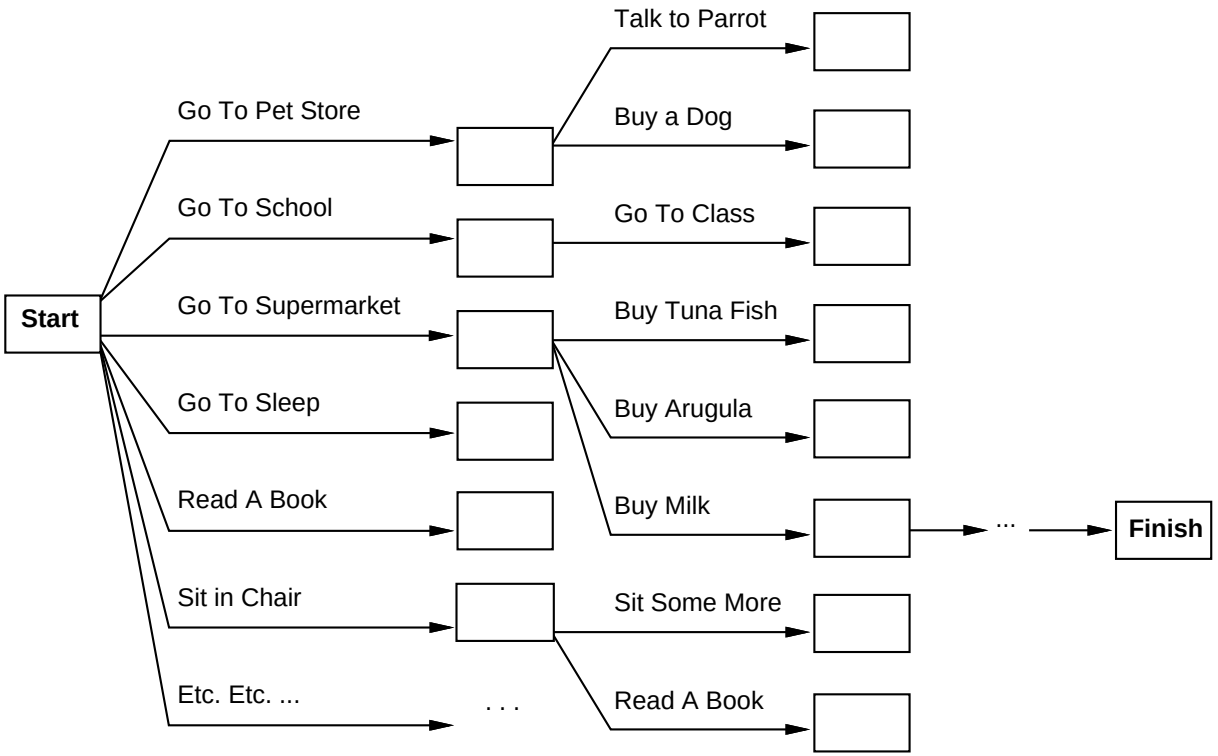
Planowanie

Każde zadanie znalezienia planu można teoretycznie rozwiązać posługując się jedną z metod przeszukiwania przestrzeni stanów. Jest to jednak często niewykonalne w praktyce, ze względu na skomplikowany opis stanów i wysoki współczynnik rozgałęzienia.

Przykład

Skonstruować plan gwarantujący posiadanie szklanki mleka, kilku bananów i wiertarki.

Mały fragment przestrzeni stanów dla tego zadania:



Systemy planujące są znacznie bardziej wyrafinowane od systemów realizujących przeszukiwanie przestrzeni stanów.

Podstawowe cechy systemów planujących

- Wykorzystanie języka formalnego (na ogół logiki) do opisu przestrzeni stanów, celu i akcji. (Zwiększenie ekspresji.)
- Konstruowanie planu metodą zstępującą. Zaczynamy od planu przybliżonego, który później ulepszamy. (Zwiększenie efektywności.)
- Wykorzystanie metody "dziel i zwyciężaj" przy konstruowaniu planu. Typowe systemy planujące dzielą cel na podcele, które próbują kolejno zrealizować. (Zwiększenie efektywności.)

Rachunek sytuacyjny jest bardzo eleganckim i ekspresyjnym formalizmem planującym. Niestety, z praktycznego punktu widzenia, jest nieefektywny.

Planowanie efektywne wymaga:

(1) Użycia podzbioru logiki pierwszego rzędu jako języka formalizacji.

(2) Użycia wyspecjalizowanych algorytmów do planowania zamiast udowadniacza twierdzeń ogólnego przeznaczenia typu metody rezolucji.

Większość systemów planujących wywodzi się z systemu STRIPS (STanford Research Institute Problem Solver, 1970).

Reprezentacja stanów, celu i akcji w systemach typu STRIPS

Stany reprezentowane są jako koniunkcja literałów nie zawierających zmiennych:

$$At(Home) \wedge \neg Have(Milk) \wedge \cdots$$

Cel jest dowolną koniunkcją literałów:

$$At(home) \wedge Have(Milk) \wedge \cdots$$

Formuła celu może zawierać zmienne:

$$At(x) \wedge Sells(x, Milk).$$

Zakłada się, że zmienne występujące w formule celu są egzystencjalnie kwantyfikowane.

Opis akcji

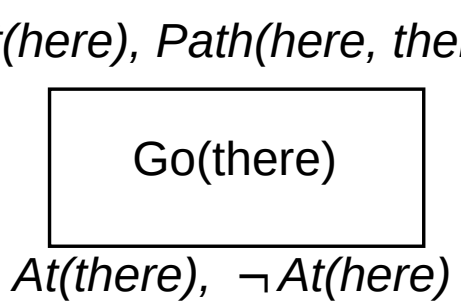
- **Nazwa akcji**.
- **Warunek wstępny (precondition)** – koniunkcja literałów dodatnich, które muszą zachodzić, aby akcja była wykonywalna.
- **Efekty akcji** – Koniunkcja literałów opisujących zmiany stanu w wyniku wykonania akcji.

