

# 6장. 계획수립

1. 계획수립
2. 계획수립 문제
3. 계획수립기
4. 계획수립 문제 기술 언어
5. 고전적 계획수립 방법
6. 상태공간 계획수립
7. 계획공간 계획수립
8. 계층적 계획수립

# 1. 계획수립(planning)

## ❖ 계획수립(planning)

- 주어진 계획수립 문제의 임의의 초기 상태에서 목표 상태 중의 하나로 도달할 수 있게 하는 일련의 행동을 생성하는 것

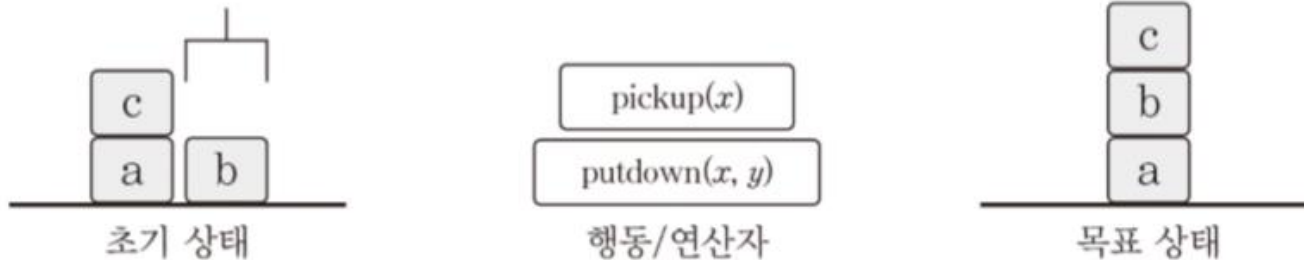


그림 6.1 계획수립 문제의 예

계획 (plan) : pickup(c) → putdown(c, floor) → pickup(b) → putdown(b, a) → pickup(c) → putdown(c, b)

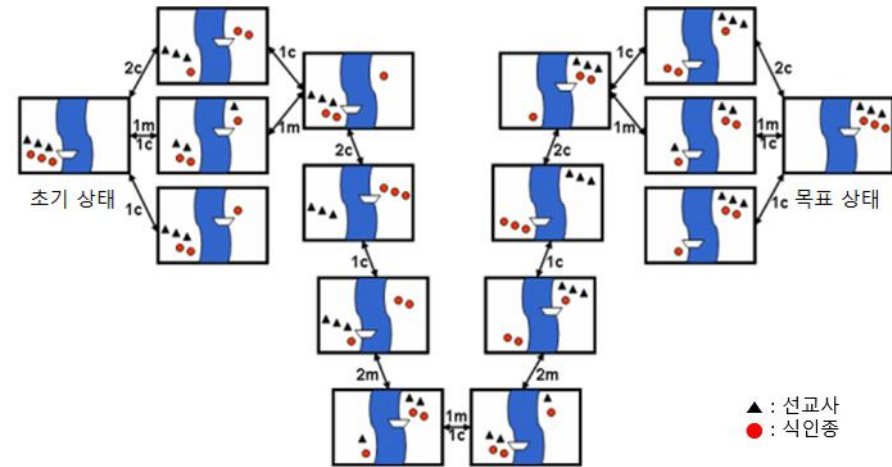
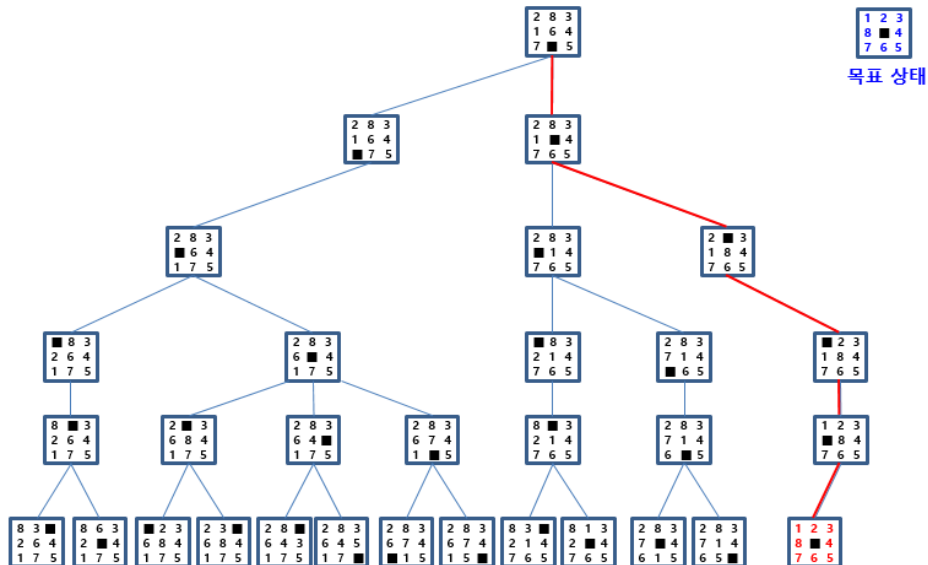
# 계획수립

## ❖ 계획수립의 사례

- 일상에서 흔하게 일어나는 일
  - 하루 일과(日課)의 계획
  - 등교길의 교통편 이용 계획

## ▪ 8-퍼즐 문제

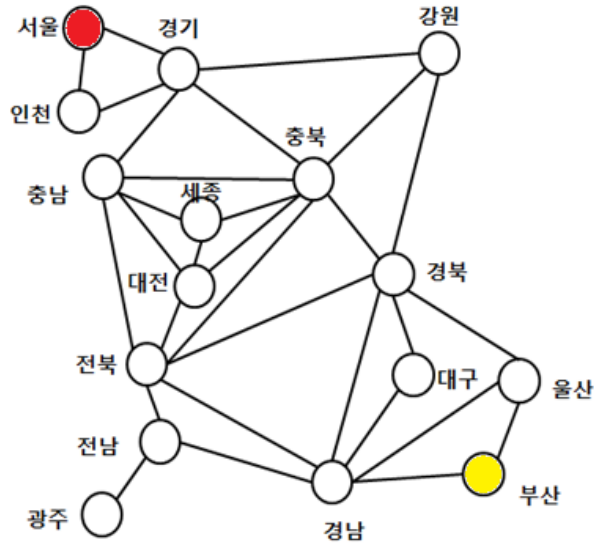
## ▪ 식인종-선교사 문제



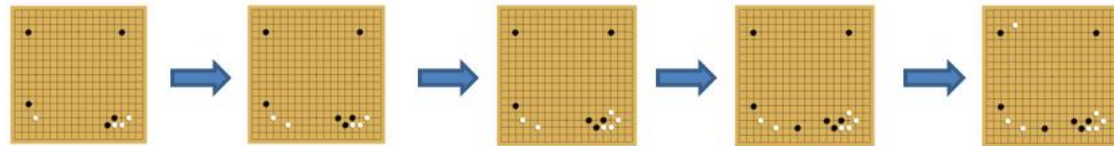
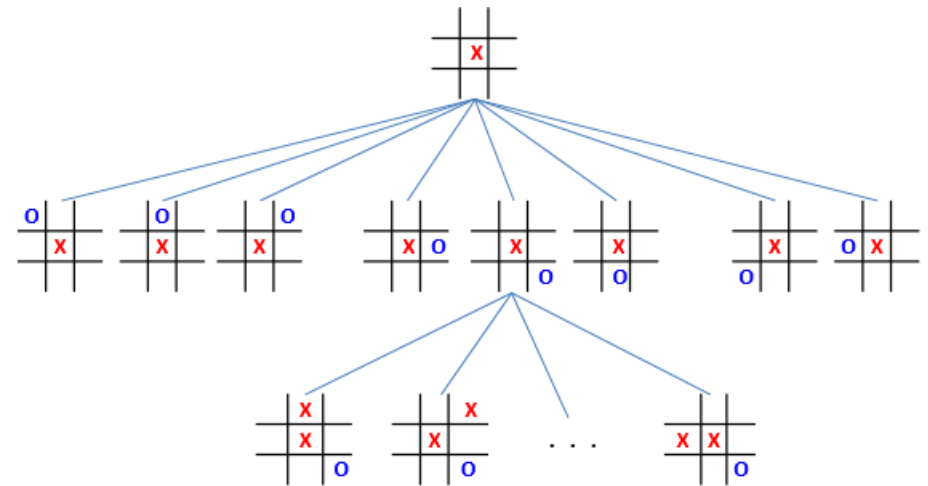
# 계획수립

## ❖ 계획수립의 사례 - cont.

### ▪ 최단경로 문제



### ▪ 보드 게임



# 계획수립

## ❖ 계획수립의 사례 - cont.

### ▪ 로봇의 계획수립

#### • 움직임 계획수립(motion planning)

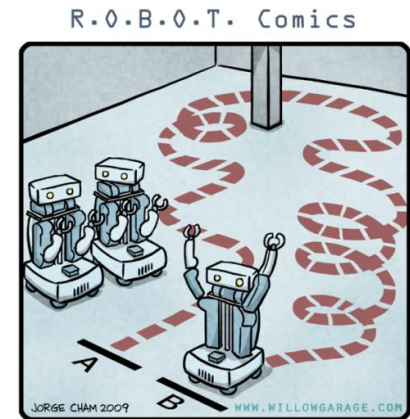
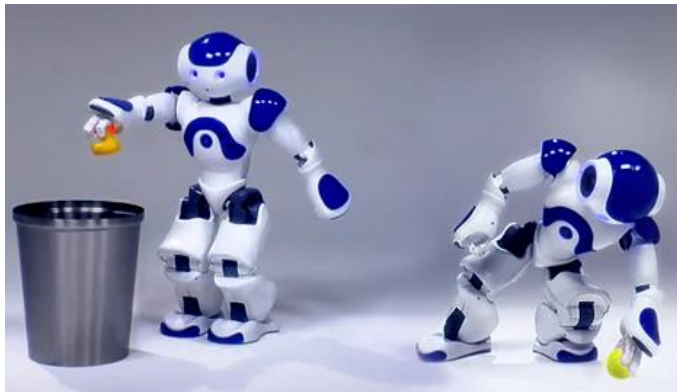
- 원하는 움직임 작업을 수행하도록 제약조건을 만족시키면서 최소의 비용으로 일련의 움직임을 찾아내는 일

#### • 경로 계획수립(path planning)

- 시작 위치에서 목적 위치로 가기 위해 관절이나 바퀴를 이동시킬 순차적인 위치를 결정하는 일

#### • 궤적 계획수립(trajjectory planning)

- 주어진 경로와 제약조건 및 물리적인 특성을 고려하여 매 시점의 관절 등의 위치, 속도, 가속도 등을 결정하는 일

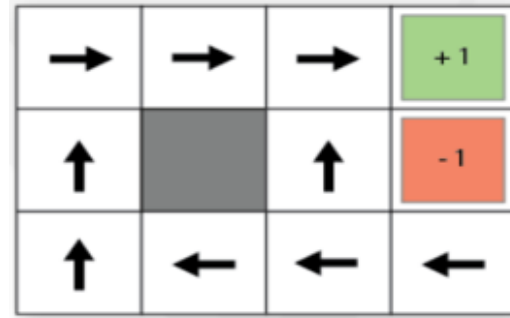
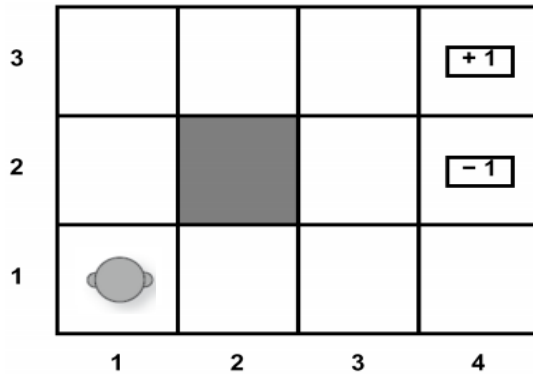


"HIS PATH-PLANNING MAY BE SUB-OPTIMAL, BUT IT'S GOT FLAIR."

# 계획수립

## ❖ 계획수립의 사례 - cont.

- 강화학습에서 정책(policy)의 학습

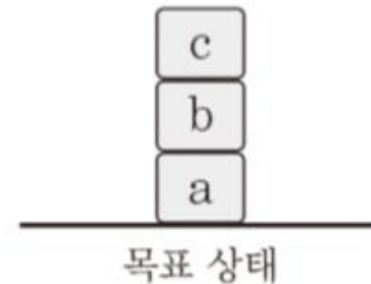
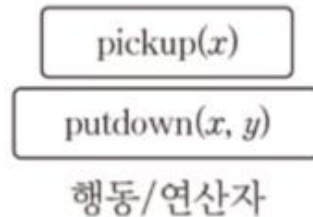
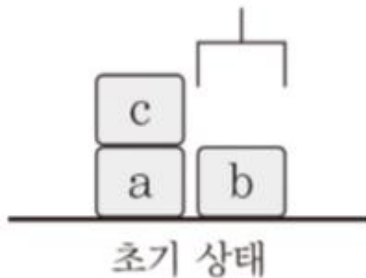


- 스케줄링 문제
- 프로젝트 관리
- 군사작전 계획
- 정보 수집
- 자원 관리
- ...

## 2. 계획수립 문제

### ❖ 계획수립 문제의 구성요소

- 초기 상태(initial state)에 대한 명세(description)
- 원하는 목표 상태(goal state)에 대한 명세
- 가능한 행동(action)들에 대한 명세





# 계획수립 문제

## ❖ 에이전트(agent)와 계획수립

### ▪ 에이전트

- 위임받은 일을 자율적(autonomous)이고 지능적(intelligent)하게 처리하는 개체
- 소프트웨어 에이전트, 물리적 에이전트(robot)

### ▪ 어떻게 일을 할지(how)를 말하지 않고, 목적(goal, what)만을 말하면 알아서 처리할 수 있는 능력 필요

- 계획수립(planning)이 핵심 요소

# 계획수립 문제

## ❖ 계획수립 문제의 형태

- 고전적 계획수립(classical planning)
- 마르코프 결정과정(Markov Decision Process, **MDP**)
- 부분관측 마르코프 결정과정(Partially Observable Markov Decision Process, **POMDP**)
- 다중 에이전트(multi-agent) 계획수립

# 계획수립 문제

## ❖ 고전적 계획수립 문제(classical planning problem)

- 가장 간단한 계획수립 문제 부류
- 기본 전제
  - 초기 상태는 하나만 주어진다.
  - 행동들은 지속시간이 없고, 행동의 결과가 결정적이고, 한 번에 하나의 행동만 수행될 수 있다.
  - 행동을 하는 에이전트는 하나뿐 이다.
- 일련의 행동들을 수행한 이후의 세계(world)의 상태 예측 가능
- 계획은 일련의 행동들로 정의
  - 목표상태에 도달하기 위해 어떤 행동들을 해야 하는지 미리 결정할 수 있음

# 계획수립 문제

- ❖ **마르코프 결정과정 문제** (Markov Decision Process, MDP)
  - 행동들의 결과는 비결정적(nondeterministic)이고, 에이전트가 행동을 통제할 수 있는 문제
  - **이산시간 마르코프 결정과정 문제** (Discrete-time Markov decision processes, discrete-time MDP)
    - 행동들은 지속시간이 없다
    - 행동의 결과가 확률에 따라 결정되어 비결정적이다.
    - 행동의 결과는 관측 가능하여 확인할 수 있다.
    - 보상함수(reward function)를 최대화하는 것을 목적으로 한다.
    - 행동을 하는 에이전트는 하나뿐 이다.
  - 강화학습(reinforcement learning)에 관심을 갖는 문제

# 계획수립 문제

## ❖ 부분관측 마르코프 결정과정

(partially observable Markov decision process, **POMDP**)

- 행동의 결과가 확률에 따라 결정되는 비결정적인 마르코프 결정과정
- 행동의 결과는 부분적으로(간접적으로) 관측
  - 현재 상태를 정확히 알 수 없고 확률적인 분포로만 추정
- 현재 상태에 대한 확률적인 분포를 믿음<sup>belief</sup>이라고 함
- 행동을 하면서 이러한 믿음을 계속 갱신

# 계획수립 문제

- ❖ **다중 에이전트 계획수립 문제**(multi-agent planning)
  - 여러 에이전트가 있는 계획수립 문제
  - 다중 에이전트의 작업에서 필요 사항
    - 하나의 **공동 목표**를 위한 에이전트들이 **계획수립**을 하는 것
    - 작업 및 자원에 대한 **협상**을 통해 **계획**을 **정제**하는 것
    - 목표의 달성을 위해 에이전트들의 **작업**을 **조정**하는 것



