The Fusemate Logic Programming System for Situational Awareness

Peter Baumgartner
Data61 | CSIRO
The Australian National University
Situational Awareness $\approx$ comprehending system state as it evolves over time

Factory Floor
Are the operations carried out according to the schedule?

Food Supply Chain
Are goods delivered within 3 hours and stored below 25ºC?
Why is the truck late?
What is the expected quality (shelf life) of the goods?

Data Cleansing
Does the database have complete, correct and relevant data?

What’s the problem?

• The domain model needs to cover multiple aspects:
  Temporal/cause/structural/physical/…

• Events happened $\neq$ events reported (errors, incomplete, late …)

• Can only hope for multiple plausible explanations
Situational Awareness  \(\approx\) comprehending system state as it evolves over time

Factory Floor
Are the operations carried out according to the schedule?

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?

Data Cleansing
Does the database have complete, correct and relevant data?

What’s the problem?
• The domain model needs to cover multiple aspects:
  Temporal/causal/structural/physical/…
• Events happened \(\neq\) events reported (errors, incomplete, late …)
• Can only hope for multiple plausible explanations
Situational Awareness ≈ comprehending system state as it evolves over time

Factory Floor
Are the operations carried out according to the schedule?

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?

Data Cleansing
Does the database have complete, correct and relevant data?

What’s the problem?
- The domain model needs to cover multiple aspects:
  Temporal/causal/structural/physical/…
- Events **happened** ≠ events **reported** (errors, incomplete, late …)
- Can only hope for **multiple** plausible explanations
Situational Awareness \(\approx\) comprehending system state as it evolves over time

**Factory Floor**
Are the operations carried out according to the schedule?

**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?

**Data Cleansing**
Does the database have complete, correct and relevant data?

**What’s the problem?**
- The domain model needs to cover multiple aspects:
  Temporal/causal/structural/physical/…
- Events *happened* \(\neq\) events *reported* (errors, incomplete, late …)
- Can only hope for multiple plausible explanations

Logic program + ontologies/event calculus
Belief revision
Models
Observation: truck is in Sydney at the warehouse
Observation: truck is in Sydney at the warehouse
Observation: tomatoes are loaded
Observation: tomatoes are loaded
Assumption as per schedule: truck is on the road
Assumption as per schedule: truck is on the road

T  T+1
Report: truck is on the road
Report: truck is on the road

T  
T+1  
T+2
Conclusion: truck is on the road for too long - tomatoes are no longer fresh
Conclusion: truck is on the road for too long - tomatoes are no longer fresh
Report: actually, at T+1 truck was still in Sydney warehouse
Report: actually, at T+1 truck was still in Sydney warehouse
Conclusion: tomatoes are still fresh at T+2
Conclusion: tomatoes are still fresh at $T+2$
No information at T+3
T+3: What if truck is on the road?
T+3: What if truck is on the road?
T+3: What if truck is on the road? At Canberra warehouse?
Report: truck at Canberra warehouse
Report: truck at Canberra warehouse
Report: truck at Canberra warehouse

We use logic programming
Logic Programming
Logic Programming

Algorithm = Logic + Control (Kowalski)

Pieces of reusable domain knowledge
Chained by inference engine
Logic Programming

Algorithm = Logic + Control (Kowalski)

Pieces of reusable domain knowledge
Chained by inference engine
Logic Programming

Algorithm = Logic + Control (Kowalski)

Pieces of reusable domain knowledge
Chained by inference engine
Logic Programming

Algorithm = Logic + Control (Kowalski)

- Pieces of reusable domain knowledge
- Chained by inference engine
Logic Programming

Algorithm = Logic + Control (Kowalski)

- Pieces of reusable domain knowledge
- Chained by inference engine

Cats drink milk
Milk is in the fridge
Tom is a cat
Tom is thirsty
Logic Programming

Algorithm = Logic + Control (Kowalski)

Pieces of reusable domain knowledge
Chained by inference engine
Logic Programming

Algorithm = Logic + Control (Kowalski)
- Pieces of reusable domain knowledge
- Chained by inference engine

Logic Programs

Tom is thirsty
Cats drink milk
Milk is in the fridge
Coles sells milk
Tom is a cat
Logic Programming

Algorithm = Logic + Control (Kowalski)

- Pieces of reusable domain knowledge
- Chained by inference engine

Logic Programs

If-then rules

\[
\text{drinks}(x, \text{Milk}) :- \text{cat}(x)
\]

if \text{cat}(x) then \text{drinks}(x, \text{Milk})
Logic Programming

Algorithm = Logic + Control (Kowalski)

- Pieces of reusable domain knowledge
- Chained by inference engine

Logic Programs

If-then rules

- drinks(x, Milk) :- cat(x)  if cat(x) then drinks(x, Milk)
- inBowl(time+1, Milk) :- inFridge(time, Milk)
Logic Programming

Algorithm = Logic + Control (Kowalski)

- Pieces of reusable domain knowledge
- Chained by inference engine

Logic Programs

If-then rules

- drinks(x, Milk) :- cat(x)
- inBowl(time+1, Milk) :- inFridge(time, Milk)

Facts

- cat(Tom)
- inFridge(5, Milk)
Logic Programming

Algorithm = Logic + Control (Kowalski)

Pieces of reusable domain knowledge
Chained by inference engine

Logic Programs

If-then rules

drinks(x, Milk) :- cat(x)  \[\text{if cat(x) then drinks(x, Milk)}\]
inBowl(time+1, Milk) :- inFridge(time, Milk)

Facts

cat(Tom)
inFridge(5, Milk)

Default negation

inFridge(time, Milk) :- not inBowl(time, Milk)
Logic Programming

Algorithm = Logic + Control (Kowalski)

- Pieces of reusable domain knowledge
- Chained by inference engine

Logic Programs

If-then rules
- drinks(x, Milk) :- cat(x)  \[\text{if cat(x) then drinks(x, Milk)}\]
- inBowl(time+1, Milk) :- inFridge(time, Milk)

Facts
- cat(Tom)
- inFridge(5, Milk)

Default negation
- inFridge(time, Milk) :- not inBowl(time, Milk)
Logic Programming

Algorithm = Logic + Control (Kowalski)

Pieces of reusable domain knowledge
Chained by inference engine

Logic Programs

If-then rules

drinks(x, Milk) :- cat(x)  # if cat(x) then drinks(x, Milk)
inBowl(time+1, Milk) :- inFridge(time, Milk)

Facts

cat(Tom)
inFridge(5, Milk)

Default negation

inFridge(time, Milk) :- not inBowl(time, Milk)

Disjunctions

drinks(x, Milk) or drinks(x, Water) :- cat(x), thirsty(x)

Tom is a cat

Cats drink milk

Milk is in the fridge

Coles sells milk

Tom is thirsty

“innocent :- not guilty”
Logic Programming

Algorithm = Logic + Control (Kowalski)

Pieces of reusable domain knowledge
Chained by inference engine

Logic Programs

If-then rules
drinks(x, Milk) :- cat(x)  \( \text{if } \text{cat}(x) \text{ then } \text{drinks}(x, \text{Milk}) \)

inBowl(time+1, Milk) :- inFridge(time, Milk)

Facts

cat(Tom)
inFridge(5, Milk)

Default negation

inFridge(time, Milk) :- not inBowl(time, Milk)

“innocent :- not guilty”

Disjunctions

drinks(x, Milk) or drinks(x, Water) :- cat(x), thirsty(x)

Integrity constraints

fail :- cat(x), mouse(x)
Logic Programming

Algorithm = Logic + Control (Kowalski)
- Pieces of reusable domain knowledge
  - Chained by inference engine

Logic Programs

If-then rules
- drinks(x, Milk) :- cat(x)  \quad \text{if cat(x) then drinks(x, Milk)}
- inBowl(time+1, Milk) :- inFridge(time, Milk)

Facts
- cat(Tom)
- inFridge(5, Milk)

Default negation
- inFridge(time, Milk) :- not inBowl(time, Milk)

Disjunctions
- drinks(x, Milk) or drinks(x, Water) :- cat(x), thirsty(x)

Integrity constraints
- fail :- cat(x), mouse(x)

Purpose

- Query answering (who drinks milk?), planning (get Tom some milk),
- abduction (why did we go to Coles?), model computation (what do we know about Tom?)
Logic Programming

Prolog - “top down query answering”

Answer Set Programming - “model computation”
Logic Programming

Prolog - “top down query answering”

append([], L, L)
append([H|T], L, [H|R]) :-
    append(T, L, R)

Answer Set Programming - “model computation”
Logic Programming

Prolog - “top down query answering”

append([], L, L)
append([H|T], L, [H|R]) :-
    append(T, L, R)

?- append([1,2], [3,4], L)
?- append([1,2], L, [1,2,3,4])
?- append(K, L, [1,2,3,4])
?- append(K, L, M)
Logic Programming

Prolog - “top down query answering”

append([], L, L)
append([H|T], L, [H|R]) :-
    append(T, L, R)

?- append([1,2], [3,4], L)
?- append([1,2], L, [1,2,3,4])
?- append(K, L, [1,2,3,4])
?- append(K, L, M)  [X,Y] ++ L = [X,Y|L] ...

Answer Set Programming - “model computation”
Logic Programming

Prolog - “top down query answering”

append([], L, L)
append([H|T], L, [H|R]) :-
   append(T, L, R)

?- append([1,2], [3,4], L)
?- append([1,2], L, [1,2,3,4])
?- append(K, L, [1,2,3,4])
?- append(K, L, M) [X,Y] ++ L = [X,Y|L] ...

Answer Set Programming - “model computation”

r(X,X)                        r(a,b)
r(X,Y) :- r(Y,X)              r(c,b)
r(X,Z) :- r(X,Y), r(Y,Z)
Logic Programming

Prolog - “top down query answering”

\[
\begin{align*}
\text{append}([], L, L) \\
\text{append}([H|T], L, [H|R]) & : - \\
& \quad \text{append}(T, L, R) \\
?- & \text{append}([1,2], [3,4], L) \\
?- & \text{append}([1,2], L, [1,2,3,4]) \\
?- & \text{append}(K, L, [1,2,3,4]) \\
?- & \text{append}(K, L, M) \ [X,Y] ++ L = [X,Y|L] ...
\end{align*}
\]

Answer Set Programming - “model computation”

\[
\begin{align*}
r(X,X) & \\
r(X,Y) & : - r(Y,X) \\
r(X,Z) & : - r(X,Y), r(Y,Z) \\
a & : - \textbf{not} a
\end{align*}
\]
Logic Programming

Prolog - “top down query answering”

```
append([], L, L)
append([H|T], L, [H|R]) :-
  append(T, L, R)

?- append([1,2], [3,4], L)
?- append([1,2], L, [1,2,3,4])
?- append(K, L, [1,2,3,4])
?- append(K, L, M)  [X,Y] ++ L = [X,Y|L] ...
```

Answer Set Programming - “model computation”

```
r(X,X)                        r(a,b)
r(X,Y) :- r(Y,X)              r(c,b)
r(X,Z) :- r(X,Y), r(Y,Z)
a :- not a                     No model
```

Logic Programming

Prolog - “top down query answering”

append([], L, L)
append([H|T], L, [H|R]) :-
    append(T, L, R)

?- append([1,2], [3,4], L)
?- append([1,2], L, [1,2,3,4])
?- append(K, L, [1,2,3,4])
?- append(K, L, M) [X,Y] ++ L = [X,Y|L] ...

Answer Set Programming - “model computation”

r(X,X)
r(a,b)
r(X,Y) :- r(Y,X)
r(c,b)
r(X,Z) :- r(X,Y), r(Y,Z)

a :- not a   No model

a :- not b
b :- not a
Logic Programming

Prolog - “top down query answering”

append([], L, L)
append([H|T], L, [H|R]) :-
append(T, L, R)

?- append([1,2], [3,4], L)
?- append([1,2], L, [1,2,3,4])
?- append(K, L, [1,2,3,4])
?- append(K, L, M) [X,Y] ++ L = [X,Y|L] ...
Logic Programming

Prolog - “top down query answering”

append([], L, L)
append([H|T], L, [H|R]) :-
    append(T, L, R)

?- append([1,2], [3,4], L)
?- append([1,2], L, [1,2,3,4])
?- append(K, L, [1,2,3,4])
?- append(K, L, M) [X,Y] ++ L = [X,Y|L] ...

Answer Set Programming - “model computation”

r(X,X) r(a,b)
\[ r(X,Y) :- r(Y,X) \]
\[ r(X,Z) :- r(X,Y), r(Y,Z) \]
a :- not a No model
\[ a :- not b \] Model 1: \{a\}
b :- not a Model 2: \{b\}

unhappy(now) :- not win(now+1)
Logic Programming

Prolog - “top down query answering”

append([], L, L)
append([H|T], L, [H|R]) :- append(T, L, R)
?- append([1,2], [3,4], L)
?- append([1,2], L, [1,2,3,4])
?- append(K, L, [1,2,3,4])
?- append(K, L, M) [X,Y] ++ L = [X,Y|L] ...

Answer Set Programming - “model computation”

r(X,X)
r(X,Y) :- r(Y,X)
r(X,Z) :- r(X,Y), r(Y,Z)
a :- not a       No model
a :- not b       Model 1: {a}
b :- not a        Model 2: {b}
unhappy(now) :- not win(now+1)

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>NP</td>
<td>Δ₂²</td>
</tr>
<tr>
<td>{w}</td>
<td>NP</td>
<td>Δ₂²</td>
<td>NP</td>
<td>Δ₂²</td>
<td>NP</td>
<td>Δ₂²</td>
</tr>
<tr>
<td>{h}</td>
<td>NS</td>
<td>Δ₂²</td>
<td>NS</td>
<td>Δ₂²</td>
<td>NS</td>
<td>Δ₂²</td>
</tr>
<tr>
<td>{v}</td>
<td>Δ₂²</td>
<td>Δ₂²</td>
<td>Δ₂²</td>
<td>Δ₂²</td>
<td>Δ₂²</td>
<td>Δ₂²</td>
</tr>
</tbody>
</table>

4.5 Summary of Results and Discussion

The complexity of Brave Reasoning and Cautious Reasoning for each answer set of M ∈ P但对于 each DLV program is in co-NP, and thus the checking problem is in co-NP. Moreover, it is well-known that, for each DLV program, an answer set can be reduced to answer set checking for DLV programs. This clearly allows us to reduce answer set checking for DLV programs to the problem of checking for DLV programs.

