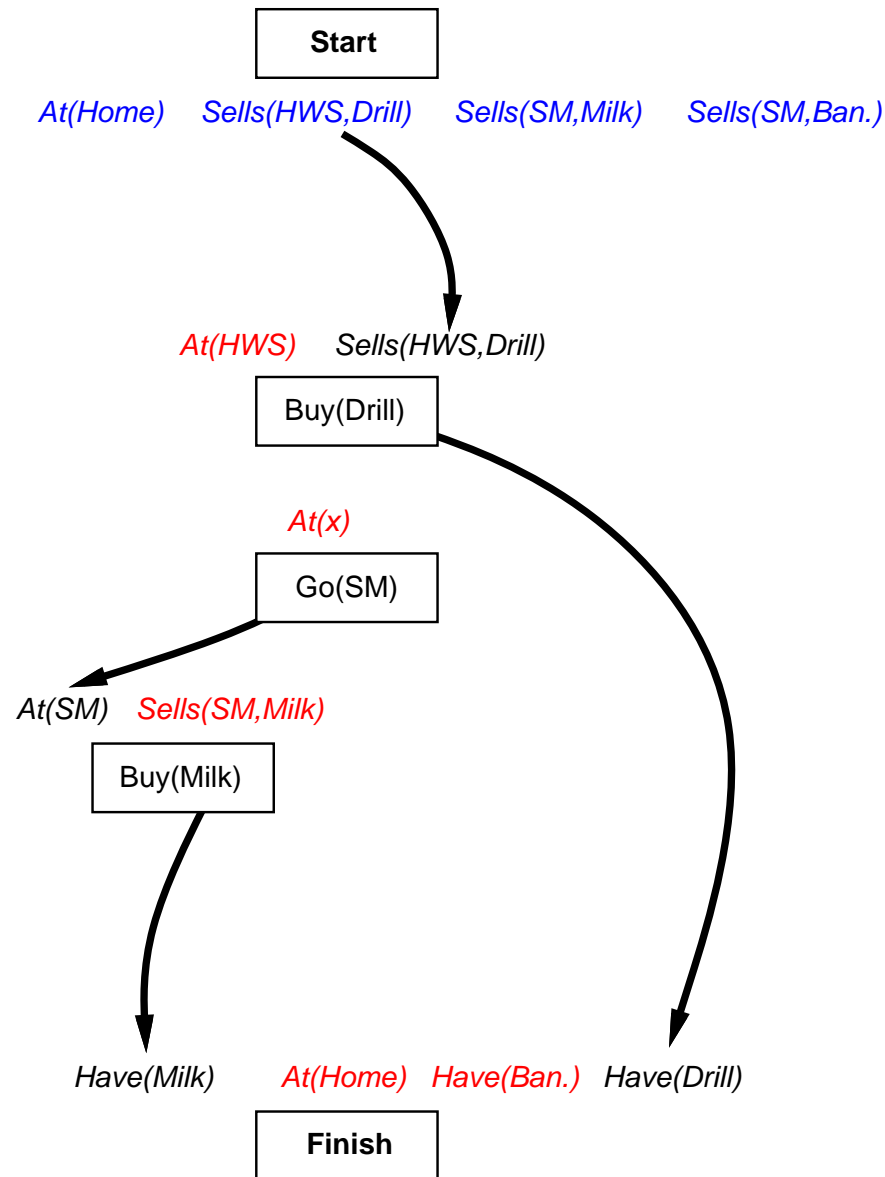
Start

At(Home) Sells(HWS,Drill) Sells(SM,Milk) Sells(SM,Ban.)

