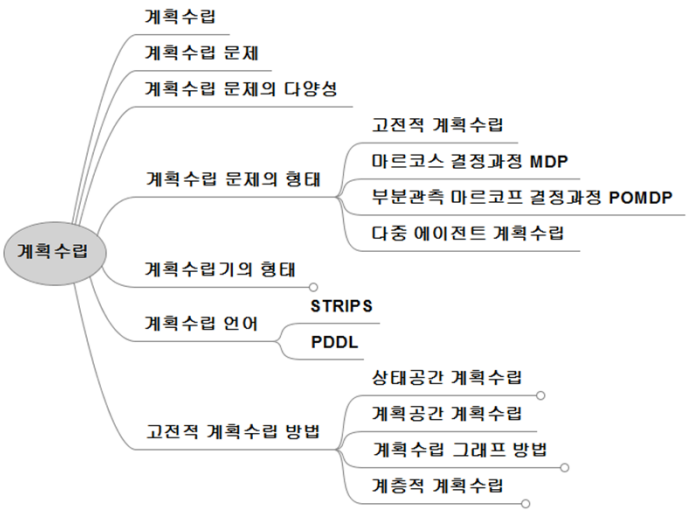
계획수립

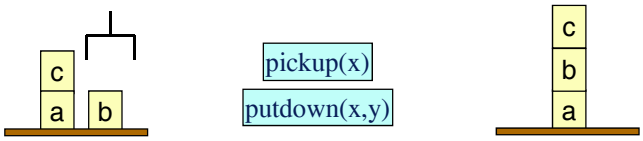
충북대학교 소프트웨어학과
이건명



1. 계획수립(planning)

❖ **계획수립(planning)**

- 주어진 계획수립 문제의 임의의 **초기 상태**에서 **목표 상태** 중의 하나로 도달할 수 있게 하는 **일련의 행동**들을 생성하는 것



초기 상태 (initial state) | **행동/연산자 (operator)** | **목표 상태 (goal state)**

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

계획수립

❖ **계획수립 문제의 구성요소**

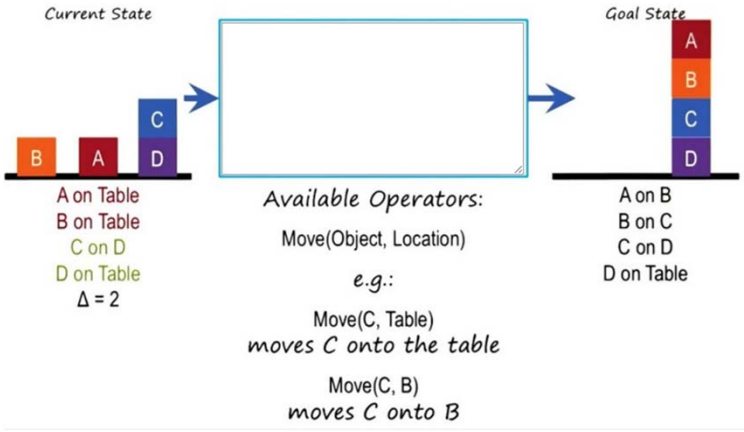
- 가능한 **초기 상태(initial state)**에 대한 기술(description)
- 원하는 **목표 상태(goal state)**에 대한 기술
- 가능한 **행동(action)**들에 대한 기술

❖ **에이전트(agent)**

- **위임받은 일**을 **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

2. 계획수립 문제

❖ **수단-목표 분석**(means-ends analysis)의 **적용예**
- 예. 블록이동 문제



Source: udacity

계획수립 문제

❖ **작업 수행 절차 계획**
❖ **로봇의 움직임 계획**

Image : spir.al, gamma.cs.unc.edu

계획수립 문제

❖ **상태 공간 그래프**(state space graph)

- **선교사-식인종 문제**

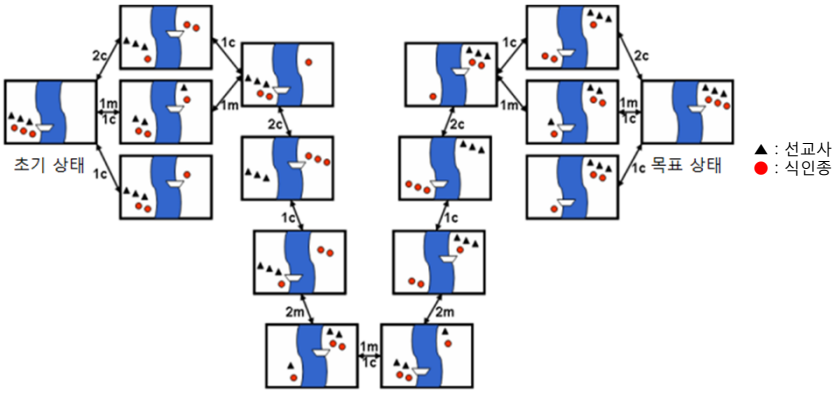
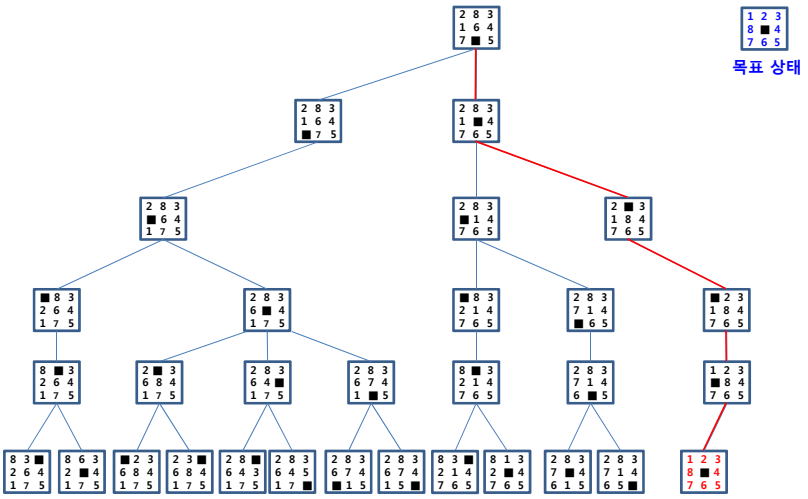


