Operationalizing Declarative and Procedural Knowledge
a benchmark on Logic Programming Petri Nets (LPPNs)

CAUSAL2020 Workshop on Causal Reasoning and Explanation in Logic Programming @ ICLP2020
19 September 2020

Giovanni Sileno  g.sileno@uva.nl
Informatics Institute, University of Amsterdam
Problem: reasoning with cases

• Regulations concern **systems of norms**, that in abstract, in a fixed point in time, may be approached atemporally.

• However, when applied, regulations deal with a **continuous flow of events**.

• Prototypical encounter: legal cases.

• More general but similar problem: narratives, stories.
Problem: reasoning with cases

While John was walking his dog, the dog ate Paul’s flowers.
Problem: reasoning with cases

While John was walking his dog, the dog ate Paul’s flowers.

How to entail that John is responsible to pay Paul?
Problem: reasoning with cases

While John was walking his dog, the dog ate Paul’s flowers.

The owner of an animal has to pay for the damages it produces. (example of underlying norm)

How to entail that John is responsible to pay Paul?
Problem: reasoning with cases

While John was walking his dog, the dog ate Paul’s flowers.

The owner of an animal has to pay for the damages it produces. (example of underlying norm)

A conceptual gap exists between the concrete domain and the legal abstraction that applies on it.

How to entail that John is responsible to pay Paul?
While John was walking his dog, the dog ate Paul’s flowers.

The owner of an animal has to pay for the damages it produces.

eating an object destroys the object

dogs are animals

flowers are objects

destruction is damage

flowers are objects
While John was walking his *dog*, the dog *ate* Paul’s *flowers*.

The owner of an *animal* has to pay for the *damages* it produces.

- **dogs are animals**
- **flowers are objects**
- **eating an object destroys the object**
- **destruction is damage**

Some connections are terminological (e.g. taxonomical relations). Other provides causal meaning.
Types of Knowledge

- **Declarative knowledge**, concerning objects (physical, mental, institutional) and their logical relationships—typically reified by means of symbols

- **Procedural knowledge**, concerning patterns of events/actions, mechanisms, or processes (involving objects)—often tacit, internalized
Perspectives on Modelling

• Physical systems can be approached from **steady state** (equilibrium) or **transient** (non-equilibrium, dynamic) perspectives

  ![Graph showing transient and steady state with time axis]

• Steady states descriptions **omit** transient characteristics
  
  ex. Ohm's Law. \( V = R \times I \)
Specifying transients and steady states

• Possible analogies:
  – steady state approach with
    • Logic
    • Declarative programming

focus on What
Specifying transients and steady states

- Possible analogies:
  - steady state approach with
    - Logic
    - Declarative programming
  - transient approach
    - Process modelling
    - Procedural programming
Specifying transients and steady states

• Possible analogies:
  – steady state approach with
    • Logic
    • Declarative programming
  – transient approach
    • Process modelling
    • Procedural programming
Specifying transients and steady states

- Possible analogies:
  - **steady state** approach
    - Logic
    - Declarative programming
  - **transient** approach
    - Process modeling
    - Procedural programming

Answer Set Programming

focus on What

focus on How

Petri Nets!
Specifying transients and steady states

- Possible analogies:
  - **steady state** approach
    - Logic
    - Declarative programming
  - **transient** approach
    - Process modeling
    - Procedural programming

**logic programming petri nets**

= **LPPNs**

**Petri Nets!**
Logic Programming Petri Nets
Logic Programming Petri Net (LPPN)

- An LPPN consists of three components:
  - a procedural net (places, transitions)
  - a declarative net for places
  - a declarative net for transitions

causal mechanisms

logical dependencies between objects

logical dependencies between events
Procedural LPPN
(same as Condition/Event PN)

Petri net: bipartite directed graph made of **places** (circles) and **transitions** (boxes).
Procedural LPPN
(same as Condition/Event PN)

- tokens may occupy places.
Procedural LPPN
(same as Condition/Event PN)

- Execution semantics (*token game*): if any of its input places is not occupied, the transition is **disabled**. It cannot **fire**.
Procedural LPPN
(same as Condition/Event PN)

- Execution semantics *(token game)*: if all of its input places are occupied, the transition is **enabled**. It can fire.
Procedural LPPN
(same as Condition/Event PN)

- Execution semantics (*token game*): when the transition *fires* it will *consume* tokens from the *input* places.
Procedural LPPN
(same as Condition/Event PN)

- Execution semantics (*token game*): ...and *produce* tokens in the *output* places.
For our purposes, this maps to a reactive rule (ECA):

\[ \#t1 : p1, p2 \Rightarrow -p1, -p2, +p3. \]
Declarative LPPN for places

- Constructed from the ASP program:
  
  \[ p6 \leftarrow p4, p5. \]
  
  \[ p5. \]
Declarative LPPN for places

- Equivalent to
  \[ p6 \leftarrow p4, p5. \]
  \[ p5. \]
Declarative LPPN for places

- Equivalent to
  
  \[ p6 :\Rightarrow p4, p5. \]
  
  \[ p4. p5. \]
Declarative LPPN for places

- Equivalent to
  \[ p6 \dashv \text{ AND } p4, p5. \]
  \[ p4. p5. \]

\[ \text{ entails } p4. p5. p6. \]
Declarative LPPN for transitions

- Equivalent to
  
  #t3 :- #t2, p9.
  