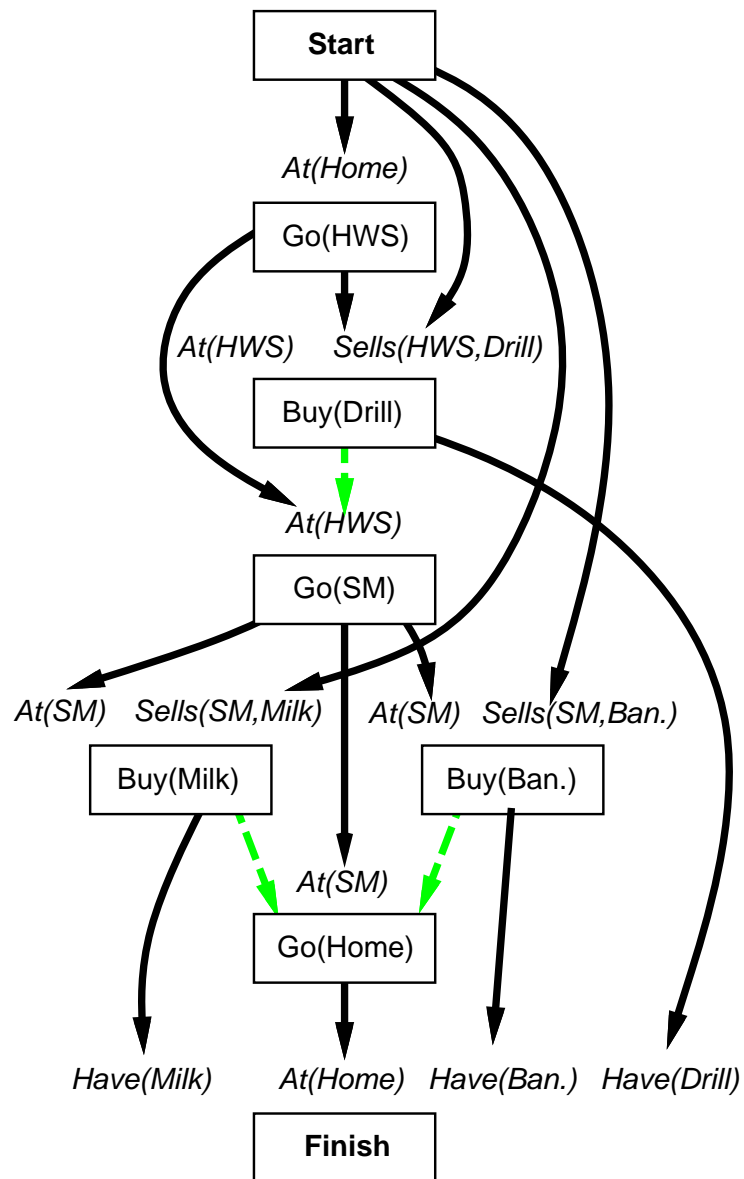
Have(Milk) At(Home) Have(Ban.) Have(Drill)

Finish

Polaczenia przyczynowe i wiezy porzadkujace

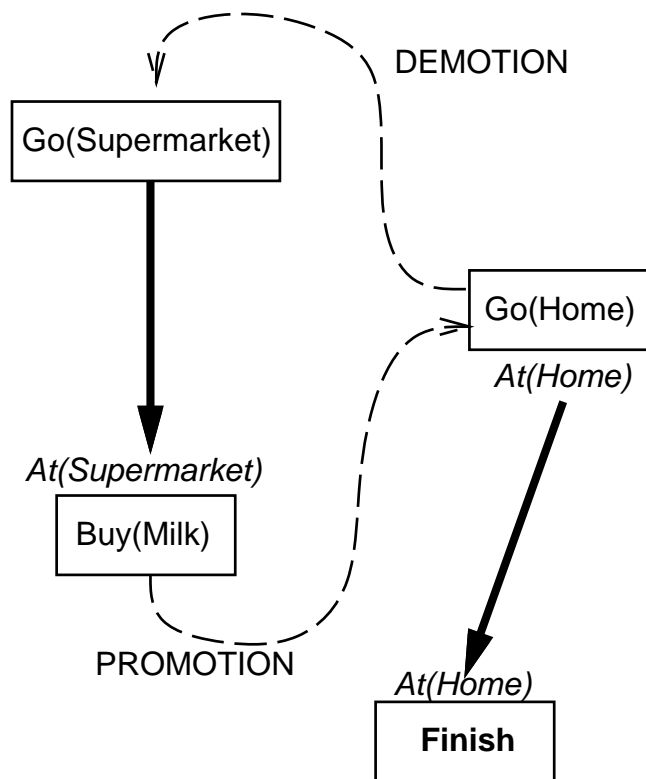


Polaczenia przyczynowe i wiezy porzadkujace



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 do

if SOLUTION?(*plan*) **then return** *plan*

S_{need}, c ← SELECT-SUBGOAL(*plan*)

