Foundations of Language Interaction

HANDOUT THREE

June 7, 2001

M. Stone
mdstone@cs.rutgers.edu

1 Introduction

Today we consider the interaction of KNOWLEDGE and INTENTIONS.

(1) Last week, we characterized a plan as an argument that demonstrates how performing a sequence of actions in the current circumstances leads to desired effects.

A rational agent need not make all its decisions immediately. It can instead defer choices to later steps of deliberation. Plans can and should guide these later choices, but only if they anticipate the new reasons to act afforded by the agent's increased future information. Our new account of plans follows [Stone, 1998]; those of you unsatisfied with [Davis, 1994] as a guide to today's meeting may want to consult this paper.

(2)
```
agitate State Goals Intentions :-
    perceive State Beliefs,
    update State Beliefs Goals Intentions
    NewIntentions,
    !,
    act State Goals NewIntentions.
```

Today's objective is to extend the representations of intentions used by our classic agent simulator, showing how to REPRESENT knowledge and choice, and how to formalize INFERENCES about knowledge and choice in plans.

2 Representing knowledge

Recall that `fact` was the type of statements in the agent's knowledge base; the agent's knowledge base was a list of facts. We now add three new declarations.

(3) a
```
kind agent, object type.
```
b
```
type k agent -> fact -> fact.
```
c
```
type sm (object -> list fact) -> fact.
```

`k` REPRESENTS knowledge:

(4) a If `a` is an agent, `(k a)` is a function from facts to facts; this is called a MODAL OPERATOR.

b use [A] to write `(k a)` in logic.

c `(k a f)` represents the fact that agent `a` knows fact `f`.

`sm` is indispensible for SPECIFYING knowledge.

(5) a `(sm P)` represents the fact that there is an object `x` for which all the facts in `(P x)` are true.
b Use λProlog's function syntax to make this look more like $\exists x P(x)$: `(sm x\ P x)`.
c Illustrates HIGHER-ORDER ABSTRACT SYNTAX. Use λ-abstraction to represent bound variables in the OBJECT LOGIC of fact-expressions as bound variables in λProlog, the META-LOGIC.
d To substitute an object-level term for an object-level bound variable, use meta-level function application.

Some key formulas, from [Hintikka, 1971].

(6) a
```
type food object -> fact.
```
b `sm x\(k self food x)::nil` $= \exists x[\text{SELF}]fx$
c `k self (sm x\ food x::nil)` $= [\text{SELF}]\exists x fx$

(6b) means that there is a specific object x about which you know that it's food. (6b) is an INDEFINITE SPECIFICATION of what you know—it constrains what you know but does not say exactly what you know (it doesn't say what that x is that you know is food). (6c) means that you know that there is some food. (6c) says exactly what you know, but indicates that you have only INDEFINITE KNOWLEDGE—you don't actually know what the food is, specifically.

3 Plans and proof

Suppose you want to achieve a goal G at some point.

(7) a You need to choose a specific action x, based on your knowledge. Your knowledge should tell you that x brings it about that (BIAT) you know G.
b Prove $\exists x[\text{SELF}](x \text{ BIAT } [\text{SELF}]G)$.

Suppose you want to achieve a goal G, and you get to choose two actions. You need to choose the first now, but you don't need to choose the second until your next cycle of perception and action.

(8) a You need to choose a specific action y, based on your knowledge. Your knowledge should tell you that y brings it about that you can achieve G in the sense of (7) afterwards.
b Prove $\exists y[\text{SELF}](y \text{ BIAT } \exists x[\text{SELF}](x \text{ BIAT } [\text{SELF}]G))$

I can't resist a peek ahead to collaboration! Suppose you and a friend want to achieve G; you act first, then the other goes.

(9) a You need to choose a specific action y, based on your knowledge. Your knowledge should tell you that y brings it about that the other can achieve G afterwards.
b Prove $\exists y[\text{SELF}](y \text{ BIAT } \exists x[\text{OTHER}](x \text{ BIAT } [\text{SHARED}]G))$

We will presume that you COORDINATE on starting: as part of the collaboration you both know the proof and have agreed to act as it lays out. At the end [SHARED] ensures that you can coordinate on stopping.