Recall that, given a DLV program, we first decide whether there exists an answer set or not. If this is not the case, we stop. Otherwise, we compute the complementary problem of checking for DLV programs is in PSPACE. The same reduction holds for the problem of checking for DLV programs.

The complexity of Brave Reasoning and Cautious Reasoning for each answer set of M ∈ P for each DLV program is in co-NP, and thus the checking problem is in co-NP. Moreover, it is well-known that, for each DLV program, an answer set can be reduced to answer set checking for DLV programs. This clearly allows us to reduce answer set checking for DLV programs to the problem of checking for DLV programs.

Recall that, given a DLV program, we first decide whether there exists an answer set or not. If this is not the case, we stop. Otherwise, we compute the complementary problem of checking for DLV programs is in PSPACE. The same reduction holds for the problem of checking for DLV programs.
Logic Programming

Prolog - “top down query answering”

\[
\text{append([], L, L)} \\
\text{append([H|T], L, [H|R]) : - append(T, L, R)} \\
?- \text{append([1,2], [3,4], L)} \\
?- \text{append([1,2], L, [1,2,3,4])} \\
?- \text{append(K, L, [1,2,3,4])} \\
?- \text{append(K, L, M) [X,Y] ++ L = [X,Y|L] ...}
\]

“More operational”
General purpose PL
Unification/DFBS

Answer Set Programming - “model computation”

\[
\text{r(X,X)} \\
\text{r(X,Y) :- r(Y,X)} \\
\text{r(X,Z) :- r(X,Y), r(Y,Z)} \\
a :- \text{not a} \quad \text{No model} \\
a :- \text{not b} \quad \text{Model 1: \{a\}} \\
b :- \text{not a} \quad \text{Model 2: \{b\}} \\
\text{unhappy(now) :- not win(now+1)}
\]

\[
\begin{array}{cccccccc}
| {} | \{w\} | \{\text{not}_a\} | \{\text{not}_a, w\} | \{\text{not}\} | \{\text{not}, w\} | \\
\hline
\{\} & P & P & P & P & NP & \Delta^F_2 \\
\{w\} & NP & \Delta^F_2 & NP & \Delta^F_2 & NP & \Delta^F_2 \\
\{v\} & \Sigma^F_2 & \Delta^F_2 & \Sigma^F_2 & \Delta^F_2 & \Sigma^F_2 & \Delta^F_2 \\
\end{array}
\]

“More declarative”
NP-complete (or harder) search problems
Grounding (SAT solving)
Logic Programming

Prolog - “top down query answering”

```
append([], L, L)
append([H|T], L, [H|R]) :-
    append(T, L, R)

?- append([1,2], [3,4], L)
?- append([1,2], L, [1,2,3,4])
?- append(K, L, [1,2,3,4])
?- append(K, L, M) [X,Y] ++ L = [X,Y|L] ...
```

“More operational”

General purpose PL

Unification/DFBS

Fusemate

Model computation
Functions/data structures
Stratified (negation) by time
Belief revision:

```
fail(+win(now-1)) :- happy(now)
```

Answer Set Programming - “model computation”

```
r(X,X)       r(a,b)
r(X,Y) :- r(Y,X) r(c,b)
r(X,Z) :- r(X,Y), r(Y,Z)
```

```
a :- not a          No model
b :- not a
```

```
unhappy(now) :- not win(now+1)
```

```
<table>
<thead>
<tr>
<th></th>
<th>{}</th>
<th>{w}</th>
<th>{not_a}</th>
<th>{not_a,w}</th>
<th>{not}</th>
<th>{not,w}</th>
</tr>
</thead>
<tbody>
<tr>
<td>{}</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>P</td>
<td>NP</td>
<td>Δ₂⁻⁺</td>
</tr>
<tr>
<td>{v₁}</td>
<td>NP</td>
<td>Δ₁⁻⁺</td>
<td>NP</td>
<td>Δ₁⁻⁺</td>
<td>NP</td>
<td>Δ₂⁻⁺</td>
</tr>
<tr>
<td>{v}</td>
<td>Δ₁⁻⁺</td>
<td>Δ₁⁻⁺</td>
<td>Δ₁⁻⁺</td>
<td>Δ₁⁻⁺</td>
<td>Δ₁⁻⁺</td>
<td>Δ₁⁻⁺</td>
</tr>
</tbody>
</table>
```

“More declarative”

NP-complete (or harder) search problems

Grounding (SAT solving)
Sokoban Answer Set Solver Program [DLV]

\[
\text{time}(T) :- \#\text{int}(T). \\
\text{actiontime}(T) :- \#\text{int}(T), \ T \neq \#\text{maxint}. \\
\text{left}(L_1,L_2) :- \text{right}(L_2,L_1). \\
\text{bottom}(L_1,L_2) :- \text{top}(L_2,L_1). \\
\text{adj}(L_1,L_2) :- \text{right}(L_1,L_2). \\
\text{adj}(L_1,L_2) :- \text{left}(L_1,L_2). \\
\text{adj}(L_1,L_2) :- \text{top}(L_1,L_2). \\
\text{adj}(L_1,L_2) :- \text{bottom}(L_1,L_2). \\
\text{location}(L) :- \text{adj}(L,\_). \\

\text{push}(B, \text{right}, B_1, T) \vee \neg \text{push}(B, \text{right}, B_1, T) :- \text{reachable}(L, T), \text{right}(L, B), \text{box}(B, T), \\
\quad \text{pushable\_right}(B, B_1, T), \text{good\_pushlocation}(B_1), \text{actiontime}(T). \\
\text{push}(B, \text{left}, B_1, T) \vee \neg \text{push}(B, \text{left}, B_1, T) :- \text{reachable}(L, T), \text{left}(L, B), \text{box}(B, T), \\
\quad \text{pushable\_left}(B, B_1, T), \text{good\_pushlocation}(B_1), \text{actiontime}(T). \\
\text{push}(B, \text{up}, B_1, T) \vee \neg \text{push}(B, \text{up}, B_1, T) :- \text{reachable}(L, T), \text{top}(L, B), \text{box}(B, T), \\
\quad \text{pushable\_top}(B, B_1, T), \text{good\_pushlocation}(B_1), \text{actiontime}(T). \\
\text{push}(B, \text{down}, B_1, T) \vee \neg \text{push}(B, \text{down}, B_1, T) :- \text{reachable}(L, T), \text{bottom}(L, B), \text{box}(B, T), \\
\quad \text{pushable\_bottom}(B, B_1, T), \text{good\_pushlocation}(B_1), \text{actiontime}(T). \\
\text{reachable}(L, T) :- \text{pushokan}(L, T). \\
\text{reachable}(L, T) :- \text{reachable}(L_1, L), \text{adj}(L_1, L), \neg \text{box}(L, T). \\

\text{pushable\_right}(B, D, T) :- \text{box}(B, T), \text{right}(B, D), \neg \text{box}(D, T), \text{actiontime}(T). \\
\text{pushable\_right}(B, D, T) :- \text{pushable\_right}(B, D_1, T), \text{right}(D_1, D), \neg \text{box}(D, T). \\
\text{pushable\_left}(B, D, T) :- \text{box}(B, T), \text{left}(B, D), \neg \text{box}(D, T), \text{actiontime}(T). \\
\text{pushable\_left}(B, D, T) :- \text{pushable\_left}(B, D_1, T), \text{left}(D_1, D), \neg \text{box}(D, T). \\
\text{pushable\_top}(B, D, T) :- \text{box}(B, T), \text{top}(B, D), \neg \text{box}(D, T), \text{actiontime}(T). \\
\text{pushable\_top}(B, D, T) :- \text{pushable\_top}(B, D_1, T), \text{top}(D_1, D), \neg \text{box}(D, T). \\
\text{pushable\_bottom}(B, D, T) :- \text{box}(B, T), \text{bottom}(B, D), \neg \text{box}(D, T), \text{actiontime}(T). \\
\text{pushable\_bottom}(B, D, T) :- \text{pushable\_bottom}(B, D_1, T), \text{bottom}(D_1, D), \neg \text{box}(D, T). \\
\text{sokoban}(L, T_1) :- \text{push}(\_, \text{right}, B_1, T), \#\text{succ}(T, T_1), \text{right}(L, B_1). \\
\text{sokoban}(L, T_1) :- \text{push}(\_, \text{left}, B_1, T), \#\text{succ}(T, T_1), \text{left}(L, B_1). \\
\text{sokoban}(L, T_1) :- \text{push}(\_, \text{up}, B_1, T), \#\text{succ}(T, T_1), \text{top}(L, B_1). \\
\text{sokoban}(L, T_1) :- \text{push}(\_, \text{down}, B_1, T), \#\text{succ}(T, T_1), \text{bottom}(L, B_1). \\
\neg \text{sokoban}(L, T_1) :- \text{push}(\_, \_, T), \#\text{succ}(T, T_1), \neg \text{sokoban}(L, T). \\
\text{box}(B, T_1) :- \text{push}(\_, \_ B, T), \#\text{succ}(T, T_1). \\
\neg \text{box}(B, T_1) :- \text{push}(\_, \_, T), \#\text{succ}(T, T_1). \\
\text{box}(L_B, T_1) :- \text{box}(L_B, T), \#\text{succ}(T, T_1), \neg \text{box}(L_B, T_1). \\
\text{sokoban}(L_S, T) :- \text{sokoban}(L_S, T), \#\text{succ}(T, T_1), \neg \text{sokoban}(L_S, T_1). \\
\neg \text{sokoban}(L_S, T) :- \text{push}(\_, \_, T), \#\text{succ}(T, T_1), \text{B} \neq \text{B}_1. \\
\neg \text{box}(B, D, T) :- \text{push}(B, D, \_ T), \#\text{succ}(T, T_1), \text{D} \neq \text{D}_1. \\
\neg \text{push}(B, D, B_1, T), \#\text{succ}(T, T_1), \text{B}_1 \neq \text{B}_1. \\
\text{good\_pushlocation}(L) :- \text{right}(L, \_), \text{left}(L, \_). \\
\text{good\_pushlocation}(L) :- \text{top}(L, \_), \text{bottom}(L, \_). \\
\text{good\_pushlocation}(L) :- \text{solution}(L). \\
\text{notgoal} :- \text{solution}(L), \neg \text{box}(L, \#\text{maxint}). \\
\text{notgoal}?
Recap: Issues

Domain Modelling

Multiple aspects
(temporal/causal/physical/epistemic/legal/...)
Incomplete

Events

Events happened ≠ events reported (errors, incomplete, late ...)

Explanations

Multiple plausible explanations
Recap: Issues

Domain Modelling

Multiple aspects
(temporal/causal/physical/epistemic/legal/...)
Incomplete

Events

Events happened ≠ events reported (errors, incomplete, late ...)

Explanations

Multiple plausible explanations
Recap: Issues

Domain Modelling

Multiple aspects
(temporal/causal/physical/epistemic/legal/...)
Incomplete

Events

Events happened ≠ events reported (errors, incomplete, late...)

Explanations

Multiple plausible explanations

Fusemate:

Logic program
+ ontologies/event calculus

Belief revision
Recap: Issues

Domain Modelling

Multiple aspects
(temporal/causal/physical/epistemic/legal/…)
Incomplete

Events

Events happened ≠ events reported (errors, incomplete, late …)

Explanations

Multiple plausible explanations

Fusemate:

Logic program
+ ontologies/event calculus

Belief revision

Models of logic program
Events happened ≠ events reported

“Fixing the event stream”
Events happened ≠ events reported

“Fixing the event stream”

Reported

Load( 10, tomatoes, pallet)
Load( 20, pallet, container)
Load( 40, container, ship)
Unload(60, apples, pallet)
Events happened ≠ events reported

“Fixing the event stream”

Reported

Load( 10, tomatoes, pallet)
Load( 20, pallet, container)
Load( 40, container, ship)
Unload(60, apples, pallet)
Events happened ≠ events reported

“Fixing the event stream”

```
Reported
Load( 10, tomatoes, pallet)
Load( 20, pallet, container)
Load( 40, container, ship)
Unload(60, apples, pallet)
```
Events happened ≠ events reported

“Fixing the event stream”

Reported

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load( 10, tomatoes, pallet)</td>
<td></td>
</tr>
<tr>
<td>Load( 20, pallet, container)</td>
<td></td>
</tr>
<tr>
<td>Load( 40, container, ship)</td>
<td></td>
</tr>
<tr>
<td>Unload(60, apples, pallet)</td>
<td></td>
</tr>
</tbody>
</table>
Events happened ≠ events reported

“Fixing the event stream”

Reported

Load( 10, tomatoes, pallet)
Load( 20, pallet, container)
Load( 40, container, ship)
Unload(60, apples, pallet)
Events happened ≠ events reported

“Fixing the event stream”

Reported

Load( 10, tomatoes, pallet)
Load( 20, pallet, container)
Load( 40, container, ship)
Unload(60, apples, pallet)
Events happened ≠ events reported

“Fixing the event stream”

*Reported*

Load( 10, tomatoes, pallet)
Load( 20, pallet, container)
Load( 40, container, ship)
Unload(60, apples, pallet)

*Happened*

Load( 10, tomatoes, pallet)
Load( 20, pallet, container)
Load( 40, container, ship)
Unload(45, container, ship)
Unload(50, pallet, container)
Unload(60, tomatoes, pallet)
Events happened ≠ events reported

“Fixing the event stream”

