Towards a standardised and efficient implementation of the normative specification language eFLINT

Olaf Erkemeij Christopher A. Esterhuyse

Tim Müller L. Thomas van Binsbergen

September 4, 2023

UNIVERSITY OF AMSTERDAM
Informatics Institute

Old takeaway messages (e.g. from ProLaLa 2022)

*At the University of Amsterdam, the Complex Cyber Infrastructure group is experimenting with approaches to **enforcing laws**, **regulations**, **agreements** and **contracts** in (distributed) systems, in particular data exchange systems*

Old takeaway messages (e.g. from ProLaLa 2022)

At the University of Amsterdam, the Complex Cyber Infrastructure group is experimenting with approaches to ***enforcing laws****,* ***regulations****,* ***agreements*** *and* ***contracts*** *in (distributed) systems, in particular data exchange systems*

The ***eFLINT DSL*** *serves as a tool to demonstrate and experiment with various aspects of our approach relating to:*

1. *dynamic compliance: access control, runtime verification, ex-post enforcement,*
2. *compliance-by-design: static checks, formal verification, compilation*
3. *accountability: explainability, log analysis and conformance checking*

Old takeaway messages (e.g. from ProLaLa 2022)

*At the University of Amsterdam, the Complex Cyber Infrastructure group is experimenting with approaches to **enforcing laws**, **regulations**, **agreements** and **contracts** in (distributed) systems, in particular data exchange systems*

*The **eFLINT DSL** serves as a tool to demonstrate and experiment with various aspects of our approach relating to:*

1. *dynamic compliance: access control, runtime verification, ex-post enforcement,*
2. *compliance-by-design: static checks, formal verification, compilation*
3. *accountability: explainability, log analysis and conformance checking*

*These experiments highlight the importance **modularity**, **reuse**, **version control**, and **inheritance** and resulted in evolution of the language and its tools*

Old takeaway messages (e.g. from ProLaLa 2022)

At the University of Amsterdam, the Complex Cyber Infrastructure group is experimenting with approaches to ***enforcing laws***, ***regulations***, ***agreements*** *and* ***contracts*** *in (distributed) systems, in particular data exchange systems*

The ***eFLINT DSL*** *serves as a tool to demonstrate and experiment with various aspects of our approach relating to:*

1. *dynamic compliance: access control, runtime verification, ex-post enforcement,*
2. *compliance-by-design: static checks, formal verification, compilation*
3. *accountability: explainability, log analysis and conformance checking*

These experiments highlight the importance ***modularity***, ***reuse***, ***version control***, *and* ***inheritance*** *and resulted in evolution of the language and its tools*

A next phase is to improve the ***practicality and usability*** *of eFLINT*

Old takeaway messages (e.g. from ProLaLa 2022)

At the University of Amsterdam, the Complex Cyber Infrastructure group is experimenting with approaches to ***enforcing laws***, ***regulations***, ***agreements*** *and* ***contracts*** *in (distributed) systems, in particular data exchange systems*

The ***eFLINT DSL*** *serves as a tool to demonstrate and experiment with various aspects of our approach relating to:*

1. *dynamic compliance: access control, runtime verification, ex-post enforcement,*
2. *compliance-by-design: static checks, formal verification, compilation*
3. *accountability: explainability, log analysis and conformance checking*

These experiments highlight the importance ***modularity***, ***reuse***, ***version control***, *and* ***inheritance*** *and resulted in evolution of the language and its tools*

A next phase is to improve the ***practicality and usability*** *of eFLINT(in this paper:)*

- *Standardized* ***syntax*** *and* ***semantics***
- *Efficient* ***reasoners*** *and standardized interaction* ***protocol***

Example – knowledge representation

(Toy Article 1) a natural person is a legal parent of another natural person if:

- *they are a natural parent, or*
- *they are an adoptive parent*

```
Fact person Identified by String
Placeholder parent    For person
Placeholder child     For person

Fact natural-parent   Identified by parent * child
Fact adoptive-parent  Identified by parent * child

Fact legal-parent     Identified by parent * child
  Holds when adoptive-parent(parent,child)
          || natural-parent(parent,child)
```