# 3. 계획수립기

## ❖ 계획수립기(planner)

- 주어진 문제에 대한 계획을 생성하는 알고리즘 또는 프로그램

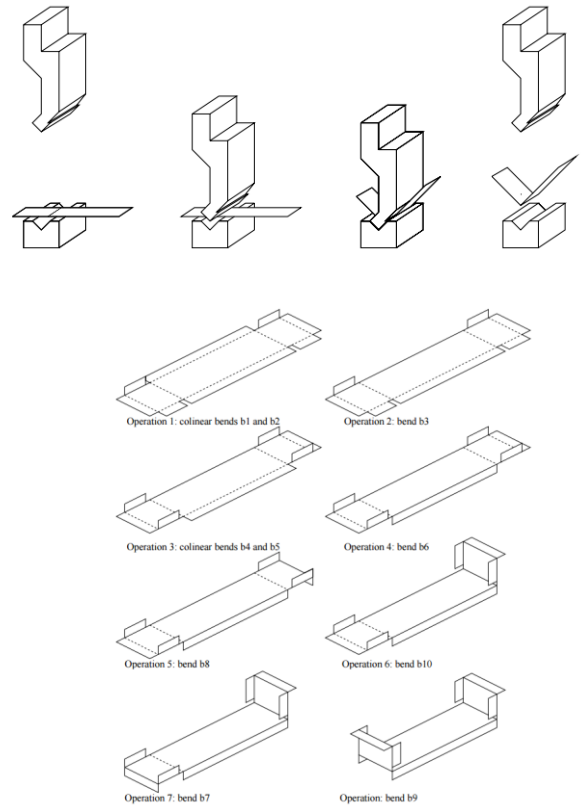
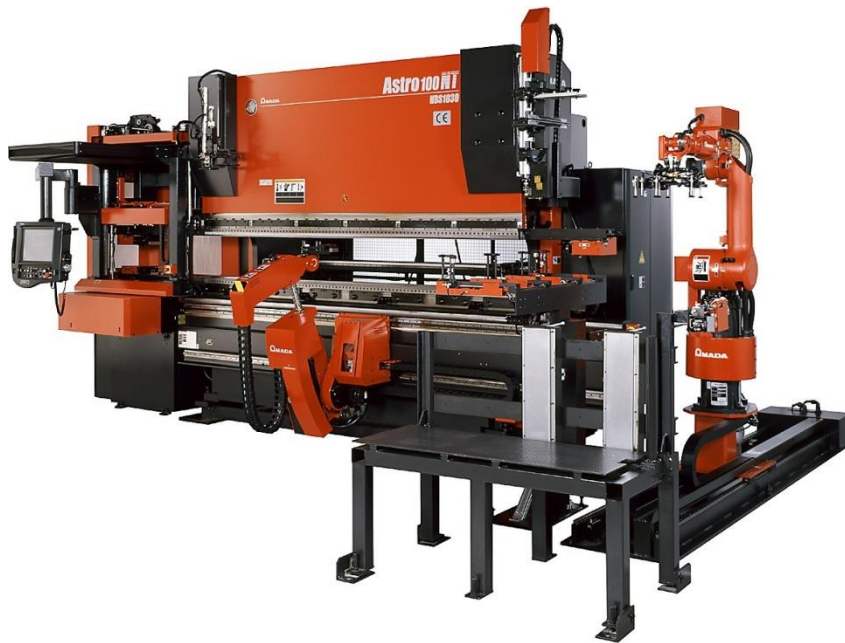
## ❖ 계획수립의 형태

- 특정 영역 계획수립기(domain-specific planner)
- 영역 독립 계획수립기(domain-independent planner)
- 설정가능 계획수립기(configurable planner)

# 계획수립기의 형태

## ❖ 특정 영역 계획수립기(domain-specific planner)

- 해당 영역에 특화된 계획수립 방법
- 다른 영역에 적용 불가
- 실제 많은 성공적인 사례



강판 벤딩(steel sheet blending) 작업계획



# 계획수립기의 형태

## ❖ 영역 독립 계획수립기(domain-independent planner)

- 영역에 상관없이 적용할 수 있는 범용 계획수립기
- 실제 모든 가능한 영역에 적용될 수 있는 계획수립기 개발 곤란
  - 적용영역 제한을 위한 가정
    - 고전적 계획수립 (classical planning)
- 계획수립 접근방법
  - 상태공간 계획수립(state-space planning)
  - 계획공간 계획수립(plan-space planning)

# 계획수립기의 형태

## ❖ 설정가능 계획수립기(configurable planner)

- 특정 영역 계획수립기에 비하여 영역 독립 계획수립기는 매우 느림
- 영역 독립 계획수립기를 사용하면서 해당 영역의 문제를 해결하는 방법에 관한 정보를 입력으로 사용
- 계층적 태스크 네트워크(hierarchical task network, HTN) 계획수립



# 4. 계획수립 언어

## ❖ 계획수립 언어

- 계획수립 문제를 표현하는데 사용하는 언어
- 고전적 계획수립 문제를 표현하는 언어
  - STRIPS, PDDL 등
  - 리터럴로 표현되는 상태변수(state variable)가 중심
  - 에이전트를 포함한 세계(world)의 상태는 상태변수에 값을 지정하여 표현
  - 행동(action)에 대한 표현은 행동전후 상태변수 값의 변화 내용을 기술
  - 상태변수들이 상태공간(state space)을 결정
    - 상태변수들의 개수가 늘어나면 상태공간의 크기는 기하급수적으로 증가

# 계획수립 언어

## ❖ STRIPS(STanford Research Institute Problem Solver)

- 미국의 SRI International의 Richard Fikes와 Nils Nilsson이 개발(1971)
  - 자동 **계획생성기**(planner)의 이름
  - 계획수립 문제를 **표현**하는 언어의 이름
- 상태와 행동을 표현하기 위해 **술어논리**(first-order predicate logic) 사용
- **상태**(state)
  - 변수와 함수를 포함하지 않은 **긍정 리터럴**(positive literal)들의 **논리곱**으로 표현
  - 예. '집에 있고 바나나가 있다'는 상태

$At(Home) \wedge Have(Banana)$

# 계획수립 언어

## ❖ STRIPS – cont.

### ▪ 목표(goal)

- 리터럴들의 논리곱으로 표현
- 부정 리터럴 및 존재한정사가 붙은 것으로 간주되는 변수 포함 가능
- 예. '집에 있는데 바나나가 없다'와 '어디에선가 바나나를 판다'

$At(\text{Home}) \wedge \neg \text{Have}(\text{Banana})$

$At(x) \wedge \text{Sells}(x, \text{Banana})$

- $x$ : 변수

# 계획수립 언어

## ❖ STRIPS – cont.

### ▪ **행동**(action)

- 이름, 매개변수 목록, precondition(사전조건), effect(효과)로 구성
- **이름**
  - 어떤 일을 하는 것인지 기술
- **매개변수 목록**
  - precondition과 effect에 값을 전달하는 변수들
- **precondition** (사전조건)
  - 행동을 실행하기 전에 만족돼야 하는 조건 기술
  - 함수를 사용하지 않는 리터럴의 논리곱으로 표현
- **effect** (효과)
  - 행동의 실행 후에 생기는 **상태변화**를 나타내는 것
  - 함수를 사용하지 않는 리터럴들의 논리곱으로 표현
  - **긍정 리터럴**들은 행동 실행으로 새로 생기는 성질 표현 (**add-list**)
  - **부정 리터럴**들은 행동 실행으로 더 이상 만족되지 않는 성질 표현 (**delete-list**)

# 계획수립 언어

## ❖ STRIPS – cont.

- 예: '상자 위에 올라간다'는 **ClimbUp** 행동 정의
  - Precondition:
    - '대상의 위치와 상자의 위치가 같고 높이는 아래쪽이다'
  - Effect:
    - '높이가 아래쪽에서 위쪽으로 바뀐다'

**Action:** ClimbUp(location)

**Precondition:** At(location), BoxAt(location), Level(Low)

**Effect:**

**add-list:** Level(High)

**delete-list:** Level(Low)

# 계획수립 언어

## ❖ PDDL(Planning Domain Definition Language)

- **고전적 계획수립 문제**의 표현 방법을 **표준화**하기 위해 Drew McDermott 등(1998)이 개발한 언어
- **국제 계획수립 대회**(International Planning Competition, IPC; [ipc.icaps-conference.org](http://ipc.icaps-conference.org))의 **표준언어**로 사용, 계속 진화
- 문제 영역 세계에 있는 **객체**(object), 객체의 **성질**에 대한 **술어**(predicate), **초기 상태**, **목표 상태**, **행동**을 기술
- 계획수립 문제를 두 개의 파일에 나누어 저장
  - **domain 파일** : 술어, 행동에 대한 정보 저장
  - **problem 파일** : 객체, 초기 상태, 목표 저장



# 계획수립 언어

## ❖ PDDL – cont.

### ▪ domain 파일의 BNF 형태

```
<domain> ::= (define (domain <name>) [  
  <require-def>] [  
  <types-def>]:typing  
  <constants-def>] [  
  <predicates-def>] [  
  <functions-def>]:fluents  
  <constraints>] <structure-def>*)
```

### ▪ Problem 파일의 BNF 형태

```
<problem> ::= (define (problem <name>) (:domain <name>) [  
  <require-def>]  
  <object declaration>] <init> <goal>  
  <constraints>]:constraints [  
  <metric-spec>]:numeric-fluents  
  <length-spec>])
```

# 계획수립 언어

## ❖ PDDL – cont.

- 예. **strips-gripper4** 문제
  - gripper-strips라는 **문제 영역**
  - **objects**에 객체들
  - **init**에 초기 조건
  - **goal**에 목표 상태

```
(define (problem strips-gripper4)
  (:domain gripper-strips)
  (:objects rooma roomb ball1 ball2 ball3 ball4 left right)
  (:init (room rooma) (room roomb)
         (ball ball1) (ball ball2) (ball ball3) (ball ball4)
         (gripper left) (gripper right)
         (at-robby rooma)
         (free left) (free right)
         (at ball1 rooma) (at ball2 rooma) (at ball3 rooma) (at ball4 rooma))
  (:goal (and (at ball1 roomb) (at ball2 roomb) (at ball3 roomb))))
```

# 계획수립 언어

## ❖ PDD – cont.

- 예. **Pick-up** 행동 표현
  - **effect** 부분에서 **not**이 되는 부분이 STRIPS에서 **delete-list**에 해당
  - 그렇지 않은 부분은 **add-list**에 해당
  - ?로 시작하는 것은 변수

```
(:action pick-up :parameters (?x ?y ?z)
  :precondition (and (BALL ?x) (ROOM ?y) (GRIPPER ?z)
                    (at-ball ?x ?y) (at-robby ?y) (free ?z))
  :effect (and (carry ?z ?x)
              (not (at-ball ?x ?y)) (not (free ?z))))
```



# 5. 고전적 계획수립 방법

## ❖ 고전적 계획수립 방법

- **상태공간 계획수립**(state-space planning)
  - 전향 탐색
  - 후향 탐색
  - STRIPS 알고리즘
- **계획공간 계획수립**(plan-space planning)
- **계획수립 그래프 방법**(planning graph method)
  - GraphPlan 알고리즘
- **계층적 계획수립**(hierarchical planning)
  - HTN 알고리즘

# 6. 상태공간 계획수립

## ❖ 상태공간 계획수립 (state-space planning)

- 상태공간(state space) 상의 초기 상태에서 목표 상태로의 **경로 탐색**
  - 노드(node) : 세계(world)의 상태
  - **에지(edge)** : 상태 전이(transition)를 일으키는 **행동(action)**
- 행동이 바로 적용할 수 있는 **기본 행동**(primitive action)
  - **연산자(operator)** = 기본 행동

# 상태공간 계획수립

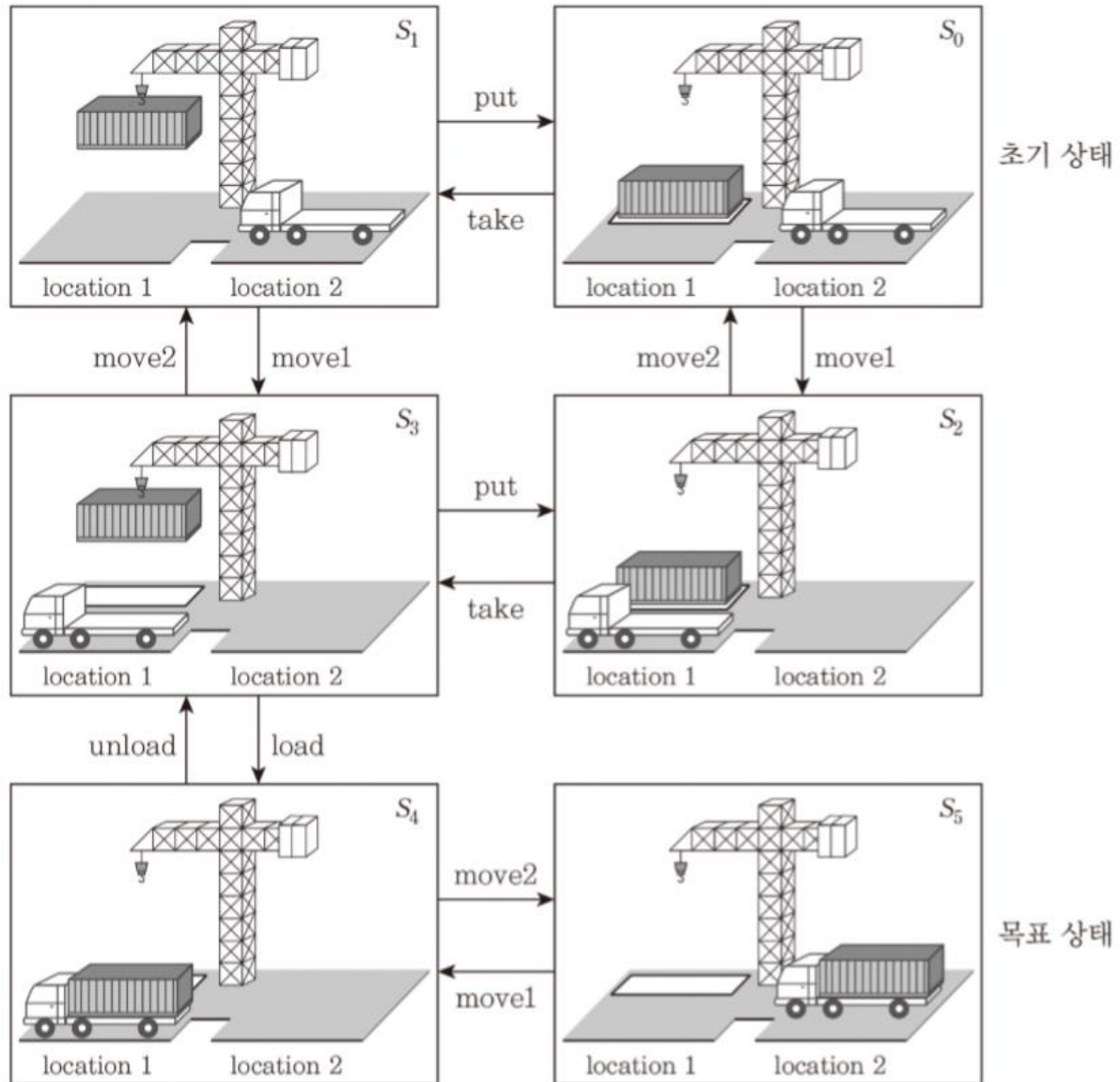


그림 6.2 차량-화물-크레인 문제의 상태공간

# 6.1 전향 탐색과 후향 탐색

## ❖ 전향 탐색(Forward Search)

- 초기 상태에서 시작
- 적용가능한 연산자를 목표 상태에 도달할 때까지 적용
- 다양한 알고리즘 적용 가능
  - 너비우선 탐색(Breadth-first search)
  - 깊이우선 탐색(Depth-first search)
  - 휴리스틱 탐색 : **A\*** 알고리즘

