Behavior-Based Arbitration

Gal A. Kaminka

Partial Plans, Recipes, and Policies

The Not-Entirely-Stupid Agent

```
W knowledge base, g goal, B actions

while g not satisfied:
    s = PERCEIVE() // new state
    Let C be the set of APPLICABLE() actions in B
    If |C|>1 then CHOOSE(g,C,W)
             else c only action in C
    EXECUTE() action c
    REMEMBER!(s, W)
```

- CHOOSE() is invoked only if there is a choice to be made
  - That's why it needs access to (g, C, W)
- CHOOSE() can call planner, or do random (biased) choice
  - The only two options we have seen

The Not-Entirely-Stupid Agent

```
W knowledge base, g goal, B actions

while g not satisfied:
    s = PERCEIVE() // new state
    Let C be the set of APPLICABLE() actions in B
    If |C|>1 then CHOOSE(g,C,W)
             else c only action in C
    EXECUTE() action c
    REMEMBER!(s, W)
```

- CHOOSE() is invoked only if there is a choice to be made
  - That's why it needs access to (g, C, W)
- CHOOSE() can call planner, or do random (biased) choice
  - The only two options we have seen

Today: look at CHOOSE() alternatives

Rewrite to emphasize CHOOSE()

CHOOSE(g,B,W) that uses a planner:

```
If have plan p in W:
  If next action b in p is APPLICABLE():
    advance p to next action ("p++")
    return b
  else:
   generate new p (p=PLANNER(g,W)), goto 1
else:
  Let C be the set of APPLICABLE() actions in B
  If |C|>1 then:
    generate new p (p=PLANNER(g,W)), goto 1
  else c only action in C; return c.
```

Why do we need to CHOOSE()?

- Planner algorithm has **perfect intelligence**
- Plan is **perfect knowledge**
- In perfect world: Never CHOOSE()
  - Call planner $\Rightarrow$ have a plan.
  - Once have a plan, never choose between actions

BUT. . . .

Imperfect planners for *perfect worlds*

Assume world is **perfect** (deterministic, transparent). Still:

- Planner algorithms search a **huge** space
  - Computationally *intractable*
- Task is made *harder* because planner has to:
  - Decide on order in advance
  - Decide on grounding in advance
  - Unroll loops

Examples of planner hardships (even in *perfect* worlds)

- Many orderings:
  - e.g., $\{$get pen, get paper, get chair$\}$ $\rightarrow$ sit $\rightarrow$ write.
  - 6 totally ordered plans to consider, only one partial plan

Examples of planner hardships (even in *perfect* worlds)

- Many orderings:
  - e.g., $\{$get pen, get paper, get chair$\}$ $\rightarrow$ sit $\rightarrow$ write.
  - 6 totally ordered plans to consider, only one partial plan
- Many groundings:
  - e.g., In soccer, action **pass ball** $\rightarrow$ **to open player**
  - Difficult to predict who will be open

Examples of planner hardships (even in *perfect* worlds)

- Many orderings:
  - e.g., {get pen, get paper, get chair} $\rightarrow$ sit $\rightarrow$ write.
  - 6 totally ordered plans to consider, only one partial plan
- Many groundings:
  - e.g., In soccer, action **pass ball** $\rightarrow$ **to open player**
  - Difficult to predict who will be open
- Loops: for (i=0; i<10,000, i++): take step forward
  - Much more compact than: step, step, ... (10,000 steps)

What about *imperfect* worlds?

- Non-deterministic actions, dynamic world:
  - Cannot predict resulting state with certainty
  - *Need Policy, not Plan*
  - Decision on ordering should be flexible

What about *imperfect* worlds?

- Non-deterministic actions, dynamic world:
  - Cannot predict resulting state with certainty
  - *Need Policy, not Plan*
  - Decision on ordering should be flexible
- Lack of transparency: cannot know everything
  - Some information only revealed while executing
  - Some information never revealed
  - Actions may be grounded only during execution

⇒ Rethink the concept of A PLAN

What's a *Plan?*

- Classic Plan: *totally-ordered* set of *grounded* actions
- But we can revise this definition:
  - *Partially-ordered* set of actions
  - *Ungrounded* actions (at least partially)
  - Allowing *loops*, *branches*
  - Durative actions, ...

---

[1] Almost all of it on hierarchical plans, see later in course.

[2] This is called *Contingency Planning.*

What's a *Plan?*

- Classic Plan: *totally-ordered* set of *grounded* actions
- But we can revise this definition:
  - *Partially-ordered* set of actions
  - *Ungrounded* actions (at least partially)
  - Allowing *loops*, *branches*
  - Durative actions, ...
- Automatic planners for generalized plans:
  - There exist planners for partially ordered plans
  - Some work on planning with ungrounded actions[1], branches[2]
  - Rare work on planning for plans allowing loops
- Let us assume such plans are given (e.g., by human)

[1]Almost all of it on hierarchical plans, see later in course.

[2]This is called *Contingency Planning.*

Plan Representations for Execution

- Late 80s, Early 90s: Move away from planning to execution
  - More accurately, away from *modeling* world, to *reacting*
  - Charge led by Rodney Brooks[3], though not alone
- Focus on hand-tailored policies, compact representation
  - Allowing for realtime control and decision-making
  - In robotics, **Behavior-Based Controller**
  - In AI, **recipes**

[3] Also, later, co-founder of iRobot, Rethink Robotics.

Plan Representations for Execution

- Late 80s, Early 90s: Move away from planning to execution
  - More accurately, away from *modeling* world, to *reacting*
  - Charge led by Rodney Brooks[3], though not alone
- Focus on hand-tailored policies, compact representation
  - Allowing for realtime control and decision-making
  - In robotics, **Behavior-Based Controller**
  - In AI, **recipes**
- Motivating Example:
  - Easier: while (nail not in): hit nail with hammer
  - Harder: model wall, nail, hammer, ... compute # of hits

---

[3] Also, later, co-founder of iRobot, Rethink Robotics.

Behavior Based Control: Basic Concepts

Basic concepts and intuitions

- Behavior: grounded controller
- Tight coupling of perception and action
  - Reactive components, little or no prediction
- Local considerations
  - Does one thing (achieves one local condition)
  - Ignores global considerations, goals
- Agent does not know goal, partially knows world state
  - Just reacts by activating behaviors

Behaviors as local control loops

Instead of this:

```
W knowledge base, g goal, B actions

while g not satisfied:
    PERCEIVE() // also REMEMBERs old states
    CHOOSE(g,B,W)
    EXECUTE() action c
```

Behaviors as local control loops

We get this:

```
W knowledge base, g goal, B BEHAVIORS

while g not satisfied:
    ARBITRATE(W,B)
```

- Each behavior in B has its own control loop
- ARBITRATE dynamically combines, selects behaviors
  - Resulting actions are a composition of behavior
- **Behavior Arbitration[4]: How to combine behaviors?**

[4] Also called *Behavior Coordination*

Overview of Behavior Based Control

Two main branches of investigation:

- Behavior Selection (one behavior takes over)
  - Key question 1: How to select?
  - Key question 2: How to de-select?

Overview of Behavior Based Control

Two main branches of investigation:

- Behavior Selection (one behavior takes over)
  - Key question 1: How to select?
  - Key question 2: How to de-select?
- Behavior Fusion (combine multiple behaviors)
  - Key question 1: How to combine?
  - Key question 2: Addressing conflicts and local minima?