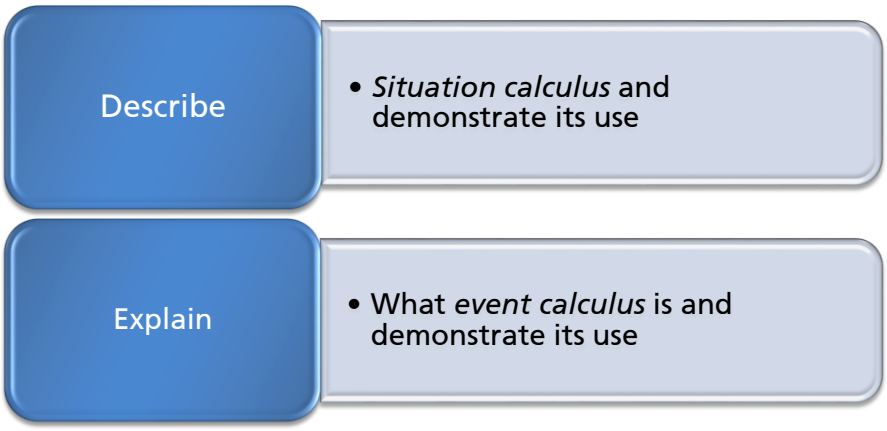
Knowledge Representation Part 2

Workshop 4 Section 2

CT621 Artificial Intelligence - MScSED

Learning Outcomes

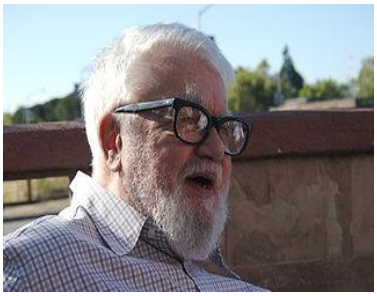
On completing this section and related learning activities, you should be able to:



This workshop will begin by introducing the concept of a situation and describing the *situation calculus*. It will also introduce *event calculus* and how it addresses some of the inherent limitations of situation calculus.

Situation Calculus

- First proposed by John McCarthy in 1963
- Disadvantage of propositional logic is its inability to deal with time
- Situation calculus is a logical language for representing and reasoning about change over time



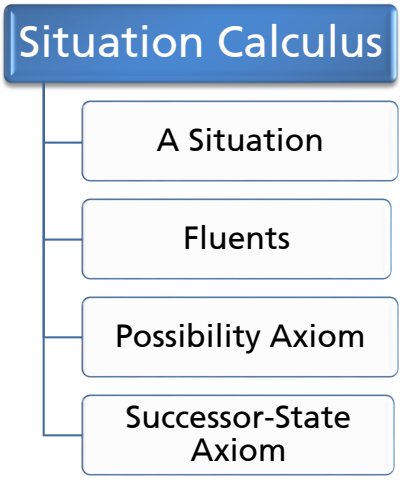
Professor John McCarthy
Image credit: (Null0 2006)

In the next few slides, we will look at how our ontology can be used to reason about actions and the result of actions using *situation calculus*.

One major drawback of propositional logic is its inability to deal with time. With this logic, there would need to be a different copy of each action description for each time at which the action may be executed.

This would result in a knowledge base becoming quickly overloaded and is certainly not a practical approach. First-order logic can be used to overcome this problem. This part of the ontology uses situation calculus to describe the situation of an agent

Components of Situation Calculus



Over the next few slides we will be examining the four core components of situation calculus.

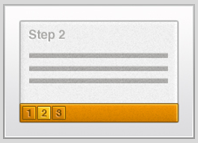
A Situation

- A situation is the result of performing a sequence of actions
  - Initial situation is labelled S_0
  - If *s* is a situation and *a* is an action then RESULT(*s*, *a*) is also a situation
- A situation is the same as its history which is the finite sequence of actions that has been performed since the initial situation S_0

The first concept we use is that of a *situation*, which denotes the state resulting from executing an action. An initial situation is typically denoted by S_0.