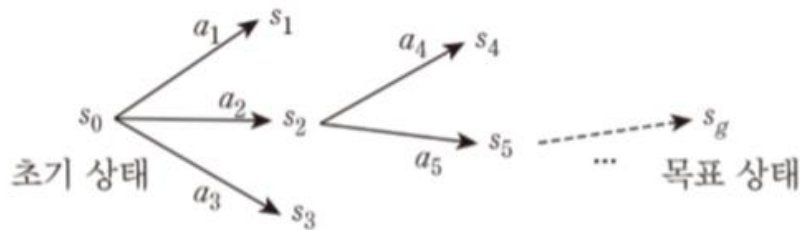


그림 6.3 목표 상태를 찾기 위한 전향 탐색

# 전향 탐색과 후향 탐색

## ❖ 후향 탐색(Backward Search)

- 목표 상태에서 시작
- 해당 상태를 만들어내는 행동 선택을 시작 상태에 도달할 때까지 반복

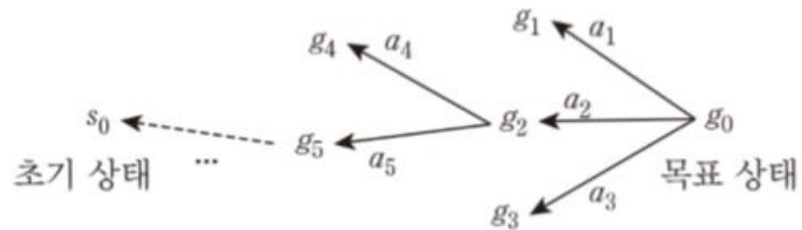


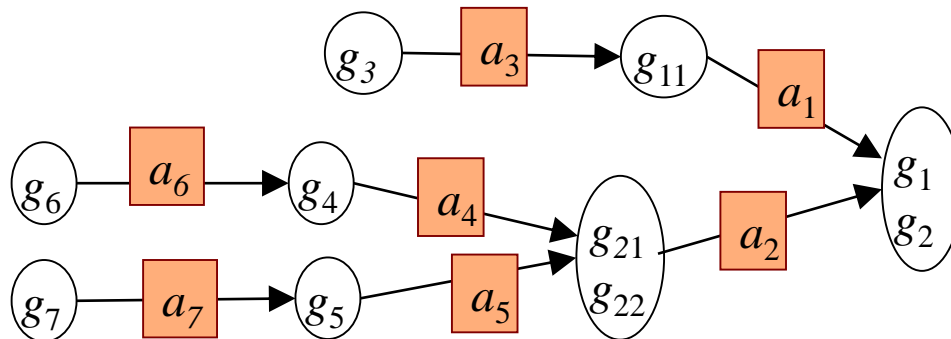
그림 6.4 초기 상태를 찾아가기 위한 후향 탐색



# 6.2 STRIPS 계획수립 알고리즘

## ❖ STRIPS 알고리즘

- 기본적으로 후향 탐색 방법
- 목표 상태가 만족되지 않으면,  
목표 상태를 effect로 만들 수 있는 연산자 선택하여  
연산자의 매개변수를 설정하고  
**precondition**이 만족되는지 확인
- precondition 중에 만족되지 않는 것이 있으면,  
그것을 effect로 하여 위 과정 반복
- 모든 precondition들이 만족되면  
사용된 매개변수 설정된 연산자들을 역순으로 나열하여 계획생성



# STRIPS 계획수립 알고리즘

## ❖ STRIPS 알고리즘의 적용 예

- 세계(world)
  - 2개의 방: R1, R2
  - 방 사이의 출입문: D
  - 로봇: I. 방 R1에 위치
  - 블록: B. 방 R2에 위치

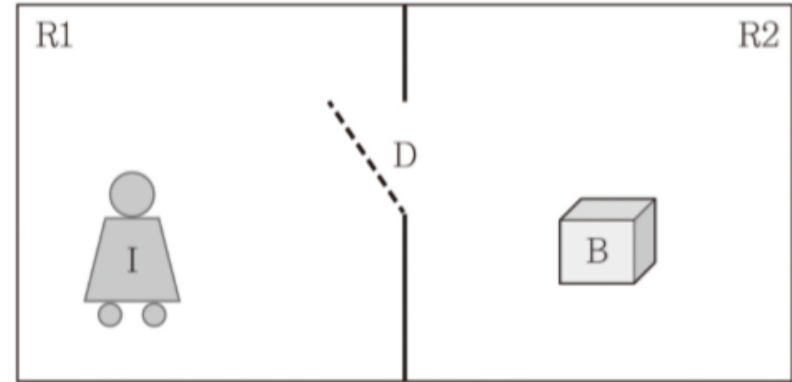
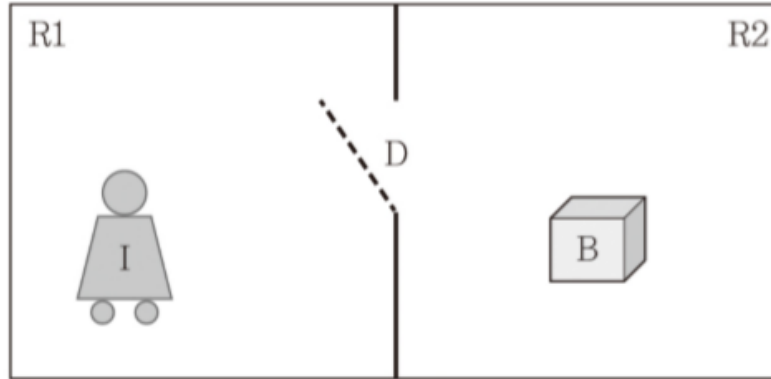


그림 6.5 계획수립 문제의 예

- 목표(goal)
  - 로봇 I와 블록 B가 방 R2에 함께 있도록 하는 것
- 술어(predicate)
  - **InRoom**(x, r) : 움직일 수 있는 물건 x가 방 r에 있다.
  - **NextTo**(x, t) : 물건 x가 방 또는 물건을 가리키는 t 옆에 있다.
  - **Status**(d, s) : 출입문 d가 s 상태(Open 또는 Closed)에 있다.
  - **Connects**(d, rx, ry) : 출입문 d가 방 rx와 방 ry를 연결한다

# STRIPS 계획수립 알고리즘

## ❖ STRIPS 알고리즘 적용 예



### ▪ 초기상태

InRoom(I, R1)  
InRoom(B, R2)  
Connects(D, R1, R2)  
Connects(D, R2, R1)  
Status(D, Open)

### ▪ 목표상태

InRoom(I, R2)  
InRoom(B, R2)  
Connects(D, R1, R2)  
Connects(D, R2, R1)  
Status(D, Open)

# STRIPS 계획수립 알고리즘

## ❖ STRIPS 알고리즘 적용 예

- 연산자(operator)

**Operator:** GoToDoor(I, dr)

**Precondition:** InRoom(I, ra), Connects(dr, ra, rb)

**Effect:**

**add-list:** NextTo(I, dr)

**delete-list:**

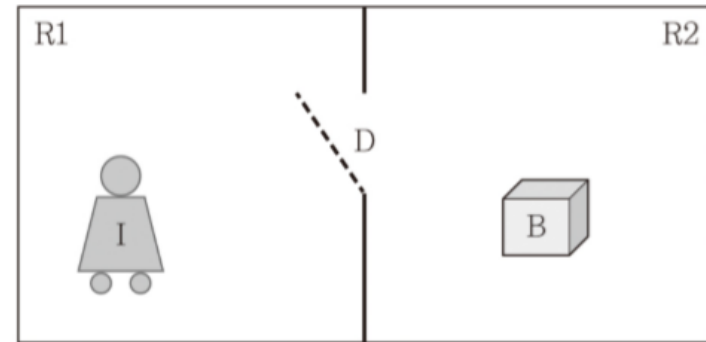
**Operator:** GoThruDoor(I, dr)

**Precondition:** Connects(dr, ra, rb), NextTo(I, dr), Status(d, Open)

**Effect:**

**add-list:** InRoom(I, rb)

**delete-list:** InRoom(I, ra)



# STRIPS 계획수립 알고리즘

## ❖ STRIPS 알고리즘 적용 예

### ▪ 초기상태

**InRoom(I, R1)**

InRoom(B, R2)

**Connects(D, R1, R2)**

Connects(D, R2, R1)

Status(D, Open)

### ▪ 목표상태

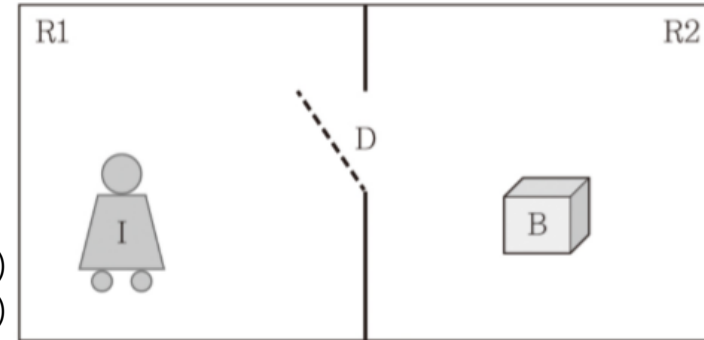
**InRoom(I, R2)**

InRoom(B, R2)

Connects(D, R1, R2)

Connects(D, R2, R1)

Status(D, Open)



### ▪ 차이

**-InRoom(I, R2)**

**Operator:** GoThruDoor(I, dr)

**Precondition:**

Connects(dr, ra, rb), NextTo(I, dr), Status(d, Open)

**Effect:**

**add-list:** InRoom(I, rb)

**delete-list:** InRoom(I, ra)

**rb = R2**

**Connects(dr, ra, R2), NextTo(I, dr), Status(dr, Open)**

**Connects(D, R1, R2)**

**dr = D, ra = R1**

**NextTo(I, D)**

**Operator:** GoToDoor(I, dr)

**Precondition:**

InRoom(I, ra), Connects(dr, ra, rb)

**Effect:**

**add-list:** NextTo(I, dr)

**delete-list:**

**dr = D**

**InRoom(I, ra), Connects(D, ra, rb)**

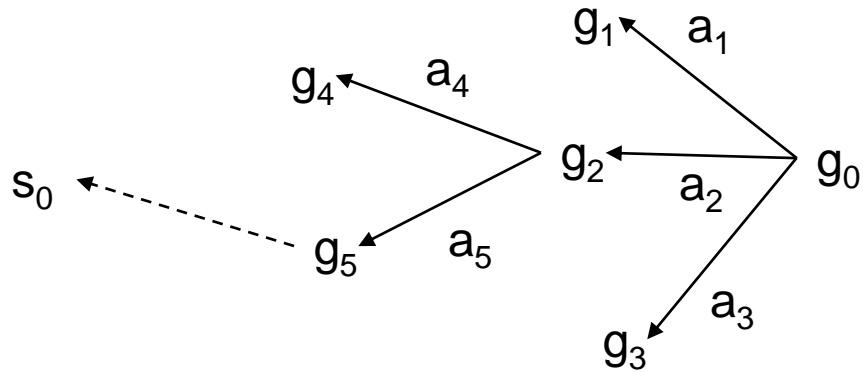
**ra = R1, rb = R2**

**InRoom(I, R1), Connects(D, R1, R2)**

**GoToDoor(I, D) → GoThruDoor(I, D)**

## 6.3 GraphPlan 알고리즘

- ❖ 계획수립 그래프 방법(planning graph method)
  - 계획수립 그래프(planning graph)를 사용하여 탐색공간 표현
  - 탐색 알고리즘에서 분기계수(branching factor) 축소
    - 후향 탐색 : 초기 상태에 도달할 수 있는 없는 행동의 시도 축소



# GraphPlan 알고리즘

## ❖ GraphPlan 알고리즘

- 기존 계획공간 계획수립(plan-space planning)에 비해 매우 빠른 속도
- 이후 GraphPlan 기반의 많은 계획수립 알고리즘 개발
  
- **변수가 없는 연산자**들로 구성된 STRIPS 문제 해결
- **변수 포함 연산자 변환**
  - 변수에 가능한 모든 객체 바인딩
  - 가능한 바인딩 조합의 개수만큼 변수가 없는 연산자 생성
    - 예. 연산자  $On(x,y)$ 에 대한 2개 객체 : 4개의 연산자 생성
  - 단점 : 연산자의 개수가 많이 늘어남
  - 장점 : 변수를 사용하지 않아 매칭 연산 용이

# GraphPlan 알고리즘

## ❖ 계획수립 그래프(planning graph)

- 명제 단계(proposition level) 와 행동 단계(action level)가 번갈아가며 구성

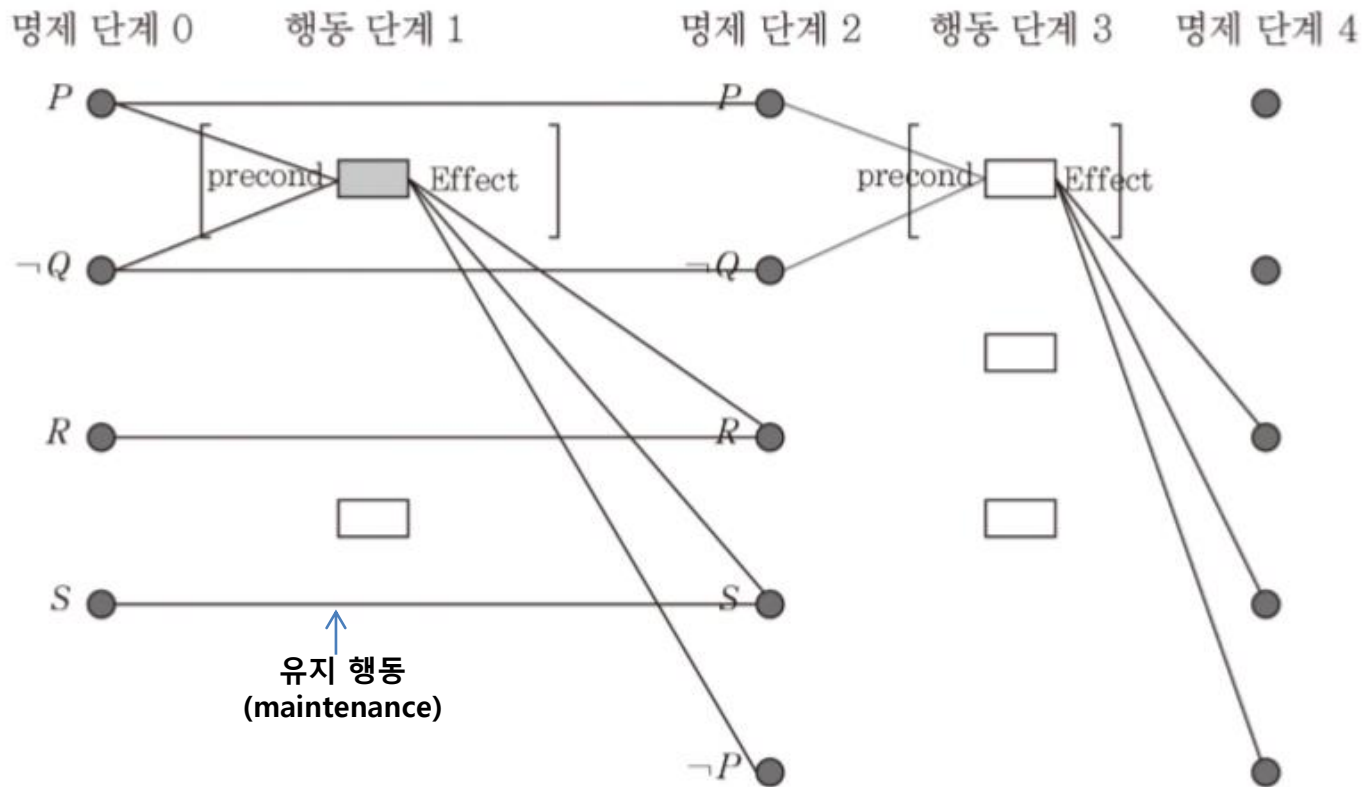


그림 6.6 계획수립 그래프(planning graph)