Reported

Load( 10, tomatoes, pallet)
Load( 20, pallet, container)
Load( 40, container, ship)
Unload(60, apples, pallet)

Happened

Load( 10, tomatoes, pallet)
Load( 20, pallet, container)
Load( 40, container, ship)
Unload(45, container, ship)
Unload(50, pallet, container)
Unload(60, tomatoes, pallet)
Events happened ≠ events reported

“Fixing the event stream”

**Reported**

Load( 10, tomatoes, pallet)
Load( 20, pallet, container)
Load( 40, container, ship)

Unload(60, apples, pallet)

**Happened**

Load( 10, tomatoes, pallet)
Load( 20, pallet, container)
Load( 40, container, ship)
Unload(45, container, ship)
Unload(50, pallet, container)
Unload(60, apples, pallet)

Load( 10, apples, pallet)
Load( 20, pallet, container)
Load( 40, container, ship)
Unload(45, container, ship)
Unload(50, pallet, container)
Unload(60, tomatoes, pallet)
Events happened ≠ events reported

“Fixing the event stream”

**Reported**

- Load(10, tomatoes, pallet)
- Load(20, pallet, container)
- Load(40, container, ship)
- Unload(60, apples, pallet)

**Happened**

- Load(10, tomatoes, pallet)
- Load(10, apples, pallet)
- Load(20, pallet, container)
- Load(40, container, ship)
- Unload(45, container, ship)
- Unload(50, pallet, container)
- Unload(60, apples, pallet)
- Unload(60, tomatoes, pallet)
Events happened ≠ events reported

“Fixing the event stream”

Reported

Load(10, tomatoes, pallet)
Load(20, pallet, container)
Load(40, container, ship)
Unload(60, apples, pallet)

Happened

Load(10, tomatoes, pallet)
Load(20, pallet, container)
Load(40, container, ship)
Unload(45, container, ship)
Unload(50, pallet, container)
Unload(60, apples, pallet)

Next:
logic program expressing this

Happened

Load(10, apples, pallet)
Load(20, pallet, container)
Load(40, container, ship)
Unload(45, container, ship)
Unload(50, pallet, container)
Unload(60, apples, pallet)
## Logic Program for the Supply Chain Example

| Derived “In” relation | Integrity constraints and revision |
Logic Program for the Supply Chain Example

Derived “In” relation

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

Integrity constraints and revision
Logic Program for the Supply Chain Example

Derived “In” relation

\[
\text{In}(\text{time}, \text{obj}, \text{cont}) : - \\
\quad \text{Load}(\text{time}, \text{obj}, \text{cont})
\]

// In transitivity
\[
\text{In}(\text{time}, \text{obj}, \text{cont}) : - \\
\quad \text{In}(\text{time}, \text{obj}, \text{c}), \\
\quad \text{In}(\text{time}, \text{c}, \text{cont})
\]

Integrity constraints and revision
Logic Program for the Supply Chain Example

Derived “In” relation

\[
\text{In}(\text{time}, \text{obj}, \text{cont}) :- \\
\text{Load}(\text{time}, \text{obj}, \text{cont})
\]

// In transitivity
\[
\text{In}(\text{time}, \text{obj}, \text{cont}) :- \\
\text{In}(\text{time}, \text{obj}, \text{c}), \\
\text{In}(\text{time}, \text{c}, \text{cont})
\]

// Frame axiom for In
\[
\text{In}(\text{time}, \text{obj}, \text{cont}) :- \\
\text{In}(\text{prev}, \text{obj}, \text{cont}), \\
\text{Step}(\text{time}, \text{prev}), \\
\textbf{not} \ \text{Unload}(\text{time}, \text{obj}, \text{cont}), \\
\textbf{not} \ (\text{In}(\text{prev}, \text{obj}, \text{c}), \\
\quad \text{Unload}(\text{time}, \text{c}, \text{cont}))
\]

Integrity constraints and revision
Logic Program for the Supply Chain Example

**Derived “In” relation**

\[
\text{In}(\text{time}, \text{obj}, \text{cont}) :\text{-} \\
\quad \text{Load}(\text{time}, \text{obj}, \text{cont})
\]

\[
// \text{In transitivity} \\
\text{In}(\text{time}, \text{obj}, \text{cont}) :\text{-} \\
\quad \text{In}(\text{time}, \text{obj}, \text{c}), \\
\quad \text{In}(\text{time}, \text{c}, \text{cont})
\]

\[
// \text{Frame axiom for In} \\
\text{In}(\text{time}, \text{obj}, \text{cont}) :\text{-} \\
\quad \text{In}(\text{prev}, \text{obj}, \text{cont}), \\
\quad \text{Step}(\text{time}, \text{prev}), \\
\quad \textbf{not} \ \text{Unload}(\text{time}, \text{obj}, \text{cont}), \\
\quad \textbf{not} \ (\text{In}(\text{prev}, \text{obj}, \text{c}), \\
\quad \quad \text{Unload}(\text{time}, \text{c}, \text{cont}))
\]

**Integrity constraints and revision**

Default negation
Logic Program for the Supply Chain Example

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(time, 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)
Logic Program for the Supply Chain Example

Derived “In” relation

\[\text{In}(\text{time}, \text{obj}, \text{cont}) :- \]
\[\text{Load}(\text{time}, \text{obj}, \text{cont})\]

// In transitivity
\[\text{In}(\text{time}, \text{obj}, \text{cont}) :- \]
\[\text{In}(\text{time}, \text{obj}, \text{c}), \]
\[\text{In}(\text{time}, \text{c}, \text{cont})\]

// Frame axiom for In
\[\text{In}(\text{time}, \text{obj}, \text{cont}) :- \]
\[\text{In}(\text{prev}, \text{obj}, \text{cont}), \]
\[\text{Step}(\text{time}, \text{prev}), \]
\[\neg \text{Unload}(\text{time}, \text{obj}, \text{cont}), \]
\[\neg (\text{In}(\text{prev}, \text{obj}, \text{c}), \]
\[\text{Unload}(\text{time}, \text{c}, \text{cont}))\]

Default negation

Integrity constraints and revision

// No Unload without earlier Load
\[\text{fail} :- \]
\[\text{Unload}(\text{time}, \text{obj}, \text{cont}), \]
\[\neg (\text{Load}(\text{t}, \text{obj}, \text{cont}), \]
\[\text{t} < \text{time})\]

// Unload a different object
\[\text{fail} :- \]
\[- \text{Unload}(\text{time}, \text{obj}, \text{cont}), \]
\[+ \text{Unload}(\text{time}, \text{o}, \text{cont}) :- \]
\[\text{Unload}(\text{time}, \text{obj}, \text{cont}), \]
\[\neg (\text{Load}(\text{t}, \text{obj}, \text{cont}), \text{t} < \text{time}), \]
\[\text{Load}(\text{t}, \text{o}, \text{cont}), \]
\[\text{t} < \text{time}, \]
\[\text{SameBatch}(\text{t}, \text{b}), \]
\[(\text{b contains obj}) \&\& (\text{b contains o})\]
Logic Program for the Supply Chain Example

Derived “In” relation

\[
\text{In}(\text{time}, \text{obj}, \text{cont}) :\neg \\
\quad \text{Load}(\text{time}, \text{obj}, \text{cont})
\]

// In transitivity

\[
\text{In}(\text{time}, \text{obj}, \text{cont}) :\neg \\
\quad \text{In}(\text{time}, \text{obj}, \text{c}), \\
\quad \text{In}(\text{time}, \text{c}, \text{cont})
\]

// Frame axiom for In

\[
\text{In}(\text{time}, \text{obj}, \text{cont}) :\neg \\
\quad \text{In}(\text{prev}, \text{obj}, \text{cont}), \\
\quad \text{Step}(\text{time}, \text{prev}), \\
\quad \text{not } \text{Unload}(\text{time}, \text{obj}, \text{cont}), \\
\quad \text{not } (\text{In}(\text{prev}, \text{obj}, \text{c}), \\
\quad \quad \text{Unload}(\text{time}, \text{c}, \text{cont}))
\]

Integrity constraints and revision

// No Unload without earlier Load

\[
\text{fail} :\neg \\
\quad \text{Unload}(\text{time}, \text{obj}, \text{cont}), \\
\quad \text{not } (\text{Load}(\text{t}, \text{obj}, \text{cont}), \\
\quad \quad \text{t} < \text{time})
\]

// Unload a different object

\[
\text{fail}(- \text{Unload}(\text{time}, \text{obj}, \text{cont}), \\
\quad + \text{Unload}(\text{time}, \text{o}, \text{cont})) :\neg \\
\quad \text{Unload}(\text{time}, \text{obj}, \text{cont}), \\
\quad \text{not } (\text{Load}(\text{t}, \text{obj}, \text{cont}), \text{t} < \text{time}), \\
\quad \text{Load}(\text{t}, \text{o}, \text{cont}), \\
\quad \text{t} < \text{time}, \\
\quad \text{SameBatch}(\text{t}, \text{b}), \\
\quad ((\text{b} \text{ contains obj}) \&\& (\text{b} \text{ contains o}))
\]

Default negation
Logic Program for the Supply Chain Example

 Derived “In” relation

\[
\text{In}(\text{time}, \text{obj}, \text{cont}) :- \\
\quad \text{Load}(\text{time}, \text{obj}, \text{cont})
\]

// In transitivity
\[
\text{In}(\text{time}, \text{obj}, \text{cont}) :- \\
\quad \text{In}(\text{time}, \text{obj}, \text{c}), \\
\quad \text{In}(\text{time}, \text{c}, \text{cont})
\]

// Frame axiom for In
\[
\text{In}(\text{time}, \text{obj}, \text{cont}) :- \\
\quad \text{In}(\text{prev}, \text{obj}, \text{cont}), \\
\quad \text{Step}(\text{time}, \text{prev}), \\
\quad \text{not} \ \text{Unload}(\text{time}, \text{obj}, \text{cont}), \\
\quad \text{not} \ (\text{In}(\text{prev}, \text{obj}, \text{c}), \\
\quad \quad \text{Unload}(\text{time}, \text{c}, \text{cont}))
\]

Default negation

 Integrity constraints and revision

// No Unload without earlier Load
\[
\text{fail} :- \\
\quad \text{Unload}(\text{time}, \text{obj}, \text{cont}), \\
\quad \text{not} \ (\text{Load}(\text{t}, \text{obj}, \text{cont}), \\
\quad \quad \text{t < time})
\]

// Unload a different object
\[
\text{fail}(- \text{Unload}(\text{time}, \text{obj}, \text{cont}), \\
\quad + \text{Unload}(\text{time}, \text{o}, \text{cont})) :- \\
\quad \text{Unload}(\text{time}, \text{obj}, \text{cont}), \\
\quad \text{not} \ (\text{Load}(\text{t}, \text{obj}, \text{cont}), \text{t < time}), \\
\quad \text{Load}(\text{t}, \text{o}, \text{cont}), \\
\quad \text{t < time}, \\
\quad \text{SameBatch}(\text{t}, \text{b}), \\
\quad ((\text{b contains obj}) \land (\text{b contains o}))
\]

+ 4 more rules
Logic Program for the Supply Chain Example

Derived “In” relation

\[ \text{In}(\text{time}, \text{obj}, \text{cont}) :- \]
\[ \text{Load}(\text{time}, \text{obj}, \text{cont}) \]

// In transitivity

\[ \text{In}(\text{time}, \text{obj}, \text{cont}) :- \]
\[ \text{In}(\text{time}, \text{obj}, \text{c}), \]
\[ \text{In}(\text{time}, \text{c}, \text{cont}) \]

// Frame axiom for In

\[ \text{In}(\text{time}, \text{obj}, \text{cont}) :- \]
\[ \text{In}(\text{prev}, \text{obj}, \text{cont}), \]
\[ \text{Step}(\text{time}, \text{prev}), \]
\[ \text{not} \ \text{Unload}(\text{time}, \text{obj}, \text{cont}), \]
\[ \text{not} \ (\text{In}(\text{prev}, \text{obj}, \text{c}), \]
\[ \text{Unload}(\text{time}, \text{c}, \text{cont})) \]

Default negation

(Frame axioms now via Event Calculus)

Integrity constraints and revision

// No Unload without earlier Load

\[ \text{fail} :- \]
\[ \text{Unload}(\text{time}, \text{obj}, \text{cont}), \]
\[ \text{not} \ (\text{Load}(\text{t}, \text{obj}, \text{cont}), \]
\[ \text{t} < \text{time}) \]

// Unload a different object

\[ \text{fail}(- \text{Unload}(\text{time}, \text{obj}, \text{cont}), \]
\[ + \text{Unload}(\text{time}, \text{o}, \text{cont})): - \]
\[ \text{Unload}(\text{time}, \text{obj}, \text{cont}), \]
\[ \text{not} \ (\text{Load}(\text{t}, \text{obj}, \text{cont}), \text{t} < \text{time}), \]
\[ \text{Load}(\text{t}, \text{o}, \text{cont}), \]
\[ \text{t} < \text{time}, \]
\[ \text{SameBatch}(\text{t}, \text{b}), \]
\[ ((\text{b} \text{ contains obj}) \&\& (\text{b} \text{ contains o})) \]

+ 4 more rules
Situational Awareness = Stratified Model Computation

“Situational awareness” task is naturally stratified
Situational Awareness = Stratified Model Computation

“Situational awareness” task is naturally stratified

- Comprehend evolving situation from “past” and “now”, not “future”
  → Stratification by time 0, 1, 2, ..., now
Situational Awareness = Stratified Model Computation

“Situational awareness” task is naturally stratified

- Comprehend evolving situation from “past” and “now”, not “future”
  → Stratification by time 0, 1, 2, ..., now
Situational Awareness = Stratified Model Computation

“Situational awareness” task is naturally stratified

- Comprehend evolving situation from “past” and “now”, not “future”
  → Stratification by time 0,1,2,...,now

- Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB  (extensional database / intensional database)

“Not known now” -> “never known”
Makes default negation possible
Situational Awareness = Stratified Model Computation

