Situation Calculus

Ulises Cortés

April 2019

First-Order Logic (FOL) (1/4)

First-Order Logic is a system of mathematical logic.

FOL is extending Propositional Logic

- Variables represent objects of a *universe of discourse*
- Quantification over variables

First-Order Logic (FOL) (2/4)

FOL Vocabulary

Set of constants which represents objects of a *universe of discourse*, e.g. *Socrates, 20, 40 …*

Set of function symbols with arity ≥ 1,

e.g. father_of (*Socrates*), average(average (*20, 40*), *40*), …

Set of predicates with arity ≥ 1, e.g. father (*Socrates*), …

Infinite set of variables, *e.g.* x, y, z …

First-Order Logic (FOL) (3/4)

FOL Vocabulary (cont.)

- Logical operators: ¬ (not), ∧ (and), ∨ (or), →(conditional), ↔(biconditional)
- Left and right parenthesis: “ (“ , “) ”
- Quantifiers: ∃ (existential), ∀ (universal)
- Equality symbol “=“ is sometimes included, not always.

Unlike predicates, function symbols are not true or false, but represent objects of the *universe*.

First-Order Logic (FOL) (4/4)

Examples:

- $\exists$x P(x), "for at least one object *a*, P(*a*) is true"
- $\forall$x P(x), "for any object *a*, P(*a*) is true"
- $\exists$x (P(x) $\wedge$ $\forall$y (P(y) $\rightarrow$ x=y)), "P(x) holds for exactly one object"

Second-Order Logic (SOL) (1/3)

- In FOL we quantify over individuals, but not over properties.
- In FOL we can find the individuals of a property, but *how can we find the properties of an individual?*
  - *e.g.* Suppose a simple knowledge base (KB): father(*John*).
    - :-? father (x).
    - KB|=x=*John*
    - But we can't ask
    - :-? X (*John*).
    - KB|=X=father

Second-Order Logic (SOL) (2/3)

- SOL extends FOL
  - Variables in predicate positions (rather than only in individual positions in FOL)
  - Quantification over predicates
- As a result, SOL has more ***expressive power*** than FOL does.
  - *e.g.* In FOL there is no way to say explicitly that individuals *a* and *b* have at least a same property in common, but in SOL we can say:

$$\exists P \, (P(a) \land P(b))$$

Second-Order Logic (SOL) (3/3)

Examples:

$\exists P\ P(\mathit{John})$, "There is a property P that *John* is member of"

$\forall F\ F(\mathit{John}) \vee \neg F(\mathit{John})$ (**principle of bivalence**)

"For every property, either *John* has it or he doesn't."

Equality in SOL can be defined by:

$$x=y \equiv_{def} \forall P\ (P(x) \leftrightarrow P(y))$$

Situation Calculus : Overview

- The Situation Calculus is a logic formalism designed for representing and reasoning about **dynamical domains**.
  - McCarthy, Hayes 1969
  - Reiter 1991

- In **FOL**, sentences are either true or false and stay that way.
- ***Nothing is corresponding to any sort of change.***

- SitCalc represents changing scenarios as a set of SOL formulae.

Situation Calculus : Basic Elements

- **Actions** that can be performed in the world
  - Actions can be quantified
- **Fluents** that describe the state of the world
- **Situations** represent a history of action occurrences
  - A dynamic world is modeled as progressing through a series of situations as a result of various actions being performed within the world
  - A finite sequence of actions
  - **A situation is not a state, but a history**

Situation Calculus : Formul*ae*

A domain is encoded in SOL by three kind of formul*ae*

- **Action precondition axioms** and **action effects axioms**
- **Successor state axioms**, one for each fluent
- **The foundational axioms** of the situation calculus

Situation Calculus : An Example (1/10)

World:

- robot
- items
- locations (x,y)
- moves around the world
- picks up or drops items
- some items are too heavy for the robot to pick up
- some items are fragile so that they break when they are dropped
- robot can repair any broken item that it is holding

Situation Calculus : An Example (2/10)

Actions

move(x, y): robot is moving to a new location **(x, y)**

pickup(o): robot picks up an object **o**

drop(o): robot drops the object **o** that holds

Situation Calculus : An Example (3/10)

Situations

Initial situation $\mathbf{S_0}$: no actions have yet occurred

A new situation, resulting from the performance of an action a in current situation **s**, is denoted using the function symbol **do(a, s)**.