image : Gillian Mosley

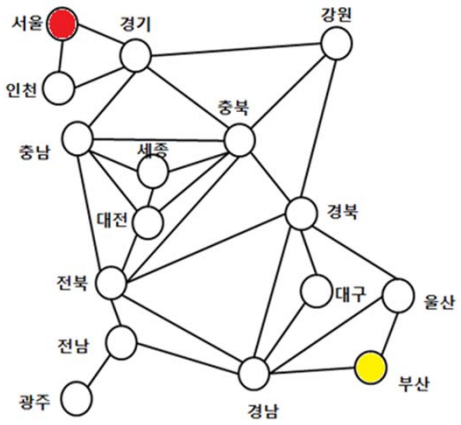
계획수립 문제

❖ **8-퍼즐 문제**



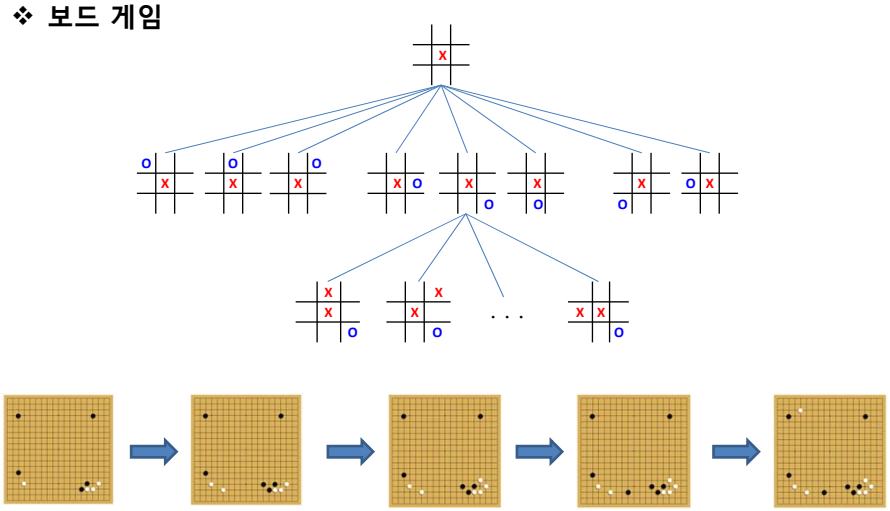
계획수립 문제

❖ 최단경로 문제



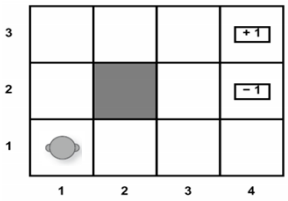
계획수립 문제

❖ 보드 게임

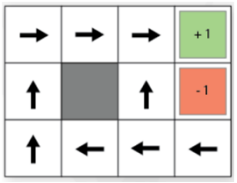


계획수립 문제

❖ 강화학습
- 정책(policy)의 학습



$c = -0.04$



계획수립 문제

❖ 생산 계획
- 강판 벤딩(steel sheet blending) 작업계획
  - 상태공간 탐색, A* 알고리즘

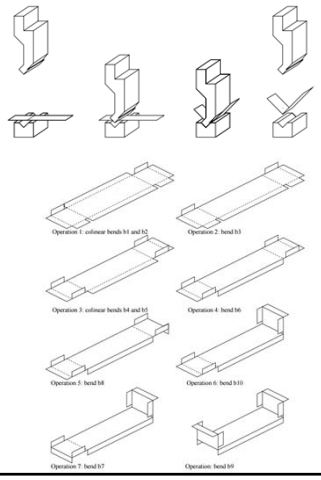


image : S. K. Gupta, et al. 1998

계획수립 문제

❖ 브리지 게임(bridge game)
- 4명이 하는 트럼프 게임
- Bridge Baron

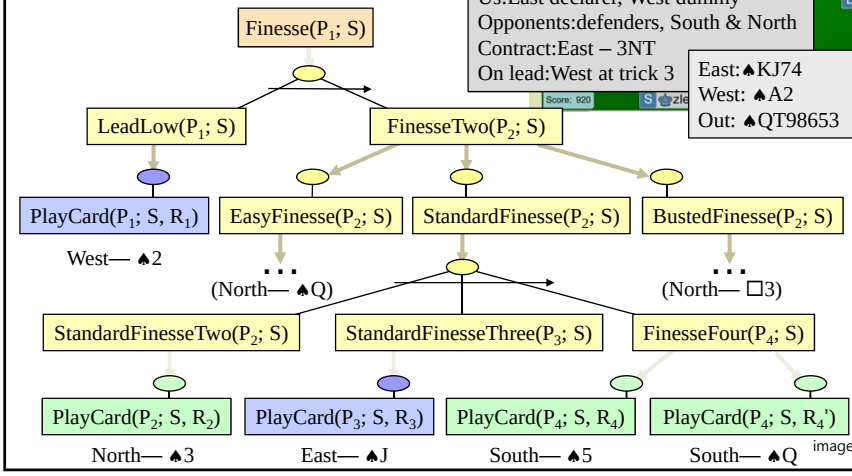


image: S. Smith et al. 1998

계획수립 문제

❖ 우주 탐사 로봇
- 화성 탐사로봇(Mars Exploration Rover: MER)
  - 자율 계획수립 및 제어

image : NASA

계획수립 문제

- ❖ 스케쥴링 (scheduling)
- ❖ 프로젝트 관리(project management)
- ❖ 군사작전 계획 (military operations planning)
- ❖ 정보 수집 (information gathering)
- ❖ 자원 관리 (resource control)
- ❖ …

3. 계획수립 문제의 다양성

- ❖ **계획수립 문제 형태의 다양성**
  - **행동**(action)의 **결과**가 **결정적**(deterministic)인가 **비결정적**(nondeterministic)인가
    - 행동에 대한 결과가 일정한가 그렇지 않은가
    - 비결정적일 때는 이에 대한 확률정보가 있는가
  - **상태 변수**(state variable)를 **이산적**(discrete)인가 **연속적**(continuous)인가
    - 이산적이라면 가능한 값의 개수가 유한한가
  - **현재 상태**(current state)가 명확히 알 수 있는가
    - 정확히 알 수 있는가 간접적인 정보만 알 수 있는가
  - **초기 상태**(initial state)의 **개수**가 얼마나 되는가
    - 유한한가 아니면 무수히 많은가
  - 행동(action)은 **지속시간**(duration)이 있는가

계획수립 문제의 다양성

❖ **계획수립 문제 형태의 다양성** – cont.

- **여러** 개의 **행동**을 **동시**에 할 수 있는가, 아니면 한 번에 하나의 동작만 하는가
- **계획의 목적**이 지정된 **목표 상태**에 도달하는 것인가, 아니면 **보상함수** (reward function)를 **최대**로 하는 것인가
- **에이전트**(예, 로봇)가 **하나**인가 **여러 개** 있는가
- 에이전트들이 서로 **협력**(cooperative)하는가 **이기적**(selfish)인가
- 에이전트 **각자**가 **자신의 계획**을 만드는가, 아니면 **전체 에이전트**를 위한 **하나의 계획**을 만드는가

4. 계획수립 문제의 형태

❖ **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 문제의 형태

❖ **고전적 계획수립 문제**(classical planning problem)

- 가장 간단한 계획수립 문제 부류
- **기본 전제**
  - **초기 상태**는 **하나**만 주어진다.
  - **행동**들은 **지속시간이 없고**, **행동의 결과가 결정적**이고, **한 번**에 **하나의 행동**만 수행될 수 있다.
  - 행동을 하는 **에이전트**는 **하나**뿐 이다.
- 일련의 **행동**들을 **수행**한 이후의 세계(world)의 **상태 예측** 가능
- **계획**은 **일련의 행동**들로 정의
  - 목표상태에 도달하기 위해 어떤 행동들을 해야 하는지 미리 결정할 수 있음

계획수립 문제의 형태

❖ **마르코프 결정과정 문제(**Markov Decision Process, MDP)

- **행동**들의 **결과**는 **비결정적**(nondeterministic)이고, 에이전트가 **행동**을 **통제**할 수 있는 문제
- **이산시간 마르코프 결정과정 문제**(Discrete-time Markov decision processes, discrete-time **MDP**)
  - 행동들은 **지속시간**이 **없다**
  - **행동의 결과**가 **확률**에 따라 결정되어 **비결정적**이다.
  - 행동의 결과는 **관측 가능**하여 확인할 수 있다.
  - **보상함수**(reward function)를 최대화하는 것을 목적으로 한다.
  - 행동을 하는 **에이전트**는 **하나**뿐 이다.

  ⇒ **강화 학습**

계획수립 문제의 형태

❖ **마르코프 결정과정 문제(**Markov Decision Process, MDP) – cont.

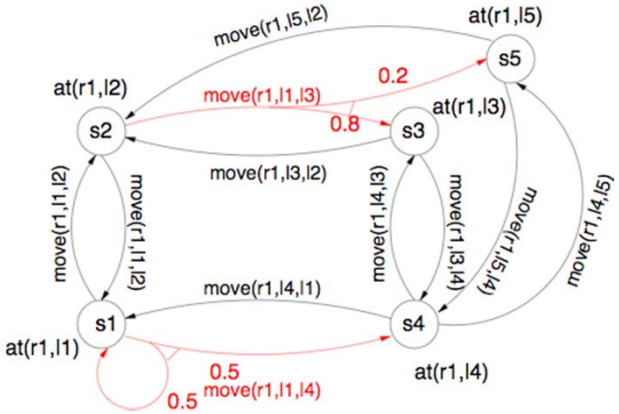


image : Dana Nau

계획수립 문제의 형태

❖ **부분관측 마르코프 결정과정**
(partially observable Markov decision process, **POMDP**)