Formalnie:

$Op(ACTION : Go(there),$

$PRECOND : At(here) \wedge Path(here, there),$

$EFFECT : At(there) \wedge \neg At(here)).$



Uwaga: Powyższy opis definiuje całą klasę akcji.

Stany będziemy często zapisywali jako zbiory literałów.

Akcja jest **stosowalna** w stanie s jeśli wszystkie warunki wstępne tej akcji są zawarte w s. W wyniku wykonania akcji A w stanie s otrzymujemy stan, który:

(1) Zawiera wszystkie literały występujące w koniunkcji efektów akcji.

(2) Zawiera wszystkie literały występujące w s, za wyjątkiem tych, których negacje występują w koniunkcji efektów akcji.

Jeśli stan s zawiera

$$At(Home), Path(Home, Supermarket), \ldots$$

to akcja $Go(Supermarket)$ jest stosowalna i w wyniku wykonania tej akcji otrzymamy:

$$\neg At(Home), At(Supermarket), Path(Home, Supermarket), \ldots$$

System planujący może być oparty na przeszukiwaniu przestrzeni stanów lub przeszukiwaniu **przestrzeni planów**. To drugie rozwiązanie jest efektywniejsze.

Jak reprezentować plany?

Cel: $RightShoeOn \wedge LeftShoeOn$.

Stan początkowy: $True$.

Akcje:

Op(ACTION: $RightShoe$,
PRECOND: $RightSockOn$,
EFFECT: $RightShoeOn$).

Op(ACTION: $LeftShoe$,
PRECOND: $LeftSockOn$,
EFFECT: $LeftShoeOn$).

Op(ACTION: $RightSock$,
PRECOND: $True$,
EFFECT: $RightSockOn$).

Op(ACTION: $LeftSock$,
PRECOND: $True$,
EFFECT: $LeftSockOn$).

Plan częściowy— częściowo uporządkowany zbiór schematów akcji.

Plan liniowy — liniowo uporządkowany zbiór schematów akcji.

Plan ukonkretniony (częściowy, liniowy) — częściowo (liniowo) uporządkowany zbiór akcji.

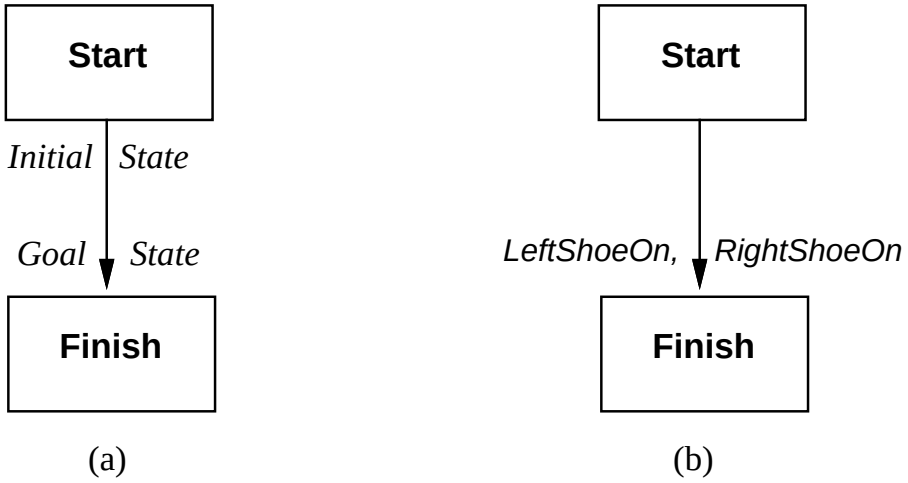
Linearyzacja planu częściowego P — każdy plan liniowy zawierający dokładnie akcje (schematy akcji) planu P, zgodny z częściowym porządkiem określonym na P.

Plan reprezentujemy jako strukturę danych zawierającą:

- Zbiór **operatorów (plan steps)**: akcje i (lub) schematy akcji.
- Zbiór **węzłów porządkujących (ordering constraints)**: wyrażenia postaci $S_i \prec S_j$, gdzie S_i, S_j są operatorami.
- Zbiór **ograniczeń zmiennych (variable constraints)**: wyrażenia postaci $v = x$, gdzie v jest zmienną, natomiast x jest inną zmienną lub stałą.
- Zbiór **powiązań przyczynowych (causal links)**: wyrażenia postaci $S_i \xrightarrow{c} S_j$, interpretowane: operator S_i zapewnia warunek wstępny c dla operatora S_j.