  #t4 :- #t2, p8.
  
  #t2. p7. p8.
Declarative LPPN for transitions

- Equivalent to
  
  \#t3 \leftarrow \#t2, p9.
  \#t4 \leftarrow \#t2, p8.

  entails

  answer set
Declarative LPPN for transitions

- Equivalent to
  
  \#t3 :- \#t2, p9.
  \#t4 :- \#t2, p8.

- Produces
  p11.

- Entails

- Answer set
while John was walking his dog, the dog ate Paul's flowers ("story")

Initial example (partial model)

dog. flower. dog-walking. #dog-eats-flower.

animal :- dog.           logical dependencies
object :- flower.        at level of objects
damage :- destruction.

#eat-object :- #dog-eats-flower. logical dependencies
#destroy-object :- #eat-object. at level of events

#destroy-object : object => -object, +destruction. causal mechanisms
Execution semantics
Execution semantics

• The paper presents two semantics:
  – a **denotational semantics**, mapping causal mechanisms to ASP using *Event Calculus* → ASP solver
Execution semantics

The paper presents two semantics:

- a denotational semantics, mapping causal mechanisms to ASP using Event Calculus
- a hybrid semantics, consisting of 4 steps:

1. solve logical dependencies of objects → ASP solver
2. select one enabled transition to fire direct computation
3. solve logical dependencies of events → ASP solver
event
4. execute the selected firing using the Petri Net direct computation
Execution semantics

- The paper presents two semantics:
  - a **denotational semantics**, mapping causal mechanisms to ASP using *Event Calculus*
  - a **hybrid semantics**, consisting of 4 steps:
    1. solve logical dependencies of objects → ASP solver
    2. select one enabled transition to fire → direct computation
    3. solve logical dependencies of events → ASP solver
    4. execute the selected firing using the Petri Net → direct computation

Question: how they compare in terms of computational performance?
Execution semantics

• The paper presents two semantics:
  – a denotational semantics, mapping causal mechanisms to ASP using Event Calculus
  – a hybrid semantics, consisting of 4 steps:
    1. solve logical dependencies of objects
    2. select one enabled transition to fire
    3. solve logical dependencies of events
    4. execute the selected firing using the Petri Net

→ ASP solver
  → direct computation

Question: how they compare in terms of computational performance? Why they should differ?
Experiment
Experiment

- We considered two basic reiterable structures at process level:
  - Serial composition (deterministic)
  - Forking (non-deterministic)

- We executed a benchmark on nets obtained by iterating these basic structures, with one token in the initial place
  - for $N$ iterations $= 1, 11, ..., 91$ (serial)
  - for $N$ iterations $= 1, 2, ..., 10$ (forking)

Code available at [http://github.com/s1l3n0/pypneu](http://github.com/s1l3n0/pypneu)
Results

Serial composition

Linear scale

Log scale
Results
Forking composition

Log scale

Linear scale
Why this difference? (intuition)
Denotational semantics: Model execution as search

- Situation Calculus, Event Calculus, Fluent Calculus all rely on some form of *timestamp*.
- Causal mechanisms are mapped to logical dependences between *timestamped snapshots*.

*Causation in model => Logical constraints*
Hybrid semantics: Model execution as execution

- Petri nets do not require to reify the global state to perform execution.
- They are directly mappable to individual instructions in imperative programs, they utilize some (local) input to produce some (local) output.

_Causation in model => Computational causation_
Conclusion

- The paper presents an empirical experiment with LPPNs, a logic programming-based extension of Petri Nets.
- LPPNs were introduced with a practical goal: a visual modelling notation, relatively simple for non-experts, handling declarative and procedural aspects of the target domain.
- Here the focus has been put on their computational properties, showing that maintaining the two levels separated has the potential to bring better performances. The benchmark needs to be extended.
Conclusion

- The paper presents an empirical experiment with LPPNs, a logic programming-based extension of Petri Nets.
- LPPNs were introduced with a practical goal: a visual modelling notation, relatively simple for non-experts, handling declarative and procedural aspects of the target domain.
- Here the focus has been put on their computational properties, showing that maintaining the two levels separated has the potential to bring better performances. The benchmark needs to be extended.
- **Future developments**: extension to predicate logic, optimization of execution model, “canonic” models