Possible Models Computation and Revision – A Practical Approach

Peter Baumgartner

Data61 | CSIRO

The Australian National University

Situational Awareness ≈ comprehending system state as it evolves over time

Factory Floor

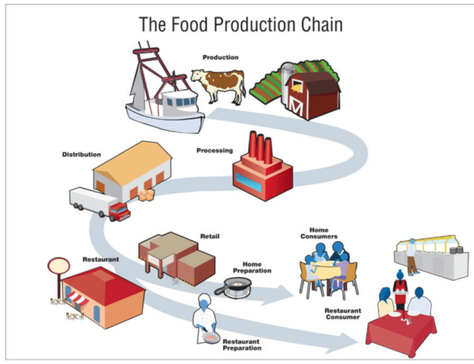
Are the operations carried out according to the schedule?

Food Supply Chain

Are goods delivered within 3 hours and stored below 25℃?

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 is the difficulty?

- Events **happened** ≠ events **reported** (errors, incomplete, late ...) ← Belief revision
- Need an integrated domain model with dependencies ← Logic program
- Can only hope for **multiple** plausible explanations ← Models

This talk: our approach to computing situational awareness

Events happened ≠ events reported

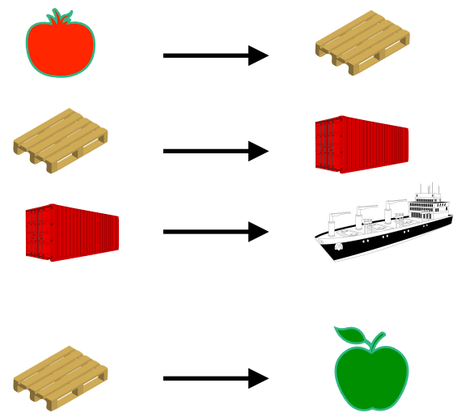
"Fixing the event stream" as computed by our implementation

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)
```

Fixed

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)
```

Fixed

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)
```

Fixed

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(next, obj, cont) :-
  In(time, obj, cont),
  Step(next, time),
  not Unload(next, obj, cont),
  not (In(time, obj, c),
       Unload(next, c, cont))
```

Default negation

Integrity constraints and revision

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

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

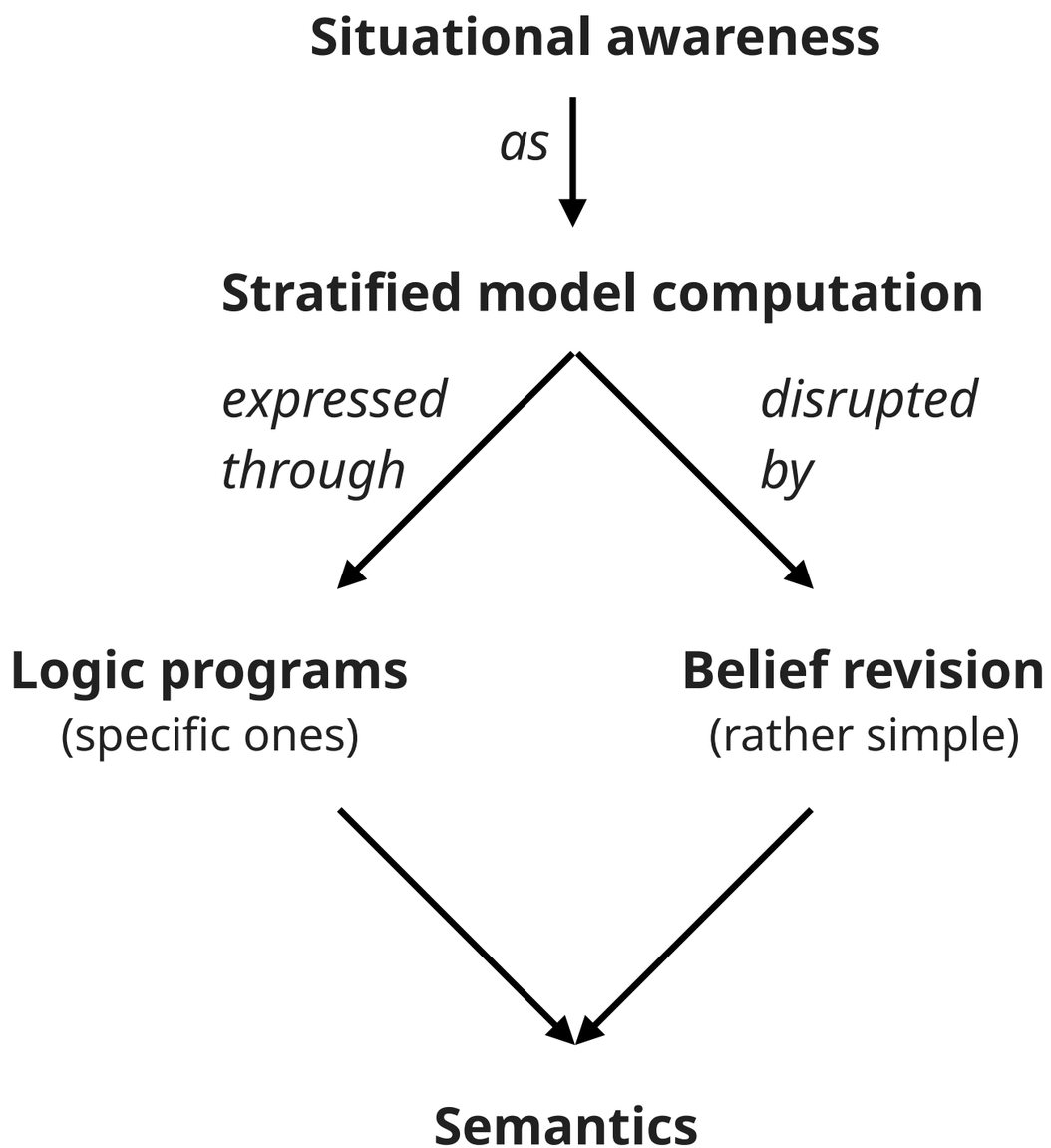
"fail" heads for fixing the event stream