 CHOOSE-OPERATOR(*plan, operators, S_{need}, c*)

 RESOLVE-THREATS(*plan*)

end

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

PopPlan: algorytm

procedure 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**

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 $ORDERINGS(plan)$

procedure 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 $ORDERINGS(plan)$

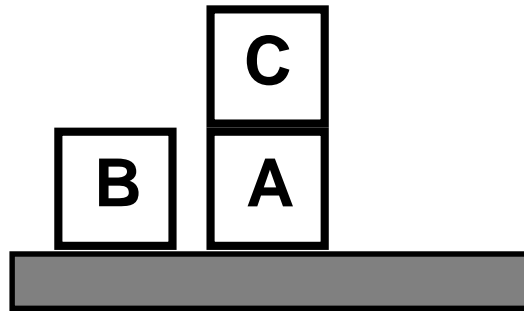
Promotion: Add $S_j \prec S_{threat}$ to $ORDERINGS(plan)$

if not CONSISTENT($plan$) **then fail**

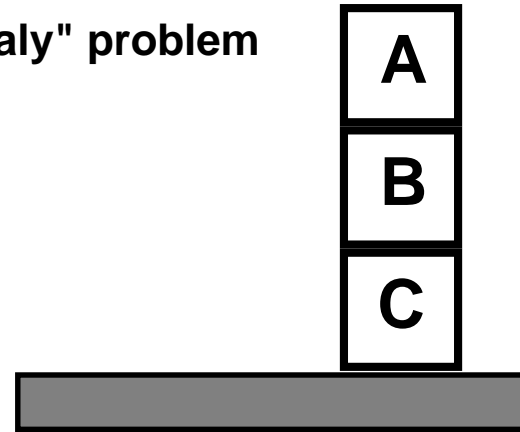
end

PopPlan: przykład

"Sussman anomaly" problem



Start State



Goal State

$Clear(x) \ On(x,z) \ Clear(y)$

PutOn(x,y)

$\sim On(x,z) \ \sim Clear(y)$
 $Clear(z) \ On(x,y)$

$Clear(x) \ On(x,z)$

PutOnTable(x)

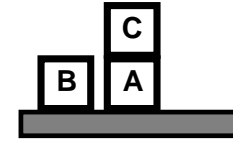
$\sim On(x,z) \ Clear(z) \ On(x, Table)$

