Blocks world – situation calculus

May 31, 2020

1 Modeling

1.1 Rigid data types

```
spec! OBJECTS =
  sorts Object %% Blocks and Table

  ops table, a, b, c : → Object
end

spec! ACTIONS =
  protecting OBJECTS
  sort Action

  op move : Object Object → Action
end
```

1.2 Nominals

```
spec! SITUATIONS =
  protecting ACTIONS
  sort Situation

  op init : → Situation
  op do : Action Situation → Situation
end
```

1.3 Flexible data types

```
spec BLOCKS-WORLD[SITUATIONS] =
  pred _on_ : Object Object %% Object is ‘on top’ of another Object
  pred clear : Object %% something can be placed on top of Object

  ∀ s : Situation; x : Object
  · @s-clear(x) ⇔ (x = table ∨ ¬ ∃ y : Object · y @s-on x)
  %% (axdef-clear)

  ∀ x, y : Object
  · x @init-on y
    ⇔ (x = b ∧ y = c)
        ∨ (x = a ∧ y = table)
        ∨ (x = c ∧ y = table)
  %% (init)

  ∀ s : Situation; x, y : Object
  · @s-clear(x) ∧ @s-clear(y)
    ⇒
    x @do(move(x,y),s)-on y
  %% (move-action-axiom)

  ∀ s, s' : Situation; a : Action; x, y : Object
  · do(a, s) = s' ∧ x @s'-on y
    ⇒
    x @s-on y ∧ ∀ z : Object · a ≠ move(x, z)
    ∨
    @s-clear(x) ∧ @s-clear(y) ∧ a = move(x, y)
  %% (move-successor-state-axiom)
end
```

2 Proof goals

1. Simple: `∃ s : Situation · a @s-on b ∧ b @s-on c`

2. A bit more complex: `∃ s : Situation · b @s-on a ∧ a @s-on c`