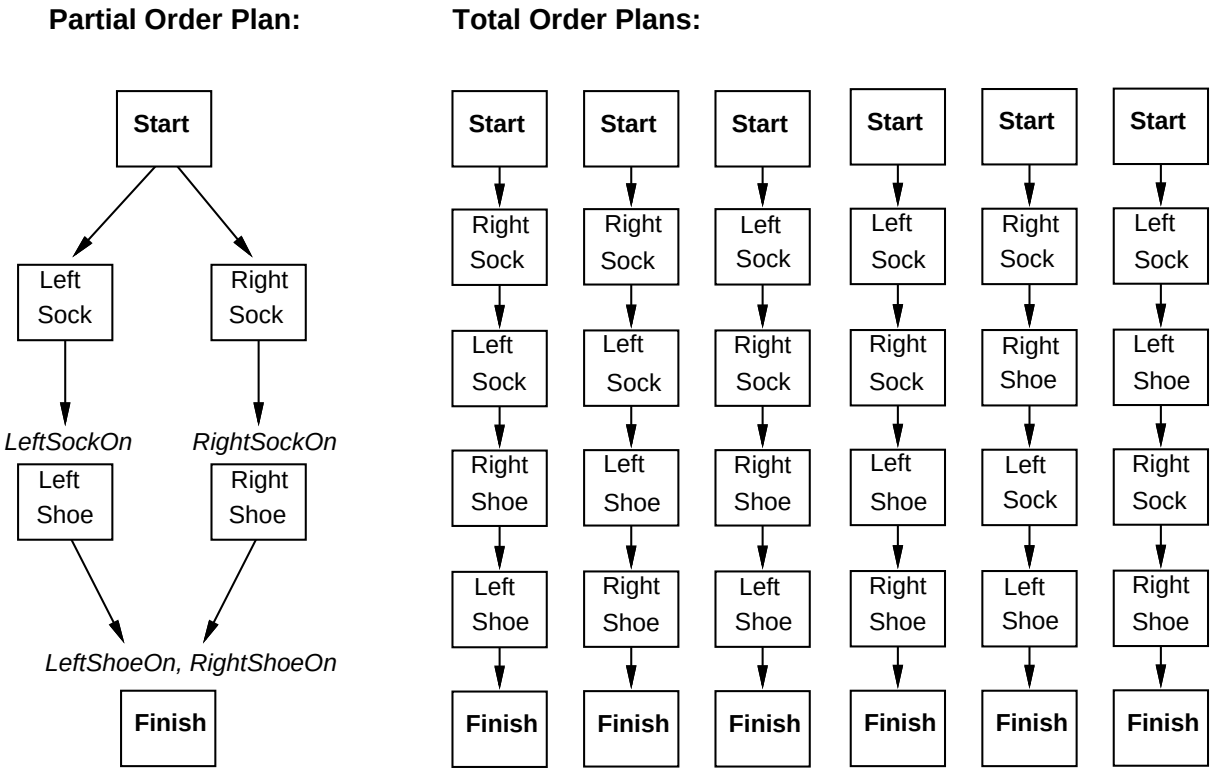
Plan początkowy: składa się z pary operatorów, $Start$ i $Finish$, powiązanych węzłem $Start \prec Finish$. Operator $Start$ nie posiada warunków wstępnych. Jego efektem działania jest realizacja stanu początkowego. Warunkiem wstępnym operatora $Finish$ jest formuła celu. Jego efektem działania jest akcja pusta. Plan początkowy nazywany jest też **planem minimalnym**.

Poniżej: (a) Ogólna struktura planu początkowego. (b) Plan początkowy dla problemu zakładania butów.



(a) (b)

Poniżej: częściowy plan (będący rozwiązaniem problemu zakładania butów) i jego sześć linearyzacji.



Co to jest rozwiązanie problemu planowania (plan końcowy)?

Plan końcowy powinien być ukonkretniony.

Dopuszczamy natomiast plany częściowe jako rozwiązanie zadania planowania, ponieważ:

1. Plan częściowy może posiadać wiele linearyzacji. Nie widać powodu, dlaczego system planujący miałby wybrać jedną z nich jako rozwiązanie.

2. Plan częściowy może być wykonany w trybie równoległym.

3. Plan częściowy można w naturalny sposób wykorzystać jako "podplan" przy rozwiązywaniu większego zadania.

Plan pełny – plan, w którym warunek wstępny każdego operatora jest zapewniony w wyniku realizacji innego operatora. Dokładniej, dla dowolnego warunku wstępnego c dowolnego operatora S_i, istnieje operator S_j taki, że:

1. $S_j \prec S_i$.
2. $c \in$ EFFECTS(S_j).
3. Plan nie zawiera operatora S_k takiego, że $\neg c \in$ EFFECTS(S_k) i $S_j \prec S_k \prec S_i$.

Plan poprawny – plan, w którym zarówno zbiór węzłów porządkujących, jak i zbiór ograniczeń zmiennych są niesprzeczne.

Każdy plan pełny i poprawny jest planem końcowym, czyli rozwiązaniem zadania planowania.

Konstruowanie planu

Ogólny pomysł:

(1) Zaczynamy od planu początkowego.

(2) W kolejnym kroku rozszerzamy plan, zapewniając jeden z warunków wstępnych istniejącego już operatora (albo poprzez dodanie nowego operatora, albo poprzez dodanie połączenia przyczynowego biegnącego od wcześniej dodanego operatora).

(3) Powtarzamy tak długo, aż otrzymamy plan końcowy lub kolejny krok jest niewykonalny.

(4) W drugim przypadku, jeśli możliwe, robimy nawrót.

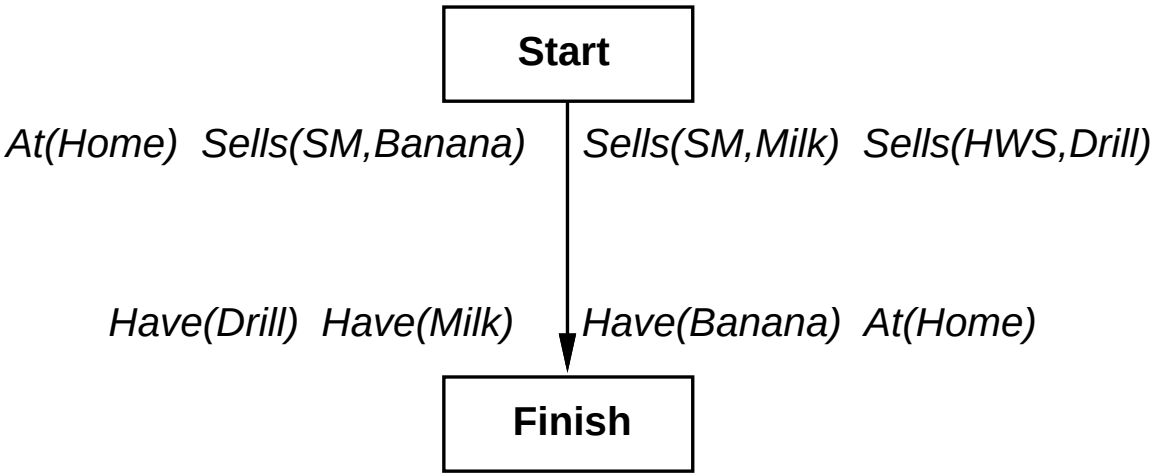
Op(ACTION: $Start$,
EFFECT: $At(Home) \wedge Sells(HWS, Drill)$
$\wedge Sells(SM, Milk) \wedge Sells(SM, Banana)$).