“Situational awareness” task is naturally stratified

• Comprehend evolving situation from “past” and “now”, not “future”
  → Stratification by time 0,1,2,...,now

• Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

  Revising events is simply addition/removal

“Not known now” \(\rightarrow\) “never known”
Makes default negation possible
Situational Awareness = Stratified Model Computation

“Situational awareness” task is naturally stratified

• Comprehend evolving situation from “past” and “now”, not “future”
  → Stratification by time 0,1,2,...,now

• Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Stratified model computation (ignoring revision)

“Not known now” → “never known”
Makes default negation possible
Situational Awareness = Stratified Model Computation

“Situational awareness” task is naturally stratified

• Comprehend evolving situation from “past” and “now”, not “future”
  → Stratification by time 0,1,2,…,now

• Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Stratified model computation (ignoring revision)

Not known now” → “never known”
Makes default negation possible

Revising events is simply addition/removal
Situational Awareness = Stratified Model Computation

“Situational awareness” task is naturally stratified

- Comprehend evolving situation from “past” and “now”, not “future”
  → Stratification by time $0, 1, 2, \ldots, \text{now}$

- Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Stratified model computation (ignoring revision)

- EDBs $E_{0, 1, 2, \ldots}$
- IDBs $I_{0, 1, 2, \ldots}$
Situational Awareness = Stratified Model Computation

“Situational awareness” task is naturally stratified

• Comprehend evolving situation from “past” and “now”, not “future”
  → Stratification by time 0, 1, 2, ..., now

• Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Stratified model computation (ignoring revision)

EDBs $E_{0,1,2,...}$

IDBs $I_{0,1,2,...}$

Bottom-up application of logic program rules until fixpoint

“Not known now” → “never known”
Makes default negation possible

Revising events is simply addition/removal
Situational Awareness = Stratified Model Computation

“Situational awareness” task is naturally stratified

- Comprehend evolving situation from “past” and “now”, not “future”
  → Stratification by time 0,1,2,...,now

- Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Stratified model computation (ignoring revision)

- Revising events is simply addition/removal

“Not known now” -> “never known”
Makes default negation possible

Time 0,1,2 ……

Bottom-up application of logic program rules until fixpoint

EDBs $E_{0,1,2,...}$

IDBs $I_{0,1,2,...}$
Situational Awareness = Stratified Model Computation

“Situational awareness” task is naturally stratified

- Comprehend evolving situation from “past” and “now”, not “future”
  → Stratification by time 0,1,2,…,now

- Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Stratified model computation (ignoring revision)

- EDBs $E_{0,1,2,…}$
- IDBs $I_{0,1,2,…}$

Bottom-up application of logic program rules until fixpoint

“Not known now” -> “never known”
Makes default negation possible

Revising events is simply addition/removal

Time 0,1,2 ……►
Situational Awareness = Stratified Model Computation

“Situational awareness” task is naturally stratified

• Comprehend evolving situation from “past” and “now”, not “future”
  → Stratification by time 0,1,2,...,now

• Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Stratified model computation (ignoring revision)

“Not known now” -> “never known”
Makes default negation possible

Revising events is simply addition/removal

Time 0,1,2 .....→
Situational Awareness = Stratified Model Computation

“Situational awareness” task is naturally stratified

- Comprehend evolving situation from “past” and “now”, not “future”
  → Stratification by time 0,1,2,…,now

- Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Stratified model computation (ignoring revision)

EDBs $E_{0,1,2,\ldots}$

IDBs $I_{0,1,2,\ldots}$

Bottom-up application of logic program rules until fixpoint

“Not known now” $\rightarrow$ “never known”
Makes default negation possible

Revising events is simply addition/removal

Time 0,1,2 …→
Situational Awareness = Stratified Model Computation

“Situational awareness” task is naturally stratified

- Comprehend evolving situation from “past” and “now”, not “future”
  → Stratification by time $0, 1, 2, ..., \text{now}$

- Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Stratified model computation (ignoring revision)

“Not known now” → “never known”
Makes default negation possible
Situational Awareness = Stratified Model Computation

“Situational awareness” task is naturally stratified

- Comprehend evolving situation from “past” and “now”, not “future”
  → Stratification by time 0,1,2,...,now

- Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Stratified model computation (ignoring revision)

- EDBs $E_{0,1,2,...}$
- IDBs $I_{0,1,2,...}$

Bottom-up application of logic program rules until fixpoint

Revising events is simply addition/removal

“Not known now” → “never known”
Makes default negation possible

Time 0,1,2 ......
Situational Awareness = Stratified Model Computation

“Situational awareness” task is naturally stratified

- Comprehend evolving situation from “past” and “now”, not “future” (*)
  → Stratification by time 0,1,2,…,now

- Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Stratified model computation (ignoring revision)

- Bottom-up application of logic program rules until fixpoint

EDBs $E_{0,1,2,…}$

IDBs $I_{0,1,2,…}$

“Not known now” → “never known”
Makes default negation possible

Revising events is simply addition/removal

Time 0,1,2 ……→

(*) Cannot change past state

“Not known now” → “never known”
Makes default negation possible
Situational Awareness = Stratified Model Computation

“Situational awareness” task is naturally stratified

- Comprehend evolving situation from “past” and “now”, not “future” (*)
  → Stratification by time 0,1,2,...,now

- Distinguish between events and states induced from these events
  → Stratification by sets of literals EDB / IDB (extensional database / intensional database)

Stratified model computation (ignoring revision)

EDBs $E_{0,1,2,...}$

IDBs $I_{0,1,2,...}$

Bottom-up application of logic program rules until fixpoint

(*): Cannot change past state

Not known now” -> “never known”
Makes default negation possible

Next: Stratified logic programs for computing models $(E \cup I)_0$, $(E \cup I)_1$, $(E \cup I)_2$, ...
Stratified Logic Programs

Consists of rules over literals

\[ head \ :- \ body, \ \ldots, \ \text{not} \ body, \ \ldots \]
Stratified Logic Programs

Consists of rules over literals

\[ \text{head} : \neg \text{body}, \ldots, \neg \text{body}, \ldots \]

s. th. (1) \( \text{var(head)} \subseteq \text{fvar(body}, \ldots, \neg \text{body}, \ldots) \)

(2) head has a time variable ("now")

(3) one body lit has same time variable

(4) other body lits have time \( \leq \) time

(5) EDB lits in not body have time \( \leq \) time

(6) IDB lits in not body have time < time
Stratified Logic Programs

Consists of rules over literals

\[ head :- body, \ldots, \textbf{not} body, \ldots \]

s. th. (1) \( \text{var}(head) \subseteq \text{fvar}(body, \ldots, \textbf{not} body, \ldots) \)
(2) \( head \) has a \textit{time} variable (“\textit{now}”)
(3) one \textit{body} lit has same \textit{time} variable
(4) other \textit{body} lits have time \( \leq \text{time} \)
(5) EDB lits in \textbf{not} \textit{body} have time \( \leq \text{time} \)
(6) IDB lits in \textbf{not} \textit{body} have time \( < \text{time} \)

Range restriction
\[ \rightsquigarrow \text{Simple model computation} \]
Stratified Logic Programs
Consists of rules over literals

\[ \text{head} :~ \text{body}, ..., \text{not} \text{ body}, ... \]

s. th.  
1. \( \text{var(head)} \subseteq \text{fvar(body, ...}, \text{not body, ...}) \)
2. head has a time variable ("now")
3. one body lit has same time variable
4. other body lits have time \( \leq \text{time} \)
5. EDB lits in not body have time \( \leq \text{time} \)
6. IDB lits in not body have time < \text{time} 

Range restriction
\(~\) Simple model computation

Stratification by time
\(~\) Effective model computation
Stratified Logic Programs
Consists of rules over literals

\[
\text{head} :\neg \text{body}, \ldots, \textbf{not} \text{body}, \ldots
\]

s. th. (1) \(\text{var(}\text{head}\text{)} \subseteq \text{fvar(}\text{body}, \ldots, \textbf{not} \text{body}, \ldots\text{)}\)
(2) \text{head} has a time variable ("now")
(3) one \text{body} lit has same time variable
(4) other \text{body} lits have time \(\leq\) time
(5) EDB lits in \textbf{not} \text{body} have time \(\leq\) time
(6) IDB lits in \textbf{not} \text{body} have time < time

Range restriction
\(\leadsto\) Simple model computation
Stratification by time
\(\leadsto\) Effective model computation
Avoids guessing whether head is true or false in final model
\(\leadsto\) Efficient model computation
Stratified Logic Programs

Consists of rules over literals

\[
\text{head} \; :\!-\!\; \text{body}, \; \ldots, \; \text{not body}, \; \ldots
\]

s. th. (1) \(\text{var(head)} \subseteq \text{fvar(body}, \; \ldots, \; \text{not body}, \; \ldots)\)
(2) \(\text{head}\) has a time variable ("now")
(3) one \(\text{body}\) lit has same time variable
(4) other \(\text{body}\) lits have time \(\leq\) time
(5) EDB lits in \(\text{not body}\) have time \(\leq\) time
(6) IDB lits in \(\text{not body}\) have time \(<\) time

Examples
Stratified Logic Programs

Consists of rules over literals

\[ \text{head} :\text{-} \text{body}, \ldots, \text{not body}, \ldots \]

s. th. (1) \( \text{var(head) } \subseteq \text{fvar(body, } \ldots, \text{not body, } \ldots) \)
(2) \( \text{head} \) has a time variable (“now”)
(3) one body lit has same time variable
(4) other body lits have time \( \leq \) time
(5) EDB lits in not body have time \( \leq \) time
(6) IDB lits in not body have time < time

Examples

\[ I(\text{time}, x) :\text{-} J(\text{time}, x, y), I(\text{time}, y) \]
Stratified Logic Programs

Consists of rules over literals

\[ \textit{head} : - \textit{body}, \ldots, \textit{not body}, \ldots \]

s. th. (1) \( \text{var}(\textit{head}) \subseteq \text{fvar}(\textit{body}, \ldots, \textit{not body}, \ldots) \)
(2) \( \textit{head} \) has a \textit{time} variable ("now")
(3) one \textit{body} lit has same \textit{time} variable
(4) other \textit{body} lits have time \( \leq \text{time} \)
(5) EDB lits in \textit{not body} have time \( \leq \text{time} \)
(6) IDB lits in \textit{not body} have time \( < \text{time} \)

Examples

\[ I(\text{time}, x) : - J(\text{time}, x, y), \ I(\text{time}, y) \]
\[ I(\text{time}, x) : - J(\text{time}, x, y), \ I(t, y), \ t \leq \text{time} \]
Stratified Logic Programs

Consists of rules over literals

\[ \text{head} :\text{-} \text{body, \ldots, not body, \ldots} \]

s. th. (1) \( \text{var(head)} \subseteq \text{fvar(body, \ldots, not body, \ldots)} \)
(2) \( \text{head} \) has a time variable ("now")
(3) one \( \text{body} \) lit has same time variable
(4) other \( \text{body} \) lits have time \( \leq \) time
(5) EDB lits in \( \text{not} \ \text{body} \) have time \( \leq \) time
(6) IDB lits in \( \text{not} \ \text{body} \) have time \( < \) time

Examples

I(time, x) :- J(time, x, y), I(time, y)
I(time, x) :- J(time, x, y), I(t, y), t \leq \text{time}
I(time, x) :- J(time, x, y), not (I(t, y), t < \text{time})
Stratified Logic Programs

Consists of rules over literals

head :- body, …, not body, …

s. th. (1) var(head) ⊆ fvar(body, …, not body, …)
(2) head has a time variable ("now")
(3) one body lit has same time variable
(4) other body lits have time ≤ time
(5) EDB lits in not body have time ≤ time
(6) IDB lits in not body have time < time

Examples

I(time, x) :- J(time, x, y), I(time, y)
I(time, x) :- J(time, x, y), I(t, y), t ≤ time
I(time, x) :- J(time, x, y), not (I(t, y), t < time)

Range restriction

Range restriction

Simple model computation

Stratification by time

Effective model computation

Avoids guessing whether head is true or false in final model

Efficient model computation

Closed world assumption

E ∪ I ⊨ not body[x] iff not exists a s.th. body[a] ⊆ E ∪ I

I, J: IDB
E: EDB
Stratified Logic Programs

Consists of rules over literals

\[ head :- body, \ldots, not \ body, \ldots \]

s. th. (1) \( \text{var}(head) \subseteq \text{fvar}(body, \ldots, not \ body, \ldots) \)
(2) \( head \) has a time variable ("now")
(3) one \( body \) lit has same time variable
(4) other \( body \) lits have time \( \leq \) time
(5) EDB lits in \( not \ body \) have time \( \leq \) time
(6) IDB lits in \( not \ body \) have time \( < \) time

Examples

\begin{align*}
I(time, x) & :- J(time, x, y), I(time, y) \\
I(time, x) & :- J(time, x, y), I(t, y), t \leq time \\
I(time, x) & :- J(time, x, y), not (I(t, y), t < time) \\
I(time, x) & :- J(time, x, y), not (I(t, y), t \leq time) \quad \text{No!}
\end{align*}

Range restriction
\( \sim \) Simple model computation

Stratification by time
\( \sim \) Effective model computation

Avoids guessing whether head is true or false in final model
\( \sim \) Efficient model computation

Closed world assumption
\( E \cup I \models \text{not} \ body[x] \) iff
not exists a s.th. \( body[a] \subseteq E \cup I \)

I, J: IDB
E: EDB
Stratified Logic Programs

Consists of rules over literals

\[
\text{head} :\text{-} \text{body, ..., not body, ...}
\]

s.th. (1) \(\text{var(head)} \subseteq \text{fvar(body, ..., not body, ...)}\)
(2) \(\text{head}\) has a time variable ("now")
(3) one \(\text{body}\) lit has same time variable
(4) other \(\text{body}\) lits have time \(\leq\) time
(5) EDB lits in \(\text{not body}\) have time \(\leq\) time
(6) IDB lits in \(\text{not body}\) have time \(<\) time