+ several inequality constraints

PopPlan: przykład

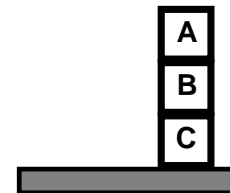
START

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

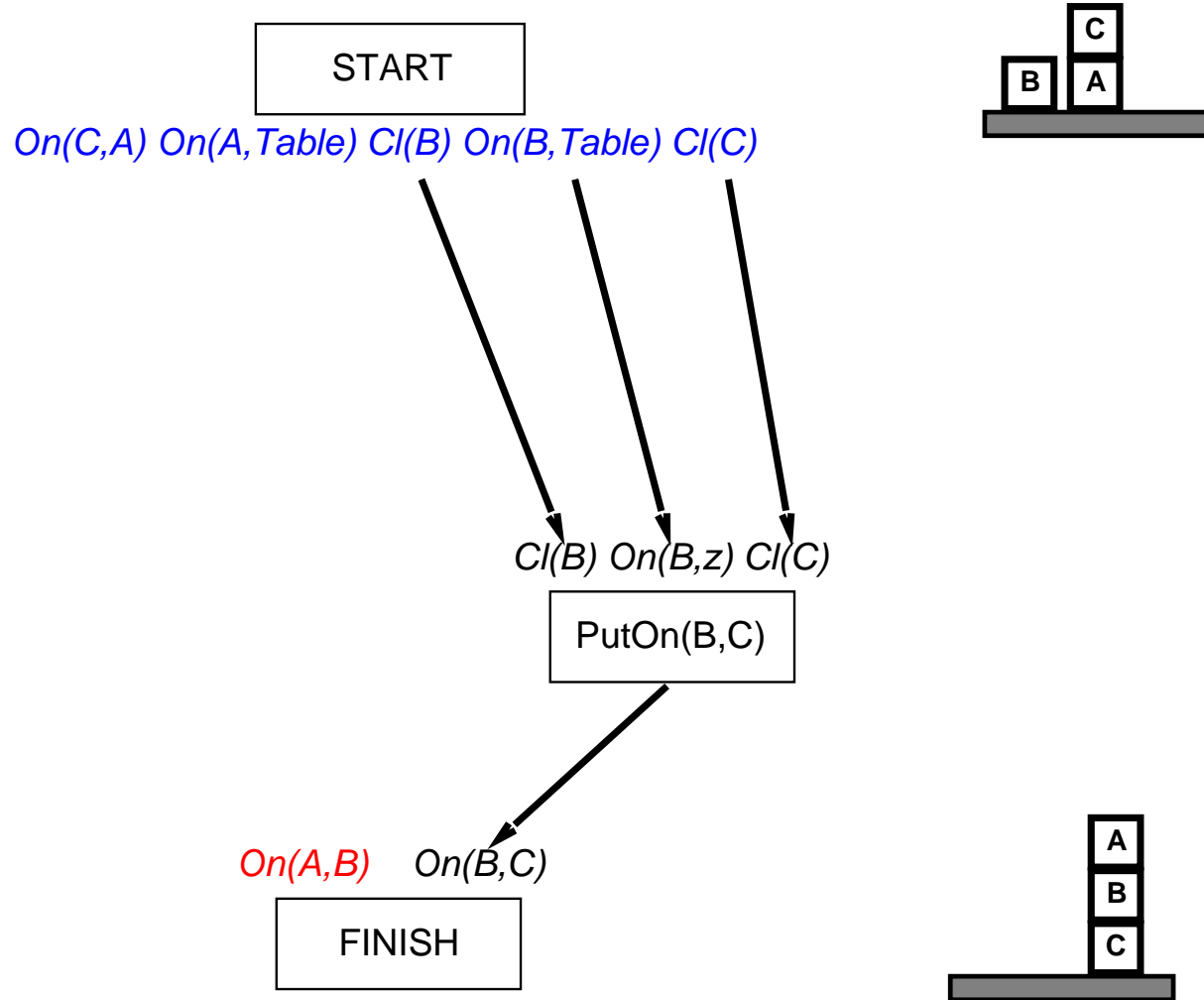


On(A,B) On(B,C)

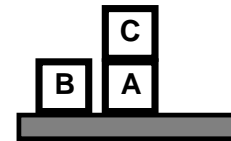
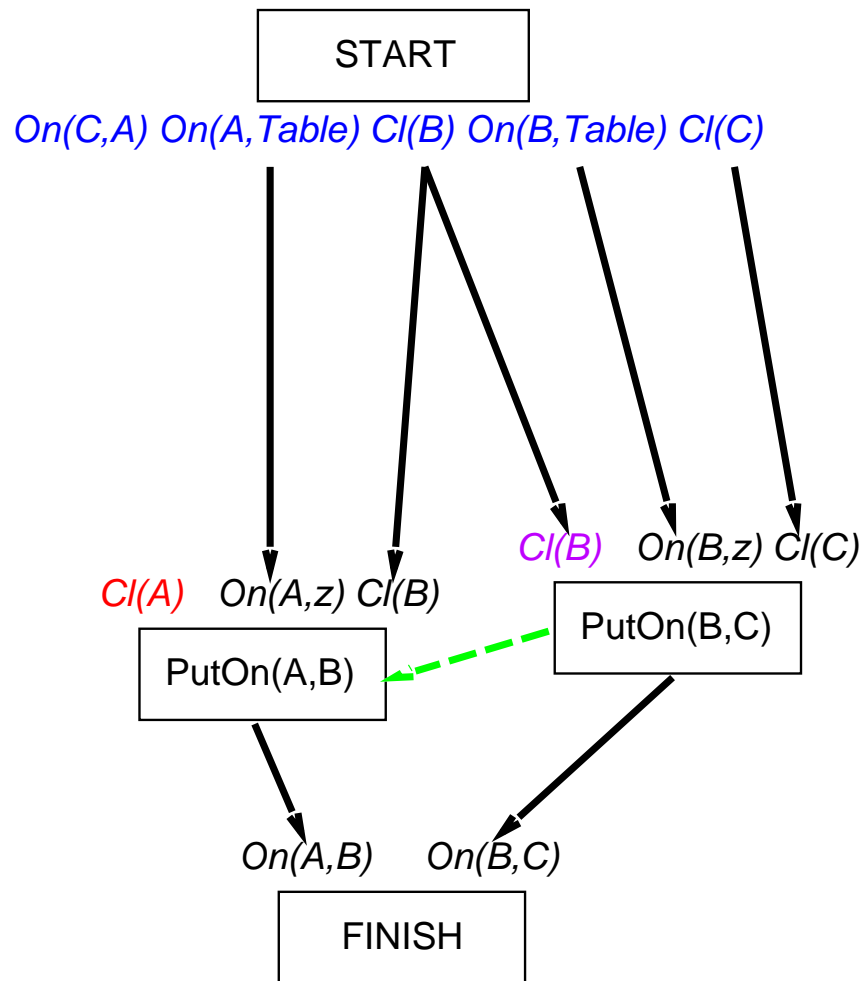
FINISH