new

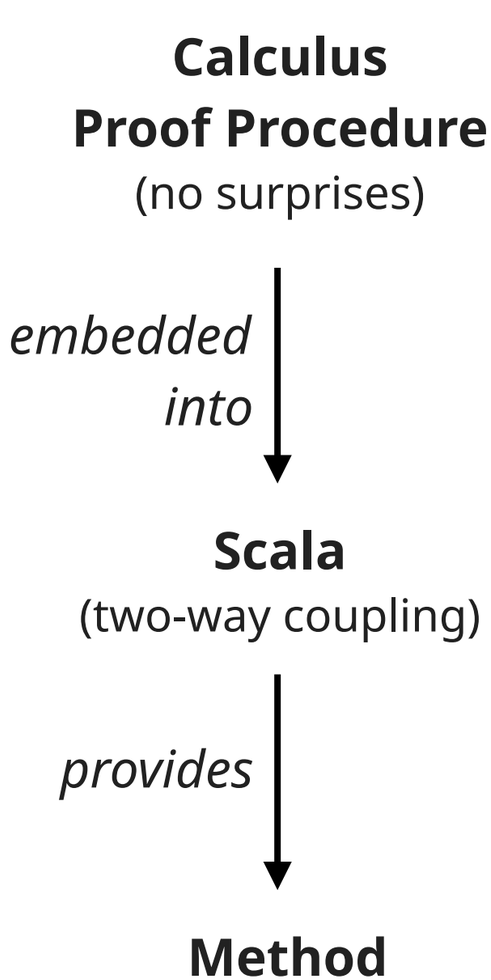
+ *4 more rules*

The Rest of This Talk Graph

"Possible Models Computation and Revision -"



"A Practical Approach"



Situational Awareness = Stratified Model Computation

"Not known now" -> "never known"
Makes default negation possible

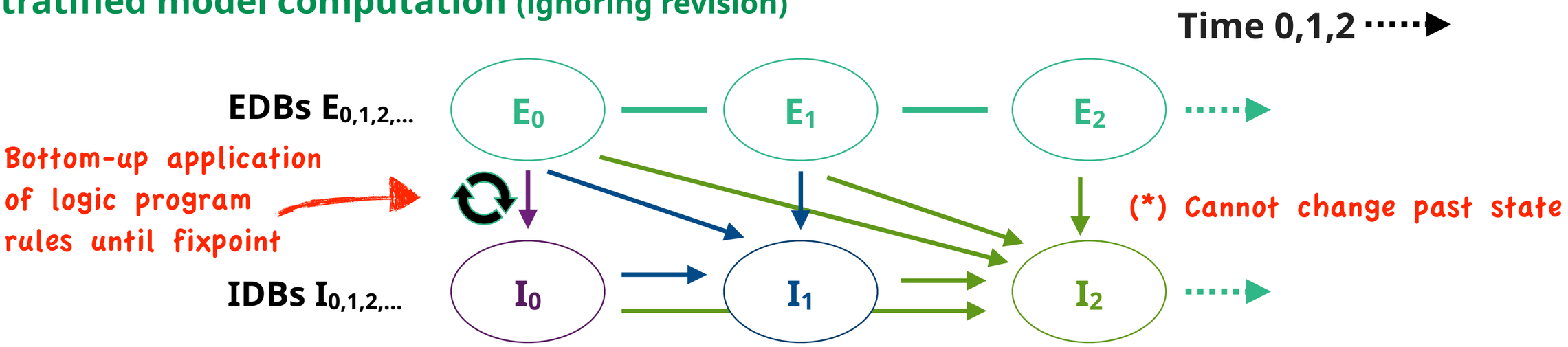
"Situational awareness" task is naturally stratified