Op(ACTION: $Finish$,
PRECOND: $Have(Drill) \wedge Have(Milk)$
$\wedge Have(Banana) \wedge At(Home)$).

Op(ACTION: $Go(there)$,
PRECOND: $At(here)$,
EFFECT: $At(there) \wedge \neg At(here)$).

Op(ACTION: $Buy(x)$,
PRECOND: $At(store) \wedge Sells(store, x)$,
EFFECT: $Have(x)$).

Plan początkowy:

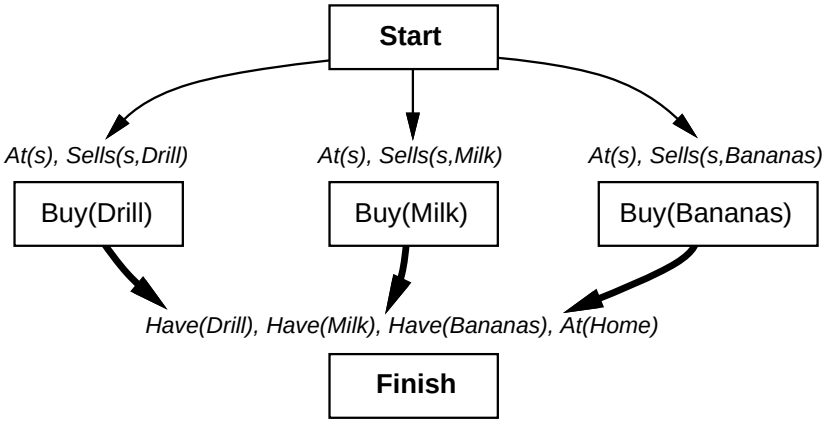


Poniżej:

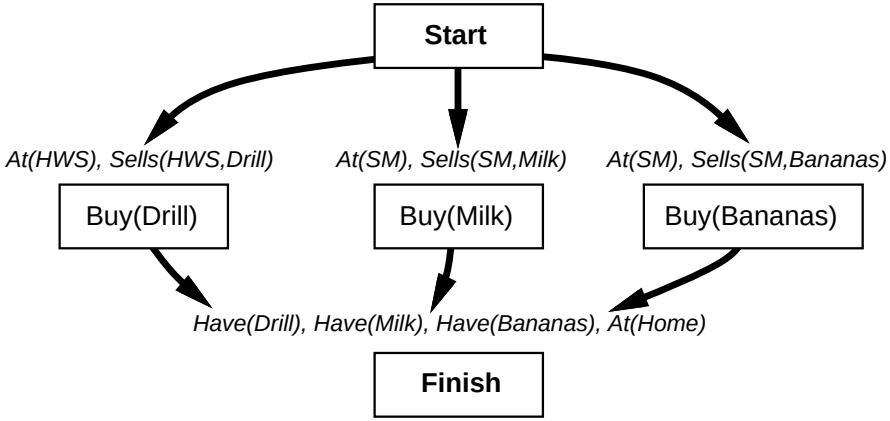
(1) Cienkie strzałki wskazują na porządek pomiędzy operatorami.

(2) Pogrubione strzałki reprezentują połączenia przyczynowe.

(System planujący "chroni" wszystkie połączenia przyczynowe.)

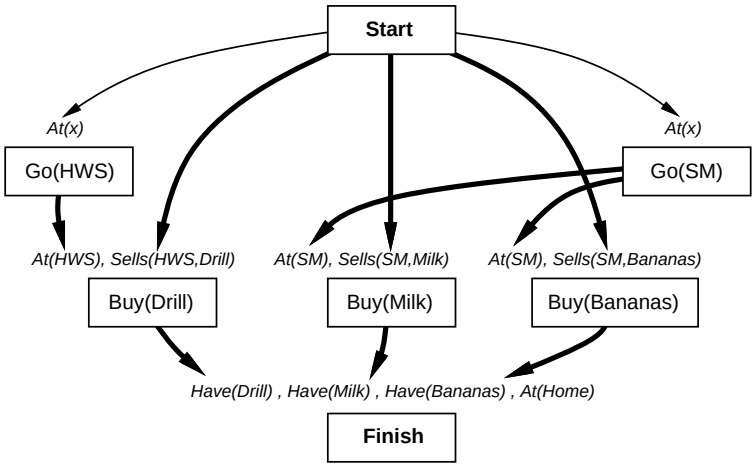


Góra: Plan zapewniający trzy spośród czterech warunków wstępnych operatora $Finish$.

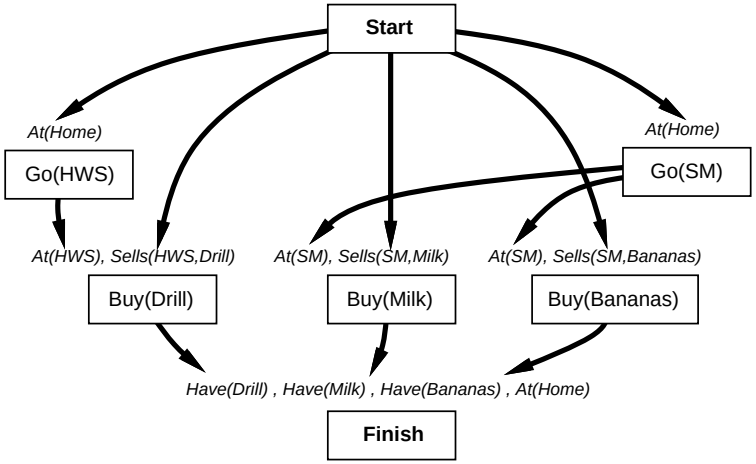


Dół: Plan zapewniający dodatkowo warunki wstępne $Sells$ w operatorach Buy.