- **do(move(**$2, 3$**), S_0):** denotes the new situation after the performance of action **move(**$2, 3$**)** in initial situation $\mathbf{S_0}$.
- **do(pickup(***Ball* **), do(move(**$2, 3$**), S_0))**
- **do(a,s)** is equal to **do(a',s')** $\Leftrightarrow$ **s=s' and a=a'**

Effect Axioms (3'/10)

- **If an action is possible, then certain fluents will hold in the situation that results from executing the action**
  - Going from **X** to **Y** results in being at **Y**
  - **Grabbing** an/the object results in holding an/the object
  - **Releasing** an/the object results in not holding it

Possibility Axioms (3''/10)

- The possibility axioms that an agent A_i can
  - **go** between adjacent locations,
  - **Grab/pick*** an object in the current location, and
  - **Release** an/the it is holding

  - *Ontological agreement

Frame Problem (3'''/10)

We run into the frame problem

- Effect axioms say *what* changes, but do not say *what* stays the same
- A real problem, because (in a non-toy domain), each **action** affects only a tiny fraction of all fluents

Situation Calculus : An Example (4/10)

Fluents: properties of the world

Relational fluents

Statements whose truth value may change

They take a situation as a final argument

is_carrying(o, s): robot is carrying object **o** in situation **s**

e.g. Suppose that the robot initially carries nothing

is_carrying(*Ball*, S_0) : FALSE

is_carrying(*Ball*, do(pickup(*Ball*), S_0)) : TRUE

Situation Calculus : An Example (5/10)

Fluents (cont.)

- **Functional fluents**
  - Functions that return a situation-dependent value
  - They take a situation as a final argument
  - **location(s)**: returns the **location(x, y)** of the robot in situation **s**

Situation Calculus : An Example (6/10)

Action Preconditions Axioms

Some actions may not be executable in a given situation

Poss(a,s): special binary predicate

denotes the executability of action **a** in situation **s**

Examples:

Poss(drop(o),s) ↔ is_carrying(o,s)

Poss(pickup(o),s) ↔ (∀z ¬is_carrying(z,s) ∧ ¬heavy(o))

Situation Calculus : An Example (7/10)

Action Effects Axioms

Specify the effects of an action **on the fluents**

Examples:

Poss(pickup(o),s) → is_carrying(o,do(pickup(o),s))

Poss(drop(o),s) ∧ fragile(o) → broken(o,do(drop(o),s))

Is that enough? No, because of the **frame problem**

Situation Calculus : An Example (8/10)

The frame problem

How can we derive the **non-effects** of axioms?

e.g. How can we derive that after picking up an object, the robot's location remains unchanged?

This requires a formul*ae* like

$$\mathbf{Poss(pickup(o),s) \wedge location(s) = (x,y) \rightarrow location(do(pickup(o),s)) = (x,y)}$$

Problem: too many of such axioms, difficult to specify all

Situation Calculus : An Example (9/10)

The solution: **Successor state axioms**

Specify all the ways the value of a particular fluent can be changed

$$\mathbf{Poss(a,s) \land \gamma^{+}_{F}(x,a,s) \rightarrow F(x,do(a,s))}$$

$$\mathbf{Poss(a,s) \land \gamma^{-}_{F}(x,a,s) \rightarrow \neg F(x,do(a,s))}$$

$\boldsymbol{\gamma^{+}_{F}}$ describes the conditions under which action **a** in situation **s** makes the fluent **F** become true in the successor situation **do(a,s)** .

$\boldsymbol{\gamma^{-}_{F}}$ describes the conditions under which performing action **a** in situation **s** makes fluent **F** false in the successor situation.

Situation Calculus : An Example (10/10)

Successor state axioms (cont.)

$$Poss(a,s) \rightarrow [F(x,do(a,s)) \leftrightarrow \gamma^{+}_{F}(x,a,s) \vee (F(x,s) \wedge \neg\gamma^{-}_{F}(x,a,s))]$$

"Given that it is possible to perform **a** in **s**, the fluent **F** would be true in the resulting situation **do(a,s)** iff performing **a** in **s** would make it true, or it is true in **s** and performing **a** in **s** would not make it false. "

Example:

$$Poss(a,s) \rightarrow [broken(o,do(a,s)) \leftrightarrow (a=drop(o) \wedge fragile(o)) \vee (broken(o,s) \wedge a \neq repair(o,s))]$$

Qualification Problem

- Ensuring that all necessary conditions for an action's success have been specified. No complete solution.

Inheritance

- If a property is true of a class, it is true of all subclasses of that class
- If a property is true of a class, it is true of all objects that are members of that class
- (If a property is true of a class, it is true of all objects that are members of subclasses of that class)
- ***There are exceptions***

Situation Calculus : A Complete Example

Precondition Axioms

Poss(pickup(o),s) ↔ ∀z ¬is_carrying(z,s) ∧ ¬heavy(o)

Poss(putonfloor(o),s) ↔ is_carrying(o,s)

Poss(putontable(o),s) ↔ is_carrying(o,s)

Successor State Axioms

is_carrying(o,do(a,s)) ↔ a=pickup(o) ∨ (is_carrying(o,s) ∧ a ≠ putontable(o) ∧ a ≠ putonfloor(o))

OnTable(o,do(a,s)) ↔ (OnTable(o,s) ∧ a ≠ pickup(o)) ∨ a = putontable(o)

OnFloor(o,do(a,s)) ↔ (OnFloor(o,s) ∧ a ≠ pickup(o)) ∨ a = putonfloor(o)

Initial state

¬is_carrying(o,S_0)

OnTable(o, S_0) ↔ o=A ∨ o=B

proc: RemoveBlock(o) : pickup(o) ; putonfloor(o) endProc;

proc: ClearTable : while ∃o OnTable(o) do RemoveBlock(o) endWhile endProc;

KB |= ∃s Do(ClearTable, S_0, s).

s=do(putonfloor(B),do(pickup(B),do(putonfloor(A),do(pickup(A), S_0))))

General Axioms

Describe formulas, which are true in all situations.