- 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

Stratified model computation (ignoring revision)



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*`,` … `,` **`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*

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

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

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

No!

I, J: IDB
E: EDB

Integrity Constraints

Usual integrity constraints

`fail :-` *body*`,` **…**`,` **`not`** *body*`,` **…**

Generalized for revision of EDB literals

`fail(-`*e*`,` **…**`,` **`+`***f*`,` **…`)`** **`:-`** *body*`,` **…**`,` **`not`** *body*`,` **…**

s. th.
- *"conditions for body as for ordinary rules"*
- EDB lits *e* and *f* have time ≤ *time*

Semantics

$E \cup I$

↓ if $E \cup I \models (body, \ldots, \texttt{not}\ body, \ldots)\sigma$

$(E \setminus e\sigma) \cup f\sigma$

Example

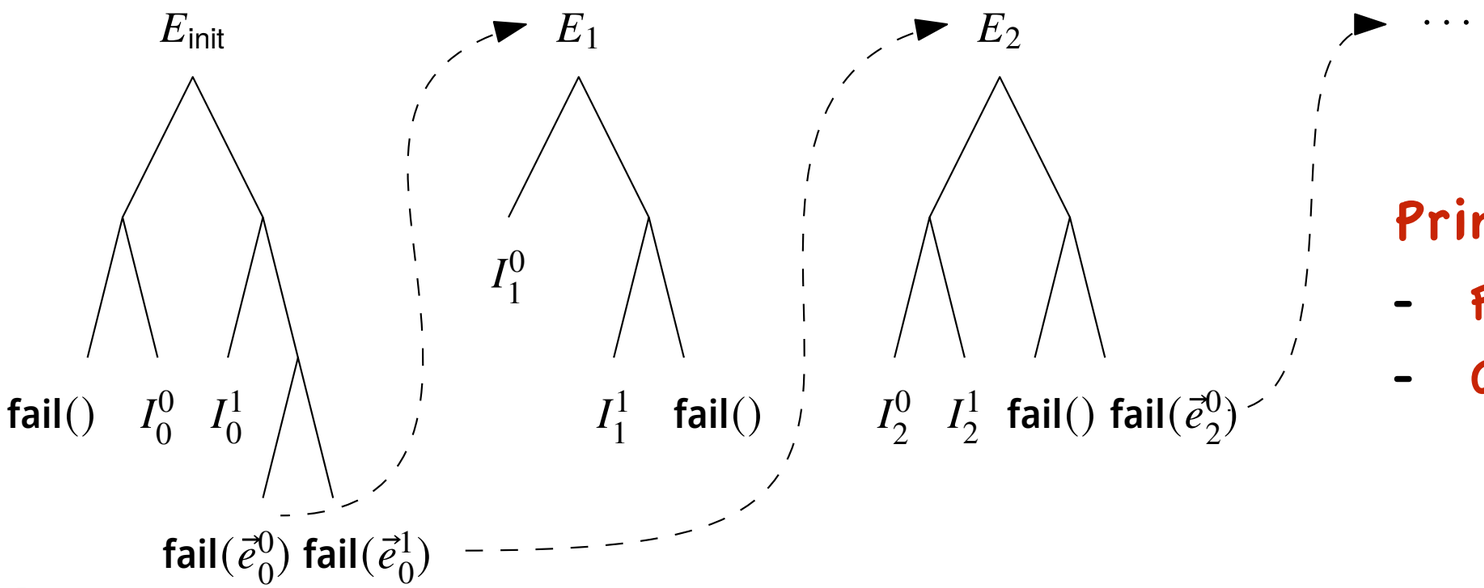
```
// Unload a different object
fail(- Unload(time, obj, cont),
     + Unload(time, o, cont)) :-
  Unload(time, obj, cont),
  not (Load(t, obj, cont), t < time),
  Load(t, o, cont), t < time,
  …
```

```
- …
  Unload(60, apples,    pallet)
```