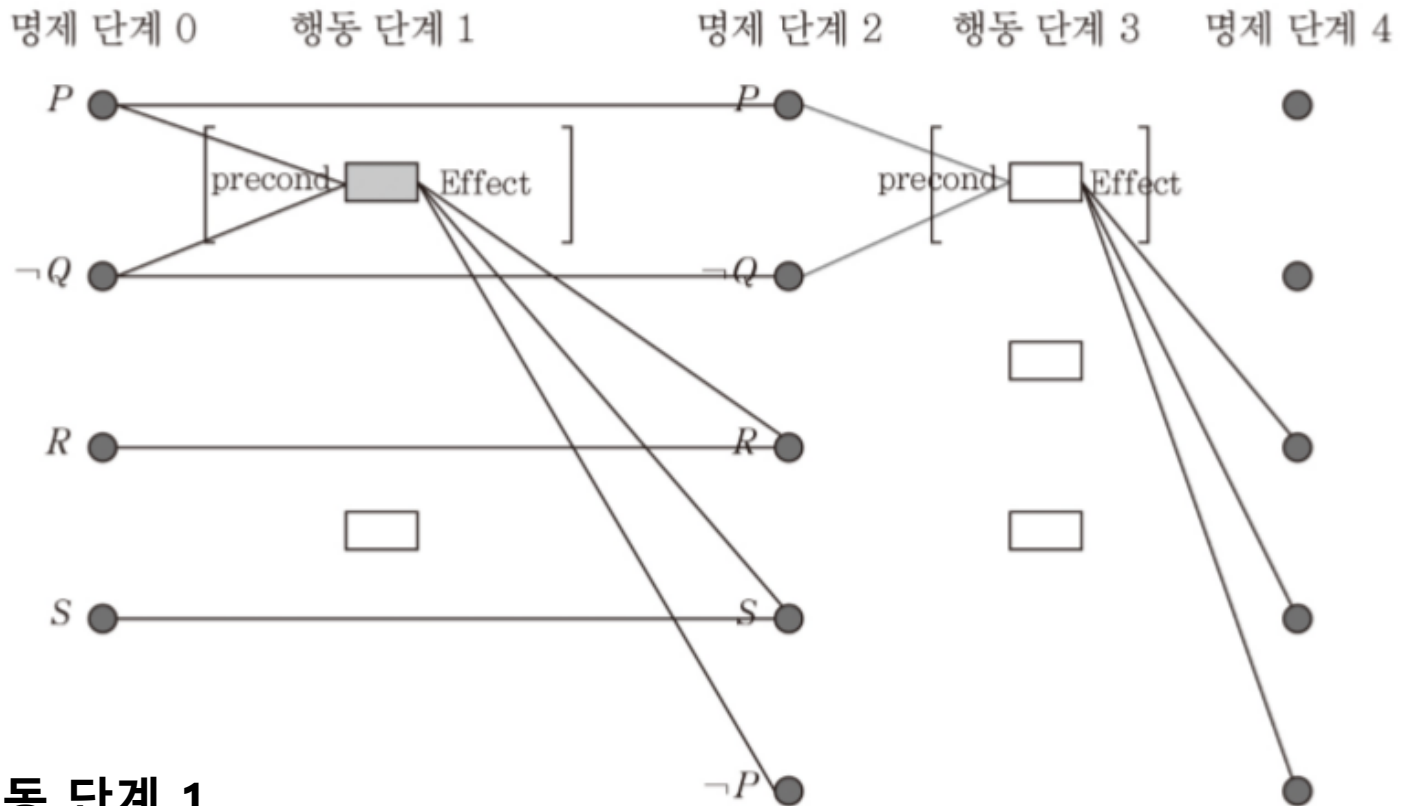
### ▪ 명제 단계 0

- 초기 상태에 주어지는 각 리터럴(literal)를 노드로 표현



# GraphPlan 알고리즘

## ❖ 계획수립 그래프(planning graph)

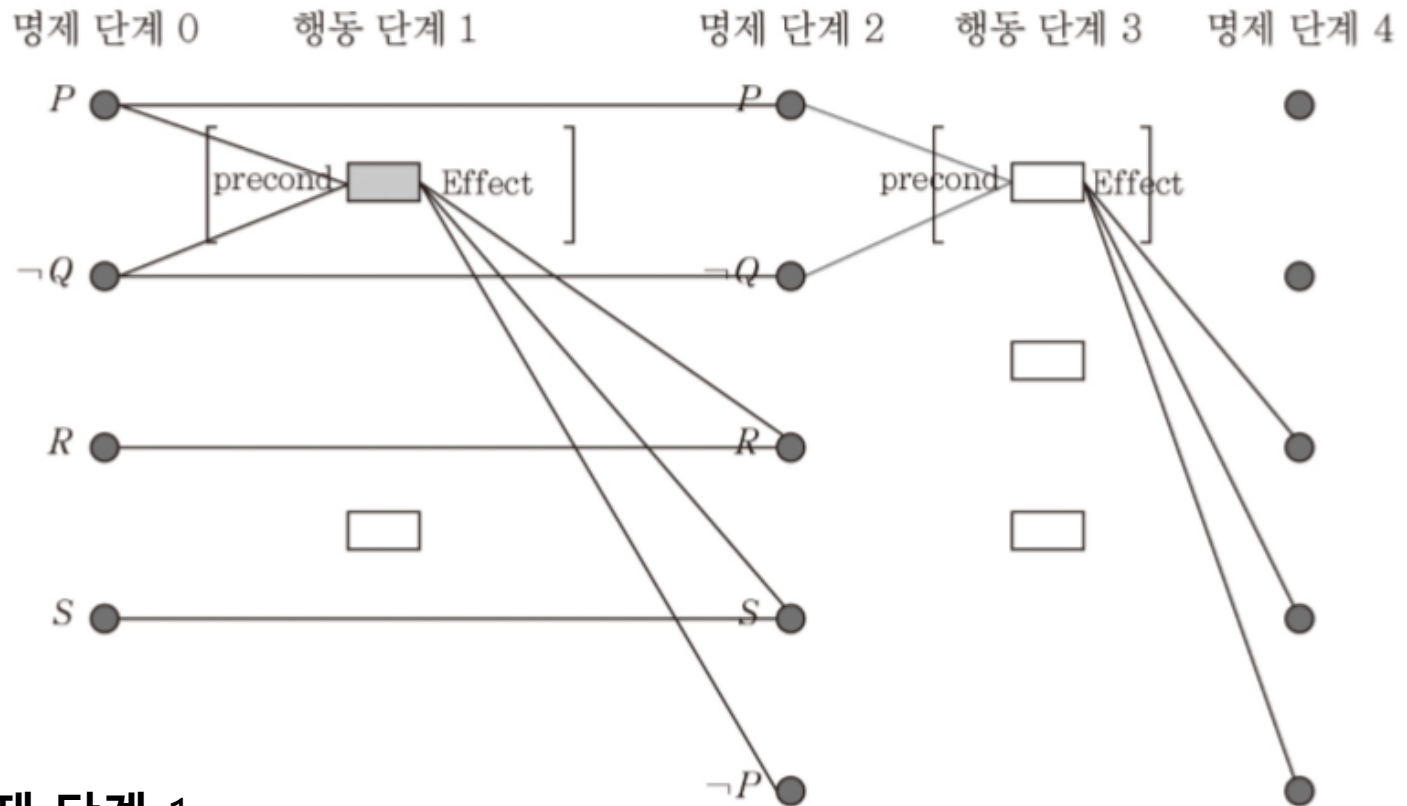


### ▪ 행동 단계 1

- 명제 단계 0의 명제들에 대해서 적용될 수 있는 각 행동을 노드로 표현
- 명제 단계 0에 나타나는 Precondition의 리터럴 노드와 연결
- 명제 단계 1에 나타나는 Effect의 리터럴 노드와 연결

# GraphPlan 알고리즘

## ❖ 계획수립 그래프(planning graph)

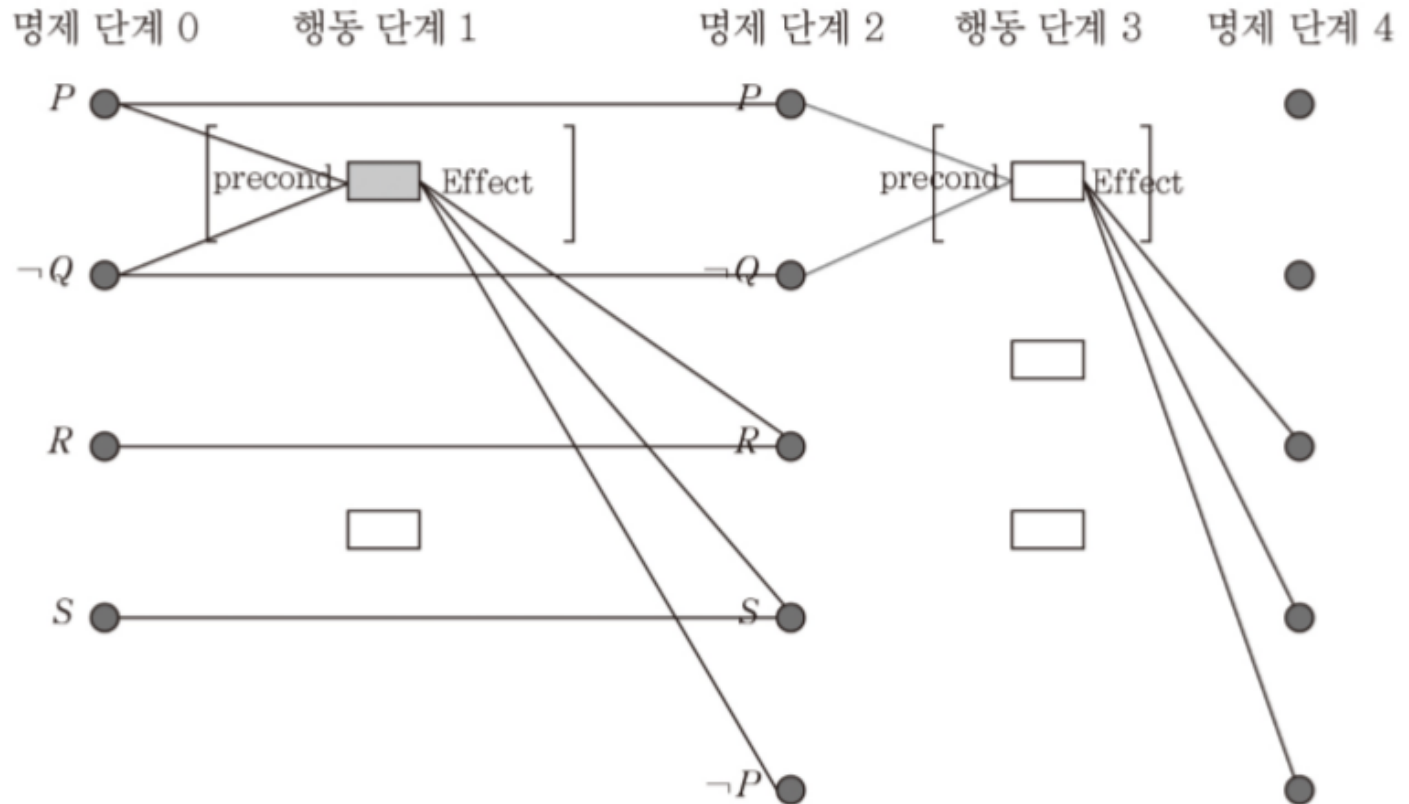


### ▪ 명제 단계 1

- 행동 단계 1의 행동의 **Effects** 부분에 나타나는 리터럴들과 명제 단계 0의 리터럴들에 대응하는 노드 생성
- 명제 단계 0의 모든 노드를 명제 단계 1에 유지

# GraphPlan 알고리즘

## ❖ 계획수립 그래프(planning graph)



- 행동 단계 2와 명제 단계 2를 동일한 방법으로 반복
- 행동 단계와 명제 단계 확장

# GraphPlan 알고리즘

## ❖ 계획수립 그래프(planning graph)의 확장

### ▪ 상호배제(mutual exclusion, mutex) 링크

- 행동 단계와 명제 단계 한 쌍의 추가시, 동시에 실행되거나 만족될 수 없는 동일 단계의 노드들 사이 연결

### ▪ 행동 단계의 상호배제

#### • 상충되는 결과 도출(inconsistent effect)

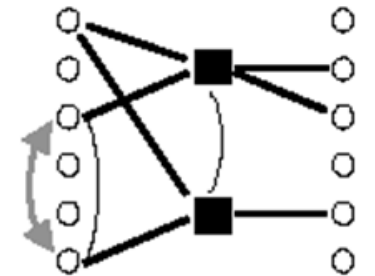
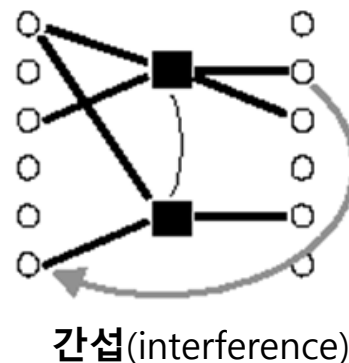
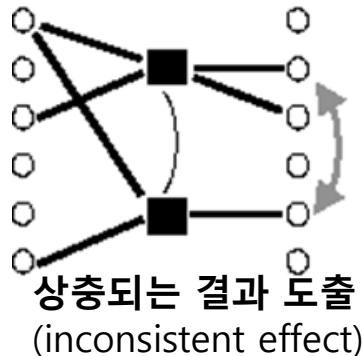
- 한 행동의 effect가 다른 행동의 effect가 만드는 명제를 제거하는 경우

#### • 간섭(interference)

- 한 행동이 다른 행동의 precondition에서 사용되는 명제를 제거하는 경우

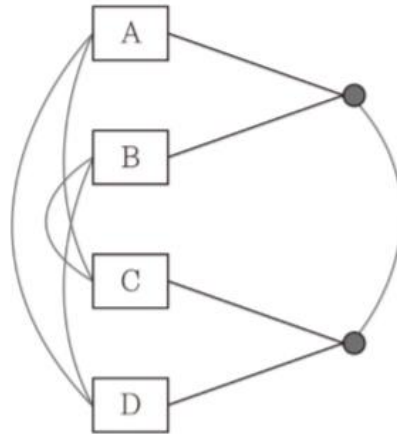
#### • 경쟁관계 사전조건(competing needs preconditions)

- 두 행동이 바로 이전 단계에서 상호배제관계에 있는 명제들을 precondition에서 사용하는 경우



# GraphPlan 알고리즘

- ❖ 계획수립 그래프(planning graph)의 확장
  - 명제 단계의 상호배제
    - 상충되는 지지(inconsistent support)
      - 대응되는 두 명제를 만들어내는 이전 단계의 모든 행동들이 서로 상호배제 관계에 있는 경우



상충되는 지지(inconsistent support)

# GraphPlan 알고리즘

---

## 알고리즘 6.1 GraphPlan 알고리즘

1. 초기 상태의 리터럴들에 해당하는 노드들을 단계 0에 만든다.
  2.  $k \leftarrow 0$
  3. 현재 명제 단계에 적용될 수 있는 행동들에 대한 노드를 행동 단계  $k+1$ 에 만들고, 이들 행동의 effect에 있는 리터럴에 해당하는 노드들을 명제 단계  $k+2$ 에 만든다.
  4.  $k \leftarrow k+2$
  5. 현재 명제 단계  $k$ 에서 목표 상태에 있는 모든 리터럴들이 상호배타적이지 않은 상태로 있으면, 단계  $k$ 의 해당 리터럴을 부분목표로 하는 계획을 찾는다. 그렇지 않으면 단계 3으로 간다.
    - 5.1 단계  $k$ 의 각 부분목표에 대해서, 단계  $k-1$ 에서 해당 부분목표를 달성할 수 있는 행동을 선택한다.
    - 5.2 선택된 행동이 이미 선택된 단계  $k-1$ 의 다른 행동과 상호배타관계에 있으면, 해당 부분목표는 실패한 것으로 간주하고, 단계 3으로 간다.
    - 5.3 선택된 행동의 precondition에 있는 리터럴들이 상호배타관계에 있지 않으면, 해당 리터럴들을 부분목표로 하고,  $k \leftarrow k-2$ 로 하여 단계 6.1로 돌아가서 반복한다.
    - 5.4  $k = 0$ 이면, 선택된 행동들을 단계 1부터 순서대로 나열한 것이 계획이 된다.
-

# GraphPlan 알고리즘

## ❖ 예. 생일저녁 준비 문제

- 생일 날 집에서 자고 있는 누군가를 위해 생일 저녁을 준비하는 일을 계획
- 목적
  - 생일 저녁을 위해 주방의 쓰레기를 치우고, 선물과 저녁식사 준비
- 명제기호
  - garb : 쓰레기가 주방에 있다
  - dinner : 저녁식사가 준비되어 있다
  - present : 선물이 준비되어있다
  - clean : 손이 깨끗하다
  - quiet : 조용하다

# GraphPlan 알고리즘의 적용 예

## ❖ 생일저녁 준비 문제 – cont.

### ▪ 초기상태

- 주방에 쓰레기가 있고 손은 깨끗하고 주변은 조용
- $\text{garb} \wedge \text{clean} \wedge \text{quiet}$

### ▪ 목표 상태

- 주방에 쓰레기가 없고, 저녁식사와 선물이 준비된 상태
- $\neg \text{garb} \wedge \text{dinner} \wedge \text{present}$



# GraphPlan 알고리즘

## ❖ 생일저녁 준비 문제 – cont.

### ▪ 행동 (action)

- 요리하기(cook), 선물 포장하기(wrap), 손으로 치우기(carry), 손수레로 치우기(dolly)