- **행동의 결과**가 **확률**에 따라 **결정**되는 비결정적인 마르코프 결정과정
- 행동의 결과는 **부분적으로(간접적으로) 관측**

$P(s'|s,a)$: 상태 s에서 행동 a를 할 때 상태 s'이 될 확률

$P(o|a,s')$: 행동 a를 하여 상태 s'이 될때 o가 관측될 확률

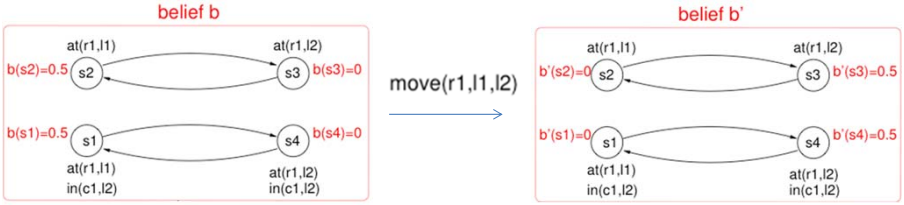


image : Dana Nau

계획수립 문제의 형태

❖ **다중 에이전트 계획수립 문제**(multi-agent planning)
- **여러 에이전트**가 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

http://gamma.cs.unc.edu/research/crowds/

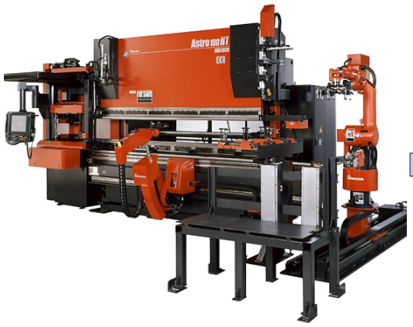
5. 계획수립기의 형태

❖ **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



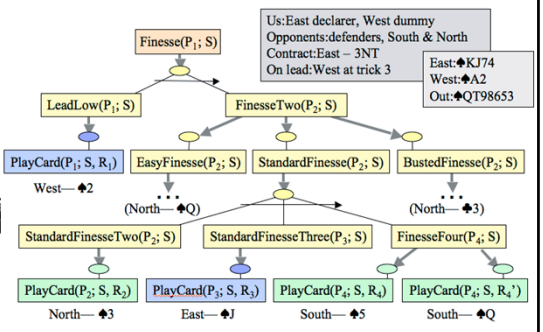


image : Dana Nau

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 제한**을 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 계획수립기**를 사용하면서
  해당 영역의 문제를 해결하는 방법에 관한 **정보**를 **입력**으로 사용
- **계층적 태스크 네트워크**(hierarchical task network, HTN) 계획수립

6. 계획수립 언어

❖ **계획수립 언어**

- **계획수립 문제**를 **표현**하는데 사용하는 언어
- 고전적 계획수립 문제를 표현하는 언어
  - **STRIPS, PDDL** 등
  - **리터럴**로 표현되는 **상태변수**(state variable)가 중심
  - 에이전트를 포함한 **세계(world)의 상태**는 상태변수에 값을 지정하여 표현
  - **행동(action)**에 대한 표현은 **행동전후 상태변수 값의 변화 내용**을 기술
  - **상태변수**들이 **상태공간(state space)**을 결정
    - 상태변수들의 개수가 늘어나면 **상태공간의 크기**는 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 **언어**의 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

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

계획수립 언어

❖ **STRIPS – cont.**
- **목표**(goal)
  - **리터럴**들의 **논리곱**으로 표현
  - **부정 리터럴** 및 **존재한정사**가 붙은 것으로 간주되는 **변수** 포함 가능
  - 예. '**집에 있는데 바나나가 없다**'와 '**어디에선가 바나나를 판다**'

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

  - 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 **변수**들
  - **precondition** (사전조건)
    - 행동을 **실행**하기 **전**에 **만족**돼야 하는 **조건** 기술
    - 함수를 사용하지 않는 **리터럴의 논리곱**으로 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
    - **부정 리터럴**들은 행동 실행으로 더 이상 만족되지 않는 성질 표현
    - 긍정 리터럴들을 **add-list**에 나타내고, 부정 리터럴들은 부정기호(¬)을 떼버리고 **delete-list** 표현하기도 함

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)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**, **행동**에 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 **not이 되는 부분**이 STRIPS에서 **delete-list**에 해당
  - 그렇지 않은 부분은 **add-list**에 해당
  - **?**로 시작하는 것은 **변수**

```
(define (domain gripper-strips)
    (: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)))))
```

6. 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

6.1 상태공간 계획수립

❖ **상태공간 계획수립**(state-space planning)

- **상태공간**(state space) 상의 **초기 상태**에서 **목표 상태**로의 **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

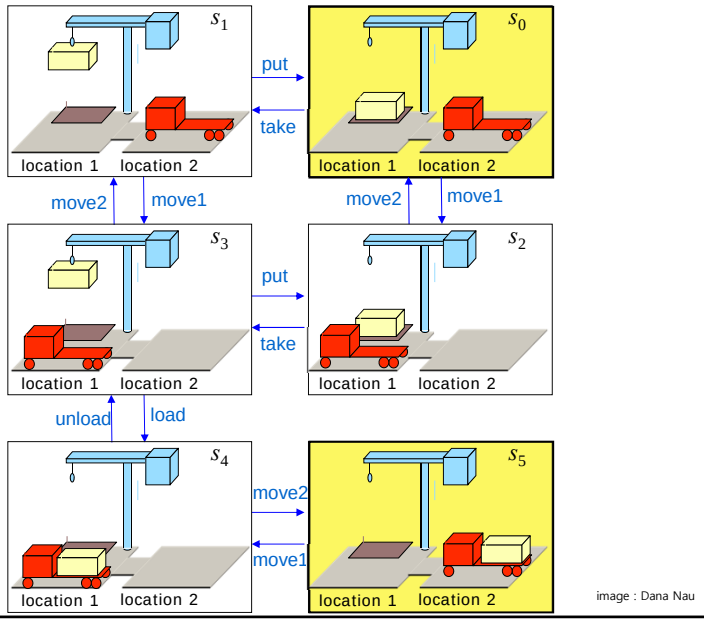
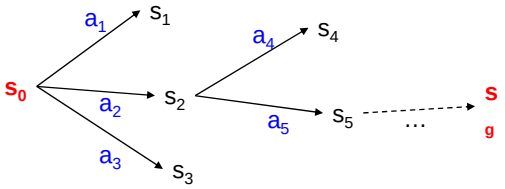


image : Dana Nau

상태공간 계획수립

❖ **전향 탐색(Forward Search)**

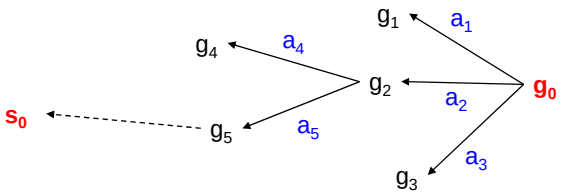
- 초기 상태에서 시작
- 적용가능한 연산자를 목표 상태에 도달할 때까지 적용
- 다양한 알고리즘 적용 가능
  - **너비우선 탐색**(Breadth-first search)
  - **깊이우선 탐색**(Depth-first search)
  - 휴리스틱 탐색 : **A* 알고리즘**



상태공간 계획수립

❖ **후향 탐색(Backward Search)**

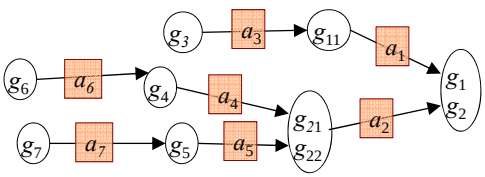
- **목표 상태**에서 **시작**
- 해당 상태를 만들어내는 행동 선택을 **시작 상태**에 **도달**할 때까지 반복



상태공간 계획수립

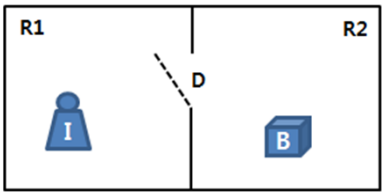
❖ **STRIPS 알고리즘**

- 기본적으로 **후향 탐색 방법**
- **목표 상태**가 **만족되지 않으면**,
  목표 상태를 effect로 만들 수 있는 **연산자 선택**하여
  연산자의 **매개변수**를 **설정**하고
  precondition이 만족되는지 **확인**
- precondition 중에 **만족되지 않는 것**이 있으면,
  그것을 effect로 하여 **위 과정 반복**
- 모든 precondition들이 만족되면
  사용된 **매개변수 설정된 연산자들**를 **역순**으로 **나열**하여 **계획생성**



상태공간 계획수립

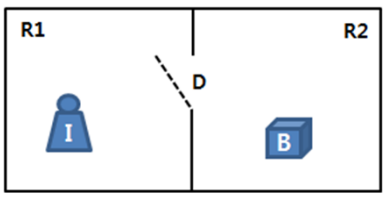
❖ **STRIPS 알고리즘의 적용 예**



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

상태공간 계획수립

❖ **STRIPS 알고리즘 적용 예**

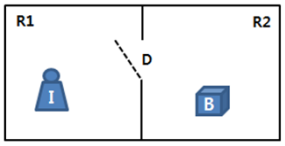


- **초기상태**
  InRoom(I, R1)
  InRoom(B, R2)
  Connects(D, R1, R2)
  Connects(D, R2, R1)
  Status(D, Open)
- **목표상태**
  InRoom(I, R2)
  InRoom(B, R2)
  Connects(D, R1, R2)
  Connects(D, R2, R1)
  Status(D, Open)