↓

```
+ …
  Unload(60, tomatoes,  pallet)
```

Semantics of Programs With Fail Rules



Principles

- Fail as early as possibly
- Collect all possible fails

Operational

for a given EDB *E*

  for time *t* = 0,1,2, ..., now

    compute { I^0, I^1, ... all IDBs for time ≤ *t* }

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

      let F = { **`fail(…)`** heads derivable from $E \cup I$ }

      if F is non-empty then

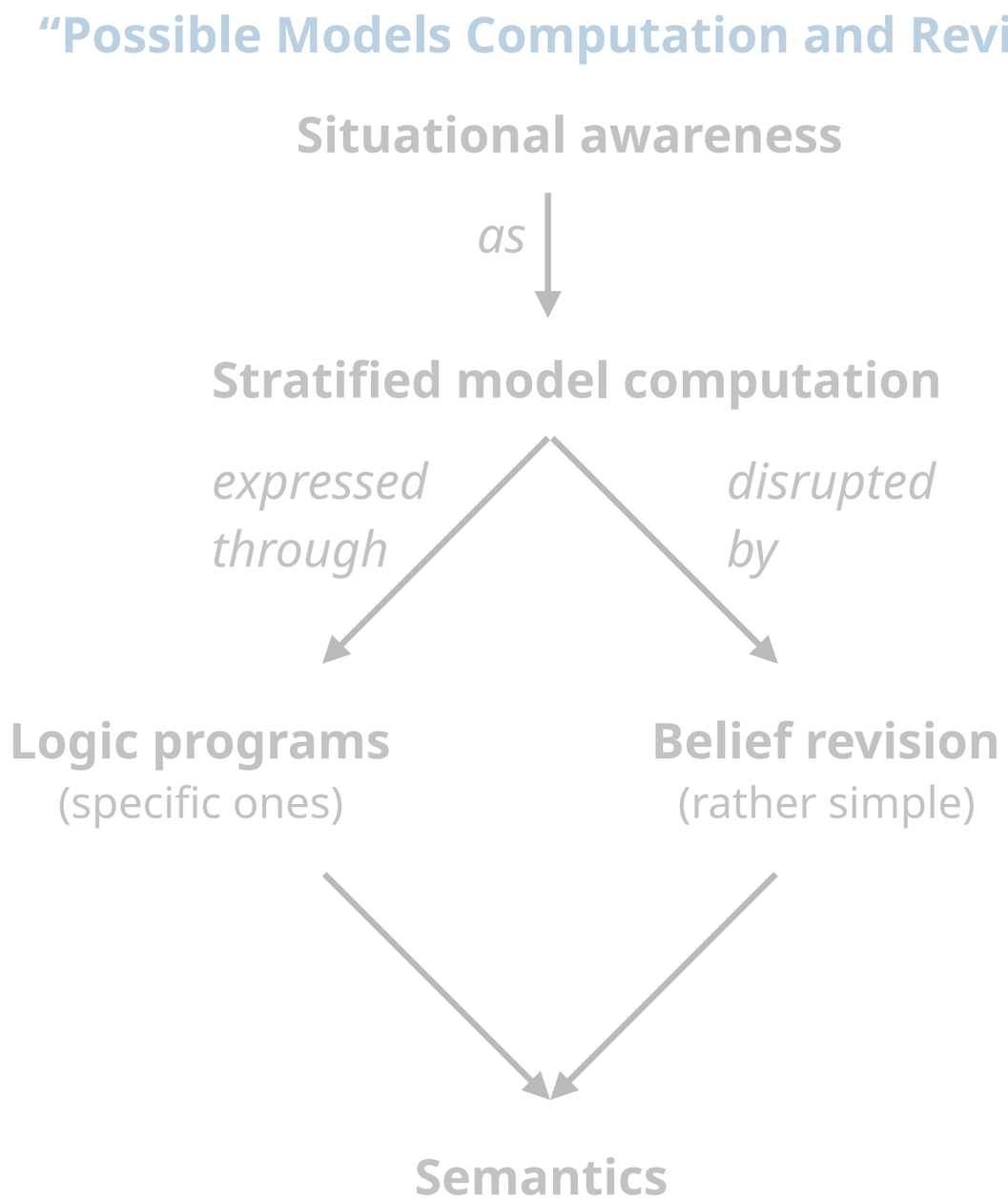
        obtain new EDBs E_1, E_2, ... as per F and