Other states are defined by applying an action or list of actions to a situation. The function Result(a,s) returns the situation after action a has been executed in situation s. Result([a1,a2, a3],S0) denotes the situation after a_1 is executed in S_0, a_2 is executed in the situation resulting from action a_1, and a_3 is executed in the situation resulting from action a_2. In this way we can describe the situation resulting from a sequence of actions.

It is important to highlight that situations describe a sequence of actions and are not a particular state. This is enforced by the following axiom: (RESULT(*s*, *a*) = RESULT(*s'*, *a'*)) <=> (*s* = *s'* ^ *a* = *a'*). If a situation was simply a state then many different actions can lead to the same state. For two situations to be the same the sequence of actions that lead to the situation must be the same.



Situation Calculus Example

Process - 4 Steps (Including Introduction)

Last modified: Monday, October 15, 2018 at 12:22:36 PM

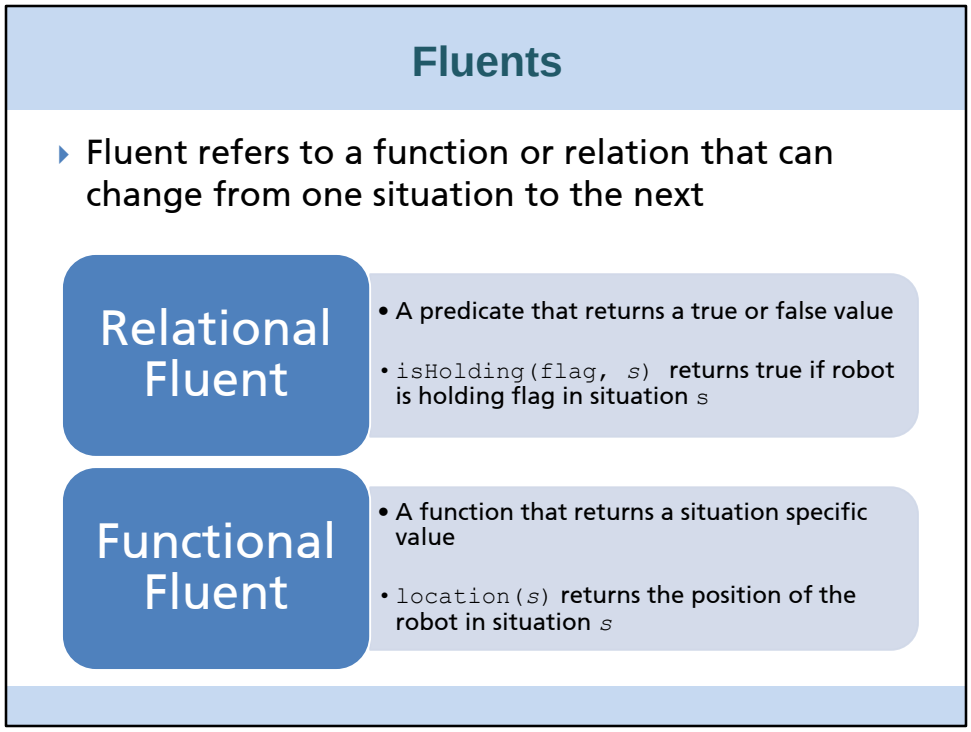
Properties

Show interaction in menu as: Multiple Items

Allow user to leave interaction: At any time

Prev/Next player buttons go to: Step in the interaction

Edit in Engage Edit Properties



Situation calculus uses *fluents* for functions or predicates that change across situations. Note that both relational and functional fluents take the situation as their last argument.

We could apply both relational and functional fluents to our previous example of the robot planting a flag at a particular grid position. For example, once the robot arrives at grid location (4, 4) it then plants the flag. However, before planting the flag it would be sensible to make sure that it is actually carrying a flag and it has not already put it down somewhere else.

We could use a relational fluent called `isHolding` to determine if the robot is holding a flag in a situation. Therefore, `isHolding(f, s)` returns true if the robot is holding the flag in the situation `s`, otherwise it returns false.

As an example of a functional fluent we could use a fluent called `location(s)` to determine the current position of the robot on the example grid in situation `s`.