상태공간 계획수립

❖ **STRIPS 알고리즘 적용 예**

- **연산자(operator)**

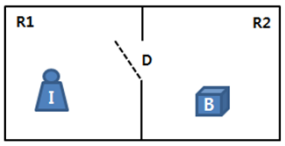


Operator: **GoToDoor**(I, dr)
Precondition:
InRoom(I, ra), Connects(dr, ra, rb)
Effect:
add-list: NextTo(I, dr)
delete-list:

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)

상태공간 계획수립

❖ **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)

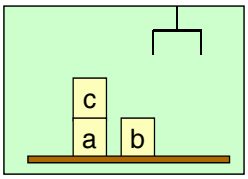
Operator: **GoToDoor**(I, dr)
Precondition:
InRoom(I, ra), Connects(dr, ra, rb)
Effect:
add-list: NextTo(I, dr)
delete-list:

dr = D
InRoom(I, ra), Connects(D, ra, rb)
ra = R1, rb = R2
InRoom(I, R1), Connects(D, R1, R2)

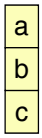
GoToDoor(I, D) → GoThruDoor(I, D)

상태공간 계획수립

❖ **Sussman 이상**(Sussman Anomaly)



초기 상태



목표 상태

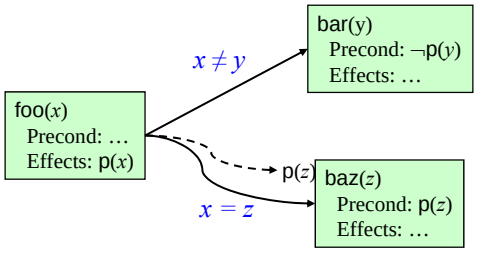
- STRIPS 알고리즘은 관련없는 상황을 만들지 않음
- STRIPS 알고리즘으로 해결 불가

6.2 계획공간 계획수립

❖ **계획공간 계획수립**(state-space planning)

- **탐색 공간**이 **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 b에서 필요한 선행조건 p를 만족시키기 위해 행동 a 사용

$$a \xrightarrow{p} b$$

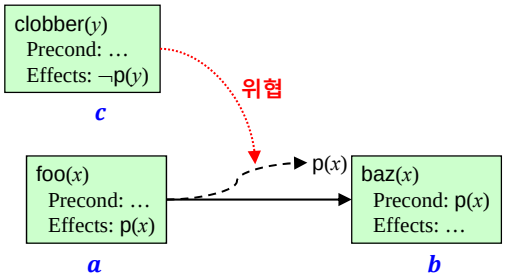


- **해답 계획**(**solution plan**)이 완성될 때까지 **점진적으로 개선**(refinement)

계획공간 계획수립

❖ **위협(threat)**

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



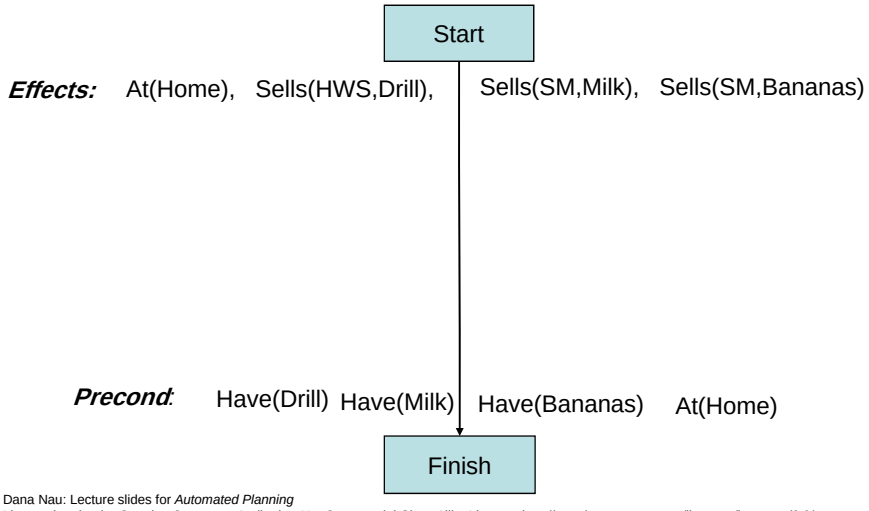
계획공간 계획수립

❖ **계획공간 계획수립 실행 예**

- **목적**
  - 하드웨어샵(HWS)에서 드릴(Drill), 슈퍼마켓(SM)에서 우유(Milk)와 바나나(Banana)를 사서 집에 오는 계획
- **행동(연산자)**
  - **Start**
    Precond: none
    Effects: At(Home), sells(HWS,Drill), Sells(SM,Milk), Sells(SM,Banana)
  - **Finish**
    Precond: Have(Drill), Have(Milk), Have(Banana), At(Home)
  - **Go**(l, m)
    Precond: At(l)
    Effects: At(m), ¬At(l)
  - **Buy**(p, s)
    Precond: At(s), Sells(s, p)
    Effects: Have(p)
  - **Start**와 **Finish**는 초기 상태와 목적 상태를 나타내는 용도