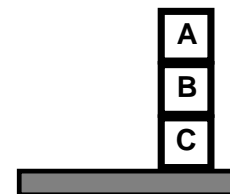
PopPlan: przykład



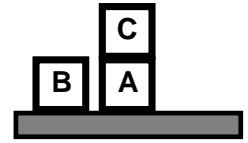
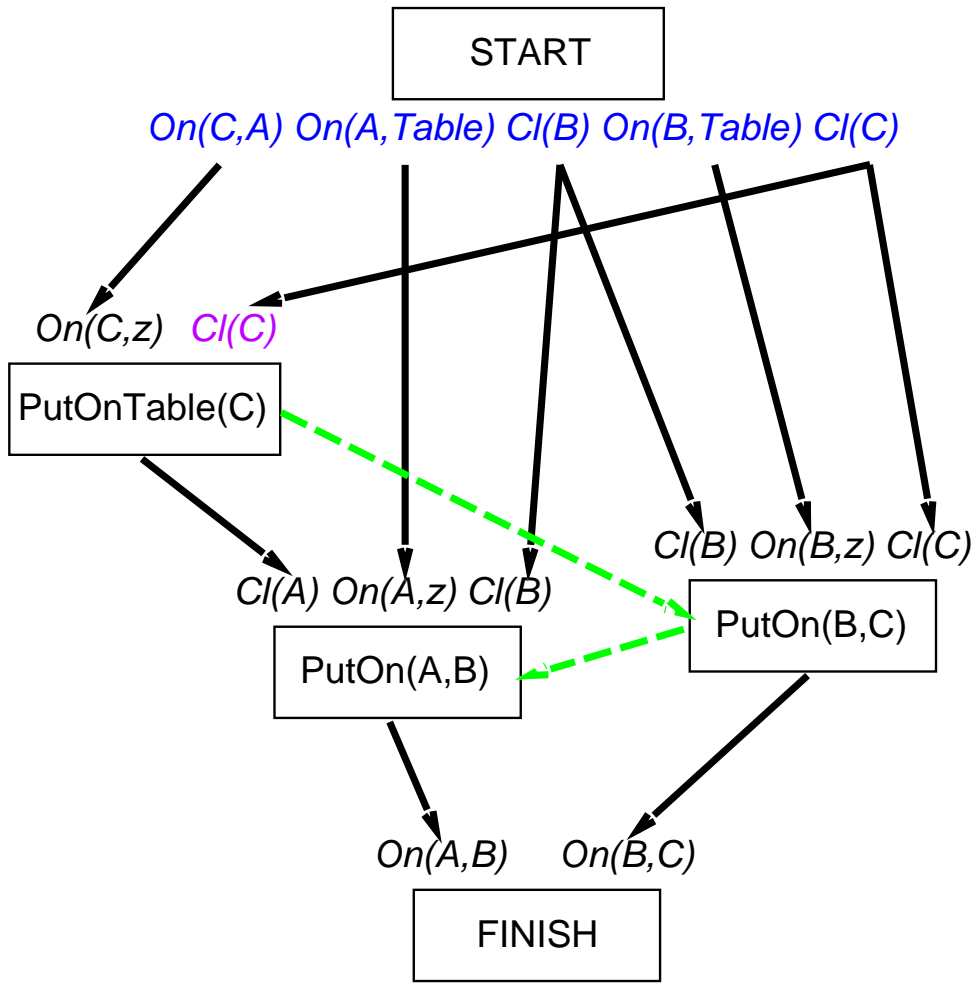
PopPlan: przykład