Action : **cook**

Precondition : clean

Effect : dinner

요리하기를 할 때는 손이 깨끗해야 하고  
요리가 끝나면 저녁식사가 준비

Action : **wrap**

Precondition : quiet

Effect : present

선물 포장하기는 조용할 때 해야 하고  
결과는 선물이 준비된 것

Action : **carry**

Precondition : garb

Effect :  $\neg$ garb  $\wedge$   $\neg$ clean

손으로 치우기는 있는 쓰레기를 치워서  
쓰레기가 없어지게 하고 손은 지저분해짐

Action : **dolly**

Precondition : garb

Effect :  $\neg$ garb  $\wedge$   $\neg$ quiet

수레로 치우기는 있는 쓰레기를 치우는데  
소란한 소리가 남

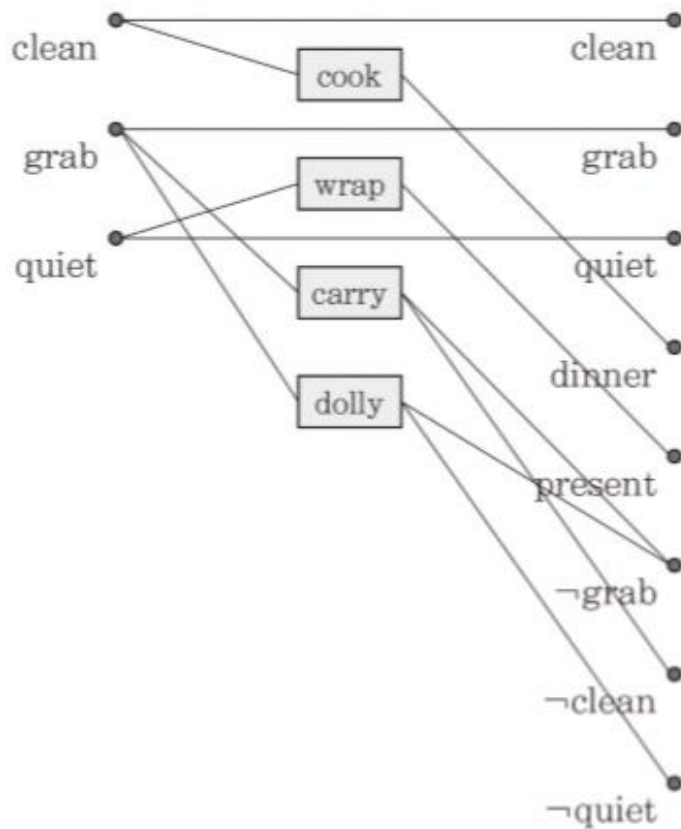
# GraphPlan 알고리즘

## ❖ 생일저녁 준비 문제 – cont.

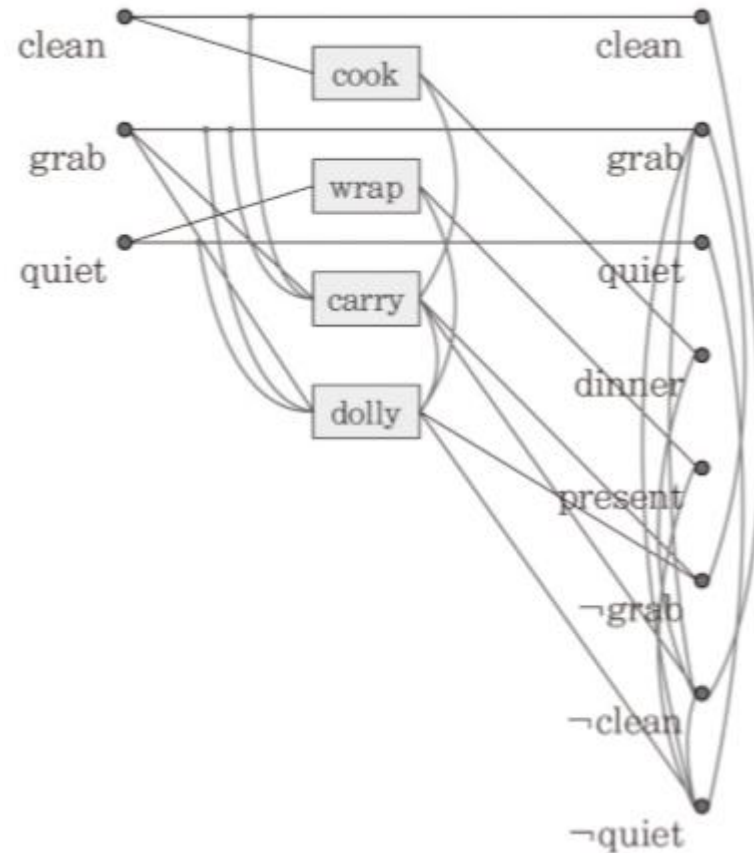
초기 상태의 리터럴

clean ●  
grab ●  
quiet ●

행동 적용 결과  
행동단계, 명제 단계 생성



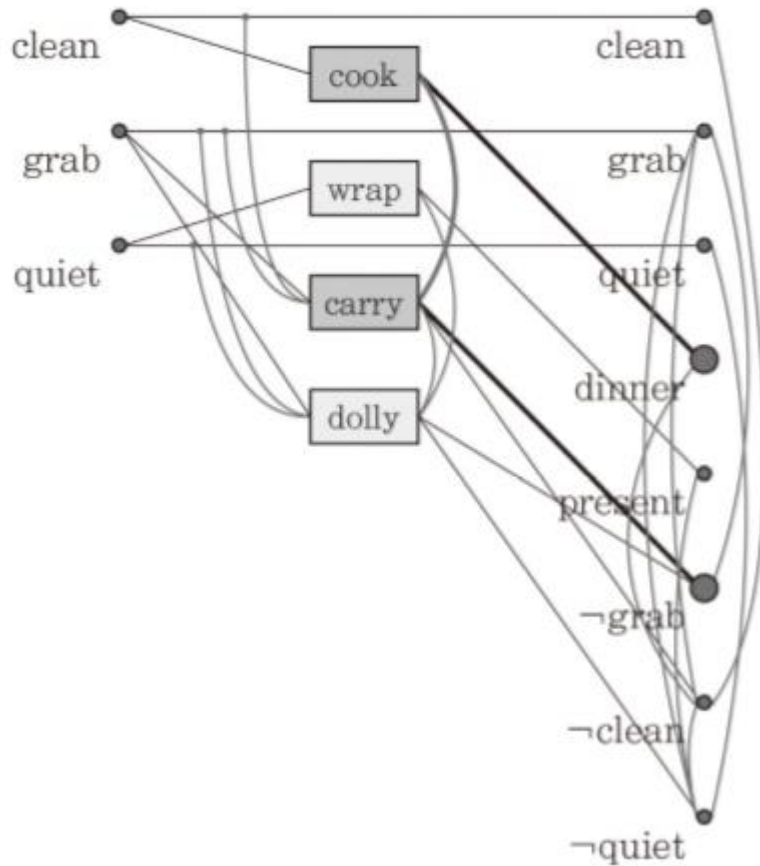
상호배제 관계 링크 추가



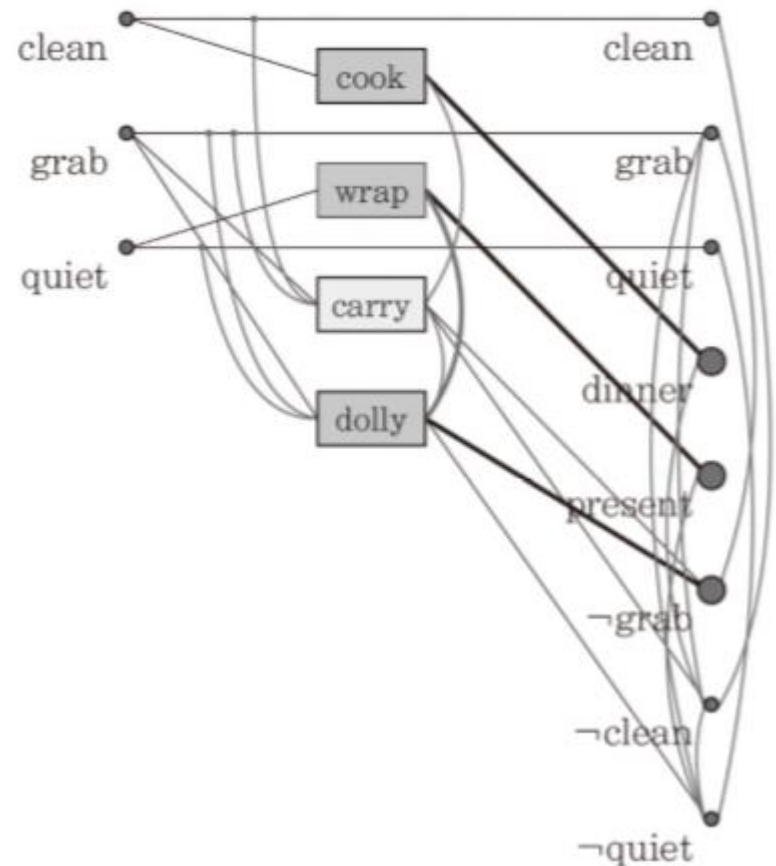
# GraphPlan 알고리즘

## ❖ 생일저녁 준비 문제 – cont.

명제 단계 1에서 목표상태 리터럴 확인 :  $\neg \text{grab} \wedge \text{dinner} \wedge \text{present}$



행동 충돌 : cook, carry



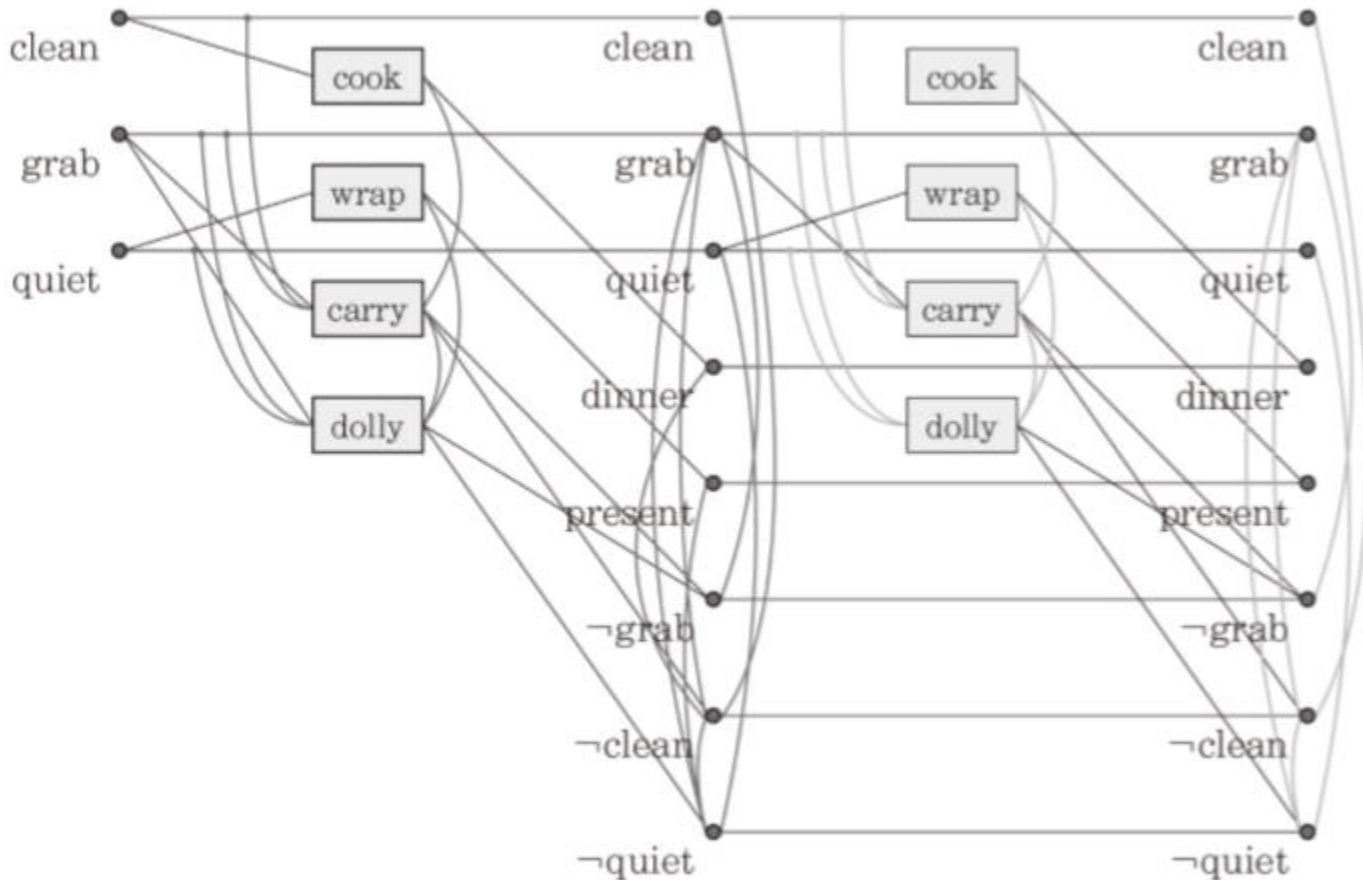
행동 충돌 : wrap, dolly

# GraphPlan 알고리즘

## ❖ 생일저녁 준비 문제 – cont.

- 그래프 확장

- 행동 단계 2와 명제 단계 2 확장



# GraphPlan 알고리즘

## ❖ 생일저녁 준비 문제 – cont.

### ▪ 계획 탐색

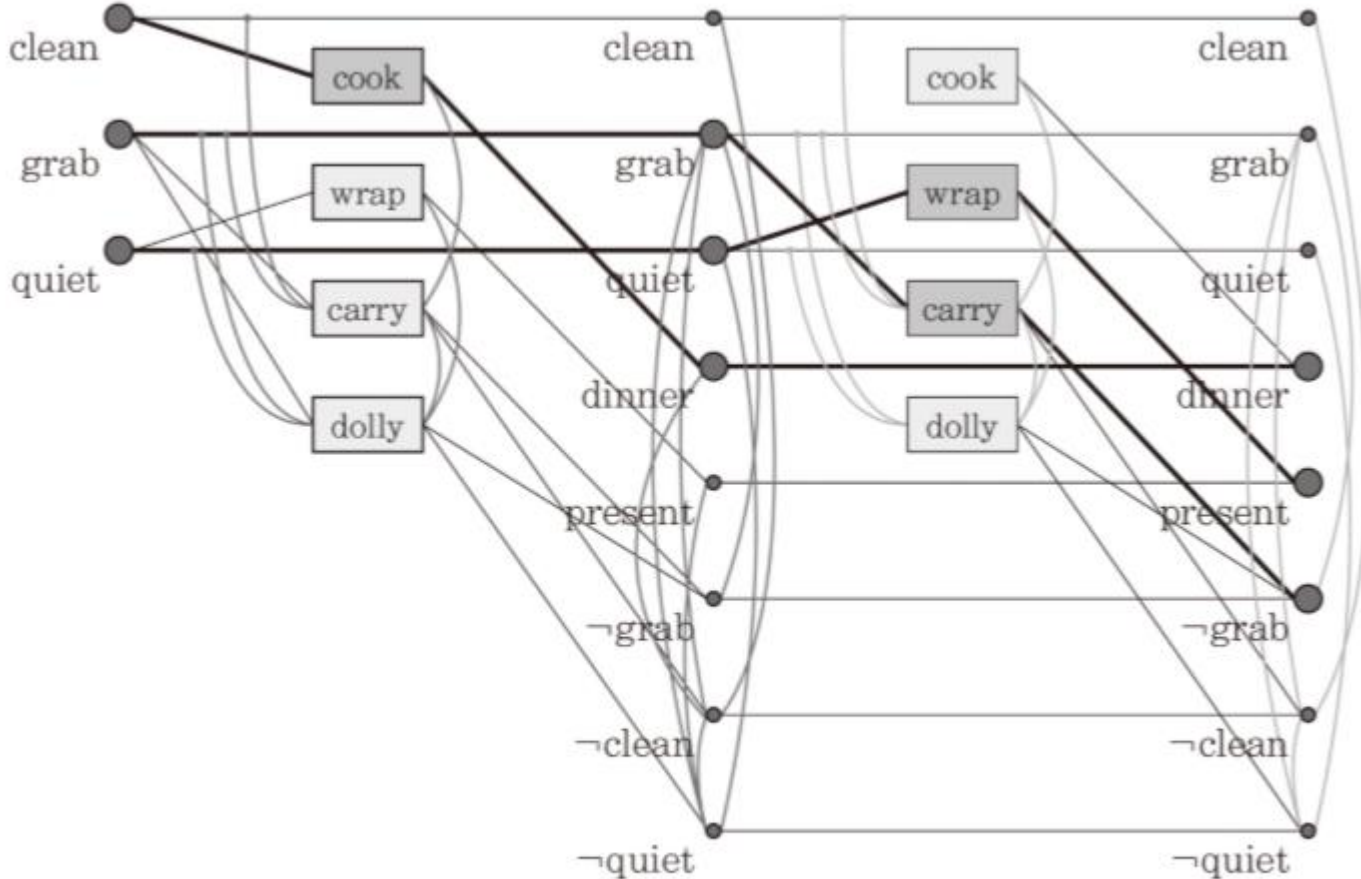


그림 6.11 목표 상태의 리터럴을 부분목표로 한 탐색 결과

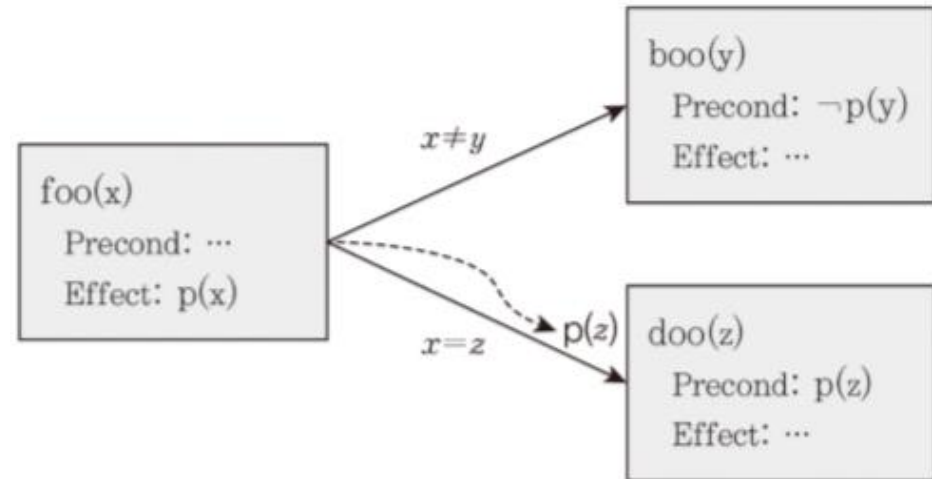
cook → {wrap, carry}



# 7. 계획공간 계획수립

## ❖ 계획공간 계획수립 (state-space planning)

- 탐색 공간이 **부분계획 (partial plan)**들로 구성
  - 부분적으로 값이 결정된 **행동 (partially instantiated actions)**의 집합
  - **제약조건 (constraints)**의 집합
    - 선행 (precedence) 제약조건 :  $a \ll b$
    - 바인딩 (binding) 제약조건
      - »  $V_1 \neq V_2$  or  $V \neq C$
      - »  $V_1 = V_2$  or  $V = C$
      - » 대체 (substitution)
    - **인과연결 (causal link)**
      - » 행동  $b$ 에서 필요한 선행조건  $p$ 를 만족시키기 위해 행동  $a$  사용
      - $a \xrightarrow{p} b$