Examples

\[
\begin{align*}
\text{I(time, x)} :&= \text{J(time, x, y), I(time, y)} \\
\text{I(time, x)} :&= \text{J(time, x, y), I(t, y), t} \leq \text{time} \\
\text{I(time, x)} :&= \text{J(time, x, y), not (I(t, y), t < \text{time})} \\
\text{I(time, x)} :&= \text{J(time, x, y), not (I(t, y), t} \leq \text{time)} \\
\text{I(time, x)} :&= \text{J(time, x, y), not (E(t, y), t} \leq \text{time)}
\end{align*}
\]

Range restriction
\(\sim\) Simple model computation

Stratification by time
\(\sim\) Effective model computation

Avoids guessing whether \text{head} is true or false in final model
\(\sim\) Efficient model computation

Closed world assumption
\(EUI \models \text{not body}[x]\) iff not exists a s.th. \(\text{body}[a] \subseteq EUI\)
Integrity Constraints and Belief Revision

Usual integrity constraints

\[ \text{fail} : - \text{body}, \ldots, \text{not body}, \ldots \]

Generalized for revision of EDB literals

\[ \text{fail}(\neg e, \ldots, +f, \ldots) : - \text{body}, \ldots, \text{not body}, \ldots \]

s. th.  
- “conditions for body as for ordinary rules”
- EDB lits e and f have time \( \leq \text{time} \)
Integrity Constraints and Belief Revision

Usual integrity constraints

\[ \text{fail} :- \ body, \ldots, \not body, \ldots \]

Generalized for revision of EDB literals

\[ \text{fail}(-e, \ldots, +f, \ldots) :- \ body, \ldots, \not body, \ldots \]

s. th. • “conditions for body as for ordinary rules”
  • EDB lits \( e \) and \( f \) have time \( \leq \) \text{time}

Example

// Unload a different object
\[ \text{fail}(- \text{Unload}(\text{time}, \text{obj}, \text{cont}), \]
\[ \quad + \text{Unload}(\text{time}, o, \text{cont})) :- \]
\[ \text{Unload}(\text{time}, \text{obj}, \text{cont}), \]
\[ \not (\text{Load}(t, \text{obj}, \text{cont}), t < \text{time}), \]
\[ \text{Load}(t, o, \text{cont}), t < \text{time}, \]
\[ \ldots \]
Integrity Constraints and Belief Revision

Usual integrity constraints

\[
\text{fail} : - \text{body}, \ldots, \text{not} \text{body}, \ldots
\]

Generalized for revision of EDB literals

\[
\text{fail}(-e, \ldots, +f, \ldots) : - \text{body}, \ldots, \text{not} \text{body}, \ldots
\]

s. th.  
• “conditions for body as for ordinary rules”
• EDB lits \(e\) and \(f\) have time \(\leq\) time

Example

// Unload a different object
false(- 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,
...

- Unload(60, apples, pallet)

+ Unload(60, tomatoes, pallet)
Integrity Constraints and Belief Revision

Usual integrity constraints

\[
\text{fail} :- \ body, \ldots, \not \ body, \ldots
\]

Generalized for revision of EDB literals

\[
\text{fail}(\neg e, \ldots, +f, \ldots) :- \ body, \ldots, \not \ body, \ldots
\]

s. th.

• “conditions for body as for ordinary rules”
• EDB lits \( e \) and \( f \) have time \( \leq \)\( time \)

Example

// Unload a different object
\[
\text{fail}(- \text{Unload}(\text{time}, \text{obj}, \text{cont}),
+ \text{Unload}(\text{time}, o, \text{cont})) :-
\text{Unload}(\text{time}, \text{obj}, \text{cont}),
\not (\text{Load}(t, \text{obj}, \text{cont}), t < \text{time}),
\text{Load}(t, o, \text{cont}), t < \text{time},
\]

Semantics

\[
\begin{align*}
E \cup I & \quad \text{if } E \cup I \models (\text{body}, \ldots, \\
& \not \text{body}, \ldots)\sigma \\
(E \setminus e\sigma) \cup f\sigma
\end{align*}
\]

Example

\[
\begin{align*}
\ldots & \quad - \text{Unload}(60, \text{apples, pallet}) \\
\ldots & \quad + \text{Unload}(60, \text{tomatoes, pallet})
\end{align*}
\]
Semantics of Programs With Fail Rules

Operational

for a given EDB E 
for time $t = 0,1,2, ...$, now 
compute $\{ I^0, I^1, ... \}$ all IDBs for time $t \leq t$ 
for $I = I^0, I^1, ...$
let $F = \{ \text{fail(...)} \}$ heads derivable from $E \cup I$
if $F$ is non-empty then 
obtain new EDBs $E_1, E_2, ...$ as per $F$ and 
abandon model candidate $I$

Principles
- Fail as early as possibly
- Collect all possible fails
Semantics of Programs With Fail Rules

for a given EDB $E$
for time $t = 0, 1, 2, \ldots$, now

compute $\{ I^0, I^1, \ldots \}$ all IDBs for time $\leq t$

for $I = I^0, I^1, \ldots$

let $F = \{ \text{fail}(\ldots) \}$ heads derivable from $E \cup I$

if $F$ is non-empty then

obtain new EDBs $E_1, E_2, \ldots$ as per $F$ and
abandon model candidate $I$
Semantics of Programs With Fail Rules

Operational

for a given EDB $E$

for time $t = 0, 1, 2, \ldots$, now

compute $\{ I^0, I^1, \ldots \}$ all IDBs for time $\leq t$

for $I = I^0, I^1, \ldots$

let $F = \{ \text{fail}(\ldots) \}$ heads derivable from $E \cup I$

if $F$ is non-empty then

obtain new EDBs $E_1, E_2, \ldots$ as per $F$ and

abandon model candidate $I$

Principles

- Fail as early as possibly
- Collect all possible fails

Declarative semantics: see paper
Description Logics

- A (usually) decidable fragment of first-order logic
- Semantic web ontologies ("is-a" and "has-a" relations)
- Reasoning on concepts and concept instances

**Concepts**

**TBox**

- Box \( \sqsubseteq \forall \text{temp}. \text{TempClass} \)
- FruitBox \( \sqsubseteq \exists \text{temp}. \text{TempClass} \)
- ToyBox \( \sqsubseteq \neg \exists \text{temp}. \text{TempClass} \)
- FruitBox \( \sqsubseteq \text{Box} \)
- ToyBox \( \sqsubseteq \text{Box} \)

  - temp is a functional role

**Instances**

**ABox**

- Box
- FruitBox
- ToyBox

```
Time QP RP SP TP UP
Action load Box P load Box Q load Box R load Box S load Box T
unload
Sensor Box P Z QP ° Box R Z QP °
```

The diagnosis is "the cooling is broken." The rest of the paper explains the underlying modelling.

I use the term "action" where usual event calculus terminology is "event." Here the term "event" is wider and means anything happening. Also, temperature sensor readings.

A (usually) decidable fragment of first-order logic

Semantic web ontologies ("is-a" and "has-a" relations)

Reasoning on concepts and concept instances

**Concepts**

**TBox**

- Box \( \sqsubseteq \forall \text{temp}. \text{TempClass} \)
- FruitBox \( \sqsubseteq \exists \text{temp}. \text{TempClass} \)
- ToyBox \( \sqsubseteq \neg \exists \text{temp}. \text{TempClass} \)
- FruitBox \( \sqsubseteq \text{Box} \)
- ToyBox \( \sqsubseteq \text{Box} \)

  - temp is a functional role

**Instances**

**ABox**

```
Box [0..1] temp TempClass
ToyBox FruitBox temp [1] temp
```

The diagnosis is "the cooling is broken." The rest of the paper explains the underlying modelling.

I use the term "action" where usual event calculus terminology is "event." Here the term "event" is wider and means anything happening. Also, temperature sensor readings.
Description Logics

- A (usually) decidable fragment of first-order logic
- Semantic web ontologies ("is-a" and "has-a" relations)
- Reasoning on concepts and concept instances

Concepts

"TBox"

- Box ⊆ ∀ temp.TempClass
- FruitBox ⊆ ∃ temp.TempClass
- ToyBox ⊆ ¬∃ temp.TempClass
- FruitBox ⊆ Box
- ToyBox ⊆ Box

  temp is a functional role

Instances

"ABox"

- Low : TempClass
- High : TempClass

![Diagram of concepts and instances]
Description Logics

- A (usually) decidable fragment of first-order logic
- Semantic web ontologies ("is-a" and "has-a" relations)
- Reasoning on concepts and concept instances

**Concepts**

**"TBox"**

- Box $\sqsubseteq \forall \text{temp. TempClass}$
- FruitBox $\sqsubseteq \exists \text{temp. TempClass}$
- ToyBox $\sqsubseteq \neg \exists \text{temp. TempClass}$
- FruitBox $\sqsubseteq \text{Box}$
- ToyBox $\sqsubseteq \text{Box}$

  `temp` is a functional role

**Instances**

**"ABox"**

- Box$_0$: FruitBox
- Box$_1$: FruitBox
- Box$_2$: Box
- Box$_3$: ToyBox
- Box$_4$: Box $\sqcap \forall \text{temp.} \neg \text{TempClass}$
- Box$_5$: Box $\sqcap \exists \text{temp. TempClass}$

**Diagram**

- Box
  - [0..1] `temp`
  - TempClass
  - ToyBox
  - FruitBox
  - [0] `temp`
  - [1] `temp`
  - Low
  - High
Description Logics

- A (usually) decidable fragment of first-order logic
- Semantic web ontologies ("is-a" and "has-a" relations)
- Reasoning on concepts and concept instances

### Concepts

**"TBox"**

- Box ⊑ ∀ temp.TempClass
- FruitBox ⊑ ∃ temp.TempClass
- ToyBox ⊑ ¬∃ temp.TempClass
- FruitBox ⊑ Box
- ToyBox ⊑ Box
- temp is a functional role

**Instances**

**"ABox"**

- Low : TempClass
- High : TempClass
- Box0 : FruitBox
- Box1 : FruitBox
- Box2 : Box
- Box3 : ToyBox
- Box4 : Box ⊍ ∀ temp.¬TempClass
- Box5 : Box ⊍ ∃ temp.TempClass

### Reasoning

- Is Box4 a FruitBox?
- Is Box5 a FruitBox?
- Are FruitBox and ToyBox disjoint?
Description Logics

- A (usually) decidable fragment of first-order logic
- Semantic web ontologies ("is-a" and "has-a" relations)
- Reasoning on concepts and concept instances

**Concepts**

**"TBox"**

- \( \text{Box} \sqsubseteq \forall \text{temp}. \text{TempClass} \)
- \( \text{FruitBox} \sqsubseteq \exists \text{temp}. \text{TempClass} \)
- \( \text{ToyBox} \sqsubseteq \neg \exists \text{temp}. \text{TempClass} \)
- \( \text{FruitBox} \sqsubseteq \text{Box} \)
- \( \text{ToyBox} \sqsubseteq \text{Box} \)

\( \text{temp} \) is a functional role

**Instances**

**"ABox"**

- \( \text{Box}_0 : \text{FruitBox} \)
- \( \text{Box}_1 : \text{FruitBox} \)
- \( \text{Box}_2 : \text{Box} \)
- \( \text{Box}_3 : \text{ToyBox} \)
- \( \text{Box}_4 : \text{Box} \cap \forall \text{temp}. \neg \text{TempClass} \)
- \( \text{Box}_5 : \text{Box} \cap \exists \text{temp}. \text{TempClass} \)

**Reasoning**

- Is Box\(_4\) a FruitBox?
- Is Box\(_5\) a FruitBox?
- Are FruitBox and ToyBox disjoint?

**[CADE-2021]:** map to Fusemate disjunctive logic program + loop check
Description Logics, Event Calculus and Rules

→ Description logics and logic programming are “very different”
  Open world vs closed world, Entailment vs Models, Infinite models vs finite models
→ Attractive to integrate for modelling complementary aspects

Box₀ : FruitBox
Box₁ : FruitBox
Box₂ : Box
Box₃ : ToyBox
Box₄ : Box ⊓ ∀ temp.¬TempClass
Box₅ : Box ⊓ ∃ temp.TempClass

The domain model also includes actions and their effects, event calculus style. The actions are “loading,” “unloading,” and the only effect is “on” as an event calculus. Effects are fluents that are “initiated” by an action and will hold until “terminated” by a later action. The actions are:
Q: A box loading action initiates the box to be on the truck
R: An unloading action terminates every box to be on the truck
S: The diagnosis rules for explanations can be stated informally as follows. Suppose a given subset of the boxes \( \{ Box\ P, \ldots, Box\ U \} \) have been unloaded at the destination:
Q: If there is no unloaded box with known high temperature then the status is ok
R: If some unloaded box has a known high temperature then this box has been tampered with or the truck cooling is broken
S: If some unloaded box has a known low temperature then the truck cooling is not broken (because a broken cooling affects all boxes)
T: Suppose that all unloaded known fruit boxes can consistently be assumed to have high temperature then box tampering can be excluded (because broken cooling is the more likely explanation)

The formalization of these rules below distinguishes between absent, unknown and known box temperature attributes. This yields different explanations depending on the actual events and what is known about the unloaded boxes at unloading time. One scenario, for instance, unfolds as follows:

Time QP RP SP TP UP
Action load Box P load Box Q load Box R load Box S load Box T
Sensor Box P Z QP ° Box R Z QP ° Box P Z RP °
Timed ABoxes

Description Logics, Event Calculus and Rules

- Description logics and logic programming are “very different”
  - Open world vs closed world, Entailment vs Models, Infinite models vs finite models
- Attractive to integrate for modelling complementary aspects

**Timed ABoxes**

<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>Unload</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Load Box₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sensor</td>
<td>Box₀ : −1⁰</td>
<td>Box₂ : 1⁰</td>
<td>Box₀ : 2⁰</td>
<td>Box₀ : 20⁰</td>
<td></td>
</tr>
</tbody>
</table>