Plan zapewniający warunki wstępne At w operatorach Buy:

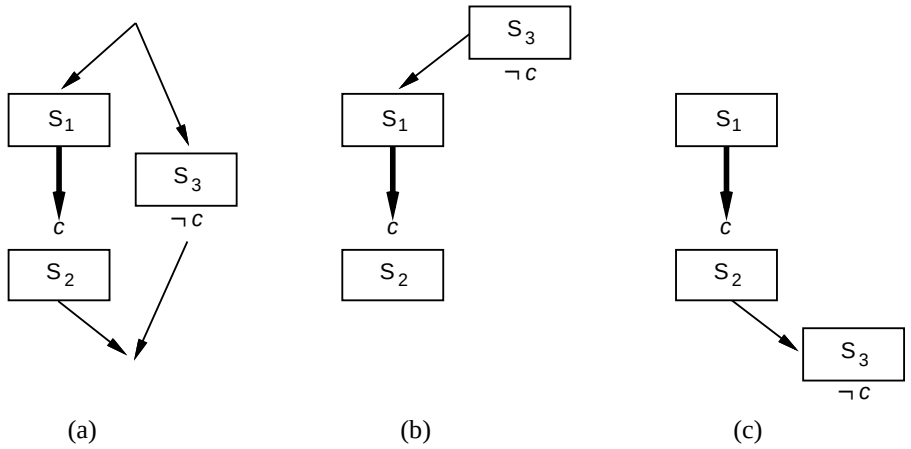


Plan zapewniający warunki wstępne At w operatorach $Go(HWS)$ i $Go(SM)$ (potrzebny nawrót):



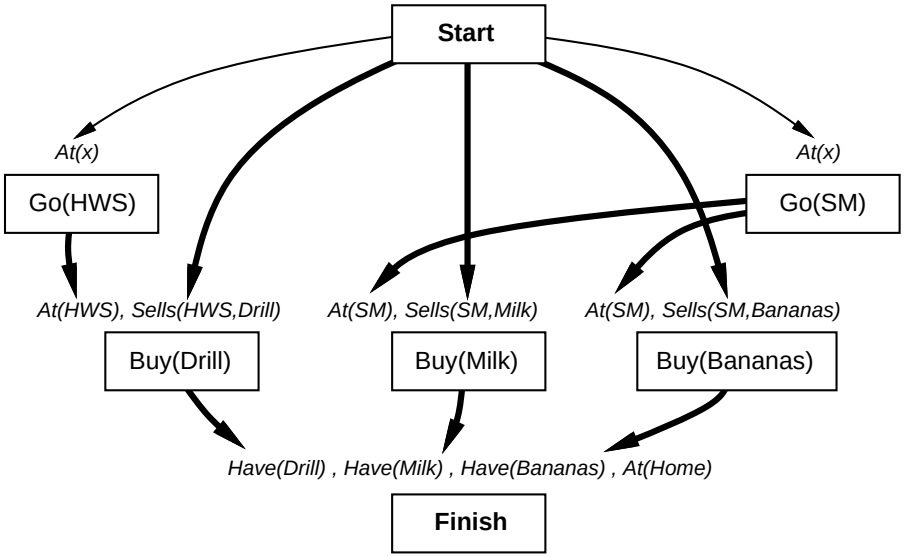
W jaki sposób system planujący potrafi stwierdzić, że konieczny jest nawrót?

Połączenia przyczynowe są chronione: system zapewnia, że **zagrożenia (threats)**, czyli operatory, które mogą zniszczyć zapewniony wcześniej warunek, są umieszczane przed (promocja) lub po (degradacja) operatorze zapewniającym ten warunek.



W naszym przypadku ani promocja, ani degradacja nie rozwiązują problemu. Dlatego system planujący wykonuje nawrót.

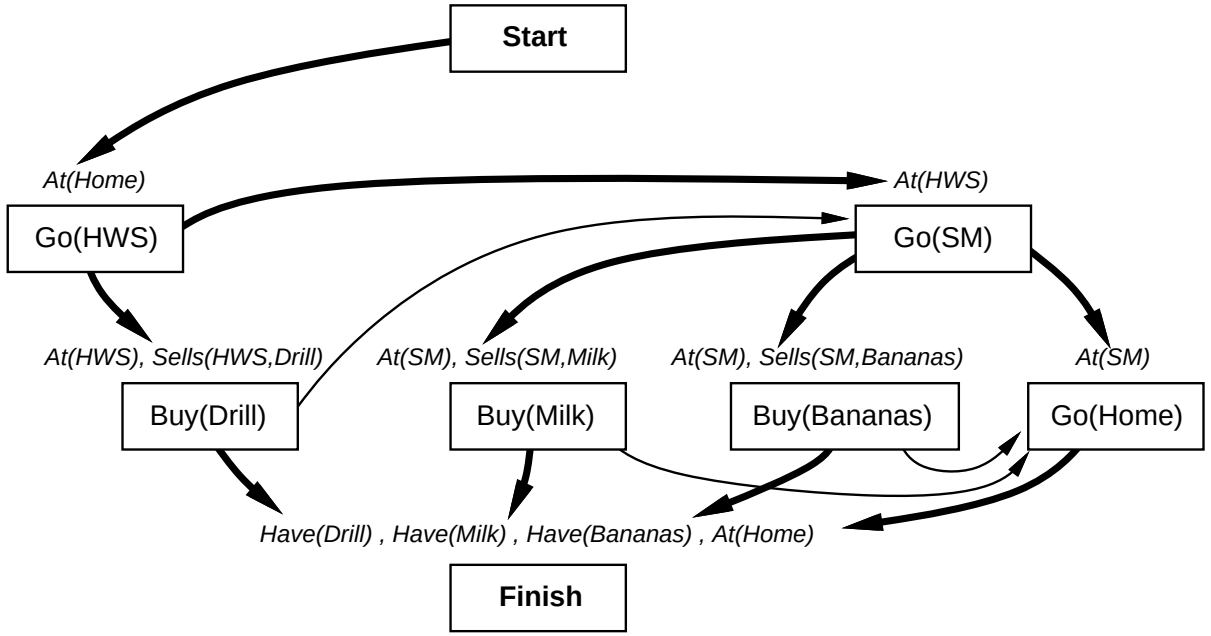
Po nawrocie:



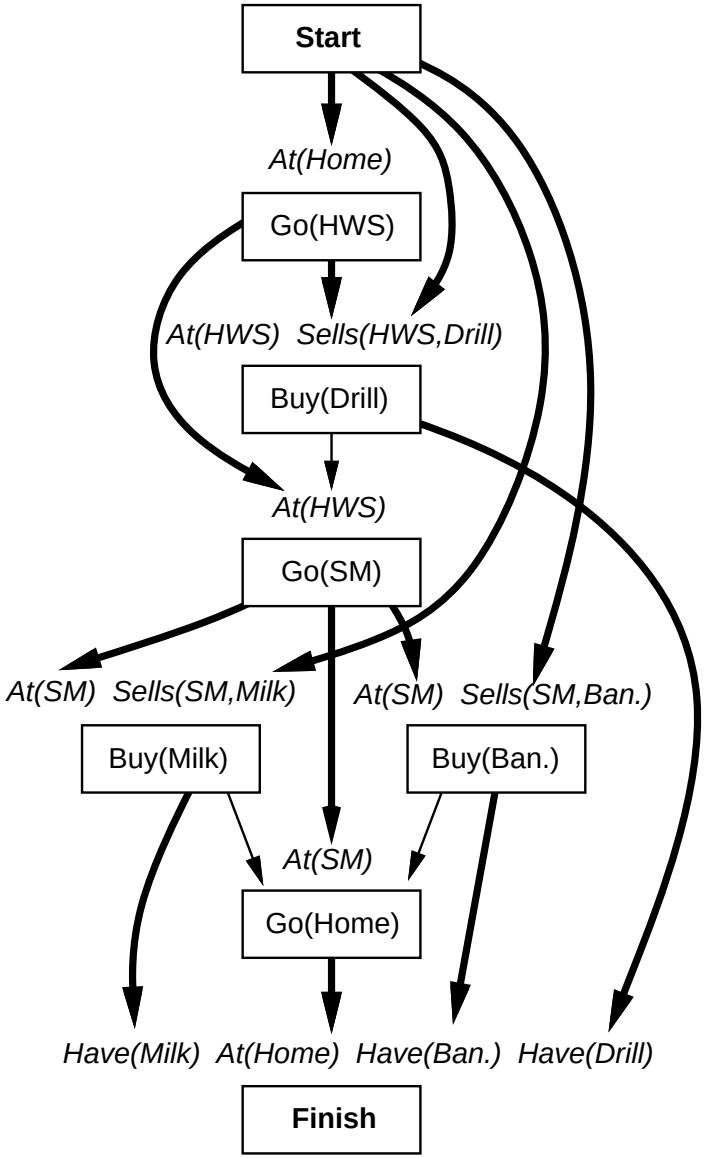
Zapewniamy warunek wstępny $At(x)$ operatora $Go(SM)$, dodając nowe połączenie postaci $Go(HWS) \overset{At(HWS)}{\rightarrow} Go(SM)$. Niestety krok ten wprowadza nową groźbę: agent może pójść ze sklepu z narzędziami do supermarketu, nie kupując przedtem wiertarki. Groźbę tę łatwo usunąć, używając metody degradacji.

Musimy jeszcze osiągnąć warunek wstępny $At(Home)$ operatora $Finish$. Wystarczy dodać operator $Go(Home)$ i zapewnić, że wystąpi on po operatorach $Buy(Milk)$ i $Buy(Banana)$.

Rozwiązanie:



W czytelniejszej formie:



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
```

---

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 --c--> S_need to LINKS(plan)
add the ordering constraint S_add ≺ S_need to ORDERINGS(plan)
if S_add is a newly added step from operators then
    add S_add to STEPS(plan)
    add Start ≺ S_add ≺ Finish to ORDERINGS(plan)
```

---

procedure RESOLVE-THREATS(*plan*)

```
for each S_threat that threatens a link S_i --c--> S_j in LINKS(plan) do
    choose either
        Promotion: Add S_threat ≺ S_i to ORDERINGS(plan)
        Demotion: Add S_j ≺ S_threat to ORDERINGS(plan)
    if not CONSISTENT(plan) then fail
end
```

Algorytm POP

(Partial-Order Planning Algorithm)

Uwagi:

- Algorytm POP jest algorytmem niedeterministycznym: instrukcja **choose** wybiera jedną z dopuszczalnych możliwości. Jeśli wszystkie dopuszczalne wybory zostały już dokonane, algorytm kończy działanie, nie znajdując rozwiązania.

- Instrukcja **fail** oznacza nawrót do ostatnio wykonywanej instrukcji **choose**.

- Algorytm POP jest algorytmem regresywnym: zaczyna od formuły celu i w każdym kroku próbuje osiągnąć jeden z warunków wstępnych wprowadzonego wcześniej operatora.

- Wybór operatora S_{need} i warunku c w procedurze SELECT-SUBGOAL nie prowadzi nigdy do nawrotu (zauważmy, że procedura nie używa instrukcji **choose**). Powód: jeśli chcemy znaleźć rozwiązanie, musimy zapewnić wszystkie warunki wstępne wszystkich operatorów.

- Algorytm POP jest poprawny i pełny: każdy znaleziony plan jest rozwiązaniem problemu; jeśli istnieje rozwiązanie problemu, zostanie ono znalezione.

- Algorytm POP, w wersji podanej powyżej, nie jest do końca wyspecyfikowany: pomija ograniczenia zmiennych.

Częściowo ukonkretnione operatory

Czy procedura RESOLVE-THREATS powinna traktować operator, którego efektem jest $\neg At(x)$ jako groźbę dla warunku $At(Home)$? Jest to groźba **potencjalna**.

Możliwy sposób rozwiązania problemu gróźb potencjalnych:

Ignorujemy każdą groźbę potencjalną do momentu, kiedy staje się ona groźbą rzeczywistą. Tzn. procedura RESOLVE-THREATS nie rozważa operatora, którego efektem jest $\neg At(x)$ jako groźby dla $At(Home)$. Ale: jeśli ograniczenie zmiennych $x = Home$ zostanie dodane do planu, powyższa potencjalna groźba staje się groźbą rzeczywistą i musi zostać rozwiązana. (Przy użyciu metody promocji lub degradacji.)