L. Thomas van Binsbergen, Lu-Chi Liu, et al. "EFLINT: A Domain-Specific Language for Executable Norm Specifications". In: GPCE. 2020, pp. 124–136

Example – powers and duties

> *(Toy Article 2) a child has the power to ask a legal parent for help with their homework, resulting in a duty for the parent to help.*

```
Act ask-for-help
  Actor      child
  Recipient  parent
  Creates    help-with-homework(parent,child)
  Holds when legal-parent(parent,child)

Duty help-with-homework
  Holder        parent
  Claimant      child
  Violated when homework-due(child)

Fact homework-due Identified by child

Act help
  Actor      parent
  Recipient  child
  Terminates help-with-homework(parent,child)
  Holds when help-with-homework(parent,child)
```

Example – scenario

```
Fact person Identified by Alice, Bob, Chloe, David
```

Domain specification

```
+natural-parent(Alice, Bob).
+adoptive-parent(Chloe, David).
```

Initial state

```
ask-for-help(Bob, Alice).                    // action permitted, creates duty
+homework-due(Bob).                          // homework deadline passed
?Violated(help-with-homework(Alice,Bob)).    // query confirms duty is violated
help(Alice,Bob).                             // action terminates duty
```

Scenario

Foundational, normative & computational concepts

computational

state

$$\boxed{\begin{array}{l} parent(A, B) = true \\ \ldots \end{array}}$$

Foundational, normative & computational concepts

computational

state

$$\boxed{\begin{array}{l} \mathit{parent}(A, B) = \mathit{true} \\ \dots \end{array}}$$

transitions

$$\begin{tikzcd}
\boxed{\begin{array}{l} \mathit{parent}(A, B) = \mathit{true} \\ \dots \end{array}} \arrow[d, bend right] \\
\boxed{\begin{array}{l} \mathit{parent}(A, B) = \mathit{false} \\ \dots \end{array}}
\end{tikzcd}$$

Foundational, normative & computational concepts

computational

state

$$\boxed{\begin{array}{l} parent(A, B) = true \\ \ldots \end{array}}$$

transitions

$$\begin{tikzcd} \boxed{\begin{array}{l} parent(A, B) = true \\ \ldots \end{array}} \arrow[d, bend right] \\ \boxed{\begin{array}{l} parent(A, B) = false \\ \ldots \end{array}} \end{tikzcd}$$

- Violations of state and transition

Foundational, normative & computational concepts

deontic

prohibitions

permissions

obligations

computational

state

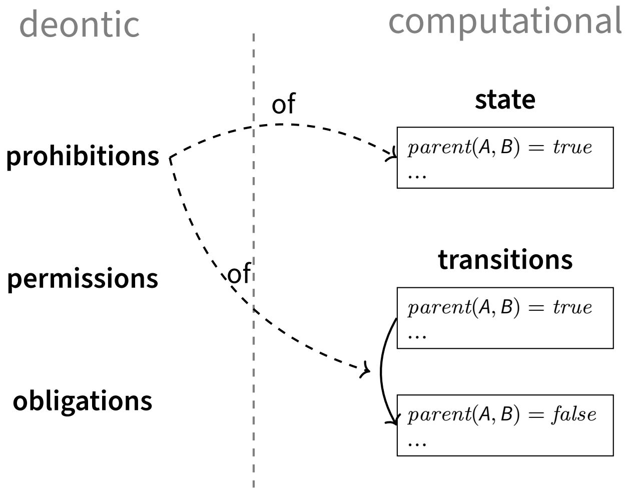
$$\boxed{\begin{array}{l} parent(A, B) = true \\ \ldots \end{array}}$$

transitions

$$\begin{tikzcd}
\boxed{\begin{array}{l} parent(A, B) = true \\ \ldots \end{array}} \arrow[d, bend right] \\
\boxed{\begin{array}{l} parent(A, B) = false \\ \ldots \end{array}}
\end{tikzcd}$$