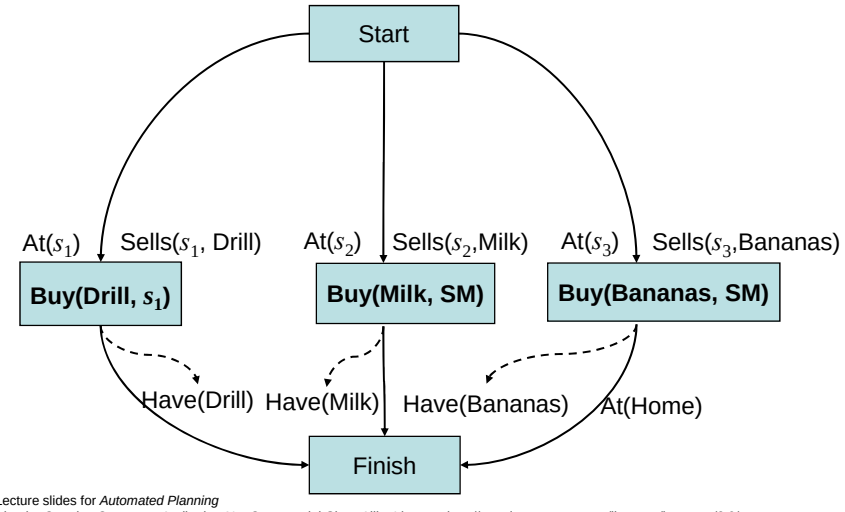
계획공간 계획수립

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



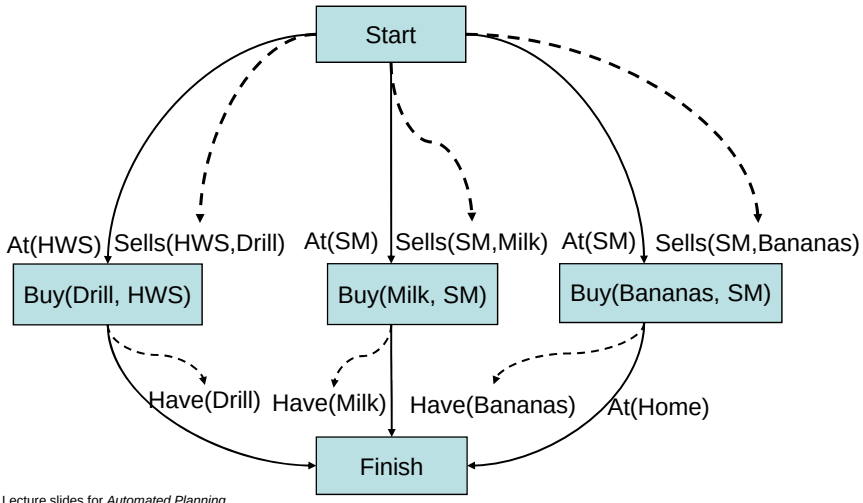
계획공간 계획수립

❖ 사전조건 Have를 만드는 3가지 행동



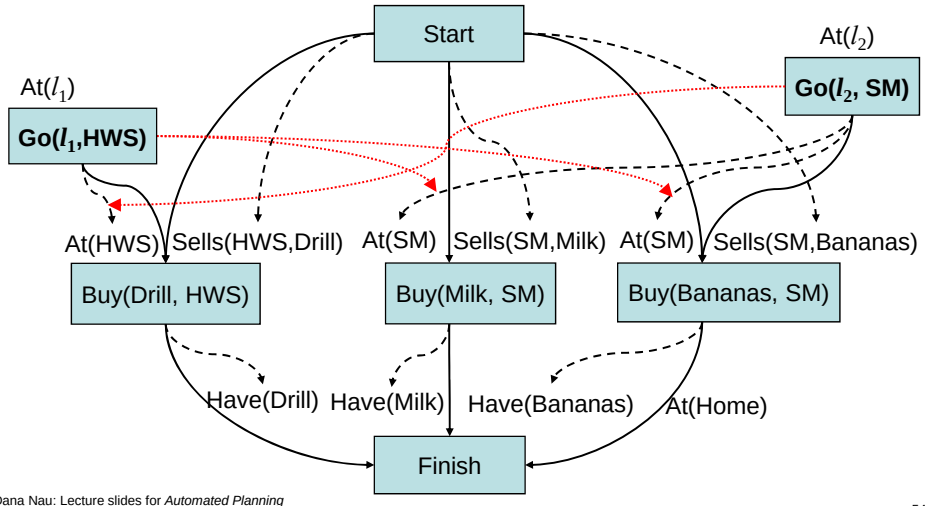
계획공간 계획수립

❖ 사전조건 Sells를 지지하는 것



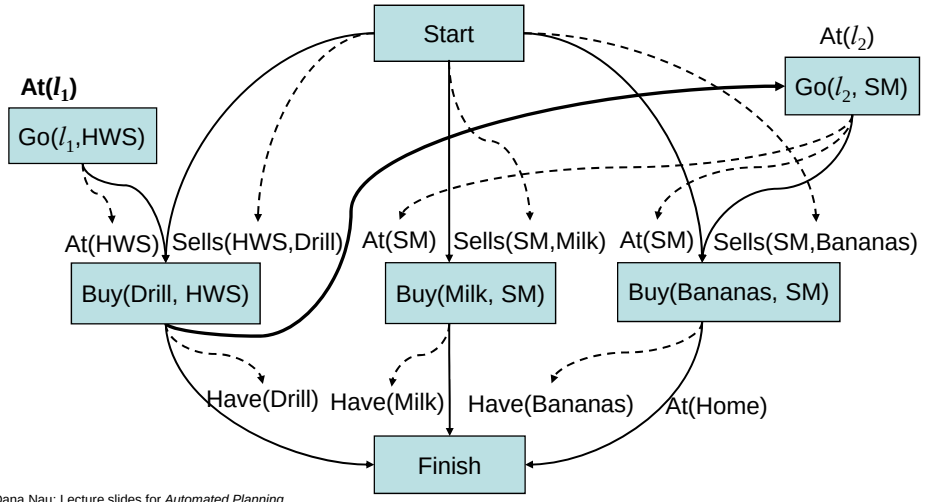
계획공간 계획수립

❖ At(HWS)와 At(SM)를 만드는 방법 : 위협 발생



계획공간 계획수립

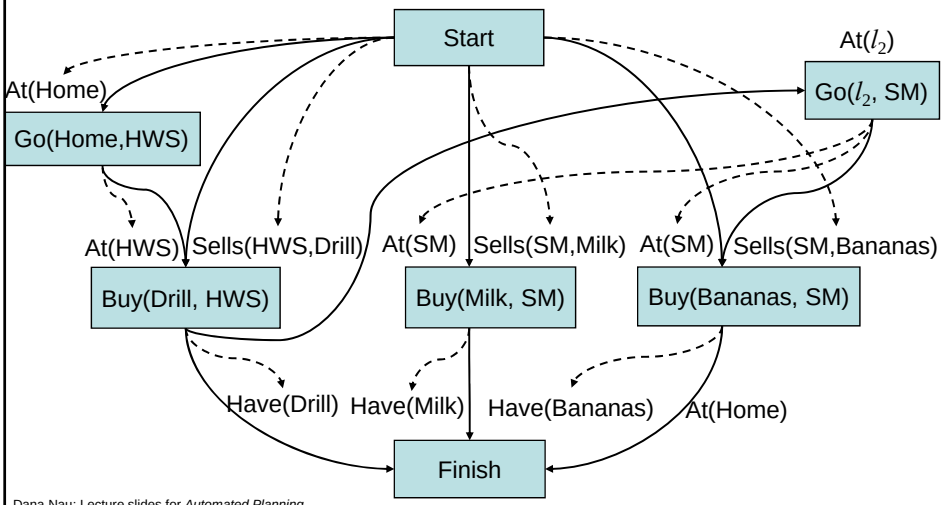
❖ At(s_1)의 위협 해결 : Buy(Drill)가 Go(l_2, SM)보다 선행



55

계획공간 계획수립

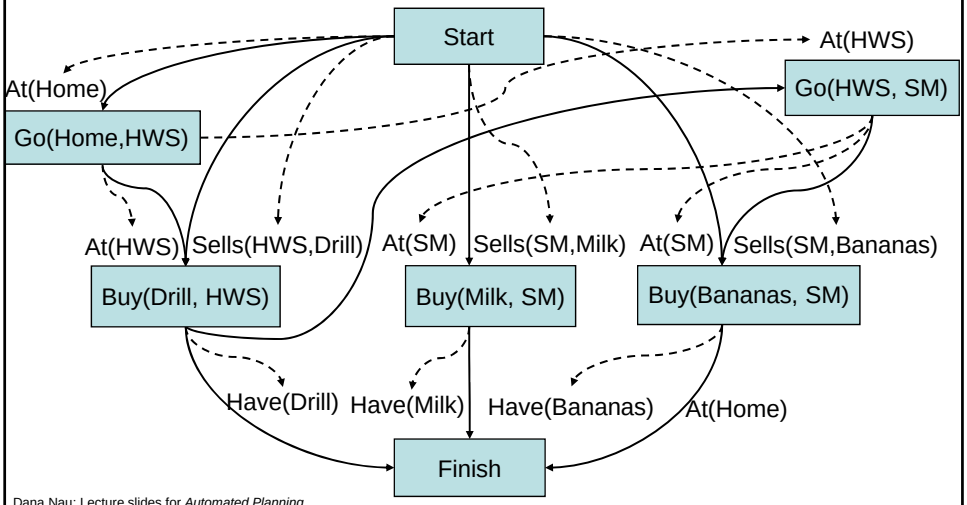
❖ At(l_1)의 결정 : Start에서 l_1=Home 로 결정



56

계획공간 계획수립

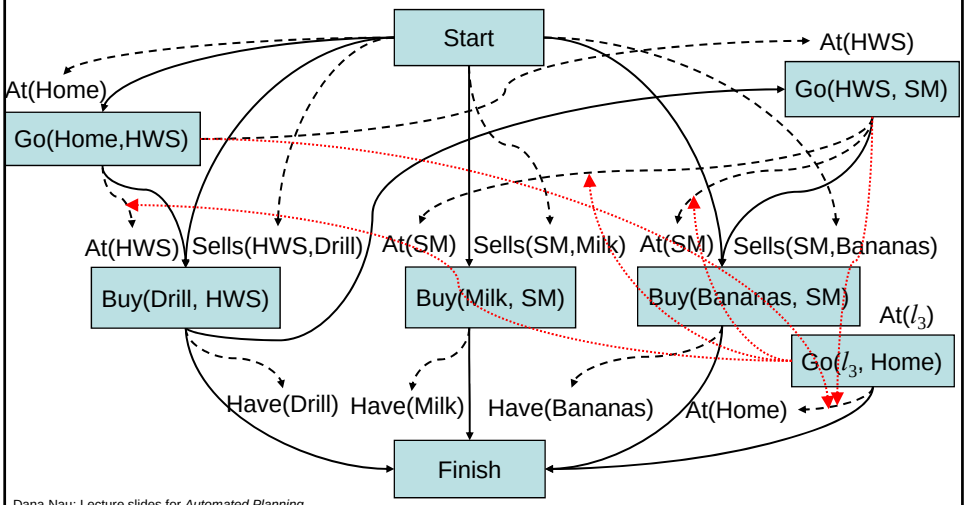
❖ At(l_2)의 결정 : Go(Home,HWS)에서 l_2= HWS로 지정