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

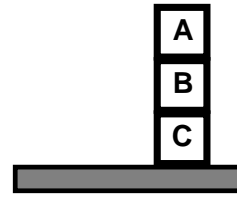


PopPlan: przykład



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 powiązаныmi 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, co 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:

— $Eat(Cake)$

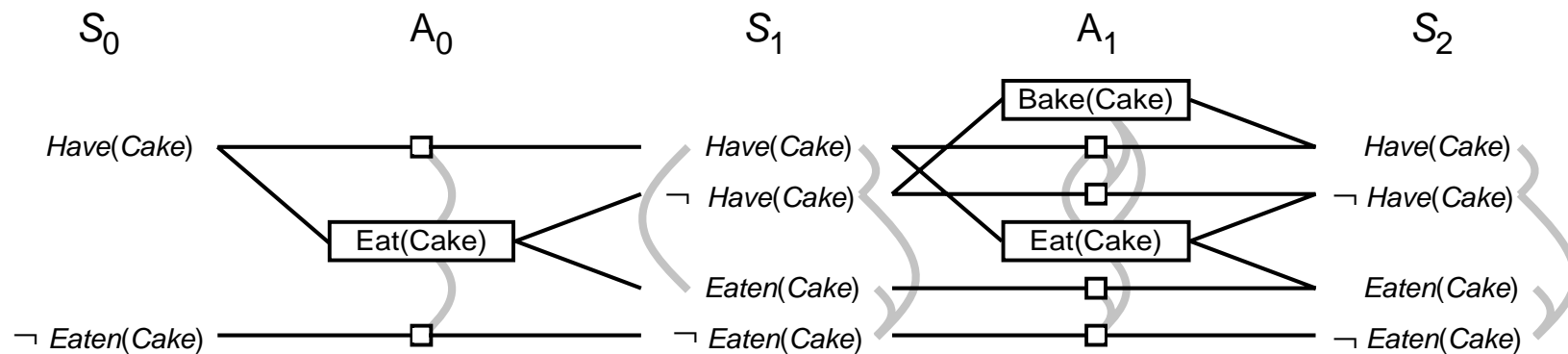
warunki wstępne: $Have(Cake)$

efekty: $\neg Have(Cake) \wedge Eaten(Cake)$

— $Bake(Cake)$

warunki wstępne: $\neg Have(Cake)$

efekty: $Have(Cake)$



Relacja wzajemnej wylacznosci (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