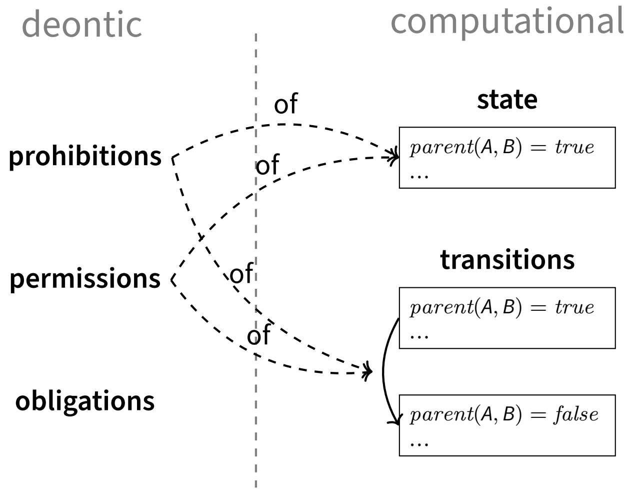
■ Violations of state and transition

Foundational, normative & computational concepts



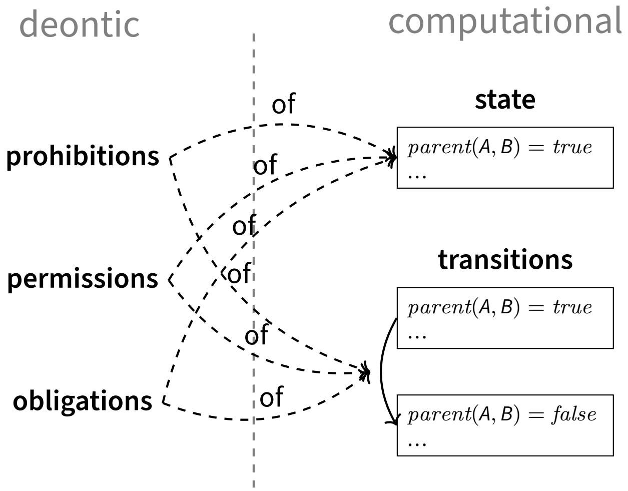
▪ Violations of state and transition

Foundational, normative & computational concepts



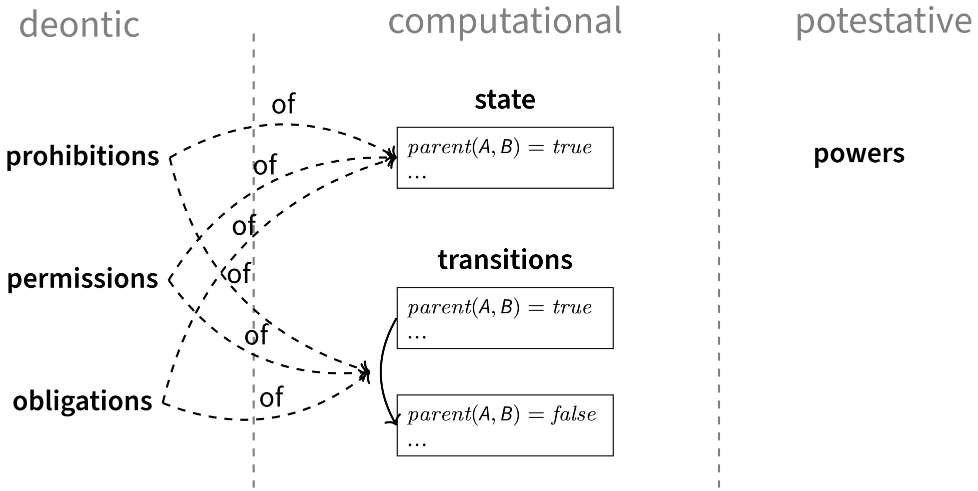
▪Violations of state and transition

Foundational, normative & computational concepts



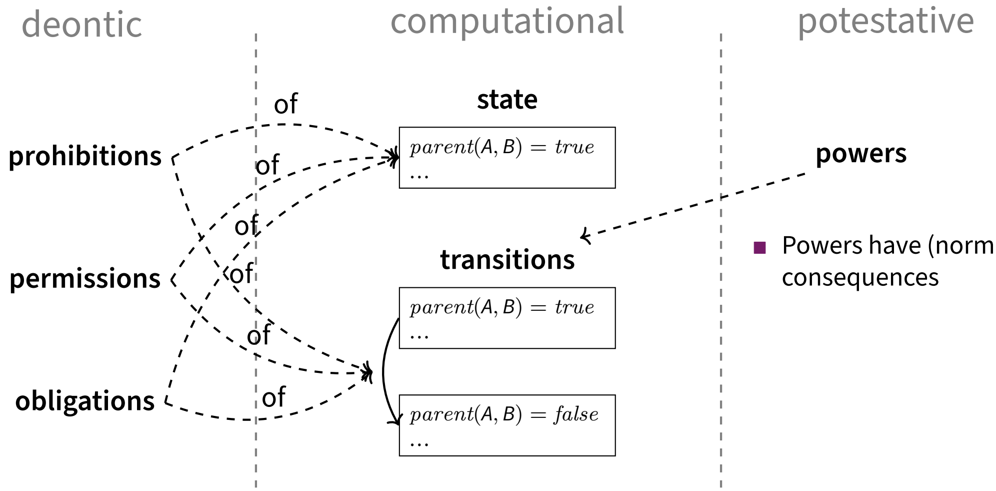
■Violations of state and transition