57

계획공간 계획수립

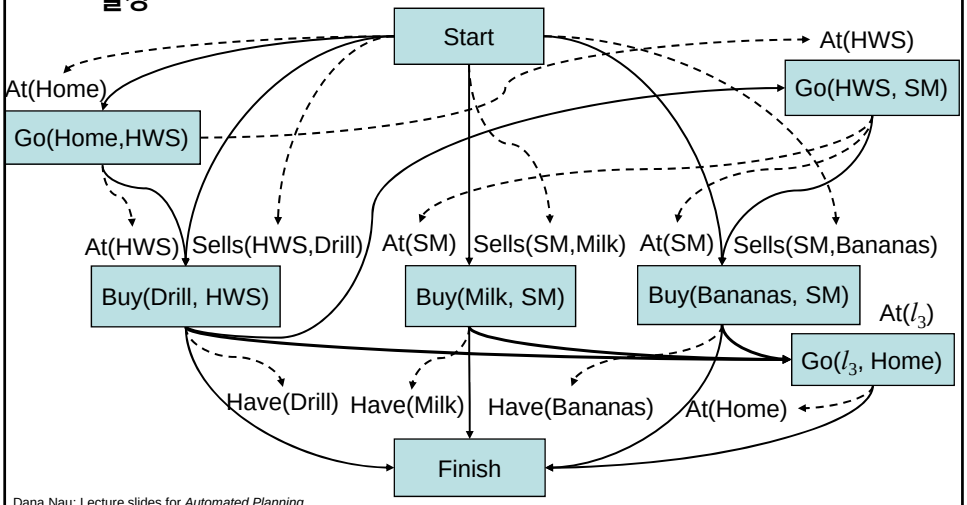
❖ Finish에서 At(Home) 생성 : Go(l_3, Home) 추가 ⇒ 여러 위협 생성



58

계획공간 계획수립

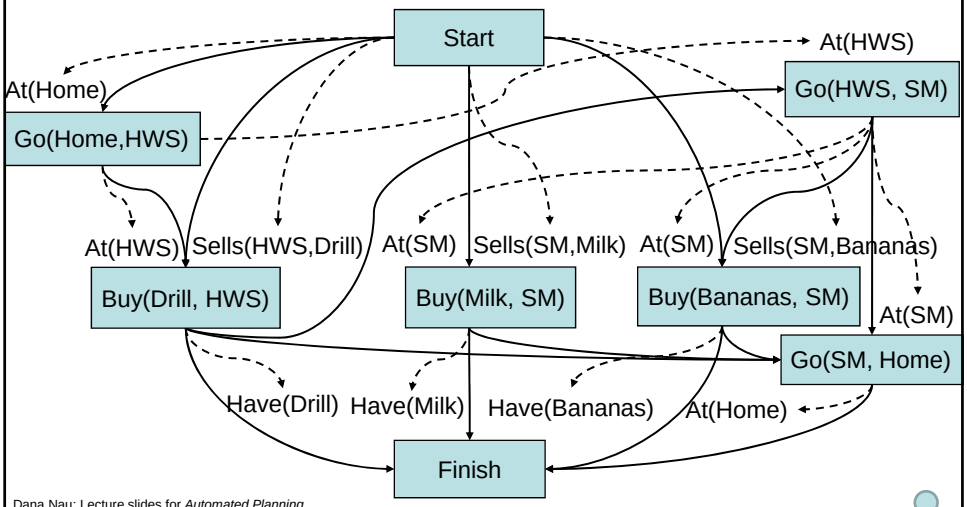
❖ **At(SM), At(HWS)에 대한 위협 제거 : Go(l_3,Home) 보다 선행하게 설정**



59

계획공간 계획수립

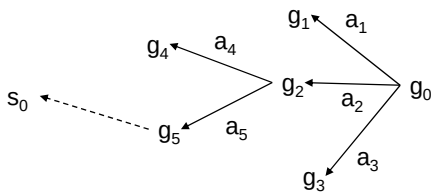
❖ **최종 계획 : At(l_3)를 l_3=SM로 설정**



60

6.3 계획수립 그래프 방법

❖ **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



계획수립 그래프 방법

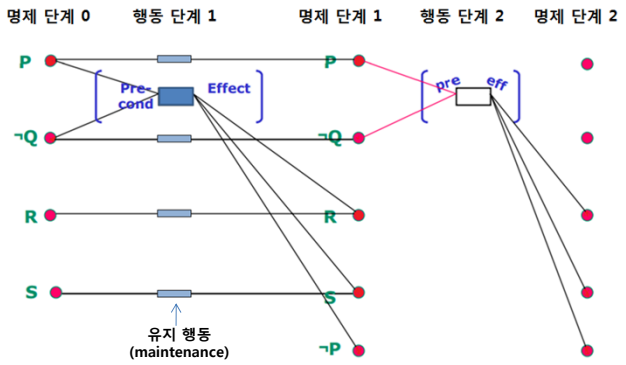
❖ **GraphPlan 알고리즘**
- 기존 계획공간 계획수립(plan-space planning)에 비해 매우 빠른 속도
- 이후 GraphPlan 기반의 많은 계획수립 알고리즘 개발

- **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)

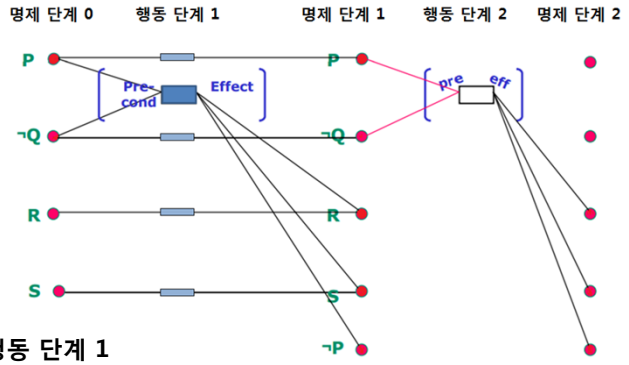
- **명제 단계(proposition level)** 와 **행동 단계(action level)**가 번갈아가며 구성



- **명제 단계 0**
  - 초기 상태에 주어지는 각 리터럴(literal)를 노드로 표현

GraphPlan 알고리즘

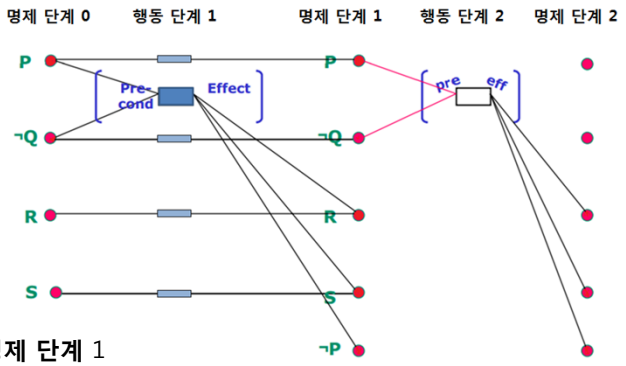
❖ **계획수립 그래프**(planning graph)



- **행동 단계 1**
  - 명제 단계 0의 명제들에 대해서 **적용될 수 있는 각 행동**을 노드로 표현
  - 명제 단계 0에 나타나는 Precondition의 리터럴 노드와 연결
  - 명제 단계 1에 나타나는 Effects의 리터럴 노드와 연결

GraphPlan 알고리즘

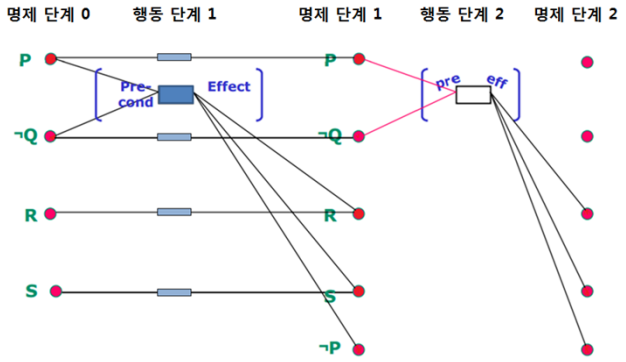
❖ **계획수립 그래프**(planning graph)



- **명제 단계** 1
  - **행동 단계 1의 행동**의 **Effects 부분**에 나타나는 **리터럴**들과 **명제 단계 0**의 **리터럴**들에 대응하는 노드 생성
  - 명제 단계 0의 모든 노드를 명제 단계 1에 유지

GraphPlan 알고리즘

❖ **계획수립 그래프**(planning graph)

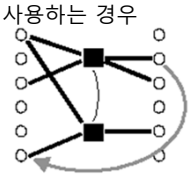


- 행동 단계 2와 명제 단계 2를 **동일한 방법으로 반복**
- **행동 단계**와 **명제 단계 확장**

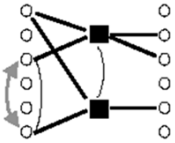
GraphPlan 알고리즘

❖ **계획수립 그래프**(planning graph)의 확장
- **상호배제**(mutual exclusion, mutex) **링크**
  - 행동 단계와 명제 단계 한 쌍이 추가시, **동시**에 **실행**되거나 **만족**될 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