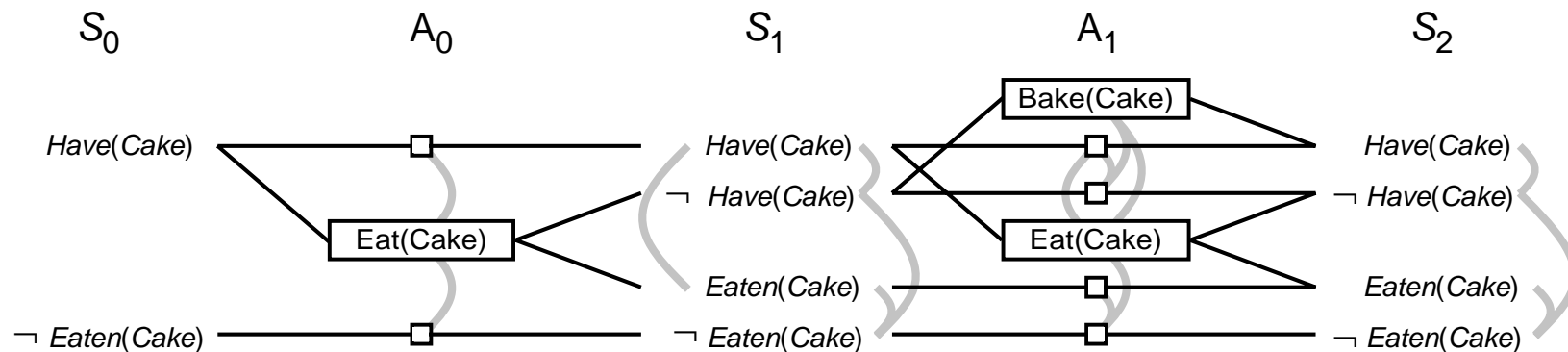
Relacja mutex: przykład

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 do
    if goals all non-mutex in last level of graph then do
      solution ← EXTRACT-SOLUTION(graph, goals, LENGTH(graph))
      if solution ≠ failure then return solution
      else if NO-SOLUTION-POSSIBLE(graph) then return failure
    graph ← EXPAND-GRAPH(graph, problem)
```

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

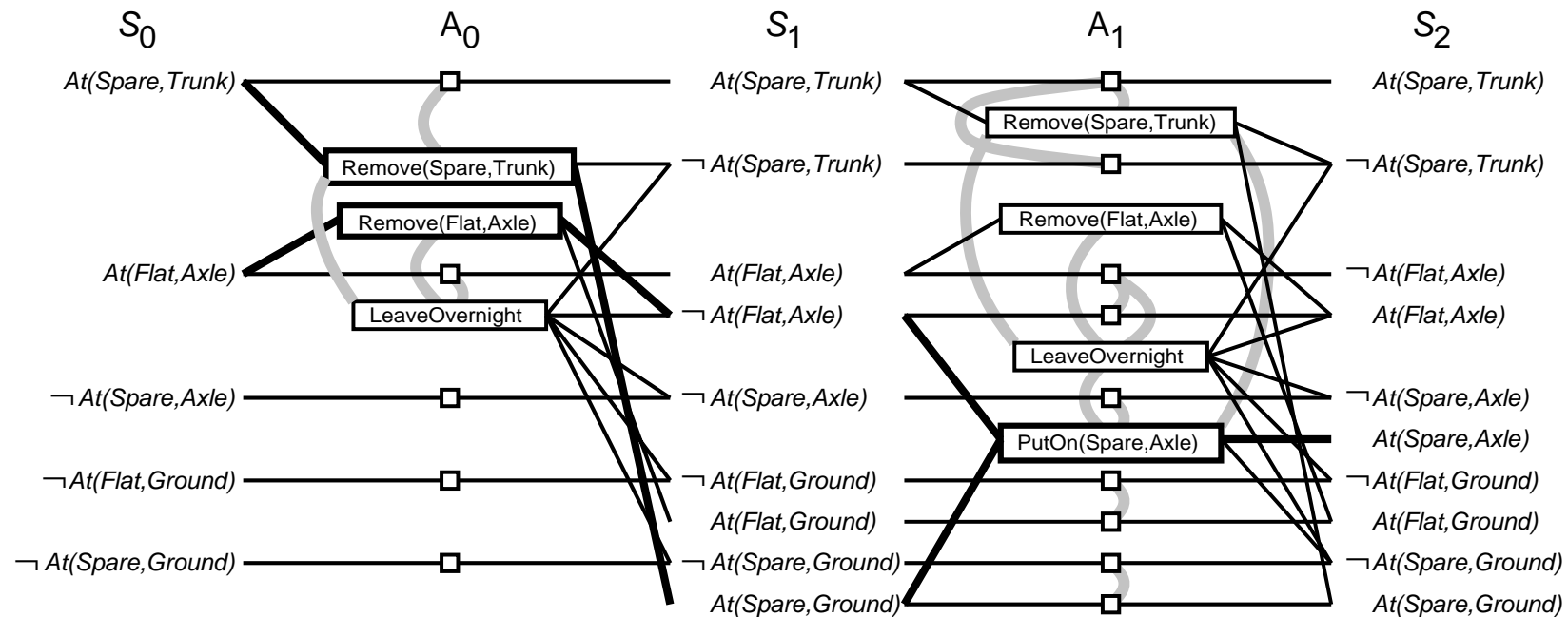
GraphPlan: szukanie rozwiązania

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: przykład

Cel: $At(Spare, Axle)$



W powyższym przykładzie zostały zaznaczone tylko niektóre relacje mutex istotne dla przykładu

GraphPlan: wybór celów i akcji