Possibility Axiom

- Possibility axiom specifies preconditions that must be satisfied for the action to be performed
- It has the form `Φ(s) => Poss(a,s)` where `Φ(s)` are the list of preconditions
- If the robot is holding a flag then it can perform the action `plantFlag`

```
isHolding(flag, s) => Poss(plantFlag(), s)
```


The *possibility axiom* specifies one or more preconditions. For a particular action to be performed then all the preconditions specified in the axiom must be satisfied.

The slide shows how we could apply the concept of a possibility axiom to our robot scenario. The action `plantFlag(f)` can only be performed if the fluent `isHolding(f, s)` is true. In other words the robot can only plant a flag if it is currently holding a flag.

The Greek letter ϕ (phi) is used to represent the list of preconditions.

Successor-State Axiom

- If an action is performed it may change the value of some of the fluents
- Each fluent has a successor state axiom that determines the change to the fluent based on the action that has been taken
- For example, if a robot performs `plantFlag()` action then the successor state axiom for this action should change the value of the `isHolding` fluent true to false

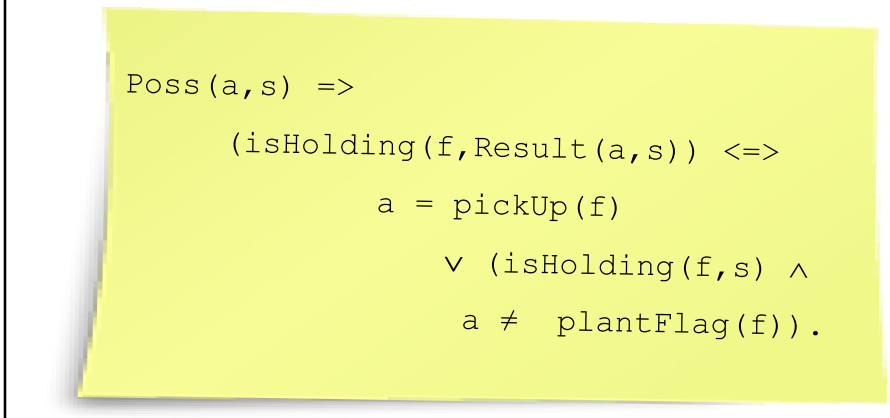
Actions undertaken in a given situation may change the value of the fluents. In our robot example the value of the `isHolding` fluent will change after a robot plants a flag. The `isHolding` fluent will return true before the flag is planted and then false after it is planted. The successor state axiom specifies how the value of a fluent will change based on the action that has been taken. We need one successor-state axiom for each fluent.

The successor-state axiom has the form:

- *Action is possible =>*

  (fluent is true in result state <=> Actions effect made it true ∨ *It was true before the action was executed and its value has not changed).*

Successor-State Axiom - Example



The successor-state axiom states how each fluent predicate evolves over time. The successor-state axiom for the `isHolding` fluent is shown here. It states that robot is currently holding a flag, is true if and only if either the executed action was `pickUp(f)`, or the agent was already holding the flag and the executed action was not `plantFlag(f)`.

Limitations of Situation Calculus

- Situation calculus designed to represent actions that are discrete and happen one at a time
  - `pickUp(f), plantFlag(f)`
- Unable to represent continuous actions that occur over a period of time such as boiling water or filling the bathtub

Situation calculus can represent discrete actions but it is unable to represent continuous actions that occur over a period.

For example, let's consider the action of boiling water. Situation calculus can describe the water before the action (water at room temperature) and the water after the action (water is at boiling temperature) but it cannot represent what happens during the action itself.

Likewise when filling a bath. Situation calculus can describe the bathtub before the action (bathtub is empty) and after the action (bathtub is full) but it cannot represent what happens during the action itself.

