Combining Event Calculus and Description Logic Reasoning via Logic Programming

Peter Baumgartner
Data61|CSIRO and ANU, Canberra, Australia
Project Background: A Logic Based System for Situational Awareness

Situational awareness ≈ comprehending system state as it evolves over time

Example: Food Supply Chain

- Are goods delivered within 3 hours and stored below 25°C?
- Why is the truck late?
- What is the expected quality (shelf life) of the goods?

What’s the problem?

- **Multiple aspects**: temporal/causal/structural/physical/…
- Events *happened* ≠ events *reported* (errors, incomplete, late …)
- **Uncertainty**: *multiple* plausible explanations for given facts

This Work

- More expressive modelling language for better domain modelling
- Extension 1: Description logic interface
- Extension 2: Event calculus
- Implementation in Fusemate system
Fusemate - Language and Model Computation Overview

Input language: Prolog-like rules
R(a,b)
R(X,Y) :- R(Y,X)
R(X,Z) :- R(X,Y), r(Y,Z)

Default negation: stratification “by time”
GoodSleep(time) :- 
  WakeUp(time), 
  GoToBed(t), t <= time - 8,
  not (t <= s, s < time, WakeUp(s))

Disjunctions: possible model semantics [Sakama 90]
Thirsty(time) or Hungry(time) :- GoodSleep(time)

Belief revision
fail(+ GoToBed(time - 8)) :-
  WakeUp(time),
  not (GoToBed(t), t <= time - 8)

Models
<table>
<thead>
<tr>
<th>R(a,b)</th>
<th>R(b,a)</th>
</tr>
</thead>
</table>

Bottom-up procedures
(Hyper tableau, Hyper resolution, ...)

GoToBed not WakeUp

21:00 6:00

unhappy(time) :- Now(time), not winLottery(time+7)

Models
<table>
<thead>
<tr>
<th>Thirsty(10)</th>
<th>Hungry(10)</th>
<th>Hungry(10)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thirsty(10)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Application: Situational awareness = model computation
Stratified Model Computation

Modelling Setup for Situational Awareness

- EDB: Timestamped facts (“events”) \( E_0, E_1, E_2, \ldots \)
- IDB: Models for derived predicates up to “now”

Model Computation

- EDBs: \( E_{0,1,2,\ldots} \)
- IDBs: \( I_{0,1,2,\ldots} \)

Bottom-up application of logic program rules until fixpoint

Revision

Revision = programmable addition/removal of events in the past + restart of model computation

Effective because default negation can refer only to the past*
Logic Program Example: Supply Chain

Derived "In" relation

In(time, obj, cont) :-
    Load(time, obj, cont)

// In transitivity
In(time, obj, cont) :-
    In(time, obj, c),
    In(time, c, cont)

// Frame axiom for In
In(time, obj, cont) :-
    In(prev, obj, cont),
    Step(time, prev),
    not Unload(prev, obj, cont),
    not (In(prev, obj, c), Unload(time, c, cont))

Integrity constraints and revision

// No Unload without earlier Load
fail :-
    Unload(time, obj, cont),
    not (Load(t, obj, cont), t < time))

// Unload a different object
fail(- Unload(time, obj, cont), + Unload(time, o, cont)) :-
    Unload(time, obj, cont),
    not (Load(t, obj, cont), t < time),
    Load(t, o, cont),
    t < time,
    SameBatch(t, b),
    ((b contains obj) && (b contains o))

Experience: Logic programs often
(a) are too low-level, and
(b) suffer from non-termination for "tuple generating dependencies"

-> Extend reasoning framework with Description Logic reasoning and Event Calculus
Description Logic Reasoner Interface
Description Logics

- A DL KB consists of a TBox (concept definitions) and an ABox (instance assertions)
- The concrete choice of DL is not important here, but must include ALC and satisfiability must be decidable

\[ \text{KB} = (\text{ABox}, \text{TBox}) \]

**Reasoning**

- Does Box\(_0\) have a temp attribute?
- Is Box\(_5\) a FruitBox?
- Are FruitBox and ToyBox disjoint?
- Is (ABox, TBox) satisfiable?