- **해답 계획 (solution plan)**이 완성될 때까지 **점진적으로 개선 (refinement)**

# 계획공간 계획수립

## ❖ 위협(threat)

- 삭제 조건 상호작용(deleted-condition interaction)
- 행동  $a$ 가 행동  $b$ 의 사전조건  $p$ 를 생성하는 **인과연결**(causal link) 관계
- 행동  $c$ 가  $p$ 를 삭제하는 Effects 보유
- $c$ 가 인과관계  $a \xrightarrow{p} b$  를 위협하는 상황

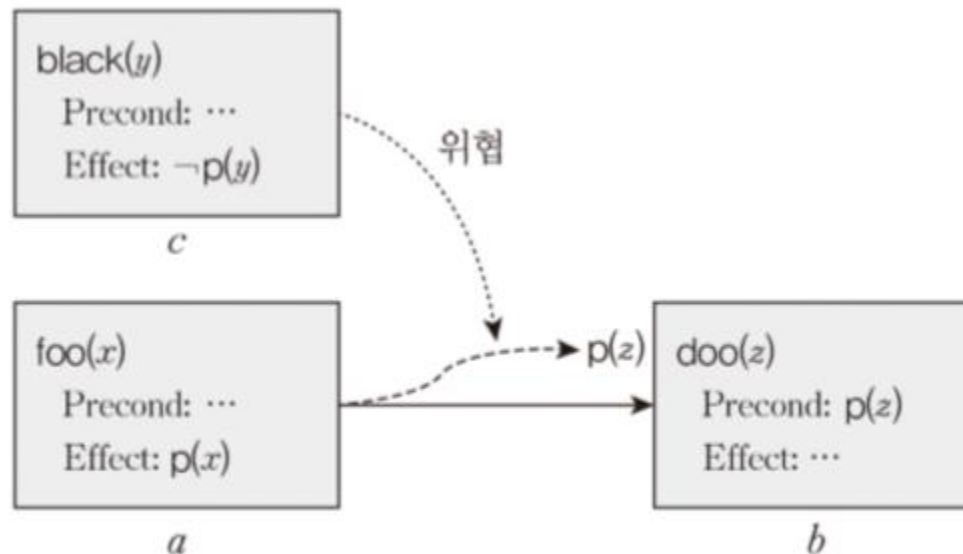


그림 6.14 위협 관계

# 계획공간 계획수립

## ❖ 예. 계획공간 계획수립

- 목적

- 공구점에서 드릴, 마트에서 우유, 빵을 사서 집에 돌아오는 계획

- 행동(연산자)

Operator: Start

Precondition: none

Effect: At(집), Sells(공구점,드릴), Sells(마트,우유), Sells(마트,빵)

Operator: Finish

Precondition: Have(드릴), Have(우유), Have(빵), At(집)

Operator: Go(l,m)

Precondition: At(l)

Effect: At(m),  $\neg$ At(l)

Operator: Buy(p,s)

Precondition: At(s), Sells(s,p)

Effect: Have(p)