상충되는 결과 도출 (inconsistent effect)



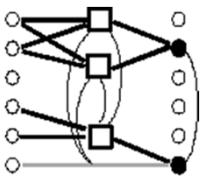
간섭(interference)



경쟁관계 사전조건 (competing needs preconditions)

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

procedure Graphplan:

`for` k = 0, 1, 2, …

Graph expansion(그래프 확장) :

create a "**planning graph**" that contains k "levels"

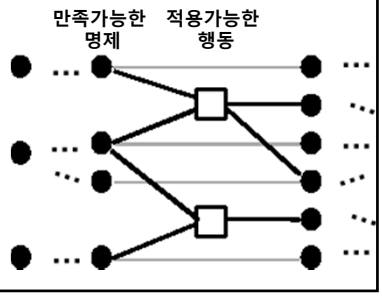
Check **whether** the planning graph **satisfies a necessary (but insufficient) condition** for plan existence

`if` it does, then

do **solution extraction (해 추출)** :

backward search, modified to consider **only the actions in the planning graph**

`if` we find a solution, `then` return it



GraphPlan 알고리즘의 적용 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
  - garb ∧ clean ∧ quiet
- **목표 상태**
  - 주방에 쓰레기가 없고, 저녁식사와 선물이 준비된 상태
  - ¬garb ∧ dinner ∧ present

GraphPlan 알고리즘의 적용 예

❖ **생일저녁 준비 문제 – cont.**

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

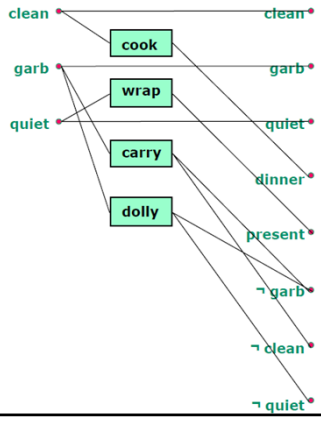
Action : **cook** Precondition : clean Effect : dinner	요리하기를 할 때는 손이 깨끗해야 하고 요리가 끝나면 저녁식사가 준비
Action : **wrap** Precondition : quiet Effect : present	선물 포장하기는 조용할 때 해야 하고 결과는 선물이 준비된 것
Action : **carry** Precondition : garb Effect : ¬garb ∧ ¬clean	손으로 치우기는 있는 쓰레기를 치워서 쓰레기가 없어지게 하고 손은 지저분해짐
Action : **dolly** Precondition : garb Effect : ¬garb ∧ ¬quiet	수레로 치우기는 있는 쓰레기를 치우는데 소란한 소리가 남

GraphPlan 알고리즘의 적용 예

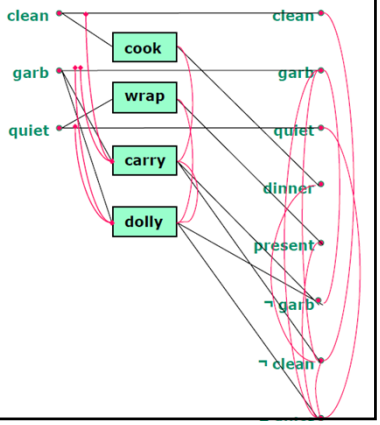
❖ 생일저녁 준비 문제 – cont.

초기 상태의 리터럴

행동 적용 결과
행동단계, 명제 단계 생성



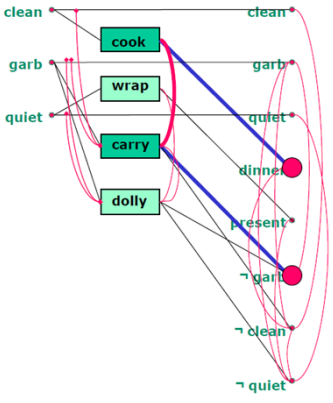
상호배타관계 링크 추가



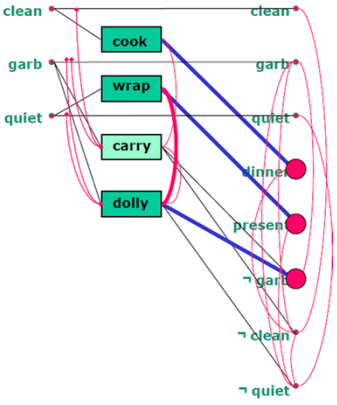
GraphPlan 알고리즘의 적용 예

❖ 생일저녁 준비 문제 – cont.

명제 단계 1에서 목표상태 리터럴 확인 : ¬garb ∧ dinner ∧ present



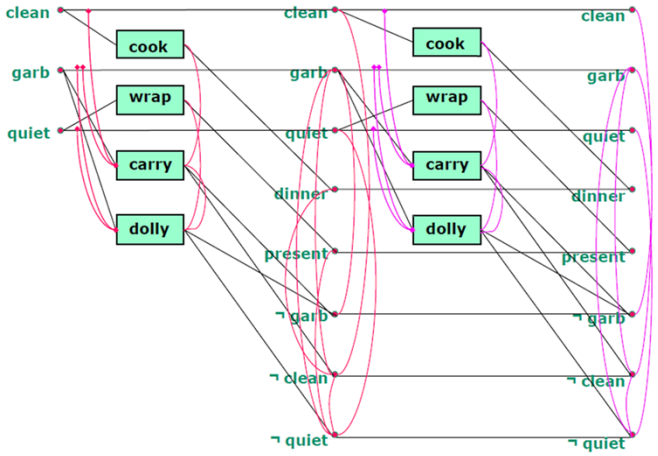
행동 충돌 : cook, carry



행동 충돌 : wrap, dolly

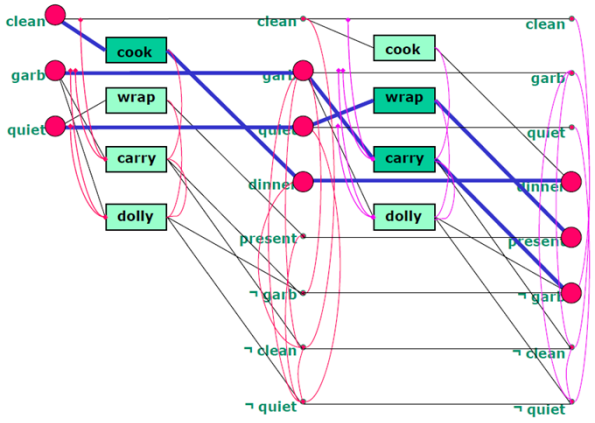
GraphPlan 알고리즘

❖ 생일저녁 준비 문제 – cont.
- ▪ 그래프 확장
  - • 행동 단계 2와 명제 단계 2 확장



GraphPlan 알고리즘

❖ 생일저녁 준비 문제 – cont.
- ▪ 계획 탐색



cook → {wrap, carry}

6.4 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**

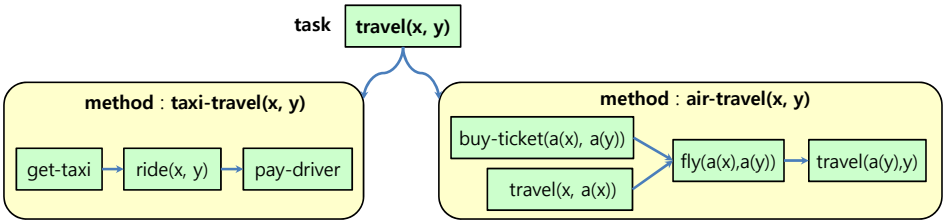
- **태스크 travel(x, y)에 대한 메소드(method)의 표현 예**

method : **taxi-travel**(x,y)
task : travel(x,y)
precondition : short-distance(x,y)
subtasks : get-taxi, ride(x,y), pay-driver
constraints : {get-taxi < ride(x,y), ride(x,y) < pay-driver}

method : **air-travel**(x,y)
task : travel(x,y)
precondition : long-distance(x,y)
subtasks : buy-ticket(a(x),a(y)), travel(x,a(x)), fly(a(x),a(y)),travel(a(y),y)
constraints : { buy-ticket(a(x),a(y)) < fly(a(x),a(y)),
travel(x,a(x)) < fly(a(x),a(y)),
fly(a(x),a(y)) < travel(a(y),y) }