Example:

$$\forall x, y, s: on\ (x, y, s) \wedge \neg(y=Table) \Rightarrow \neg clear\ (y, s)$$

For all situations s and all objects x and y: if something is on object y in s, and y is not the table, then y is not clear in s.

$$\forall s: clear\ (Table, s)$$

The table (or floor) is always clear.

Situation Calculus Axioms

- Effect axioms describe how an action changes a situation, when the action is performed.
- Frame axioms describe, what remains unchanged between situations.
- Successor-state axioms combine effect and frame axioms.

Add domain knowledge!

Situation Calculus : DB-Updates Overview

We can formalize the evolution of a DB during a sequence of transactions

- Transactions are same as actions
- During the evolution, a DB pass through different states
- Updatable DB-relations

Once again: The frame problem

Situation Calculus : DB-Updates An Example (1/7)

Suppose that DB involves 3 relations:

1. **enrolled (st, course, s):** student **st** is enrolled in course **course** when DB is in state **s**

2. **grade (st, course, grade, s):** The grade of student **st** in course **course** is **grade** when the DB is in state **s**

3. **prerequ (pre, course): pre** is a prerequisite course for course **course**

(**Notice that last relation is state independent so is not expected to change during the evolution of the database**.)

Situation Calculus : DB-Updates An Example (2/7)

Three transactions:

1. **register (st, course):** Register student **st** in course **course**
2. **change (st, course, grade):** Change the current grade of student **st** in course **course** to **grade**
3. **drop (st, course):** Student **st** drops course **course**

Situation Calculus : DB-Updates An Example (3/7)

Initial DB state

First-order specification of what is true in $\mathbf{S_0}$

e.g.

enrolled (*Mary*, *C100*, $\mathbf{S_0}$)

grade (Bill, *M200*, **70,** $\mathbf{S_0}$**)**

(**∀p**) **¬prerequ (p,** *C100***)**

Situation Calculus : DB-Updates An Example (4/7)

Transaction Preconditions

- Transactions have preconditions which must be satisfied by the current DB state

1. Poss(register (st,c),s) ↔ [(∀p) prerequ (p,c) → (∃g) grade(st,p,g,s) ∧ g>=*50*]

2. Poss(change (st,c,g),s) ↔ (∃g') grade(st,c,g',s) ∧ g'≠g

3. Poss(drop(st,c),s) ↔ enrolled(st,c,s)

Situation Calculus : DB-Updates An Example (5/7)

Update Specifications

- Specify the effects of all transactions on all updatable DB relations

$$\textbf{Poss(a,s)} \rightarrow \textbf{[enrolled(st,c,do(a,s))} \leftrightarrow \textbf{a=register(st,c)} \vee \textbf{enrolled(st,c,s)} \wedge \textbf{a}{\neq}\textbf{drop(st,c)]}$$

Situation Calculus : DB-Updates An Example (6/7)

Update Specifications (cont.)

It is the update specification axioms which “solve” the frame problem

- $\mathbf{Poss(a,s)} \wedge \mathbf{a{\neq}register(st,c)} \wedge \mathbf{a{\neq}drop(st,c)} \rightarrow$
  $\mathbf{(enrolled(st,c,do(a,s))} \leftrightarrow \mathbf{enrolled(st,c,s))}$

- **register()** and **drop()** are the only transactions that can affect the truth value of **enrolled()**

Situation Calculus : DB-Updates An Example (7/7)

Querying a Database

- All updates are virtual, DB is never physically changed
- To query a DB, resulting from a sequence of transactions, we must refer to this sequence in the query

To determine if John is enrolled in any courses after the transaction sequence

drop (John, C100), register (Mary, C100)
has been "*executed*", we must determine whether

DB|= ∃c enrolled(John, c, do(register (Mary, C100), do(drop (John, C100), s_0))).

References

- J.McCarthy and P.J.Hayes, *Some Philosophical Problems from the Standpoint of Artificial Intelligence*, in *Machine Intelligence 4*, ed D.Michie and B.Meltzer, Edinburgh University Press (1969)

- R. Reiter, *On Specifying Database Updates*, Journal of Logic Programming, 25(1):53–91, 1995.