- Start와 Finish는 초기 상태와 목적 상태를 나타냄



# 계획공간 계획수립

❖ 초기 계획(initial plan): Start, Finish, 선행 제약조건

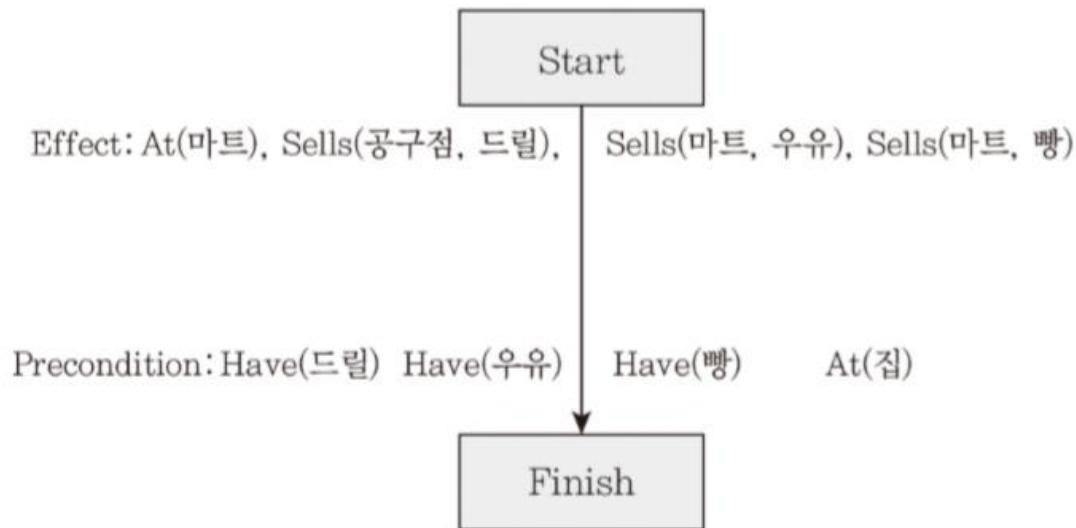


그림 6.15 계획공간 계획수립 과정 - 1

# 계획공간 계획수립

## ❖ 사전조건 Have를 만드는 3가지 행동

- 선행관계 제약조건(실선 화살표)과 인과관계 링크(점선 화살표) 추가

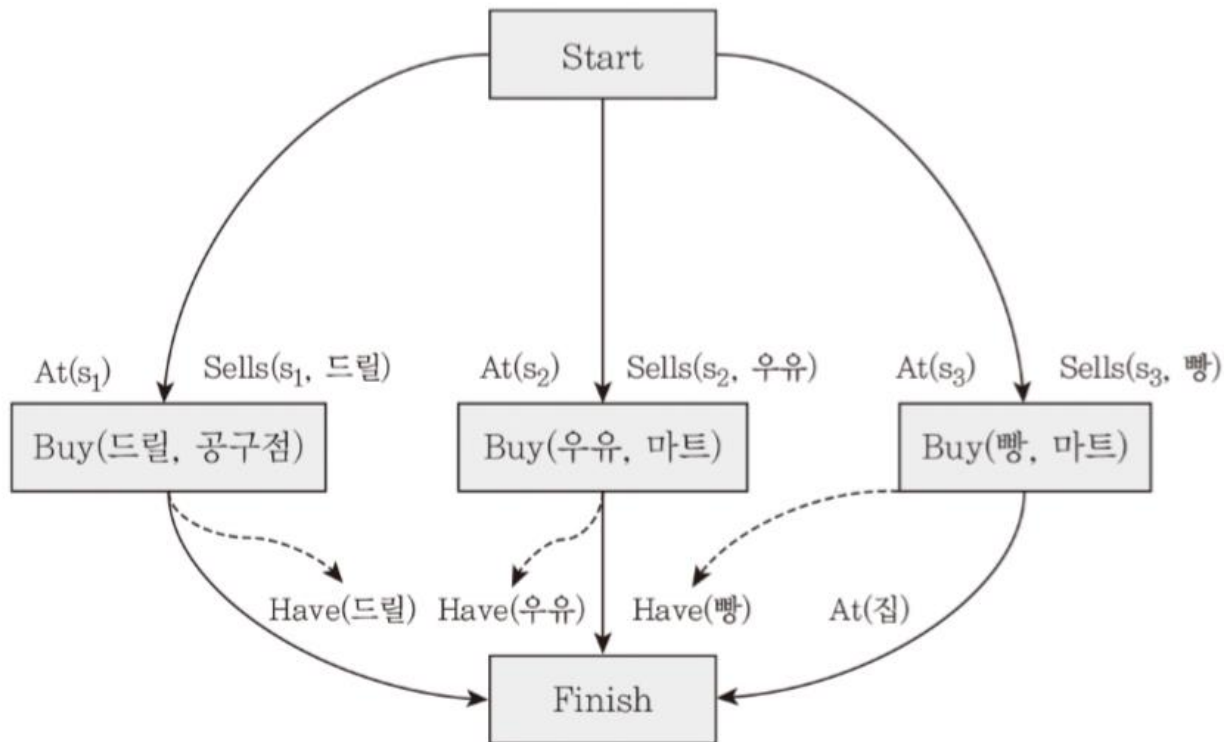


그림 6.16 계획공간 계획수립 과정 - 2

# 계획공간 계획수립

❖ 사전조건 Sells를 지지하는 **인과관계 링크** 추가

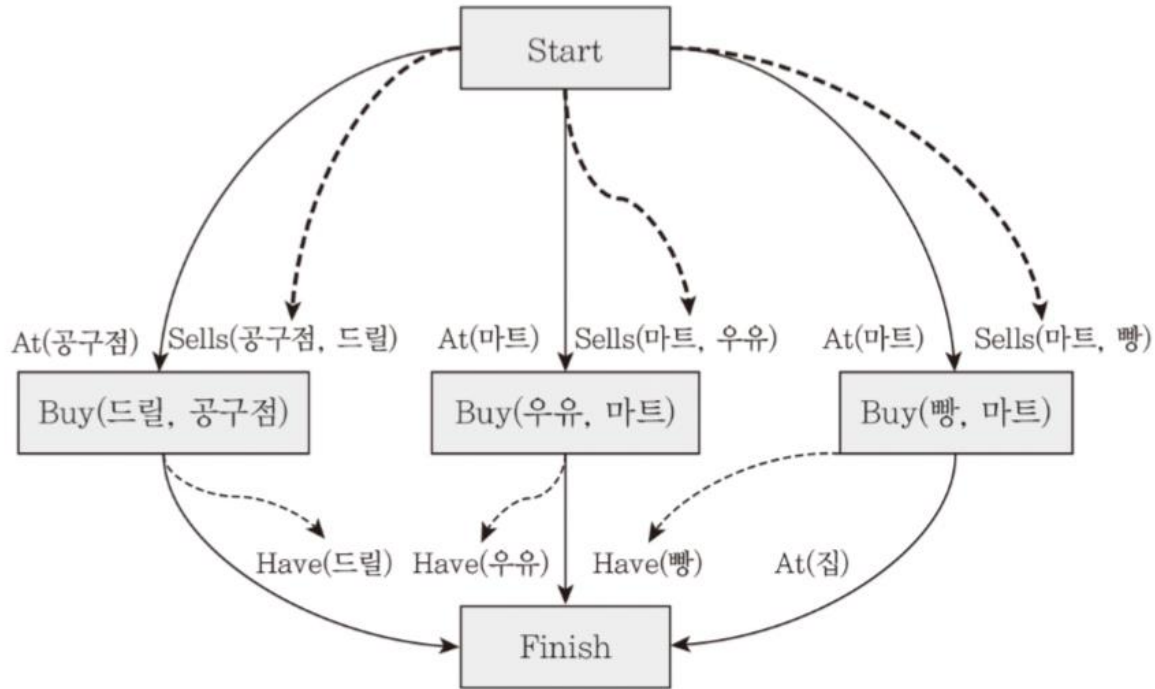


그림 6.17 계획공간 계획수립 과정 - 3

# 계획공간 계획수립

- ❖ At(공구점)와 At(마트)를 만들기 위해 Go 행동 추가
  - 위험 관계(가늘고 조밀한 점선 화살표)로 표시가 발생

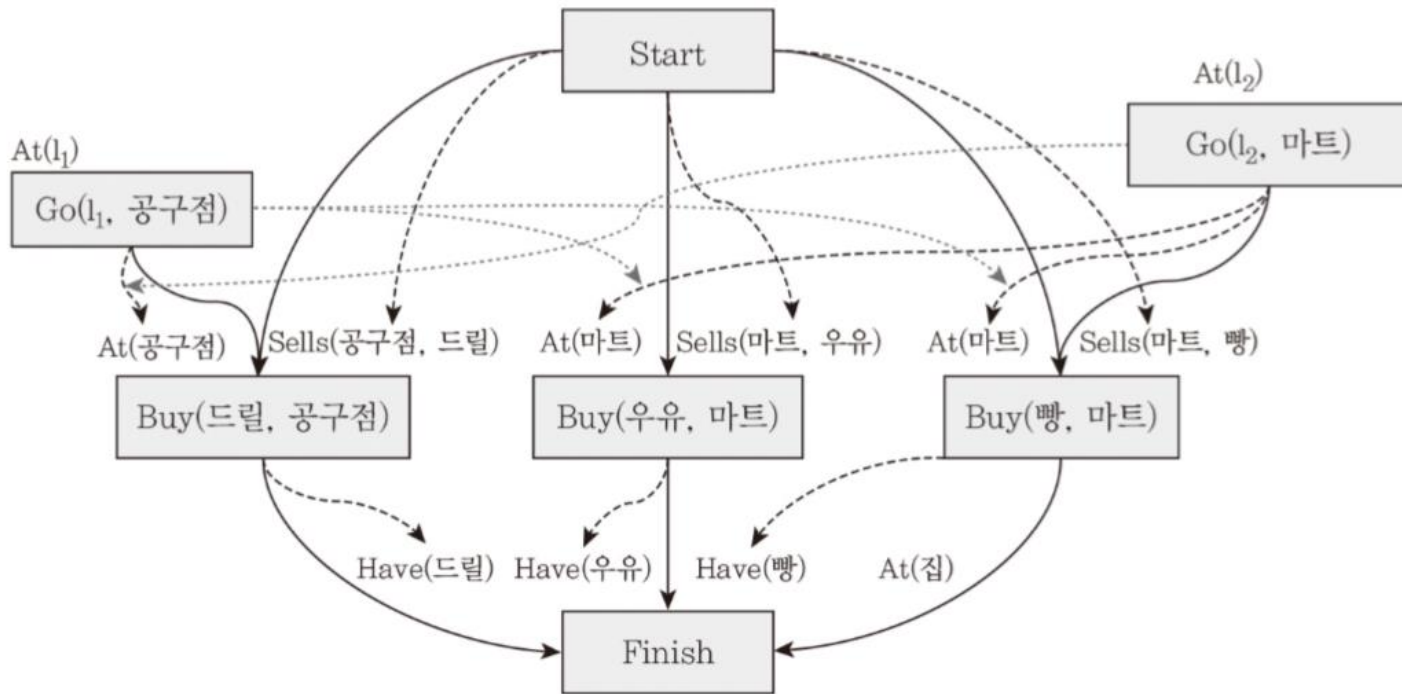


그림 6.18 계획공간 계획수립 과정 - 4

# 계획공간 계획수립

❖  $At(s_1)$ 의 위협 해결 : Buy(드릴)이  $Go(l_2, \text{마트})$ 보다 선행

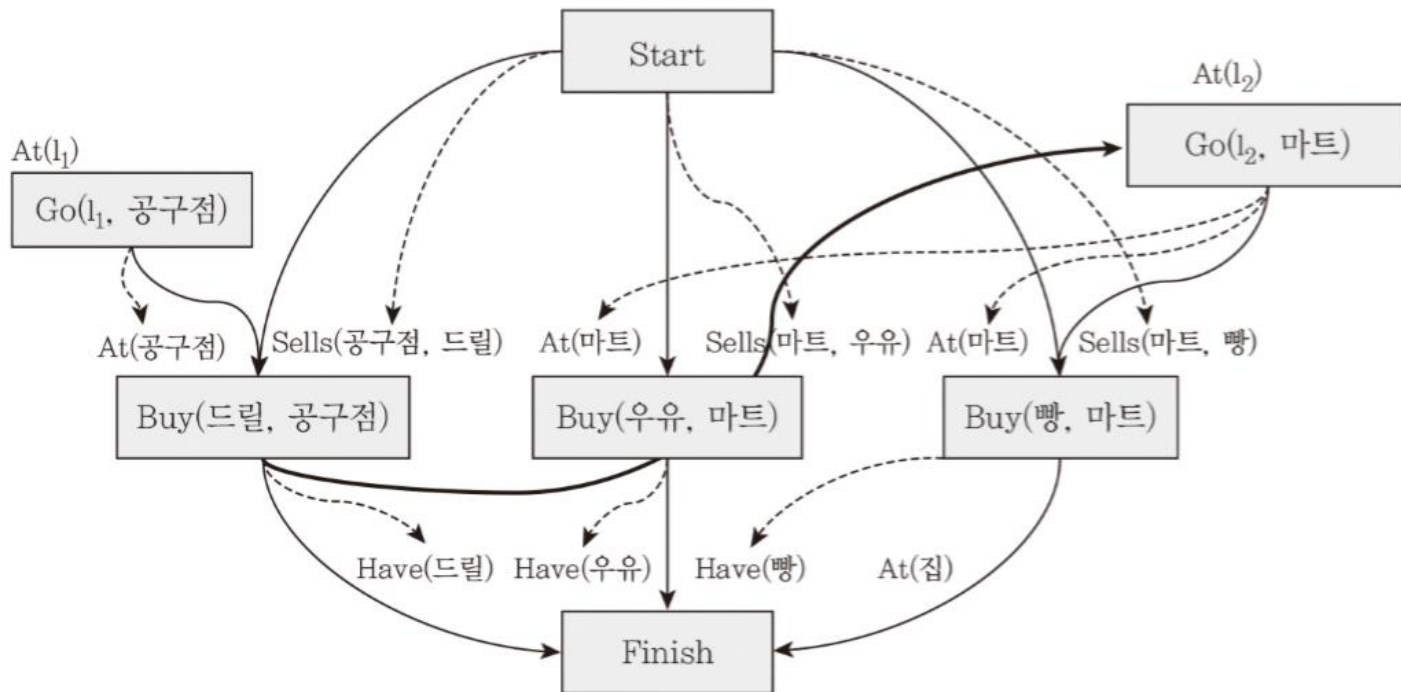


그림 6.19 계획공간 계획수립 과정 - 5

# 계획공간 계획수립

❖  $At(l_1)$ 의 결정 : Start에서  $l_1 = \text{'집'}$ 으로 결정

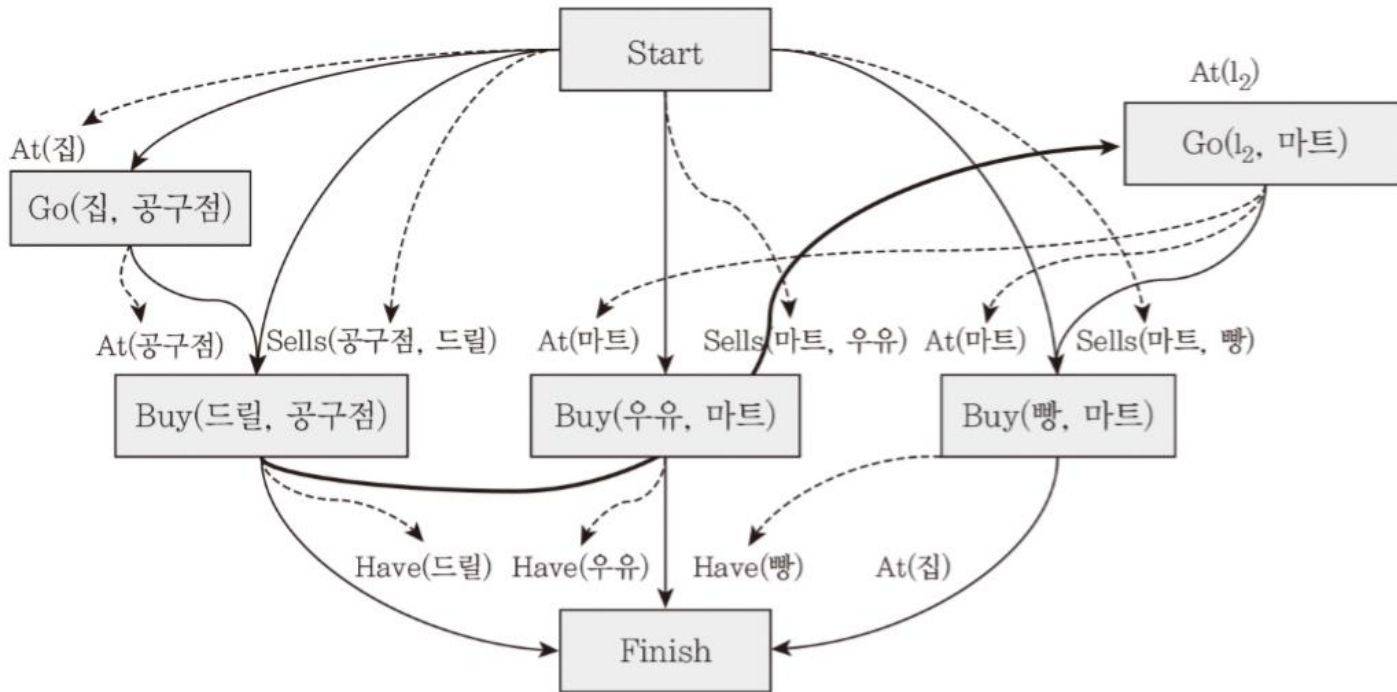


그림 6.20 계획공간 계획수립 과정 - 6

# 계획공간 계획수립

❖  $At(I_2)$ 의 결정 :  $Go(\text{집}, \text{공구점})$ 에서  $I_2 = \text{'공구점'}$ 으로 지정

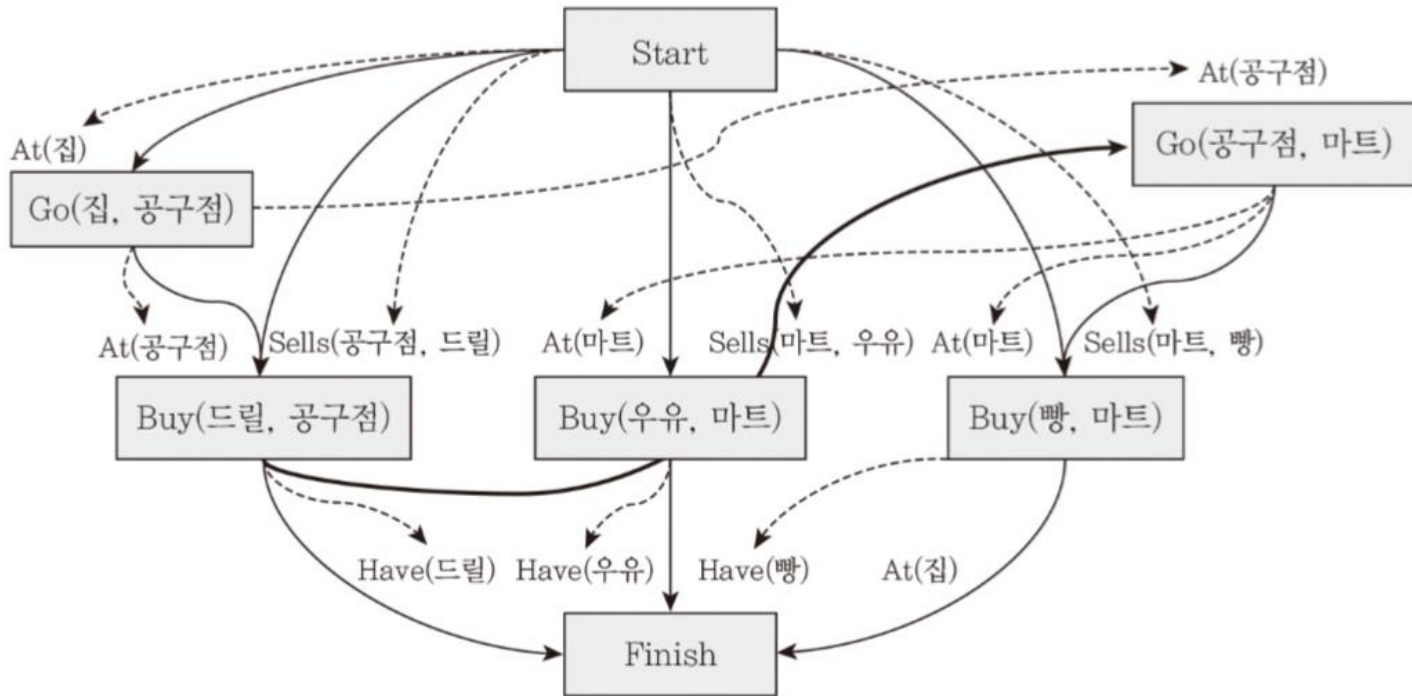


그림 6.21 계획공간 계획수립 과정 - 7

# 계획공간 계획수립

## ❖ Finish에서 At(집) 생성

- $Go(I_3, \text{집})$  추가  $\rightarrow$  여러 위협 생성

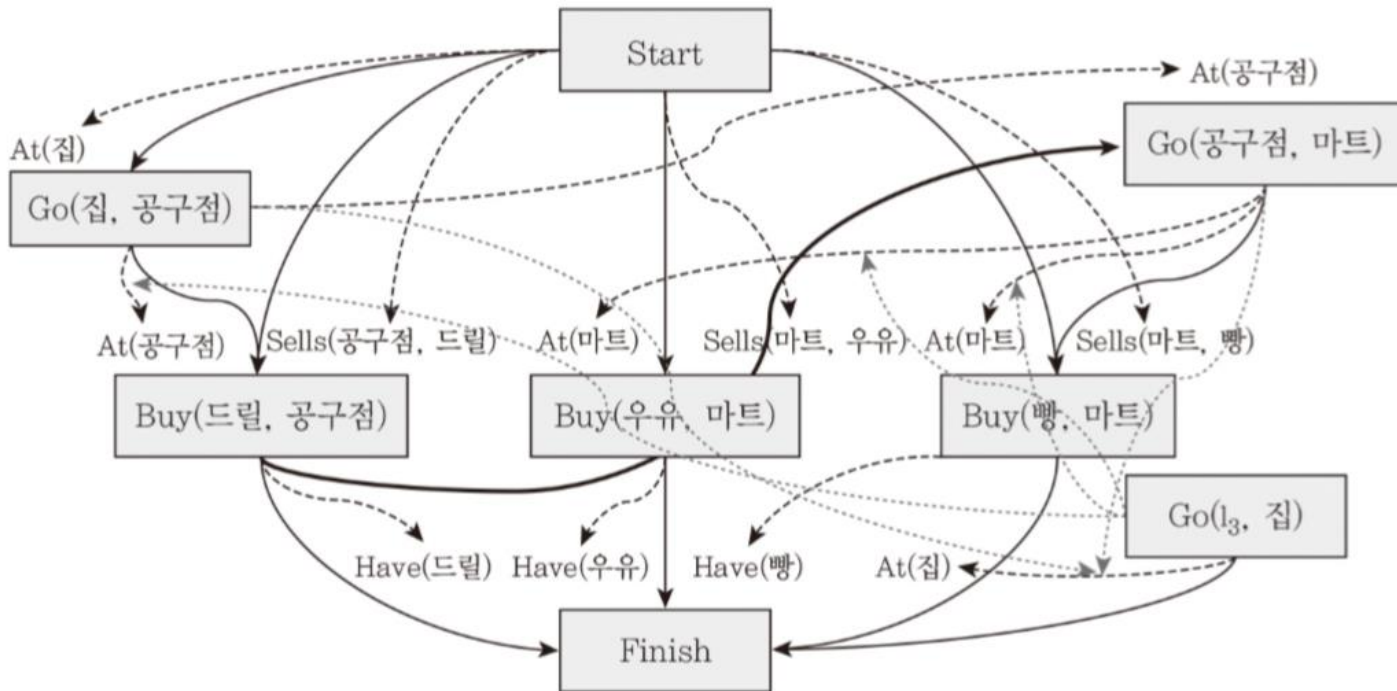


그림 6.22 계획공간 계획수립 과정 - 8



# 계획공간 계획수립

- ❖ At(마트), At(공구점)에 대한 위협 제거
  - Go(I<sub>3</sub>, 집) 보다 선행하게 설정

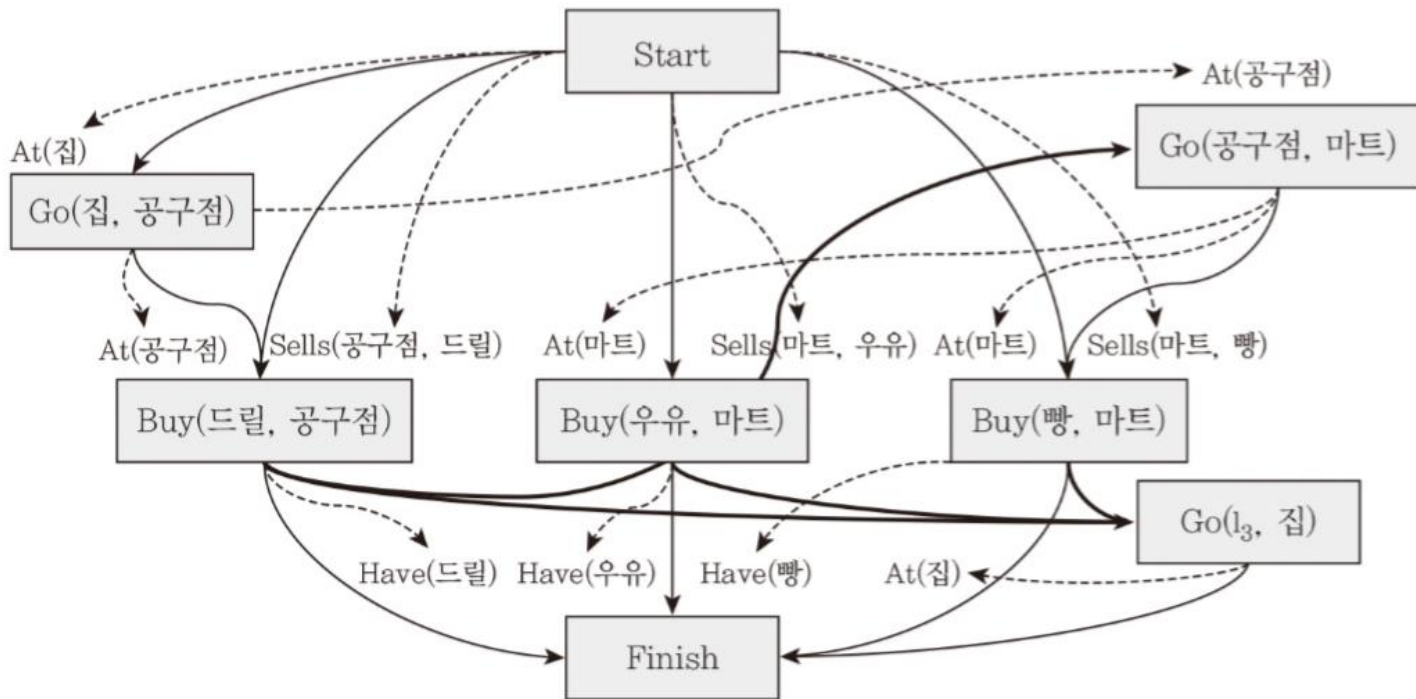


그림 6.23 계획공간 계획수립 과정 - 9

# 계획공간 계획수립

❖ 최종 계획 : At( $I_3$ )를  $I_3$ = '마트'로 설정

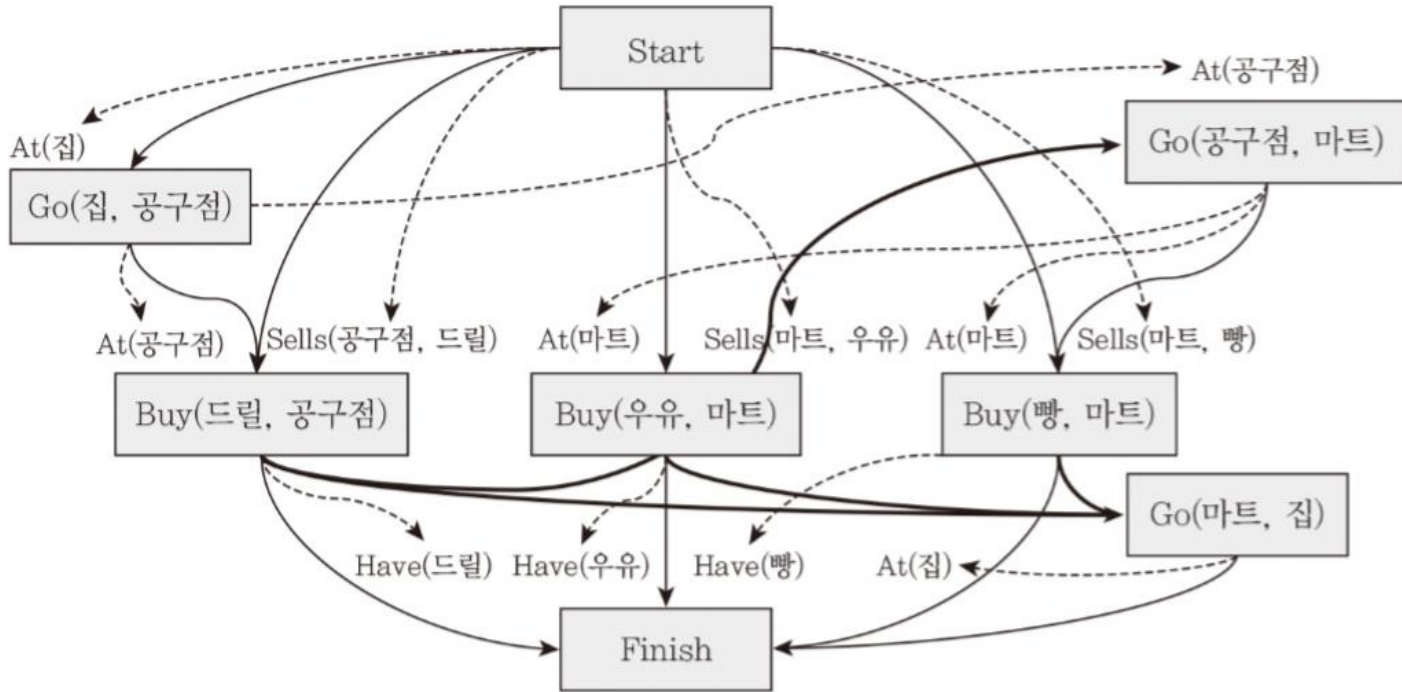


그림 6.24 계획공간 계획수립 과정 - 10

Start → Go(집, 공구점) → Buy(드릴, 공구점) → Go(공구점, 마트)  
 → Buy(우유, 마트) → Buy(빵, 마트) → Go(마트, 집) → Finish