Event Calculus

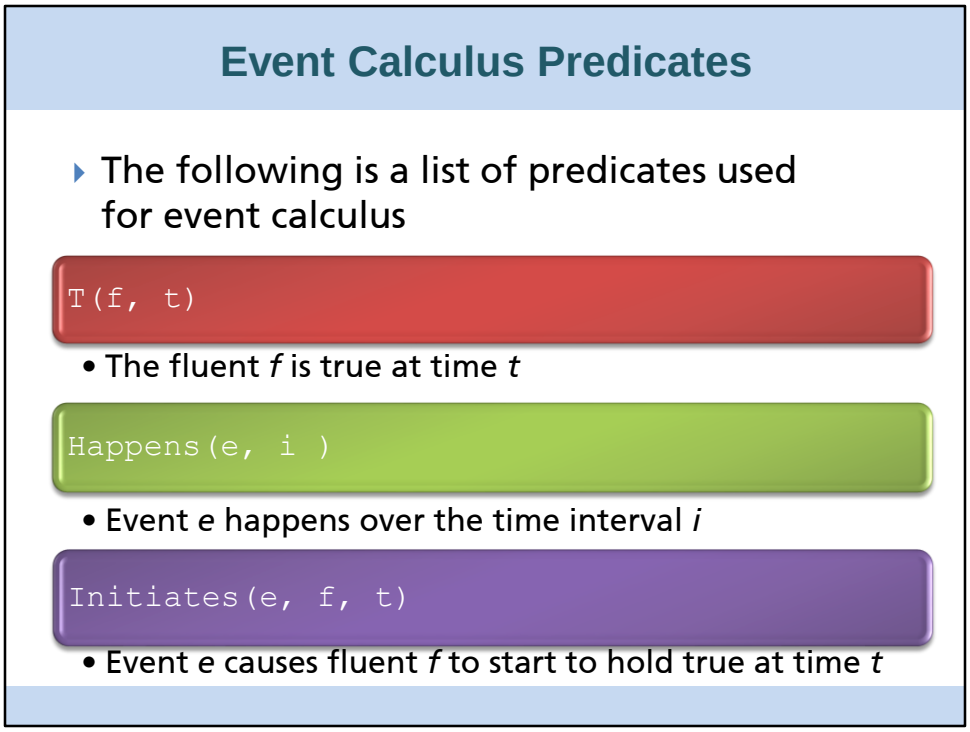
- First proposed by Robert Kowalski and Marek Sergot in 1986
- Event calculus can represent and reason about actions that occur over a period of time
- Event calculus is based on points of time rather than on situations

Robert Kowalski

Image Credit: (Permpoontanalarp 2009)

Situation calculus deals with discrete actions, i.e. actions that occur instantaneously. To better represent the real world, a general ontology must allow actions that have duration, allowing for actions that overlap. Event calculus provides this functionality.

Event calculus allows reasoning over intervals of time with actions that have duration or overlap. It can specify the value of fluents at particular point of time, the actions that took place at particular time, and their subsequent effects. Note that an event and action are the same thing.



In event calculus, fluents hold at points in time instead of situations.

This slide presents a number of the predicates used in event calculus. For example the `Happens` predicate represents an event `e` that occurs within the time interval `i`. Please note that a time interval **i** is represented by a `(start, end)` pair of times; this is, `i=(t1, t2)` such that `t1` is when the time interval starts and `t2` is where the time interval finishes.

Therefore, if we want to represent the boiling of a kettle between time 6 and time 8 then we could express it as follows:

```
Happens(kettleBoiling, (6, 8))
```

Event Calculus Predicates

`Terminates(e, f, t)`

- Event *e* causes fluent *f* to cease to hold at time *t*

`Clipped(f, i )`

- Fluent *f* ceases to be true at some point during the time interval *i*

`Restored(f, i)`

- Fluent *f* becomes true sometime during the time interval *i*

This slide presents additional predicates used in event calculus.

Example – Defining the fluent *T*(*f*, *t*)

$$T(f,t) \iff \exists\, e, t_1 \ \text{Happens}(e, t_1) \wedge \text{Initiates}(e, f, t_1) \wedge (t_1 < t) \wedge \neg \text{Clipped}(f, t_1, t)$$

The `T(f,t)` event calculus axiom is shown in the slide. The first part of this axiom states that fluent `f` is true for the time `t` if and only if an event initiated the fluent before `t` and no event terminated the fluent between the time of its initiation and time `t2`.