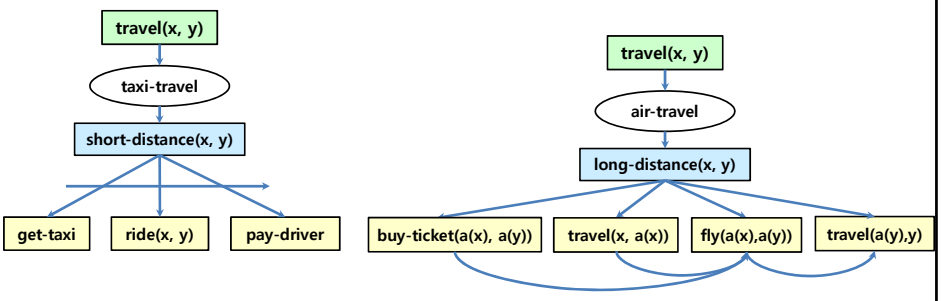
계층적 태스크 네트워크 HTN

❖ **계층적 태스크 네트워크 HTN – cont.**
- 태스크 분할 방법을 표현하는 메소드



계층적 태스크 네트워크 HTN

❖ **계층적 태스크 네트워크 HTN – cont.**
- 메소드의 네트워크 표현



계층적 태스크 네트워크 HTN

❖ **계층적 계획수립 HTN**

- 주어진 복합 태스크에 대해
  조건에 맞는 메소드를 찾아 적용, 기본 태스크로 표현될 때까지 분할

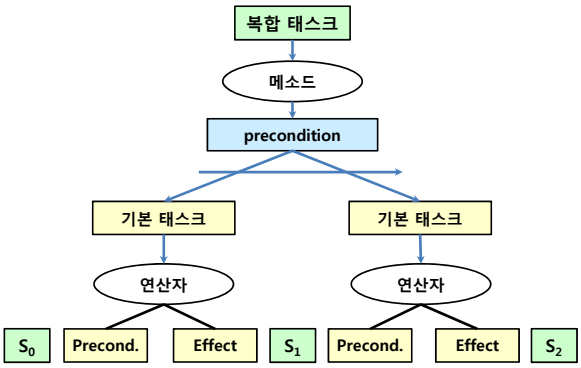
- **계획구성**
  - 기본 태스크를 수행하는 연산자를 기본 태스크에 부여된 순서관계에 따라 순차적으로 나열

- **계획실행**
  - 초기 상태에서 순차적으로 연산자를 적용하면 최종적으로 주어진 태스크가 완료되는 목표 상태에 도달

계층적 태스크 네트워크 HTN

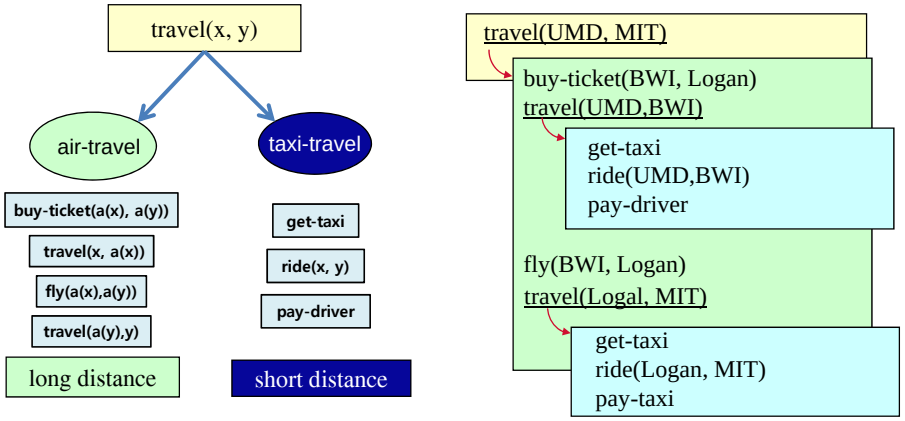
❖ **계층적 태스크 네트워크 HTN – cont.**

- **계층적 계획수립 형태**



계층적 태스크 네트워크 HTN

❖ HTN 적용 예
- UMD(University of Maryland)에서 MIT로 가는 방법



계층적 태스크 네트워크 HTN

❖ HTN 적용 예
- Bridge Baron

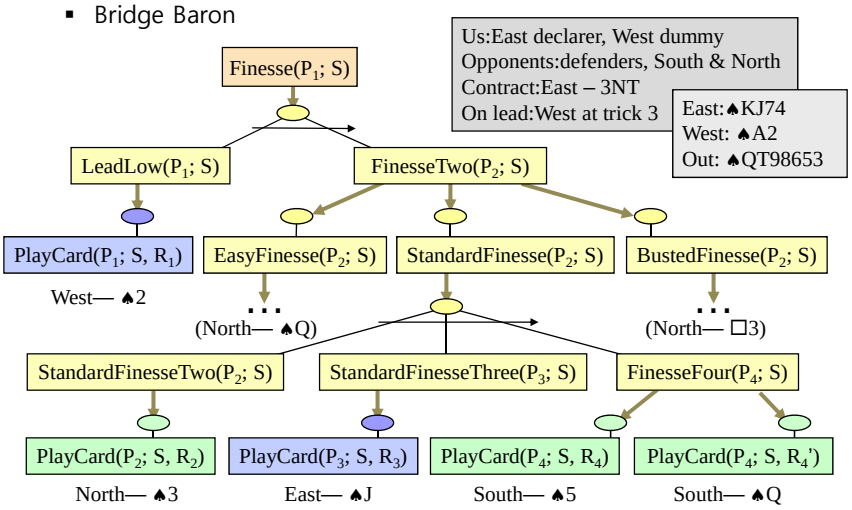
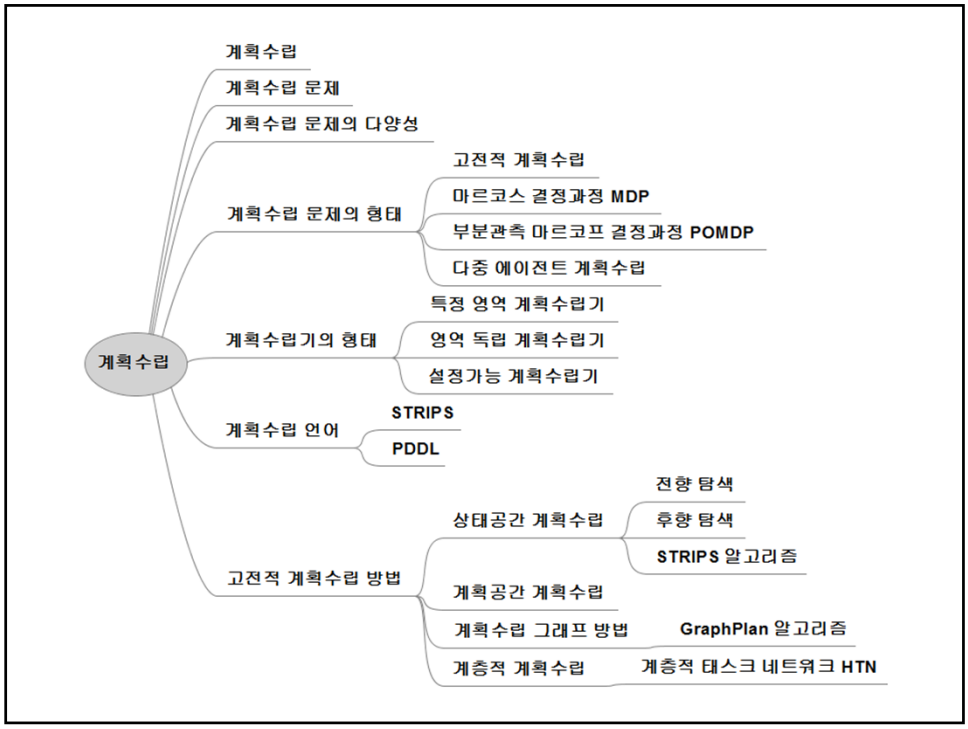


image: S. Smith et al. 1998


계획수립
계획수립
계획수립 문제
계획수립 문제의 다양성
계획수립 문제의 형태
고전적 계획수립
마르코스 결정과정 MDP
부분관측 마르코프 결정과정 POMDP
다중 에이전트 계획수립
계획수립기의 형태
특정 영역 계획수립기
영역 독립 계획수립기
설정가능 계획수립기
계획수립 언어
STRIPS
PDDL
고전적 계획수립 방법
상태공간 계획수립
전향 탐색
후향 탐색
STRIPS 알고리즘
계획공간 계획수립
계획수립 그래프 방법
GraphPlan 알고리즘
계층적 계획수립
계층적 태스크 네트워크 HTN