# 계획공간 계획수립

---

## 알고리즘 6.2 계획공간 계획수립

1. 임의로 초기 부분계획을 생성한다.
  2. 미해결 목표이거나 위협 상황인 결함을 찾는다.
  3. 결함이 없으면, 현재 부분계획 내의 행동들을 선행관계 제약조건에 맞춰 일렬로 배치하여 계획으로 출력한다.
  4. 부분계획에 있는 미해결 목표가 있으면, 이를 해소하는 행동을 찾고, 이들 간에 인과관계 링크를 추가한다.
  5. 인과관계 링크에 대한 위협이 있으면, 행동 간의 선행관계 제약조건을 추가하거나 변수들에 바인딩 제약조건을 추가하여 위협을 해결한다.
  6. 단계 2로 돌아간다.
-

# 8. 계층적 계획수립

## ❖ 계층적 계획수립

- 복잡한 태스크를 더 단순한 태스크로 분할

## ❖ 계층적 태스크 네트워크(hierarchical task network, HTN) 방법

- 태스크(task)가 목표로 주어질 때, 이 태스크를 추상적 단계에서 분할하여 점차 구체적인 기본 작업들로 구성하여 계획을 수립하는 방법
- 주어지는 정보
  - 기본 태스크(primitive task) : 연산자(행동)에 의해 수행
  - 복합 태스크(non-primitive task)
    - 메소드(method)를 사용하여 더 작은 부분태스크(subtask)들로 분할하는 방법 표현
    - 복합 태스크는 여러가지 방법으로 분할 가능
    - 부분태스크 수행에 제약조건(constraint) 존재 가능

# 계층적 태스크 네트워크 HTN

## ❖ 계층적 태스크 네트워크 HTN

- 태스크  $\text{travel}(x, y)$ 에 대한 메소드(method)의 표현 예

method : **taxi-travel**(x,y)

task :  $\text{travel}(x,y)$

precondition :  $\text{short-distance}(x,y)$

subtasks :  $\text{get-taxi}$ ,  $\text{ride}(x,y)$ ,  $\text{pay-driver}$

constraints : {  $\text{get-taxi} \prec \text{ride}(x,y)$ ,  $\text{ride}(x,y) \prec \text{pay-driver}$  }

method : **air-travel**(x,y)

task :  $\text{travel}(x,y)$

precondition :  $\text{long-distance}(x,y)$

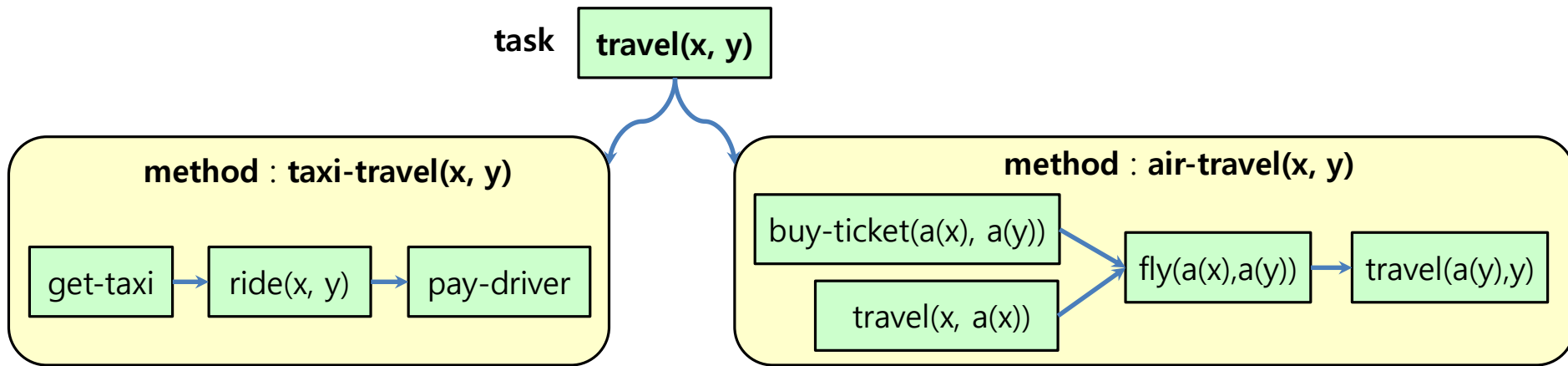
subtasks :  $\text{buy-ticket}(a(x),a(y))$ ,  $\text{travel}(x,a(x))$ ,  $\text{fly}(a(x),a(y))$ ,  
 $\text{travel}(a(y),y)$

constraints : {  $\text{buy-ticket}(a(x),a(y)) \prec \text{fly}(a(x),a(y))$ ,  
 $\text{travel}(x,a(x)) \prec \text{fly}(a(x),a(y))$ ,  
 $\text{fly}(a(x),a(y)) \prec \text{travel}(a(y),y)$  }

# 계층적 태스크 네트워크 HTN

## ❖ 계층적 태스크 네트워크 HTN – cont.

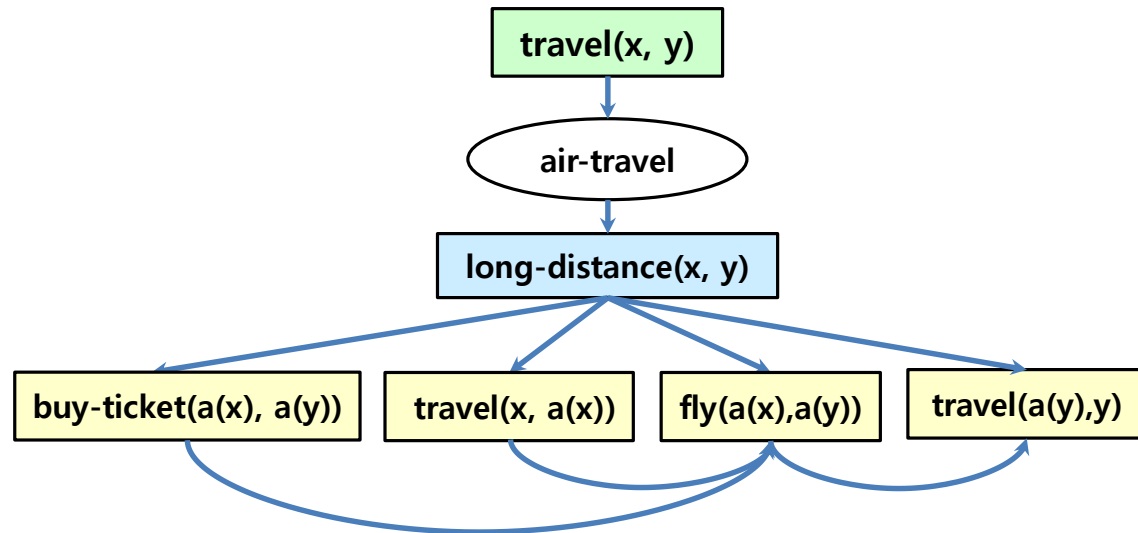
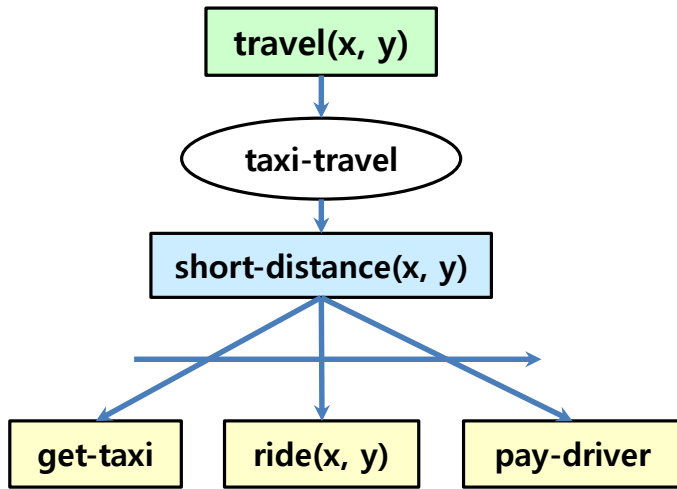
- 태스크 분할 방법을 표현하는 메소드
  - 하나의 태스크를 여러 방법으로 분할 가능



# 계층적 태스크 네트워크 HTN

## ❖ 계층적 태스크 네트워크 HTN – cont.

- 메소드의 네트워크 표현



# 계층적 태스크 네트워크 HTN

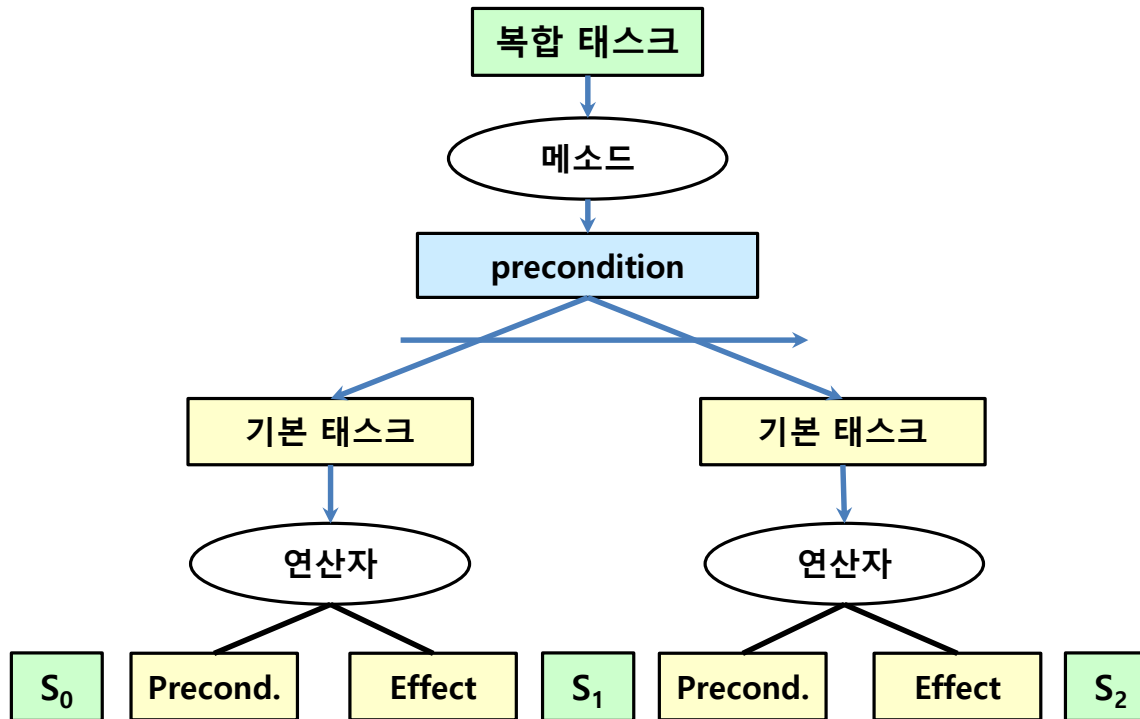
## ❖ 계층적 태스크 네트워크 HTN 계획수립

- 주어진 복합 태스크에 대해 조건에 맞는 메소드를 찾아 적용, 기본 태스크로 표현될 때까지 분할
- **계획구성**
  - 기본 태스크를 수행하는 연산자를 기본 태스크에 부여된 **순서관계**에 따라 순차적으로 **나열**
- **계획실행**
  - 초기 상태에서 순차적으로 연산자를 적용하면 최종적으로 주어진 태스크가 완료되는 목표 상태에 도달



# 계층적 태스크 네트워크 HTN

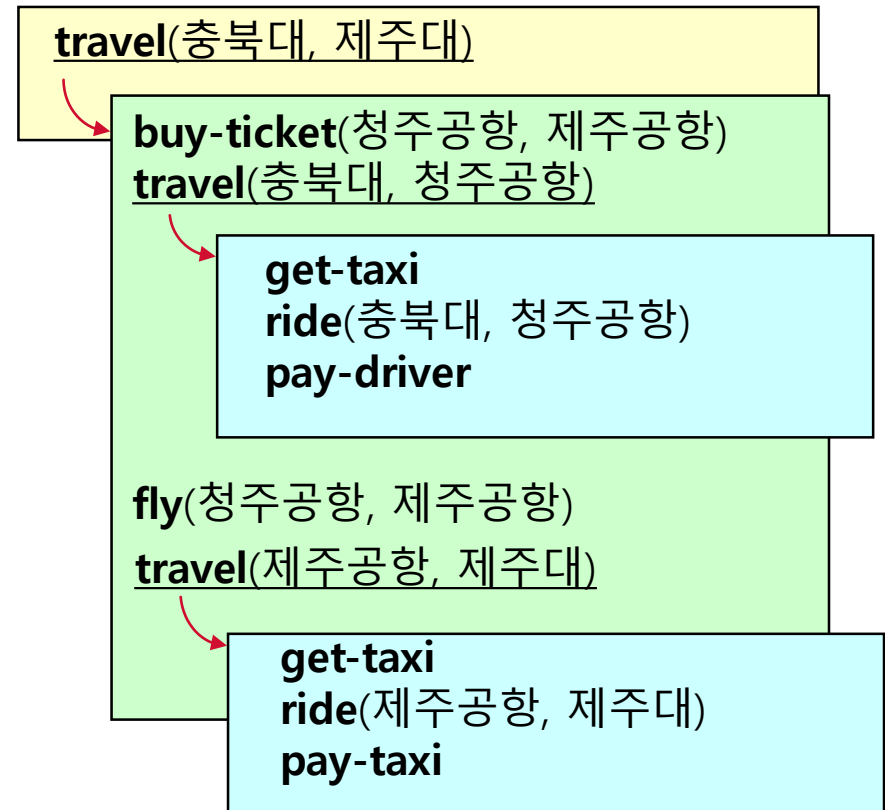
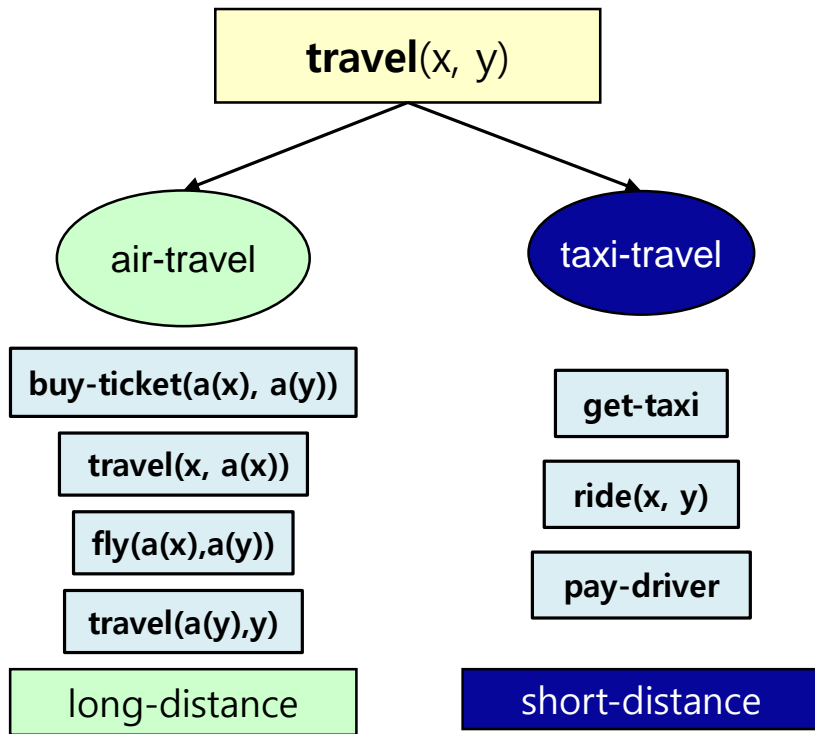
- ❖ 계층적 태스크 네트워크 HTN – cont.
  - 계층적 계획수립 형태



# 계층적 태스크 네트워크 HTN

## ❖ HTN 적용 예

- travel(충북대, 제주대)에 대한 계획



buy-ticket(청주공항, 제주공항) → get-taxi → ride(충북대, 청주공항) → pay-driver → fly(청주공항, 제주공항) → get-taxi → ride(제주공항, 제주대) → pay-driver