Planning and Acting

Chapter 12
Outline

♦ The real world
♦ Conditional planning
♦ Monitoring and replanning
The real world

START

~Flat(Spare) Intact(Spare) Off(Spare)
On(Tire1) Flat(Tire1)

On(x) ~Flat(x)

FINISH

On(x)  ~Flat(x)

On(x)

Off(x) ClearHub

Remove(x)

Puton(x)

Intact(x) Flat(x)

Inflate(x)

Off(x) ClearHub

On(x) ~ClearHub

~Flat(x)
Things go wrong

**Incomplete information**
- Unknown preconditions, e.g., $\text{Intact}(\text{Spare})$?
- Disjunctive effects, e.g., $\text{Inflate}(x)$ causes
  $$\text{Inflated}(x) \lor \text{SlowHiss}(x) \lor \text{Burst}(x) \lor \text{BrokenPump} \lor \ldots$$

**Incorrect information**
- Current state incorrect, e.g., spare NOT intact
- Missing/incorrect postconditions in operators

**Qualification problem:**
- can never finish listing all the required preconditions and possible conditional outcomes of actions
Solutions

Conformant or sensorless planning
Devise a plan that works regardless of state or outcome

*Such plans may not exist*

Conditional planning
Plan to obtain information (observation actions)
Subplan for each contingency, e.g.,

\[
\text{[Check(Tire1), if Intact(Tire1) then Inflate(Tire1) else Call AAA]}
\]

*Expensive because it plans for many unlikely cases*

Monitoring/Replanning
Assume normal states, outcomes
Check progress *during execution*, replan if necessary

*Unanticipated outcomes may lead to failure (e.g., no AAA card)*

(Really need a combination; plan for likely/serious eventualities, deal with others when they arise, as they must eventually)
Conformant planning

Search in space of belief states (sets of possible actual states)
Conditional planning

If the world is nondeterministic or partially observable then percepts usually provide information, i.e., split up the belief state
Conditional planning contd.

Conditional plans check (any consequence of KB +) percept

[... if $C$ then $Plan_A$ else $Plan_B$, ...]

Execution: check $C$ against current KB, execute “then” or “else”

Need *some* plan for *every* possible percept

(Cf. game playing: *some* response for *every* opponent move)
(Cf. backward chaining: *some* rule such that *every* premise satisfied

AND–OR tree search (very similar to backward chaining algorithm)
Example

Double Murphy: sucking or arriving may dirty a clean square
Example

Triple Murphy: also sometimes stays put instead of moving

\[ L_1 : \text{Left}, \text{if } AtR \text{ then } L_1 \text{ else [if } \text{CleanL then [] else Suck]} \]

or \[ \text{while } AtR \text{ do [Left], if } \text{CleanL then [] else Suck} \]

“ Infinite loop” but will eventually work unless action always fails
Execution Monitoring

“Failure” = preconditions of *remaining plan* not met

Preconditions of remaining plan
   = all preconditions of remaining steps not achieved by remaining steps
   = all causal links *crossing* current time point

On failure, resume POP to achieve open conditions from current state

IPEM (Integrated Planning, Execution, and Monitoring):
   keep updating *Start* to match current state
   links from actions replaced by links from *Start* when done
Example

Start

At(Home)

Go(HWS)

At(HWS) Sells(HWS,Drill)

Buy(Drill)

At(HWS)

Go(SM)

At(SM) Sells(SM,Milk)

Buy(Milk)

At(SM)

Sells(SM,Ban.)

Buy(Ban.)

At(SM)

Go(Home)

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

Finish

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

At(SM)  Sells(SM,Ban.)  Sells(SM,Milk)
At(Home)  Have(Ban.)  Have(Drill)
At(HWS)  Sells(HWS,Drill)
Buy(Drill)  Sells(SM,Milk)
Buy(Ban.)  Have(Drill)
Buy(Milk)  Have(Milk)
Go(Home)  At(SM)  At(HWS)
Go(HWS)  Go(SM)
Start
Example
Example

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

Have(Drill)
Have(Ban.)
Have(Milk)

Start

Finish
Example

At(SM)  At(Home)  At(HWS)  Sells(HWS,Drill)  Buy(Drill)  At(HWS)  Go(HWS)  At(SM)  Sells(SM,Milk)  Buy(Milk)  At(SM)  Sells(SM,Ban.)  Buy(Ban.)  At(SM)  Go(Home)  At(Home)  Have(Drill)  Have(Ban.)  Have(Milk)  Finish
Emergent behavior

**PRECONDITIONS**

- Color(Chair,Blue)
- ~Have(Red)

**START**

- Get(Red)

**FAILURES RESPONSE**

- Fetch more red

- Have(Red)

- Paint(Red)

- Color(Chair,Red)

**FINISH**
Emergent behavior

PRECONDITIONS

START

Color(Chair,Blue) ~Have(Red)

Get(Red)

Have(Red)

Paint(Red)

Color(Chair,Red)

FINISH

FAILURE RESPONSE

Extra coat of paint
Emergent behavior

“Loop until success” behavior emerges from interaction between monitor/replan agent design and uncooperative environment.