**[CADE-2021 SD]:**

DL ALCIF by mapping to Fusemate disjunctive logic program + loop check
**Description Logics + Logic Programming Approach - Overview**

**DL and LP are Complementary**
Open world vs closed world, entailment vs models, unique name assumption no/yes

**Here: Timed Setting**

<table>
<thead>
<tr>
<th>Time</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Load Box₀</td>
<td>Load Box₂</td>
<td>Load Box₃</td>
<td>Load Box₄</td>
<td>Unload</td>
</tr>
<tr>
<td>Sensor</td>
<td>Box₀ : −10°</td>
<td>Box₂ : 10°</td>
<td>Box₀ : 2°</td>
<td>Box₀ : 20°</td>
<td></td>
</tr>
</tbody>
</table>

**Goal: Understanding situation as it evolves over time**

- **t=10**
  - Box₀ has a known* low temp
  - Box₁ has some unknown temp
  - Box₂ is not known to have a temp
  - Box₃ is known to have no temp

- **t=20**
  - Box₂ has a known high temp

- **t=30**
  - Box₀ has a known high temp

* “known” = “follows wrt FOL”

**Approach: DL+Rules(+Event Calculus)**

- **DL**: black-box theory reasoner - can talk about implicitly existing individuals
- **EC**: actions and their effects over time - can add “from now on unless change” to above properties
- **Rules**: glue between DL+EC - can bring in concrete domains (numbers)
Description Logic Interface - Queries

DL Query Example (Body Literal)

(ABox(I, 20), TBox) \models Box_2 : FruitBox, (Box_2, Temp) : High

ABox in interpretation I at time 20

DL Query Syntax

The following forms can be used in rule bodies

\[ T \models \vec{q} \quad \text{DLISSAT}(T) \quad \text{DLISUNSAT}(T) \]
\[ (A, T) \models \vec{q} \quad \text{DLISSAT}(A, T) \quad \text{DLISUNSAT}(A, T) \]

- \( T \) is a TBox
- \( A \) is an ABox, implicitly \( A(\text{current I}, \text{now}) \) where
  \[ A(I, t) = \{ a : C \mid a : C @ t \in I \} \cup \{(a, b) : R \mid (a, b) : R @ t \in I \} \]
- \( \vec{q} \) is a query, i.e., a sequence of terms representing an ABox
- \( (A, T) \models \vec{q} \) means “\( A \cup T \models \bigwedge \vec{q} \)” as FOL formulas
Description Logic Interface - Examples

**Materialization**

Can derive new ABox assertions (even in the past)!

\[
x : \text{Box} @ \text{time} :-
\]

\[(x : _ @ \text{time}), \quad \text{// x is an individual in an ABox assertion at “time”}
\]

TBox |= x : \text{Box} // Implicit ABox (A(I, time)

Variables in DL Queries grounded now

**Box has temp**

TempBox(time, box) :-

\[\text{box} : \text{Box} @ \text{time},\]

TBox |= box : \exists \text{Temp} . \text{TempClass}

**Box has known temp**

KnownTempBox(time, box) :-

\[\text{box} : \text{Box} @ \text{time},\]

\[\text{temp} \in \{\text{Low}, \text{High}\}, \text{// Guess}\]

TBox |= (box, temp): \text{Temp}

**Box never had known high temp in the past**

ColdBox(time, box) :-

\[\text{box} : \text{Box} @ \text{time},\]

\[\text{not} ( \quad \text{t < time},\]

\[\quad (A(I, t), \text{TBox}) |= \text{box} : \text{Box}, (\text{box, High}) : \text{Temp} )\]

(Stratified) DL call under default negation!
Description Logic Interface - Semantics

Query Evaluation
Reduce query evaluation to standard DL knowledge base satisfiability

\[(A, T) \models a : C \quad \text{iff } (A \cup \{a : \neg C\}, T) \text{ is unsatisfiable}\]

\[(A, T) \models (a, b) : r \quad \text{iff } (A \cup \{a : \forall r. \neg B, b : B\}, T) \text{ is unsatisfiable, with } B \text{ fresh}\]

Stratification

- Implicit ABox \(A(I, t)\) - use concept and role assertions timed \(t\)
- Explicit ABox: not automatically, use with care :)  

Unique Name Assumption (UNA)

- DL does not assume UNA

  E.g. \(A = \{(c, a) : r, (c, b) : r\}\) with functional \(r\) is satisfiable only if \(I(a) = I(b)\)

- LP does assume unique name assumption, i.e., \(I(a) \neq I(b)\)

- Solution: enforce UNA in DL by adding axioms

  E.g. \(N = \{a, b, c\}\) are all current named ABox individuals

  Add to ABox \(\{a : N_{ab}, b : \neg N_{ab}, a : N_{ac}, c : \neg N_{ac}, \ldots\}\) where \(N_{xy}\)'s are fresh concept names
Description Logic Interface - Soundness and Completeness

Model computation soundness and completeness rests on the following properties

**DL-safe rules**
- **Named** individuals: those that appear explicitly in ABox assertions
- **Unnamed** individuals: implicitly constructed (Skolem)
- Rules are DL-safe: unnamed individuals cannot escape their query scope