Box₀ : FruitBox
Box₁ : FruitBox
Box₂ : Box
Box₃ : ToyBox
Box₄ : Box ⊃ ∀ temp.¬TempClass
Box₅ : Box ⊃ ∃ temp.TempClass
Description Logics, Event Calculus and Rules

- Description logics and logic programming are “very different”
  - Open world vs closed world, Entailment vs Models, Infinite models vs finite models
- Attractive to integrate for modelling complementary aspects

Timed ABoxes

<table>
<thead>
<tr>
<th>Time</th>
<th>Action</th>
<th>Sensor</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Load Box_0</td>
<td>Box_0: -10°</td>
</tr>
<tr>
<td>20</td>
<td>Load Box_2</td>
<td>Box_2: 10°</td>
</tr>
<tr>
<td>30</td>
<td>Load Box_3</td>
<td>Box_0: 2°</td>
</tr>
<tr>
<td>40</td>
<td>Load Box_4</td>
<td>Box_0: 20°</td>
</tr>
<tr>
<td>50</td>
<td>Unload</td>
<td></td>
</tr>
</tbody>
</table>

Fusemate + DL integration

- Rules can call description logic reasoner
- Rules can extend current ABox / fix past ABox
Description Logics, Event Calculus and Rules

- Description logics and logic programming are “very different”
  - Open world vs closed world, Entailment vs Models, Infinite models vs finite models
- Attractive to integrate for modelling complementary aspects

Timed ABoxes

<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 Box_0</td>
<td>Load Box_2</td>
<td>Load Box_3</td>
<td>Unload</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Load Box_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sensor</td>
<td>Box_0: -10°</td>
<td>Box_2: 10°</td>
<td>Box_0: 2°</td>
<td>Box_0: 20°</td>
<td></td>
</tr>
</tbody>
</table>

[DL]: Box2 is “High temp box” at t=20

[Fusemate + DL integration]

[EC rules]: ... and temp stays at 10° at t=30, 40, 50

- Rules can call description logic reasoner
- Rules can extend current ABox / fix past ABox
Description Logics, Event Calculus and Rules

- Description logics and logic programming are “very different”
  - Open world vs closed world, Entailment vs Models, Infinite models vs finite models
- Attractive to integrate for modelling complementary aspects

Timed ABoxes

<table>
<thead>
<tr>
<th>Time</th>
<th>Action</th>
<th>Sensor</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Load Box₀</td>
<td>Box₀ : −10°</td>
</tr>
<tr>
<td></td>
<td>Load Box₁</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Load Box₂</td>
<td>Box₂ : 10°</td>
</tr>
<tr>
<td></td>
<td>Load Box₃</td>
<td>Box₀ : 2°</td>
</tr>
<tr>
<td>30</td>
<td>Load Box₄</td>
<td>Box₀ : 20°</td>
</tr>
<tr>
<td>40</td>
<td>Unload</td>
<td></td>
</tr>
</tbody>
</table>

Fusemate + DL integration

- Rules can call description logic reasoner
- Rules can extend current ABox / fix past ABox
Description Logics, Event Calculus and Rules

- Description logics and logic programming are “very different”
  - Open world vs closed world, Entailment vs Models, Infinite models vs finite models
  - Attractive to integrate for modelling complementary aspects

Timed ABoxes

<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>Unload</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Load Box₁</td>
<td></td>
<td></td>
<td></td>
<td></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>

Fusemate + DL integration

- Rules can call description logic reasoner
- Rules can extend current ABox / fix past ABox

\[
\text{ColdBox}(\text{time}, \text{box}) : - \\
\quad \text{IsxAA}(\text{time}, x, \text{Box}), \\
\quad \text{NOT} (t < \text{time}, (\text{I.aboxAt}(t), \text{tbox}) \models \text{IsA}(x, \text{Box}), \text{HasA}(x, \text{Temp, High}))
\]

\(\models\) means “provably” (not “consistently”)
Description Logics, Event Calculus and Rules

- Description logics and logic programming are “very different”
  - Open world vs closed world, Entailment vs Models, Infinite models vs finite models
- Attractive to integrate for modelling complementary aspects

Timed ABoxes

<table>
<thead>
<tr>
<th>Time</th>
<th>Action</th>
<th>Sensor</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Load Box₀</td>
<td>Box₀ : −10°</td>
</tr>
<tr>
<td></td>
<td>Load Box₁</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Load Box₂</td>
<td>Box₂ : 10°</td>
</tr>
<tr>
<td>30</td>
<td>Load Box₃</td>
<td>Box₀ : 2°</td>
</tr>
<tr>
<td>40</td>
<td>Unload</td>
<td>Box₀ : 20°</td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Fusemate + DL integration

- Rules can call description logic reasoner
- Rules can extend current ABox / fix past ABox

ColdBox(time, box) :-
  IsAAAt(time, x, Box),
  NOT (t < time, (IaboxAt(t), tbox) |= IsA(x, Box), HasA(x, Temp, High))

|= means “provably” (not “consistently”)
Implementation Aspects
Embedding Into Scala: Translation

Input program ≈ Scala source code

<table>
<thead>
<tr>
<th>Logic</th>
<th>Scala</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pred/Fun signature</td>
<td>Class</td>
</tr>
<tr>
<td>Interpretation</td>
<td>Set of class instances</td>
</tr>
<tr>
<td>Variable</td>
<td>Variable</td>
</tr>
<tr>
<td>Rule</td>
<td>Partial function</td>
</tr>
<tr>
<td>Matching subst</td>
<td>Pattern matching</td>
</tr>
</tbody>
</table>
Embedding Into Scala: Translation

Input program \( \approx \) Scala source code

type Time = Int

case class Load(time: Time, obj: String, cont: String) extends Atom

case class In(time: Time, obj: String, cont: String) extends Atom

@rules
val rules = List( In(time, obj, cont) :- (In(time, obj, c), In(time, c, cont) ) )
Embedding Into Scala: Translation

Input program \(\approx\) Scala source code

type Time = Int
case class Load\((\text{time}: \text{Time}, \text{obj}: \text{String}, \text{cont}: \text{String})\) extends Atom
case class In\((\text{time}: \text{Time}, \text{obj}: \text{String}, \text{cont}: \text{String})\) extends Atom

@rules
val rules = List( In\((\text{time}, \text{obj}, \text{cont})\) :- (In\((\text{time}, \text{obj}, \text{c})\), In\((\text{time}, \text{c}, \text{cont})\) )

<table>
<thead>
<tr>
<th>Logic</th>
<th>Scala</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pred/Fun signature</td>
<td>Class</td>
</tr>
<tr>
<td>Interpretation</td>
<td>Set of class instances</td>
</tr>
<tr>
<td>Variable</td>
<td>Variable</td>
</tr>
<tr>
<td>Rule</td>
<td>Partial function</td>
</tr>
<tr>
<td>Matching subst</td>
<td>Pattern matching</td>
</tr>
</tbody>
</table>
Embedding Into Scala: Translation

Input program \(\approx\) Scala source code

```scala
type Time = Int
case class Load(time: Time, obj: String, cont: String) extends Atom
case class In(time: Time, obj: String, cont: String) extends Atom

@rules
val rules = List(In(time, obj, cont) :- (In(time, obj, c), In(time, c, cont)) )
```

```scala
case List(In(time, obj, c), In(time0, c1, cont))
  if c == c1 && time == time0
  => In(time, obj, cont)
```

---

<table>
<thead>
<tr>
<th>Logic</th>
<th>Scala</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pred/Func signature</td>
<td>Class</td>
</tr>
<tr>
<td>Interpretation</td>
<td>Set of class instances</td>
</tr>
<tr>
<td>Variable</td>
<td>Variable</td>
</tr>
<tr>
<td>Rule</td>
<td>Partial function</td>
</tr>
<tr>
<td>Matching subst</td>
<td>Pattern matching</td>
</tr>
</tbody>
</table>
Embedding Into Scala: Translation

Input program ≈ Scala source code

type Time = Int
case class Load(time: Time, obj: String, cont: String) extends Atom
case class In(time: Time, obj: String, cont: String) extends Atom

@rules
val rules = List( In(time, obj, cont) :- (In(time, obj, c), In(time, c, cont) )

  case List(In(time, obj, c), In(time0, c1, cont))
    if c == c1 && time == time0
  => In(time, obj, cont)

+ given-clause loop operating on such rules-as-partial-functions
(In reality the macro expansion is more complicated because of default negation)
Embedding into Scala: Method

```scala
val eventsCSV = List("Load,10,tomatoes,pallet","Load,20,pallet,container", ...)

// Compute alternative “fixes” and extract their Load/Unload events a CSV again
eventsCSV map { line =>
  line.split"," match {
    case Array("Load", time, obj, cont) => Load(time.toInt, obj, cont)
    ...
  }
} saturate { @rules ...
  fail(...) :-
    ...
    (b ∈ obj) && (b ∈ o),
    where { val b = sameBatch(t) }
} map { I =>
  I.toList.sortBy(_.time) flatMap {
    case Load(time, obj, cont) => List(s"Load,$time,$obj,$cont")
    ...
  }
}
```

List(Load,10,tomatoes,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,tomatoes,pallet)
List(Load,10,tomatoes,pallet, Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)
List(Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)
Embedding into Scala: Method

```
val eventsCSV = List("Load,10,tomatoes,pallet","Load,20,pallet,container", ...)

// Compute alternative “fixes” and extract their Load/Unload events a CSV again
eventsCSV map { line =>
  line.split(",",") match {
    case Array("Load", time, obj, cont) => Load(time.toInt, obj, cont)
    ...  
  }
} saturate { @rules ...
fail(...) :-
  ...
  (b ∋ obj) && (b ∋ o),
  where { val b = sameBatch(t) }
} map { I =>
  I.toList.sortBy(_.time) flatMap {
    case Load(time, obj, cont) => List(s"Load,$time,$obj,$cont")
    ...  
  }
}
```

Natural” integration into Scala and vice versa
Embedding into Scala: Method

```
val eventsCSV = List("Load,10,tomatoes,pallet","Load,20,pallet,container","...

// Compute alternative “fixes” and extract their Load/Unload events a CSV again
eventsCSV map { line =>
  line.split("","") match {
    case Array("Load", time, obj, cont) => Load(time.toInt, obj, cont)
    ...
  }
} saturate {
  fail(...) :-
  ...
  (b ∈ obj) && (b ∈ o),
  where { val b = sameBatch(t) }
} map { I =>
  I.toList.sortBy(_.time) flatMap {
    case Load(time, obj, cont) => List(s"Load,$time,$obj,$cont")
    ...
  }
}
```

List(Load,10,tomatoes,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,tomatoes,pallet)
List(Load,10,tomatoes,pallet, Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)
List(Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)

“Natural” integration into Scala and vice versa
Embedding into Scala: Method

```
val eventsCSV = List("Load,10,tomatoes,pallet","Load,20,pallet,container", ...)

// Compute alternative “fixes” and extract their Load/Unload events a CSV again
eventsCSV map { line =>
    line.split"," match {
        case Array("Load", time, obj, cont) => Load(time.toInt, obj, cont)
        ...
    }
} saturate {
    @rules ...
    fail(...) :-
    ...
    (b ∈ obj) && (b ∈ o),
    where { val b = sameBatch(t) }
} map { I =>
    I.toList.sortBy(_.time) flatMap {
        case Load(time, obj, cont) => List(s"Load,$time,$obj,$cont")
        ...
    }
}
```

“Natural” integration into Scala and vice versa

List(Load,10,tomatoes,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,tomatoes,pallet)
List(Load,10,tomatoes,pallet, Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)
List(Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)
val eventsCSV = List("Load,10,tomatoes,pallet","Load,20,pallet,container", ...)

// Compute alternative “fixes” and extract their Load/Unload events a CSV again
eventsCSV map { line =>
  line.split(",",") match {
    case Array("Load", time, obj, cont) => Load(time.toInt, obj, cont)
    ...
  }
}

List(Load,10,tomatoes,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,tomatoes,pallet)
List(Load,10,tomatoes,pallet, Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)
List(Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)
Embedding into Scala: Method

```
val eventsCSV = List("Load,10,tomatoes,pallet","Load,20,pallet,container", ...)

// Compute alternative "fixes" and extract their Load/Unload events a CSV again
eventsCSV map { line =>
  line.split(",",") match {
    case Array("Load", time, obj, cont) => Load(time.toInt, obj, cont)
    ...
  }
} saturate {
  @rules ...
  fail(...) :-
  ...
  (b ∈ obj) && (b ∈ o),
  where { val b = sameBatch(t) }
} map { I =>
  I.toList.sortBy(_.time) flatMap {
    case Load(time, obj, cont) => List(s"Load,$time,$obj,$cont")
    ...
  }
}
```

"Natural" integration into Scala and vice versa

```
List(Load,10,tomatoes,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,tomatoes,pallet)
List(Load,10,tomatoes,pallet, Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)
List(Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)
```
val eventsCSV = List("Load,10,tomatoes,pallet","Load,20,pallet,container", ...)