4 Formalizing Deductions

(10) a Suppose you have a bunch of premises `Fs` including `sm x\(C x)`, and you're trying to prove `G`. And let `w` be a symbol that doesn't occur in `Fs` or `G`. If you can prove `G` from `Fs` together with `C w` then you can prove `G` from `Fs`.

b If the subproof with `w` is `Plan w`, then the overall proof is `whatever Fs C Plan`.

c
```
type whatever
list fact -> (object -> list fact) -> (object -> plan) ->
plan.
```

(11) a Suppose you have a bunch of premises `Fs` which all take the form `k Agent F`, and you're trying to prove `k Agent G`. If you can prove `G` from `Fs` then you can prove `k Agent G` from `Fs`.

b If the subproof is `Plan`, then the overall proof is `know Agent Fs Plan`.

c
```
type know
agent -> list fact -> plan -> plan.
```

(12) a If you want to prove an existential statement, just prove an instance. We will only have to prove existential statements which quantify over the action the agent chooses.

b If the subproof is `Plan` and the action selected is `Action`, the overall proof is `find Action Plan`.

c `type find action -> plan -> plan.`

Here's a plan that assumes that we know of something that it's food. We plan to eat it, and thereby to sate our hunger.

(13)
```
(whatever ((sm x\ k self (food x)::nil)::nil)
 (x\ k self (food x)::nil)
 (f\(find ((k self (food f))::nil)
     (eat f)
     (know self ((k self (food f))::nil)
      (step ((food f)::nil)
       self (eat f)
       (know self ((k self full)::nil)
        (finish (full::nil)))))))
```

As always, we keep only the premises we need to continue. We also assume that if anyone knows something, it's true. (Remember we're using this inference to guide deliberation, so if we have reason to disbelieve something that a plan may depend on, we should thrash out the discrepancy now rather than assess what we would otherwise expect about mental states.)

(14)
```
(find look
 (know self
  ((k self (sm x\((food x)::nil)))::nil)
  (whatever ((sm x\((food x)::nil))::nil)
   (x\((food x)::nil))
   (f\(step ((food f)::nil)
       self look
       (find (eat f)
        (know self ((k self (food f))::nil)
         (step ((food f)::nil)
          self (eat f)
          (know self ((k self full)::nil)
           (finish (full::nil)))))))))))
```

(14) shows the kind of reasoning involved in (13) would appear as a subplan of a larger plan that describes first getting more information, then what you'll do in the future once you have that information.

5 Using Plans

Finding the next action:

(15) Scan down into the plan structure until you find the first `step` operation. (The agent should be `self`.) Do the specified action.

Finding the intention for the next round of deliberation

(16) Scan down into the plan structure until you find the first `know` operation after your current `step`. (The agent should be `self`.) Save this as your intention for next time.

Handling indefinite information.

(17) When you reach a substructure of the form `whatever _ _ PF`, create a new, unspecified value with some variable `X` and process `PF X`.

For the case of plan (14).

(18) a Next action is `look`

b Subplan is
```
(know self ((k self (food X1))::nil)
 (step ((food X1)::nil)
  self (eat X1)
  (know self ((k self full)::nil)
   (finish (full::nil)))))
```

c Logic variable `X1` replaces bound variable `f` in plan instance.

Monitoring plan execution—actually, same as before:

(19)
```
update _ Facts _ Plan Plan :-
    circ Plan C, entail_all Facts C.
```

In this case, `Facts` gives the specific knowledge you have

(20)
```
know self food o7
```

And the circumstances `C` is an indefinite specification of this knowledge, involving variables:

(21)
```
know self food X1
```

As in any query processing, when you establish that (20) entails (21), you compute the substitution `X1 = o7`. Thus after the call to `update` succeeds in clause (19), our current intention is as in (22).

(22)
```
(know self ((k self (food o7))::nil)
 (step ((food o7)::nil)
  self (eat o7)
  (know self ((k self full)::nil)
   (finish (full::nil)))))
```

From (22) we get

(23) a Next action: `eat o7`

b Next plan:
```
(know self ((k self full)::nil)
        (finish (full::nil)))
```

References

[Davis, 1994] Davis, E. (1994). Knowledge preconditions for plans. *Journal of Logic and Computation*, 4(5):721–766.

[Hintikka, 1971] Hintikka, J. (1971). Semantics for propositional attitudes. In Linsky, editor, *Reference and Modality*, pages 145–167. Oxford.

[Stone, 1998] Stone, M. (1998). Abductive planning with sensing. In *AAAI*, pages 631–636, Madison, WI.