In more detail the formula means that the fluent represented by the term `f` is true at time `t` if:

1. An event `e` is taking place at time `t1: Happens(e, t1 )`
2. The event `e` took place in the past because we know that `t1 < t`
3. This event begins at time `t1` and sets the fluent `f` to true : `Initiates(a,f,t_1);`
4. The fluent has not been made false in the meantime: `Clipped(t_1,f , t)`

Time Interval

- Event calculus allow us to reason about time and time intervals
  - Function `Begin` and `End` pick out the earliest and latest moments in an time interval
  - The function `Time` returns the point on the timescale
  - `Duration` returns the difference between two points on the timescale

A number of functions are defined for dealing with time intervals. `Start` and `End` refer to the earliest and latest moments in an interval. `Time` returns the point on the timescale for the moment. `Duration` gives the difference between the end and start time of an interval.

Defining a Time Interval

```
Interval(i)  =>  Duration(i)  =
        (Time(End(i)) - Time (Start(i)))
```


We can now more precisely define the `Interval` function. The function `Duration` returns the difference between the end time and the start time.

Time Interval Relations

- We can use the **Begin** and **End** functions to define a more expressive range of relations for representing time intervals

```
Meet(i, j) <=> End(i) = Begin(j)
```

- Two intervals meet if the end time of the first equals the start time of the second

```
Before(i, j) <=> End(i) < Begin(j)
```

- One interval is before another if the end time of the first is less than the start time of the second

The following are examples of the relations defined in the slide:

The interval represented by the reign of Edward VIII ends when the reign of George the VI begins. George VI became British Monarch upon the abdication of Edward VIII

•`Meet( ReignOf(Edward VIII), ReignOf(George VI) )`

The interval represented by the 12th century occurs before the interval represented by the 15th century.

•`Before(1200AD , 1500AD)`

Time Interval Relations

```
During(i, j) <=> Begin(j) < Begin(i) < End(i) < End(j)
```

- The interval `i` occurs during interval `j` if `j` begins before `i` and `j` end after `i`

```
Overlap(i, j) <=> Begin(i) < Begin(j) < End(i) < End(j)
```

- The interval `i` overlaps with interval `j` if `i` begins before `j` and `i` ends before `j`

```
Begins(i, j)  <=> Begin(i) = Begin(j)
```

- Interval `i` begins at the same time as interval `j`. Similar definition can be used to define `After(i,j)`

The following are examples of the relations defined in the slide:

The interval represented by the 1960s occurs during the 20th century.
•`During(20thCentury , 1960s)`

The interval represented by the 1950s decade overlaps with the reign of Elvis.
•`Overlap(1950s, Reign(Elvis))`

The interval represented by the 20th century begins at the same time as the interval represented by the 1900 decade.
•`Begins(20hCentury, 1900s)`

Event Calculus Review Question

Quiz - 1 Question

Last modified: Monday, October 15, 2018 at 12:22:21 PM

Properties

On passing, 'Finish' button:	Goes to next slide
On failing, 'Finish' button:	Goes to next slide
Allow user to leave quiz:	At any time
User may view slides after quiz:	Any time
Show quiz in menu as:	Multiple Items

Edit in Quizmaker Edit Properties

Summary

Described Situation Calculus	Introduced the area of situation calculus. Described the concept of a situation and a fluent. Outlined the limitations of situation calculus.
Described Event Calculus	Described the concepts of an event and outlined how event calculus addresses the limitations of situation calculus. Examined event predicates and demonstrated how they can be used to reason about time intervals.

References

ArsElectronica (2010) ASIMO [photo], (CC BY-NC-ND 2.0), available: https://www.flickr.com/photos/36085842@N06/4945217670/ [accessed 18 Apr 2019].

Null0 (2006) *Professor John McCarthy discusses the Summit* [photo], CC-BY-SA-2.0, available: https://www.flickr.com/photos/null0/272015955/ [accessed 29 Apr 2019].

Permpoontanalarp, Y. (2009) Robert Kowalski [photo], CC BY 3.0, available: https://en.wikipedia.org/wiki/Robert_Kowalski#/media/File:Robert_Kowalski.jpg [accessed 29 Apr 2019].

All CC0 licensed images were accessed from Articulate 360's Content Library