Knowledge Representation and Reasoning
Logics for Artificial Intelligence

Stuart C. Shapiro

Department of Computer Science and Engineering
and Center for Cognitive Science
University at Buffalo, The State University of New York
Buffalo, NY 14260-2000

shapiro@cse.buffalo.edu

Contents

10 The Situation Calculus

Motivation (McCarthy)

I'm in my study at home. My car is in the garage. I want to get to the airport. How do I decide that I should walk to the garage and drive to the airport, rather than vice versa?

A commonsense planning problem.

Solution Sketch

My study and garage are in my home.
To get from one place to another in my home, I should walk.

My garage and the airport are in the county.
To get from one place to another in the county, I should drive.

Situations

When an agent acts, some propositions change as a result of acting, and some are independent of acting.

E.g. the fact that the airport is in the county is independent of my acting, but whether I'm in my study, in the garage, or at the airport, changes when I act.

We say that an act takes us from one situation to another.

Propositions that are dependent on situations are called propositional fluents. E.g. *At(study, S0)*, *At(garage, S1)* vs. *In(study, home)*, *In (airport, county)*

Situational Fluents

We can view an act as something that's done in some situation, and takes us to another situation.

Let *do(a, s)* be a two-argument functional term.

$[\![do(a, s)]\!]$ = the situation that results from doing the act $[\![a]\!]$ in the situation $[\![s]\!]$.

So, *At(study, S0)*, *At(garage, do(walk(study, garage), S0))*

Planning in the Situational Calculus

Describe the situation *S0*.

Give domain rules describing the effects of actions.

Find a solution for *At(airport, ?s)*

Formalization in SNARK
Non-Fluent Propositions

```
(assert '(Walkable home))
(assert '(Drivable county))
(assert '(In study home))
(assert '(In garage home))
(assert '(In garage county))
(assert '(In airport county))
```

Effect Axioms

```
(assert '(all (x y z s)
          (implies (and (At x s) (In x z) (In y z)
                        (Walkable z))
           (At y (do (walk x y) s)))))

(assert '(all (x y z s)
          (implies (and (At x s) (In x z) (In y z)
                        (Drivable z))
           (At y (do (drive x y) s)))))
```

Initial Situation

```
(assert '(At study S0))
```

SNARK Solves the Problem

```
(ask '(At airport ?s) :answer '(At airport ?s))
(At airport (do (drive garage airport)
                (do (walk study garage) S0)))
```

Example 2: BlocksWorld Non-Fluent Propositions

```
(assert '(all s (Clear Table s)))

(assert '(all (x y s) (implies (and (Block y) (On x y s))
                                (not (Clear y s)))))

(assert '(all (x s) (implies (Held x s)
                             (not (Clear x s)))))
```

BlocksWorld Effect Axioms

```
(assert
    '(all (x y s) (implies (and (On x y s) (Clear x s))
                           (and (Held x (do (pickUp x) s))
                                (Clear y (do (pickUp x) s))))))
(assert
    '(all (x y s) (implies (and (Held x s) (Clear y s))
                           (and (On x y (do (putOn x y) s))
                                (not (Held x (do (putOn x y) s)))
                                (Clear x (do (putOn x y) s))))))
```

Initial Situation

```
(assert '(Block A))
(assert '(Block B))
(assert '(Block C))
(assert '(On A B S0))
(assert '(On B Table S0))
(assert '(On C Table S0))
(assert '(Clear A S0))
(assert '(Clear C S0))
```

Solving A Simple Problem

```
(ask '(Held A ?s) :answer '(Held A ?s))

(Held A (do (pickUp A) S0))
```

A Harder Problem

```
(ask '(On A C ?s) :answer '(On A C ?s))
```

Just loops!

The Frame Problem

We want

`(On A C (do (putOn A C) (do (pickUp A) S0)))`

but this requires C to be clear in situation
`(do (pickUp A) S0)`

That can't be decided.

We need to specify what propositional fluents **don't change** when an action is performed.

A Frame Axiom

```
(assert
    '(all (x y s) (implies (and (Clear x s) (not (= x y)))
                      (Clear x (do (pickUp y) s)))))
```

Another Problem

Still doesn't work, because we don't know that

```
(not (= C A))
```

Uniqueness Axioms

```
(assert '(not (= A B)))
(assert '(not (= A C)))
(assert '(not (= B C)))
```

Also need

```
(use-paramodulation)
```

after `(initialize)`

This includes the theory of equality with resolution.

Success!

```
(ask '(On A C ?s) :answer '(On A C ?s))

(On A C (do (putOn A C) (do (pickUp A) S0)))
```