Planowanie hierarchiczne
(Hierarchical planning)

Planowanie odpowiada konstruowaniu programu. Planowanie hierarchiczne odpowiada konstruowaniu programu metodą zstępującą.

Plan [$Go(Supermarket)$, $Buy(Milk)$, $Go(home)$] jest dobry dla człowieka, ale nie dla robota. Z drugiej strony, posługując się algorytmem POP, nie jesteśmy w stanie skonstruować planu [$Forward(1m), TurnLeft(90deg), \ldots$].

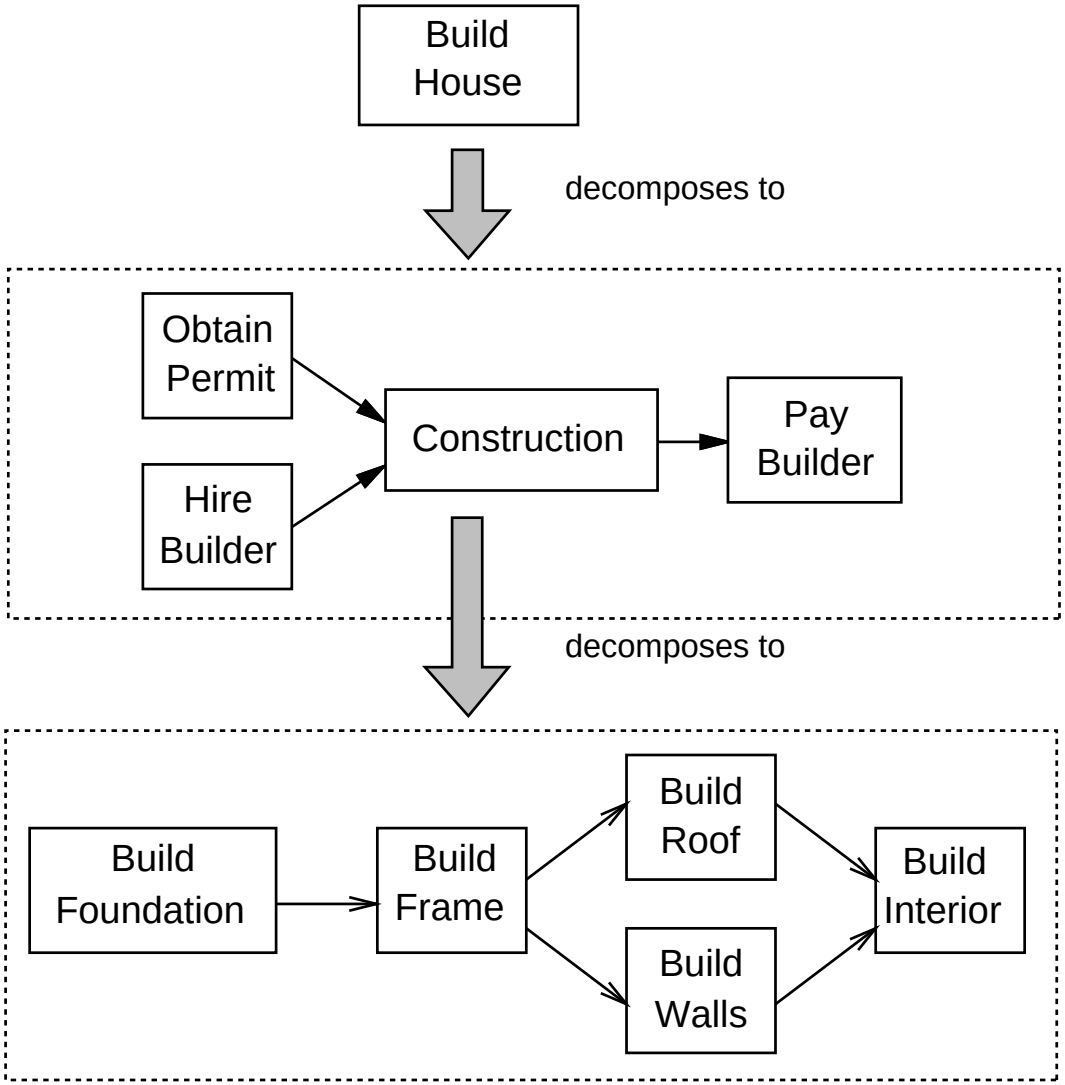
Praktyczne systemy planujące oparte są na metodzie **hierarchicznej dekompozycji**:

Startujemy, posługując się operatorami wysokiego poziomu (operatory abstrakcyjne).

W kolejnych krokach każdy abstrakcyjny operator zostaje zastąpiony swoją **implementacją**, tzn. planem, który go realizuje i który używa operatorów niższego poziomu.

Kontynuujemy ten proces tak długo, aż otrzymany plan zawiera tylko operatory **podstawowe (prymitywne)**.

Hierarchiczna dekompozycja:



Rozszerzenie języka

Dzielimy zbiór dopuszczalnych operatorów na operatory podstawowe i abstrakcyjne.