Foundational, normative & computational concepts



▪Violations of state and transition

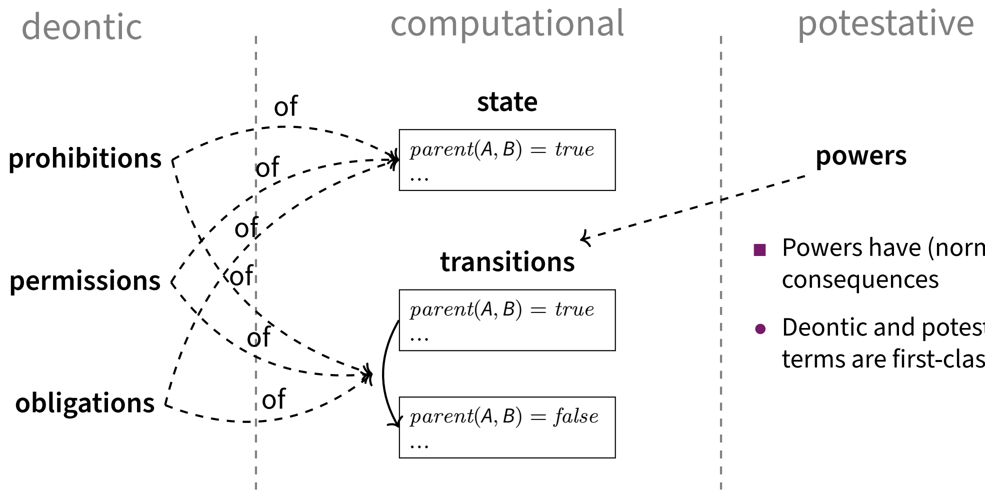
Foundational, normative & computational concepts



- Powers have (normative) consequences

- Violations of state and transition

Foundational, normative & computational concepts



- Powers have (normative) consequences
- Deontic and potestative terms are first-class

- Violations of state and transition

Incremental program development

General approach:

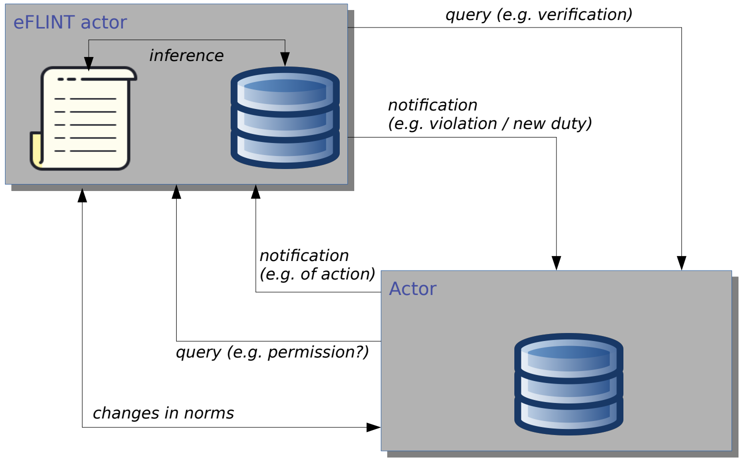
- Identify the top-level ‘phrases’ of your language (syntax)
- Define your interpreter as a (pure) transition function (semantics)

```
interpreter :: Phrase ->
                 Input ->
                 (Spec, State) -> -- current `configuration'
                   (Spec    -- holds type decl info
                   ,State   -- holds truth assignments
                   ,Output) -- triggered transitions, violations, ...
```

Top-level function of Haskell reference interpreter

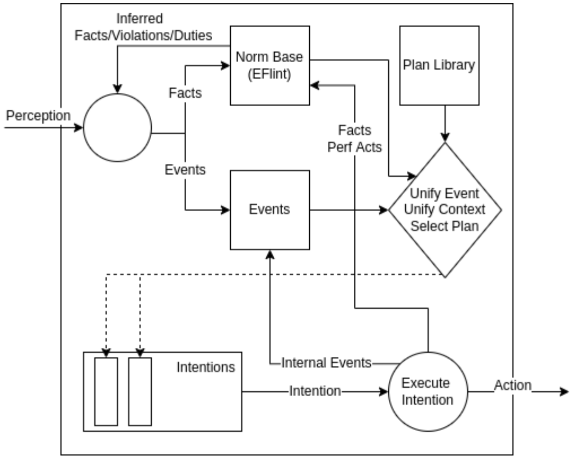
L. Thomas van Binsbergen, Mauricio Verano Merino, et al. “A Principled Approach to REPL Interpreters”. In: Onwards! 2020

Actor-oriented programming and simulations



Goals: Dynamic extensions to norms and scenario assessment

Normative multi-agent systems



An internal architecture for normative BDI agents

Mostafa Mohajeri Parizi et al. "A Modular Architecture for Integrating Normative Advisors in MAS". In: EUMAS. 2022

Specification of qualification rules

```
Extend Act make-data-available Syncs with (Foreach donor:
  collect-personal-data(controller = member
                       ,subject    = donor
                       ,data       = dataset
                       ,processor  = "DCOG"
                       ,purpose    = "DIPGResearch")
    When subject-of(donor, dataset))

Extend Act write Syncs with
  make-data-available(member, DCOG, asset) When affiliated(actor, member)
```

L. Thomas van Binsbergen et al. “Dynamic generation of access control policies from social policies”. In: ICTH (2022)

Ongoing/unpublished work

- Model checking abstract properties (Florine de Geus, MSc)
- Compilation to Solidity and other smart contract languages
- ...

Plans for IFL paper

- Standardise syntax and semantics for dynamic compliance
  - By releasing the Haskell reference interpreter
- Standardise interaction protocol for various usage scenarios
  - By releasing JSON API specification
- Experiment with alternative derivation semantics for negative antecedents
- Experiment with implementation strategies to improve runtime

Towards a standardised and efficient implementation of the normative specification language eFLINT

Olaf Erkemeij   Christopher A. Esterhuyse

Tim Müller   L. Thomas van Binsbergen

September 4, 2023

University of Amsterdam
Informatics Institute

Research questions

What approaches can be taken to develop a **scalable server-based implementation** of the eFLINT language, and how does its **performance** compare to the existing reference interpreter?

Research questions

What approaches can be taken to develop a **scalable server-based implementation** of the eFLINT language, and how does its **performance** compare to the existing reference interpreter?