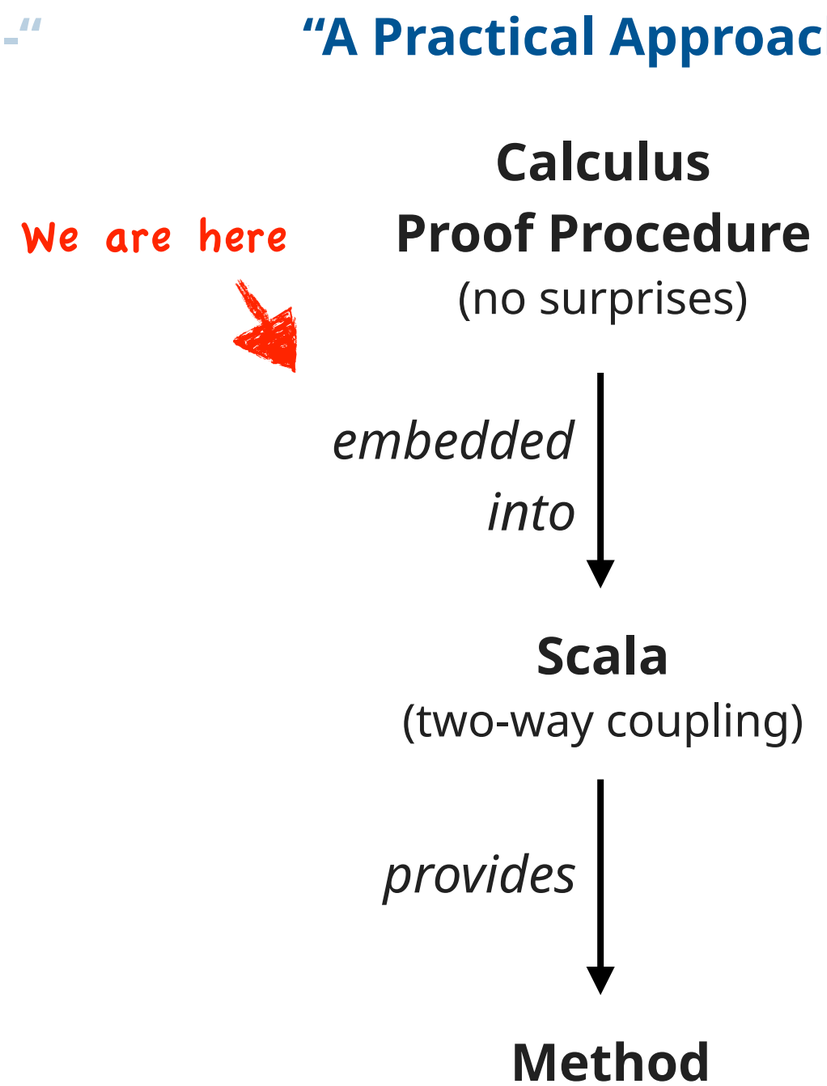
        abandon model candidate I

Can branch out because of disjunctive heads

Declarative: see paper

The Rest of This Talk Graph





Embedding Into Scala: Translation

Logic	Scala
Pred/Fun signature	Class
Interpretation	Set of class instances
Variable	Variable
Rule	Partial function
Matching subst	Pattern matching

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) )
```

Macro annotation

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

Macro expansion into partial function

+ 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

"Natural" integration into Scala and vice versa

```
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]
```

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)

Conclusions

Talk Summary

"Situational awareness = time-stratified logic programming + belief revision"

Practical? (a) Scala embedding (b) structured data (c) controllable complexity

In the Paper

Disjunctive heads, possible model semantics: `Hungry(t) v Thirsty(t) :- GetUp(t)`

Partial correctness: soundness and model completeness theorem

Current and Future Work

Generalize two-layer EDB/IDB stratification to arbitrary many levels [implemented]

Classical negation [implemented]

Proper belief revision

Timed LTL constraints $\Box t \,.\, \mathsf{shipped}(B) \rightarrow \Diamond s \,.\, s \leq t + 5 \wedge \mathsf{received}(B)$

Probabilistic EDBs a la ProbLog `Load(10, "tomatoes", "pallet") : 0.3`

Get the implementation at https://bitbucket.csiro.au/users/bau050/repos/fusemate/