// Compute alternative “fixes” and extract their Load/Unload events a CSV again

val eventsCSV = List("Load,10,tomatoes,pallet","Load,10,apples,pallet","Load,20,pallet,container","Load,40,container,ship","Unload,45,container,ship","Unload,50,pallet,container","Unload,60,tomatoes,pallet","Unload,60,apples,pallet")
Embedding into Scala: Method

```
val eventsCSV = List("Load,10,tomatoes,pallet","Load,20,pallet,container", ...)

// Compute alternative "fixes" and extract their Load/Unload events a CSV again
eventsCSV map { line =>
  line.split("","") match {
    case Array("Load", time, obj, cont) => Load(time.toInt, obj, cont)
    ...
  }
} saturate { @rules ...
  fail(...) :-
    ...
    (b ∈ obj) && (b ∈ o),
    where { val b = sameBatch(t) }
} map { I =>
  I.toList.sortBy(_._time) flatMap {
    case Load(time, obj, cont) => List(s"Load,$time,$obj,$cont")
    ...
  }
}
```

List(Load,10,tomatoes,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,tomatoes,pallet)
List(Load,10,tomatoes,pallet, Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)
List(Load,10,apples,pallet, Load,20,pallet,container, Load,40,container,ship, Unload,45,container,ship, Unload,50,pallet,container, Unload,60,apples,pallet)

“Natural” integration into Scala and vice versa
Embedding into Scala: Method

```
val eventsCSV = List("Load,10,tomatoes,pallet","Load,20,pallet,container", ...)

// Compute alternative "fixes" and extract their Load/Unload events a CSV again
eventsCSV map { line =>
    line.split"," match {
        case Array("Load", time, obj, cont) => Load(time.toInt, obj, cont)
        ...}
} saturate {
    @rules ...
fail(...) :-
    ... (b ∋ obj) && (b ∋ o),
    where { val b = sameBatch(t) }
} map { I =>
    I.toList.sortBy(_.time) flatMap {
        case Load(time, obj, cont) => List(s"Load,$time,$obj,$cont")
        ...
    }
}
```

`def` sameBatch(time: Time) =
if (time == 10) Set("tomatoes", "apples") else Set∅[String]
Embedding into Scala: Discussion

Two-way calling interface

• Scala -> Rules calls trivial
• Rules -> Scala calls trivial

Data structures integration is trivial

• Use any Scala data structure in rules
• Logic data structures (models) are Scala data structures
• Unmatched aggregation and introspection capabilities

Disadvantage

• Must rely on Scala pattern matching implementation
• Difficult to implement efficiently
Embedding into Scala: Discussion

Two-way calling interface
• Scala -> Rules calls trivial
• Rules -> Scala calls trivial

Data structures integration is trivial
• Use any Scala data structure in rules
• Logic data structures (models) are Scala data structures
• Unmatched aggregation and introspection capabilities

Disadvantage
• Must rely on Scala pattern matching implementation
• Difficult to implement efficiently

- Tighter coupling than in every other system (I know of)
- Adds “interpretations” as a container data structure to functional/OO programming with “logic programming” as an operator
Three and a Half Case Studies
Case Study 1 - Deer Supply Chain

The Use of EPC RFID Standards for Livestock and Meat Traceability

2013
Case Study 1 - Deer Supply Chain

2013

The Use of EPC RFID Standards for Livestock and Meat Traceability

Events: from farm (NZ) to retailer (DE) encoded in EPCIS
Case Study 1 - Deer Supply Chain

The Use of EPC RFID Standards for Livestock and Meat Traceability

2013

Events: from farm (NZ) to retailer (DE) encoded in EPCIS
Case Study 1 - Deer Supply Chain

The Use of EPC RFID Standards for Livestock and Meat Traceability

2013

Gary Hartley
New Zealand RFID Pathfinder Group
January 2013

What? Where? When? Why?

Events: from farm (NZ) to retailer (DE) encoded in EPCIS
The Use of EPC RFID Standards for Livestock and Meat Traceability

Case Study 1 - Deer Supply Chain

Events: from farm (NZ) to retailer (DE) encoded in EPCIS
Case Study 1 - Deer Supply Chain

From Farm to Supermarket

Of Interest

- Handling structured XML data
- Speculating whereabouts of missing item
  - A box enters supply chain but does not arrive at destination
  - Track same batch boxes as proxies
Case Study 2 - D61 Project “Supply Chain Awareness”

- Partner company BeefLedger ships boxed meat products
- Stringent cooling requirements ensure quality of products
- D61 sensors measure box temperatures
  (S. Khalifa / K. v. Richter)
- Task: Pricing model, anomaly detection
Case Study 2 - D61 Project “Supply Chain Awareness”

- Partner company BeefLedger ships boxed meat products
- Stringent cooling requirements ensure quality of products
- D61 sensors measure box temperatures
  (S. Khalifa / K. v. Richter)
- Task: Pricing model, anomaly detection

![Temperature chart with anomalies highlighted]
Case Study 2 - D61 Project “Supply Chain Awareness”
Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts

10:05
Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts

10:05

10:06
Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts

10:05  10:06  10:07
Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts

10:05

10:06

10:07
Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts

10:05
10:06
10:07

Fix:
10:06
Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts

10:05
10:06
10:07

Fix:

+BoxEvent(t, truckAt, id, (temp + prevTemp) / 2) :-
  BoxAtCoord(time, at, id, temp),
  BoxAtCoord(prev < time,
  SECONDS.between(prev, time),
  HoldsAt(time, On(id, truckId)),
  HoldsAt(prev, On(id, truckId)),
  TruckAtCoordT(t > prev, truckAtT, truckId),
  t < time
Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts

“Behaves differently”
Anomaly
Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts

“Behaves differently”
Anomaly

Fix:
+BoxEvent(t, truckAt, id, (temp + prevTemp) / 2) :-
  BoxAtCoord(time, at, id, temp),
  BoxAtCoord(prev < time, ...
  SECONDS.between(prev, time) ...
  HoldsAt(time, On(id, truckId)),
  HoldsAt(prev, On(id, truckId)),
  TruckAtCoordT(t > prev, truckAt, truckId),
  t < time

10:05
10:06
10:07
10:06
Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts

“Behaves differently” Anomaly
Fix sensor dropouts, anomalies

Fix GPS dropouts

“Behaves differently” Anomaly

Fix:
Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts

“Behaves differently”
Anomaly

Fix sensor dropouts, anomalies

Box moved to cabin?
Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

**Fix GPS dropouts**

- “Behaves differently”
- Anomaly

- “Is different”
- Anomaly

Box moved to cabin?
Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts

“Behaves differently”
Anomaly

“Is different”
Anomaly

Box moved to cabin?
Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts

“Behaves differently”
Anomaly

“Is different”
Anomaly

Box moved to cabin?

Worth checking?
Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

**Fix GPS dropouts**

10:05

10:06

10:07

**“Behaves differently”**

Anomaly

Box moved to cabin?

**“Is different”**

Anomaly

Worth checking?

Clustering based on similarity measure for feature vector
Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts
10:05

“Behaves differently”
Anomaly

“Is different”
Anomaly

Cooling OK?

Pricing?

Worth checking?

Box moved to cabin?

Clustering based on similarity
measure for feature vector

+BoxEvent(t, truckAt, id, (temp + prevTemp) / 2) :-
BoxAtCoord(time, at, id, temp),
BoxAtCoord(prev < time, ...
SECONDS.between(prev, t), ...
HoldsAt(time, On(id, truckAt)), ...
HoldsAt(prev, On(id, truckAt)), ...
TruckAtCoordT(t > prev, truckAt, truckAt),
t < time
Case Study 2 - D61 Project “Supply Chain Awareness”

Fix sensor dropouts, anomalies

Fix GPS dropouts
10:05

“Behaves differently”
Anomaly

“Is different”
Anomaly

Cooling OK?
Pricing?

Worth checking?

Box moved to cabin?

Clustering based on similarity measure for feature vector

Concrete scenarios: normal, latecool, missingbox, cabinbox
Case Study 2 - D61 Project “Supply Chain Awareness”

Rule for recovering sensor dropout

\[
\text{FAIL}(+ \ \text{BoxEvent}(t, \ \text{truckAt}, \ id, (\text{prevTemp} + \text{temp})/2)) : - ( \\
\quad \text{BoxAtCoord}(\text{time}, \ \text{at}, \ id, \ \text{temp}), \\
\quad \text{BoxAtCoord}(\text{prev} < \text{time}, \ _, \ id, \ \text{prevTemp}) \ \text{STH} \\
\quad \text{SECONDS.betwee}(\text{prev}, \ \text{time}) \leq \ \text{Sensor Dropout Allowance}, \\
\quad \text{BoxOnTruck}(\text{prev}, \ id), \\
\quad \text{BoxOnTruck}(\text{time}, \ id), \\
\quad \text{TruckAtCoord}(t, \ \text{truckAt}) \ \text{STH} \ \text{prev} < t \land t < \text{time}, \\
\quad \text{NOT} ( \ \text{TruckAtCoord}(s, \ _) \ \text{STH} \ \text{prev} < s \land s < t)
\]

Time
Loc
Case Study 2 - D61 Project “Supply Chain Awareness”

Rule for recovering sensor dropout

```
FAIL(+ BoxEvent(t, truckAt, id, (prevTemp + temp)/2)) :- (  
  BoxAtCoord(time, at, id, temp),  
  BoxAtCoord(prev < time, _, id, prevTemp) STH  
  SECONDS.between(prev, time) ≤ SensorDropoutAllowance,  
  BoxOnTruck(prev, id),  
  BoxOnTruck(time, id),  
  TruckAtCoord(t, truckAt) STH prev < t ∧ t < time,  
  NOT ( TruckAtCoord(s, _) STH prev < s ∧ s < t )
```

2°C

Time  10
Loc   A
Rule for recovering sensor dropout

FAIL(+ BoxEvent(t, truckAt, id, (prevTemp + temp)/2)) :- (  
  ② BoxAtCoord(time, at, id, temp),  
  ① BoxAtCoord(prev < time, _, id, prevTemp) STH  
          SECONDS.between(prev, time) ≤ SensorDropoutAllowance,  
          BoxOnTruck(prev, id),  
          BoxOnTruck(time, id),  
          TruckAtCoord(t, truckAt) STH prev < t ∧ t < time,  
          NOT ( TruckAtCoord(s, _) STH prev < s ∧ s < t )

<table>
<thead>
<tr>
<th>Time</th>
<th>Loc</th>
<th>Temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>A</td>
<td>2°C</td>
</tr>
<tr>
<td>20</td>
<td>C</td>
<td>10°C</td>
</tr>
</tbody>
</table>
Rule for recovering sensor dropout

\[
\text{FAIL}(+ \text{BoxEvent}(t, \text{truckAt}, \text{id}, (\text{prevTemp} + \text{temp})/2)) : - ( \\
\begin{align*}
\text{BoxAtCoord}(\text{time}, \text{at}, \text{id}, \text{temp}), \\
\text{BoxAtCoord}(\text{prev} < \text{time}, \_\_\_, \text{id}, \text{prevTemp}) \ \text{STH}
\end{align*}
\]

\[
\begin{align*}
\text{SECONDS.between}(\text{prev}, \text{time}) \leq \text{SensorDropdownAllowance}, \\
\text{BoxOnTruck}(\text{prev}, \text{id}), \\
\text{BoxOnTruck}(\text{time}, \text{id}), \\
\text{TruckAtCoord}(t, \text{truckAt}) \ \text{STH} \ \text{prev} < t \land t < \text{time}, \\
\text{NOT} ( \ \text{TruckAtCoord}(s, \_\_\_) \ \text{STH} \ \text{prev} < s \land s < t)
\end{align*}
\]

<table>
<thead>
<tr>
<th>Time</th>
<th>Loc</th>
<th>Temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>A</td>
<td>2°C</td>
</tr>
<tr>
<td>?</td>
<td></td>
<td>?</td>
</tr>
<tr>
<td>20</td>
<td>C</td>
<td>10°C</td>
</tr>
</tbody>
</table>
Rule for recovering sensor dropout

\[
\text{FAIL}(+, \text{BoxEvent}(t, \text{truckAt}, \text{id}, (\text{prevTemp} + \text{temp})/2)) : - ( \\
\text{② BoxAtCoord}(t, \text{at}, \text{id}, \text{temp}), \\
\text{① BoxAtCoord}(\text{prev} < \text{time}, \_, \text{id}, \text{prevTemp}) \text{STH} \\
\quad \text{SECONDS.between}(\text{prev}, \text{time}) \leq \text{SensorDropoutAllowance}, \\
\text{③ BoxOnTruck}(\text{prev}, \text{id}), \\
\quad \text{BoxOnTruck}(\text{time}, \text{id}), \\
\quad \text{TruckAtCoord}(t, \text{truckAt}) \text{STH} \text{ prev} < t \land t < \text{time}, \\
\quad \text{NOT} (\text{TruckAtCoord}(s, \_) \text{STH} \text{ prev} < s \land s < t))
\]
Rule for recovering sensor dropout

FAIL(+ BoxEvent(t, truckAt, id, (prevTemp + temp)/2)) :- (  
  2 BoxAtCoord(time, at, id, temp),  
  1 BoxAtCoord(prev < time, _, id, prevTemp) STH  
      SECONDS.between(prev, time) ≤ SensorDropdownAllowance,  
  3 BoxOnTruck(prev, id),  
  4 BoxOnTruck(time, id),  
    TruckAtCoord(t, truckAt) STH prev < t ∧ t < time,  
    NOT ( TruckAtCoord(s, _) STH prev < s ∧ s < t ) )

- Time 10  
- Loc A  
- 2°C  
- ?  
- 10°C  
- Time 20  
- Loc C
Case Study 2 - D61 Project “Supply Chain Awareness”

Rule for recovering sensor dropout

$$\text{FAIL}(+ \text{BoxEvent}(t, \text{truckAt}, \text{id}, (\text{prevTemp} + \text{temp})/2)) :- ( \quad \text{BoxAtCoord}(t, \text{at}, \text{id}, \text{temp}),$$

$$\text{BoxAtCoord}(\text{prev < time}, \_ , \text{id}, \text{prevTemp}) \quad \text{STH}$$

$$\text{SECONDS.between}(\text{prev, time}) \leq \text{SensorDropoutAllowance},$$

$$\text{BoxOnTruck}(\text{prev, id}),$$

$$\text{BoxOnTruck}(\text{time, id}),$$

$$\text{TruckAtCoord}(t, \text{truckAt}) \quad \text{STH} \quad \text{prev < t} \land \text{t < time},$$

$$\text{NOT} \left( \text{TruckAtCoord}(s, \_ ) \quad \text{STH} \quad \text{prev < s} \land \text{s < t} \right)$$