Koszt poziomy 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 poziomego
- do osiągnięcia danego celu wybierana jest akcja z najmniejszą sumą (lub maksimum) kosztów poziomych warunków wstępnych akcji

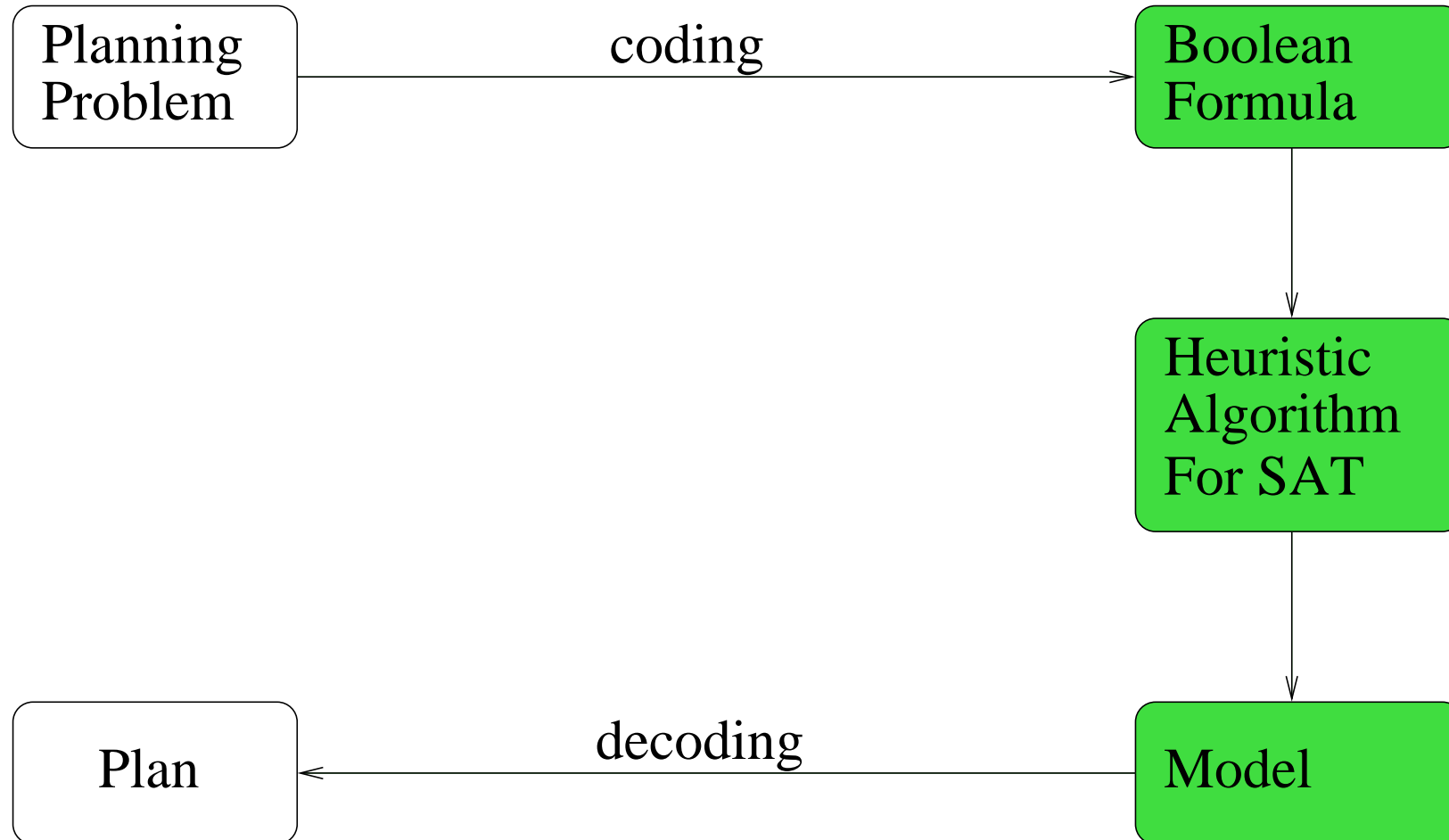
GraphPlan: własności

Twierdzenie

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

SATPlan

Planowanie przez sprowadzenie do problemu spełnialności w logice zdaniowej



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  $\leftarrow$  TRANSLATE-TO-SAT(problem,  $T$ )
    assignment  $\leftarrow$  SAT-SOLVER(cnf)
    if assignment is not null then
      return EXTRACT-SOLUTION(assignment, mapping)
  return failure
```

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

SAT-SOLVER: DPLL lub WalkSAT