Dodajemy zbiór **metod dekompozycyjnych**. Każda metoda jest wyrażeniem postaci $Decompose(o, p)$, gdzie o jest operatorem abstrakcyjnym, natomiast p jest planem.

$$
\begin{aligned}
&Decompose(Construction,\\
&\ Plan(\text{STEPS: } \{S_1 : Build(Foundation),\\
&\qquad S_2 : Build(Frame), S_3 : Build(Roof),\\
&\qquad S_4 : Build(Walls), S_5 : Build(Interior)\}\\
&\qquad \text{ORDERINGS: } \{S_1 \prec S_2 \prec S_3 \prec S_5,\\
&\qquad\qquad S_2 \prec S_4 \prec S_5\},\\
&\qquad \text{BINDINGS: } \{\},\\
&\qquad \text{LINKS: } \{S_1 \overset{Foundation}{\longrightarrow} S_2, S_2 \overset{Frame}{\longrightarrow} S_3,\\
&\qquad\ S_2 \overset{Frame}{\longrightarrow} S_4, S_3 \overset{Roof}{\longrightarrow} S_5, S_4 \overset{Walls}{\longrightarrow} S_5\})).
\end{aligned}
$$

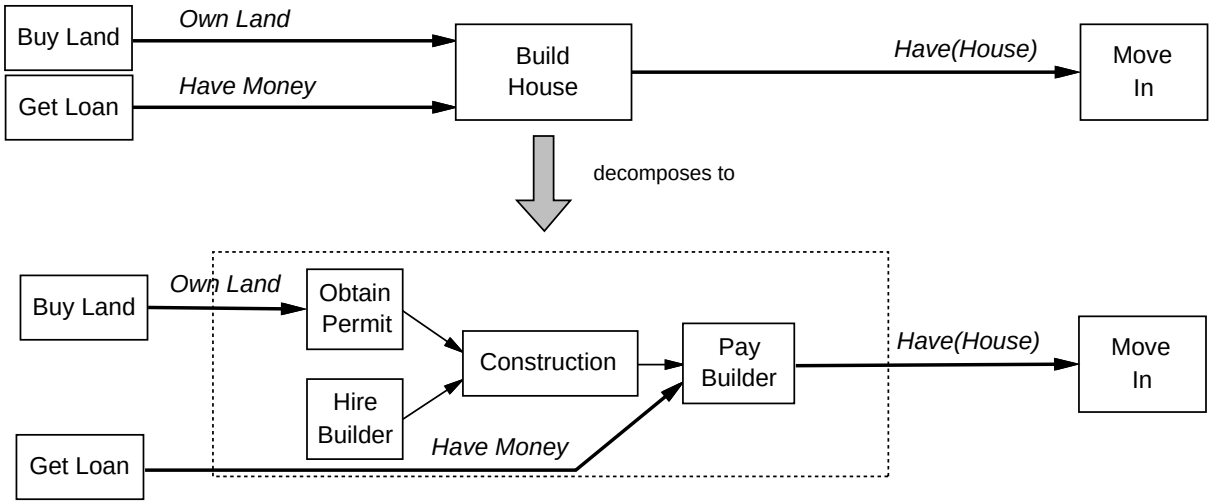
Plan p poprawnie implementuje operator o jeśli jest pełnym i poprawnym rozwiązaniem problemu znalezienia planu z warunkami wstępnymi i formułą celu operatora o. Dokładniej:

- p jest poprawny, tzn. zarówno zbiór węzłów porządkujących, jak i zbiór ograniczeń zmiennych są niesprzeczne.
- Każdy efekt operatora o musi być zapewniony przez pewien operator planu p (i nie może być "unieważniony" przez inny, późniejszy operator planu p).
- Każdy warunek wstępny każdego operatora planu p musi być osiągnięty w wyniku realizacji pewnego operatora planu p lub być jednym z warunków wstępnych operatora o.

Załóżmy, że wybrana została metoda $Decompose(S_{nonprim}, p)$. Aktualny plan, P, zostaje zmodyfikowany następująco.

- Do zbioru operatorów planu P dodajemy wszystkie operatory planu *p*; usuwamy operator $S_{nonprim}$.
- Do zbioru ograniczeń zmiennych planu P dodajemy ograniczenia zmiennych planu *p*. Jeśli operacja ta prowadzi do sprzeczności, robimy nawrót.
- Zastępujemy każdy węzeł porządkujący postaci $S_a \prec S_{nonprim}$ zbiorem węzłów $\{S_a \prec S_{m_1}, \ldots, S_a \prec S_{m_i}\}$, gdzie $S_{m_1}, \ldots, S_{m_i}$ są maksymalnymi (względem relacji $\prec$) operatorami planu *p*. Każdy węzeł porządkujący postaci $S_{nonprim} \prec S_a$ zastępujemy zbiorem $\{S_{m_1} \prec S_a, \ldots, S_{m_j} \prec S_a\}$, gdzie $S_{m_1}, \ldots, S_{m_j}$ są minimalnymi operatorami planu *p*. Następnie wywołujemy procedurę RESOLVE-THREADS w celu ewentualnego dodania nowych węzłów.

- Każde powiązanie przyczynowe występujące w P, postaci $S_i \xrightarrow{c} S_{nonprim}$, zastępujemy zbiorem powiązań postaci $S_i \xrightarrow{c} S_m$, gdzie S_m jest operatorem w p z warunkiem wstępnym c i c nie jest warunkiem wstępnym żadnego wcześniejszego operatora w p. Podobnie, każde powiązanie postaci $S_{nonprim} \xrightarrow{c} S_j$, występujące w P, zastępujemy zbiorem powiązań postaci $S_m \xrightarrow{c} S_j$, gdzie S_m jest operatorem w p z efektem c i c nie jest efektem żadnego późniejszego operatora w p.



Rozwiązanie abstrakcyjne: pełny i poprawny plan zawierający przynajmniej jeden operator abstrakcyjny.

Rozwiązanie: pełny i poprawny plan zawierający wyłącznie operatory podstawowe.

Plan P nazywamy **abstrakcją** planu *p*, jeśli *p* można otrzymać z P stosując metodę dekompozycji.

Planowanie warunkowe
(Conditional planning)

Realizując plan warunkowy, agent musi obserwować świat, w którym działa - akcje obserwacyjne.

Język planu musi zawierać zdania warunkowe, np.

if $Intact(Tire1)[Inflate(Tire1)]$
else $[Remove(Tire1, PutOn(Spare)]$.

Planowanie w zmieniającym się świecie

Etapy:

- Konstrukcja planu:
  - Obserwacja, czy nie zaszły zmiany w świecie powodujące niezachodzenie warunków początkowych planu.
  - Ewentualna rekonstrukcja planu.
- Wykonanie planu:
  - Przed wykonaniem każdego operatora konieczna jest obserwacja, czy jego warunki wstępne są spełnione.
  - Ewentualna rekonstrukcja planu.