<table>
<thead>
<tr>
<th>Time</th>
<th>Loc</th>
<th>Temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>A</td>
<td>2°C</td>
</tr>
<tr>
<td>15</td>
<td>B</td>
<td>?</td>
</tr>
<tr>
<td>20</td>
<td>C</td>
<td>10°C</td>
</tr>
</tbody>
</table>
Case Study 2 - D61 Project “Supply Chain Awareness”

Rule for recovering sensor dropout

FAIL(+ BoxEvent(t, truckAt, id, (prevTemp + temp)/2)) :- (  
  2 BoxAtCoord(time, at, id, temp),  
  1 BoxAtCoord(prev < time, __, id, prevTemp) STH  
  6 SECONDS.between(prev, time) ≤ SensorDropoutAllowance,  
  3 BoxOnTruck(prev, id),  
  4 BoxOnTruck(time, id),  
  5 TruckAtCoord(t, truckAt) STH prev < t ∧ t < time,  
      NOT ( TruckAtCoord(s, __) STH prev < s ∧ s < t ) )
Case Study 2 - D61 Project “Supply Chain Awareness”

Rule for recovering sensor dropout

7 \text{FAIL}(+ \text{BoxEvent}(t, \text{truckAt}, \text{id}, (\text{prevTemp} + \text{temp})/2)) :- (  
2 \text{BoxAtCoord}(\text{time}, \text{at}, \text{id}, \text{temp}),  
1 \text{BoxAtCoord}(\text{prev < time, }, \_, \text{id}, \text{prevTemp}) \text{ STH}  
6 \text{SECONDS.between}(\text{prev, time}) \leq \text{SensorDropoutAllowance},  
3 \text{BoxOnTruck}(\text{prev, id}),  
4 \text{BoxOnTruck}(\text{time, id}),  
5 \text{TruckAtCoord}(t, \text{truckAt}) \text{ STH prev < t \land t < time},  
\text{NOT ( TruckAtCoord}(s, \_, \) \text{ STH prev < s \land s < t) )}

\begin{tabular}{|c|c|}
\hline
\text{Time} & 10 \\
\hline
\text{Loc} & A \\
\hline
\end{tabular}

\begin{tabular}{|c|c|}
\hline
\text{Time} & 15 \\
\hline
\text{Loc} & B \\
\hline
\end{tabular}

\begin{tabular}{|c|c|}
\hline
\text{Time} & 20 \\
\hline
\text{Loc} & C \\
\hline
\end{tabular}
Rule for recovering sensor dropout

7 FAIL(+ BoxEvent(t, truckAt, id, (prevTemp + temp)/2)) :- 
   2 BoxAtCoord(time, at, id, temp),
   1 BoxAtCoord(prev < time, _, id, prevTemp) STH
   6 SECONDS.between(prev, time) ≤ SensorDropoutAllowance,
   3 BoxOnTruck(prev, id),
   4 BoxOnTruck(time, id),
   5 TruckAtCoord(t, truckAt) STH prev < t ∧ t < time,
      NOT ( TruckAtCoord(s, _) STH prev < s ∧ s < t )

- Similar rule for truck location recovery
- 25 rules altogether
Fusemate System Demo

Fusemate Messages
Model 1 at 16:00:08
Fusemate System Demo

- GPS -> Symbolic Loc
- Integrating information sources
- Noisy sensor data
- Robust anomaly detection
Case Study 3 - Taxi Rides Anomalies

2 Million taxi rides in New York City
Ride(taxi,license,from,to,start,end,fare)

Fusemate
(1) Rules for hotspot clustering and concave hull
(2) Rules for anomaly detection
Case Study 3 - Taxi Rides Anomalies

From Scala to Fusemate and back

```scala
val gaps42 = rides filter {
    _.license == "42"
} saturateFirst {
  Gap(taxi, license, prevEnd, start, prevTo, from) :- ( 
    Ride(taxi, license, start, end, _, _, from, _, _, _),
    Ride(taxi, license, _, prevEnd, _, _, _, prevTo, _, _),
    start isAfter prevEnd,
    NOT ( 
        Ride(taxi, license, otherStart, otherEnd, _, _, _, _, _, _),
        (start isAfter otherStart) ∧ (otherStart isAfter prevEnd)
    )
  )} collect {
  case g:Gap ⇒ g
}
```
Case Study 3 - Taxi Rides Anomalies

From Scala to Fusemate and back

```scala
val gaps42 = rides filter {
  _.license == "42"
} saturateFirst {
  Gap(taxi, license, prevEnd, start, prevTo, from) :- (  
    Ride(taxi, license, start, end, __, __, from, __, __, __),
    Ride(taxi, license, __, prevEnd, __, __, __, prevTo, __, __),
    start isAfter prevEnd,
    NOT (    
      Ride(taxi, license, otherStart, otherEnd, __, __, __, __, __, __),
      (start isAfter otherStart) ^ (otherStart isAfter prevEnd)   
    )
  )
} collect {
  case g:Gap => g
}
```
Case Study 3 - Taxi Rides Anomalies

From Scala to Fusemate and back

```scala
val gaps42 = rides filter {
  _.license == "42"
} saturateFirst {
  Gap(taxi, license, prevEnd, start, prevTo, from) :- (
    Ride(taxi, license, start, end, _, _, from, _, _, _),
    Ride(taxi, license, _, prevEnd, _, _, _, prevTo, _, _),
    start isAfter prevEnd,
    NOT (
      Ride(taxi, license, otherStart, otherEnd, _, _, _, _, _, _),
      (start isAfter otherStart) ∧ (otherStart isAfter prevEnd)
    )
  )} collect {
  case g:Gap ⇒ g
}
```
Case Study 3 - Taxi Rides Anomalies

From Scala to Fusemate and back

```scala
val gaps42 = rides filter {
  _.license == "42"
} saturateFirst {
  Gap(taxi, license, prevEnd, start, prevTo, from) :- (  
    Ride(taxi, license, start, end, _, _, from, _, _, _),
    Ride(taxi, license, _, prevEnd, _, _, _, prevTo, _, _),
    start isAfter prevEnd,
    NOT (  
      Ride(taxi, license, otherStart, otherEnd, _, _, _, _, _, _),
      (start isAfter otherStart) ∧ (otherStart isAfter prevEnd)
    )
  )
} collect {
  case g:Gap ⇒ g
}
```
Case Study 3 - Taxi Rides Anomalies

From Scala to Fusemate and back

```scala
val gaps42 = rides filter {
  _.license == "42"
} saturateFirst {
  Gap(taxi, license, prevEnd, start, prevTo, from) :- ( 
    Ride(taxi, license, start, end, _, _, from, _, _, _), 
    Ride(taxi, license, _, prevEnd, _, _, _, prevTo, _, _), 
    start isAfter prevEnd, 
    NOT ( 
      Ride(taxi, license, otherStart, otherEnd, _, _, _, _, _, _), 
      (start isAfter otherStart) ∧ (otherStart isAfter prevEnd)
    )
  )} collect {
    case g:Gap ⇒ g
}
```
Case Study 3 - Taxi Rides Anomalies

From Scala to Fusemate and back

```scala
val gaps42 = rides filter {
  _.license == "42"
}saturateFirst {
  Gap(taxi, license, prevEnd, start, prevTo, from) :- (
    Ride(taxi, license, start, end, _, _, from, _, _, _),
    Ride(taxi, license, _, prevEnd, _, _, _, prevTo, _, _),
    start isAfter prevEnd,
    NOT (
      Ride(taxi, license, otherStart, otherEnd, _, _, _, _, _, _),
      (start isAfter otherStart) ∧ (otherStart isAfter prevEnd)
    )
  }
} collect {
  case g:Gap ⇒ g
}
```
Case Study 3 - Taxi Rides Anomalies

From Scala to Fusemate and back

```scala
val gaps42 = rides.filter {
  _.license == "42"
} saturateFirst {
  Gap(taxi, license, prevEnd, start, prevTo, from) :- (
    Ride(taxi, license, start, end, _, _, from, _, _, _),
    Ride(taxi, license, _, prevEnd, _, _, prevTo, _, _),
    start isAfter prevEnd,
    NOT (
      Ride(taxi, license, otherStart, otherEnd, _, _, _, _, _, _),
      (start isAfter otherStart) ∧ (otherStart isAfter prevEnd)
    )
  ) } collect {
    case g:Gap ⇒ g
  }
```
Case Study 3 - Taxi Rides Anomalies

From Scala to Fusemate and back

```scala
val gaps42 = rides.filter {
  _.license == "42"
} saturateFirst {
  Gap(taxi, license, prevEnd, start, prevTo, from) :- (
    Ride(taxi, license, start, end, _, _, from, _, _, _),
    Ride(taxi, license, _, prevEnd, _, _, _, prevTo, _, _),
    start isAfter prevEnd,
    NOT (
      Ride(taxi, license, otherStart, otherEnd, _, _, _, _, _),
      (start isAfter otherStart) ∧ (otherStart isAfter prevEnd)
    )
  )
} collect {
  case g:Gap => g
}
```
Case Study 3 - Taxi Rides Anomalies

From Scala to Fusemate and back

```scala
val gaps42 = rides filter {
  _.license == "42"
} saturateFirst {
  Gap(taxi, license, prevEnd, start, prevTo, from) :- ( 
    Ride(taxi, license, start, end, __, __, from, __, __, __), 
    Ride(taxi, license, __, prevEnd, __, __, __, prevTo, __, __),
    start isAfter prevEnd,
    NOT ( 
      Ride(taxi, license, otherStart, otherEnd, __, __, __, __, __, __),
      (start isAfter otherStart) \ (otherStart isAfter prevEnd)
    )
  ) } collect {
  case g:Gap => g
}
```

Fusemate invocation
Case Study 3 - Taxi Rides Anomalies

From Scala to Fusemate and back

```scala
val gaps42 = rides.filter {
  _.license == "42"
} saturateFirst { Fusemate invocation
  Gap(taxi, license, prevEnd, start, prevTo, from) :- ( 
    Ride(taxi, license, start, end, _, _, from, _, _, _),
    Ride(taxi, license, _, prevEnd, _, _, _, prevTo, _, _),
    start isAfter prevEnd,
    NOT ( 
      Ride(taxi, license, otherStart, otherEnd, _, _, _, _, _, _),
      (start isAfter otherStart) ∧ (otherStart isAfter prevEnd)
    )
  )} collect {
  case g:Gap ⇒ g
}

Functional + Logic programming
Declarative and concise :)
```
Case Study 3 - Taxi Rides Anomalies

Anomaly: gap at a busy pickup hotspot

| hour: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
|------|---|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| pickups: | 16 | 34 | 35 | 30 | 26 | 20 | 7 | 20 | 8 | 5 | 9 | 25 | 36 | 36 | 31 | 55 | 50 | 44 | 24 | 64 | 69 | 38 | **189** | 21 |
| dropoffs: ( | 16 | 49 | 70 | 73 | 48 | 22 | 33 | 17 | 22 | 28 | 44 | 43 | 116 | 76 | 76 | 83 | 57 | 74 | 70 | 76 | 36 | 13 | **34** | 18 |)
Case Study 3 - Taxi Rides Anomalies

Anomaly: gap at a busy pickup hotspot

Of Interest

- Reasoning with non-trivially sized data sets
- Deploying Logic Programming as a method for data analysis (as a Jupyter notebook)
- Interaction Fusemate with host programming language Scala
## Data Cleansing as Situational Awareness

### Example: Employments Database

<table>
<thead>
<tr>
<th>Company</th>
<th>Employee</th>
<th>Since</th>
<th>Full-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABM</td>
<td>Alice</td>
<td>1/3/18</td>
<td>No</td>
</tr>
<tr>
<td>BBM</td>
<td>Bob</td>
<td>5/3/18</td>
<td>No</td>
</tr>
<tr>
<td>ABM</td>
<td>Alice</td>
<td>1/6/19</td>
<td>Yes</td>
</tr>
</tbody>
</table>
# Data Cleansing as Situational Awareness

## Example: Employments Database

<table>
<thead>
<tr>
<th>Company</th>
<th>Employee</th>
<th>Since</th>
<th>Full-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABM</td>
<td>Alice</td>
<td>1/3/18</td>
<td>No</td>
</tr>
<tr>
<td>BBM</td>
<td>Bob</td>
<td>5/3/18</td>
<td>No</td>
</tr>
<tr>
<td>ABM</td>
<td>Alice</td>
<td>1/6/19</td>
<td>Yes</td>
</tr>
</tbody>
</table>

*Problem: More than a full-time contract at the same time*
Data Cleansing as Situational Awareness

Example: Employments Database

<table>
<thead>
<tr>
<th>Company</th>
<th>Employee</th>
<th>Since</th>
<th>Full-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABM</td>
<td>Alice</td>
<td>1/3/18</td>
<td>No</td>
</tr>
<tr>
<td>BBM</td>
<td>Bob</td>
<td>5/3/18</td>
<td>No</td>
</tr>
<tr>
<td>ABM</td>
<td>Alice</td>
<td>1/6/19</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Problem: More than a full-time contract at the same time

How to explain and fix this inconsistency?

Approach

• There is a fixed set of contract operators: *cessation*, *conversion*, *new contract*
• Try them out as “fixes” for the problem
• Or was it Bob? Or someone else?
Conclusions

Summary

“Situational awareness = time-stratified logic programming + belief revision”

-> Good balance between expressivity and declarativity
The implementation is meant to be practical (workflow integration, ease of use)

Current and Future Work

Classical negation
Proper belief revision (ramification problem)
Timed LTL constraints  □ t . shipped(B) → ◇ s . s ≤ t + 5 ∧ received(B)

Probabilities and combination with machine learning

• Probabilistic EDBs a la ProbLog  Load(10, “tomatoes”, “pallet”) : 0.3
• ML as a subroutine for anomaly detection?
  Context may help to avoid false positives

Implementation at https://bitbucket.csiro.au/users/bau050/repos/fusemate/