**Monotonicity**
- Rules \( H \models B \) must be monotonic: if \( I \models B \) and \( J \supseteq I \) then \( J \models B \)
- No problem with stratified negation
- [OK] DL queries \( T \models \overrightarrow{q} \) and DLISUNSAT(\( T \)) are always monotonic by monotonicity of FOL
- DL queries \((A, T) \models \overrightarrow{q}, \text{DLISUNSAT}(A, T), \text{DLISSAT}(T)\) and \(\text{DLISSAT}(A, T)\) use with care

**Compactness**
- Fixpoint model requires transfinite induction in general
- Not effective for aggregation operator \{\( P(x, t) \mid Q(x, s), s < t \)\}
- However not a problem because interest only in finite models
- (DL query evaluation always compact because of FOL)
Event Calculus
Event Calculus [Kowalski & Sergot 1986]

- The event calculus (EC) is a logical language for representing and reasoning about actions and their effects
- The formulation below follows the original logic program, with adaptations and extensions for DL

**Actions and Fluents**

- A **fluent** is a property that **HoldsAt** over a time period
- Fluents are **initiated** or **terminated** by **actions** that **happen** at given time point

<table>
<thead>
<tr>
<th>Time</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>Load Box₀</td>
<td>Load Box₂</td>
<td>Load Box₃</td>
<td>Load Box₄</td>
<td>Unload</td>
</tr>
<tr>
<td>Sensor</td>
<td>Box₀ : −10°</td>
<td>Box₂ : 10°</td>
<td>Box₀ : 2°</td>
<td>Box₀ : 20°</td>
<td></td>
</tr>
</tbody>
</table>

\[ t=20 \] Load(Box₂) initiates OnTruck(Box₂) \[ \text{HoldsAt}(20, \text{OnTruck}(\text{Box}_2)) \in I \]
\[ t=50 \] Unload terminates OnTruck(Boxᵢ) \[ \text{HoldsAt}(50, \text{OnTruck}(\text{Box}_i)) \notin I \]

**Problem Specific Axioms**

- Initiates(time, Load(box), OnTruck(box)) :- box : Box @ time
- Terminates(time, Unload(box), OnTruck(box)) :- HoldsAt(time, OnTruck(box))

**Problem Specific Events**

- Happens(20, Load(Box₂))
- Happens(50, Unload)

**EC Library**

- HoldsAt(time+1, f) :- Initiates(time, f), not Terminated(time, f)
- HoldsAt(time, f) :- HoldsAt(time-1, f), not Terminated(time, f)
Event Calculus

Linking DL with EC

- Often, ABox assertions are meant to hold over time instead of time points only.
- That is, timed ABox assertions can be fluents now.

```
“From time 0 on” vs “At time 0”
HoldsAt(0, Box5: Box ⊨ ⊪ Temp . TempClass) vs Box5: Box ⊨ ⊪ Temp . TempClass @ 0
```

- Add axioms for turning ABox fluents into timed ABox assertions again (but not vice versa).

```
x : c @ time :-
HoldsAt(time, x : c)

(x, y): r @ time :-
HoldsAt(time, (x, y): r)
```

Rule with ABox Fluent, Action and Concrete Data

- Initiates(time, SensorEvent(box, temp), (box, High): Temp) and
- Terminates(time, SensorEvent(box, temp), (box, Low): Temp) :-
  Happens(time, SensorEvent(box, temp)),
  temp > 0

If box temp sensor > 0 then box temp is “high” from now on and no longer “low” from on
Conclusion

Contributions

- **Theoretical**: very liberal Rule + DL combination, conditions for soundness and completeness
- **KR language design**: extension of LP with DL + EC “very useful” for situational awareness
  Hard to quantify, but see paper for “complex” anomaly detection example

Open Problem

- The [ramification problem](#) is concerned with indirect consequences of actions, such as **conflicts**
  - It occurs in a pronounced way here
  - Example: rule for terminating a box’ temp fluent
    \[
    \text{Terminated}(\text{time} + 1, (\text{box}, \text{temp}) : \text{Temp}) :- \\
    \text{RemoveTemp}(\text{time}, \text{box}), \ // \text{Some condition for removing box Temp} \\
    (\text{box}, \text{temp}) : \text{Temp} @ \text{time} \ // \text{Attribute to be removed}
    \]
  - This rule does not always work
    E.g, for a FruitBox the box’ temp attribute is entailed by the “black box” TBox
  - AFAIK “repairing” ABoxes is ongoing research but can be done in special cases

### Box ▰ ∀ temp.TempClass
FruitBox ▰ ∃ temp.TempClass
ToyBox ▰ ¬∃ temp.TempClass
FruitBox ▰ Box
ToyBox ▰ Box

    temp is a functional role