- What are the **key challenges** in implementing eFLINT in a scalable server-based environment?

Research questions

What approaches can be taken to develop a **scalable server-based implementation** of the eFLINT language, and how does its **performance** compare to the existing reference interpreter?

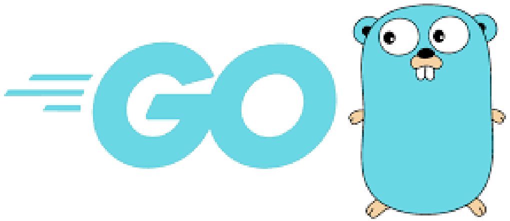
- What are the **key challenges** in implementing eFLINT in a scalable server-based environment?
- What are the **important design choices** regarding the semantics of the eFLINT language?

Research questions

What approaches can be taken to develop a **scalable server-based implementation** of the eFLINT language, and how does its **performance** compare to the existing reference interpreter?

- What are the **key challenges** in implementing eFLINT in a scalable server-based environment?
- What are the **important design choices** regarding the semantics of the eFLINT language?
- What implementation choices can be made within the eFLINT language, allowing **optimizations** in runtime and memory usage?

Technology stack



- HTTPS server through Golang
- Chosen for ease of development
  - Built-in HTTPS handler library
  - Built-in JSON library
  - Prior experience :)
- eFLINT to JSON parser in Golang

Intermediate representation

- Closely resembles JSON specification
- Automatic conversion from JSON to IR
- Store instances of facts in two maps
  - Map of instances known to be true
  - Map of instances known to be false
  - Instances are unknown if in neither map
  - Hashed instance as key, instance as value of map
  - Amortized `O(1)` lookup, addition, and removal

Interpreter control flow

Phrases

- Interpret each phrase sequentially
- After each phrase:
  - store state changes for instances
  - store violations
  - store triggers
- After interpreting all phrases:
  - Combine stored data into output format
  - Serialize output and send to the user

Expressions

- Treat each unbound variable as an implicit for-loop
- As long as a variable is present:
  1. Iterate over the instances for the variable
  2. Fill in the value for all occurrences of the variable
  3. Interpret the new expression

Expression evaluation

Derivation algorithm requires evaluation of expressions

1. Go over all derivation rules
2. Evaluate each rule and gather all results
3. Modify the program state using the results
4. Repeat until stable

```
Fact name Identified by Alice,
   Bob, Charlie.
Fact person Identified by String
            Derived from name.
```

This approach often requires many iterations

- Can take a long time
- Inefficient

Expression evaluation

The new interpreter supports on-the-fly state modifications

- During derivation process, each evaluation result can immediately be used to modify the state
- Can only be used during derivation process
- Cuts down on iterations needed

```
Fact name Identified by Alice, Bob, Charlie.
Fact person Identified by String
            Derived from name.
```

Derivation logic

1. $p \leftarrow q$
2. $q \leftarrow r$
3. $r \leftarrow s$
4. $s \leftarrow \top$

A set of derivation rules.

Haskell eFLINT interpreter supports derivation logic

- Greedily tries to apply rules
- Repeatedly considers each rule until stable
- However, eFLINT *syntax* also supports negative literals

Derivation logic

1. $p \leftarrow \neg q$
2. $q \leftarrow p$
3. $a \leftarrow \neg b$

A set of derivation rules containing a circular dependency.

- Derivations involving negated literals
- eFLINT takes a greedy approach
- Could end up with `{p, q, a}` as knowledge base
- What is the desired behaviour?

Derivation logic

Property (Satisfaction)

Each given rule is satisfied, i.e., if each of its conditions are true in the model, then its consequent is true in the model.

Property (Explainability)

Each true fact was either postulated true, or is derived by a given rule which is satisfied.

Property (Consistency)

Each model attributes a unique, Boolean truth value to each fact.

- Derivation model can conform to 3 properties
- Current approach breaches property 2
- Other approaches fulfil other properties
    - Stable model semantics
    - Well founded semantics
    - Simply don’t allow negative literals

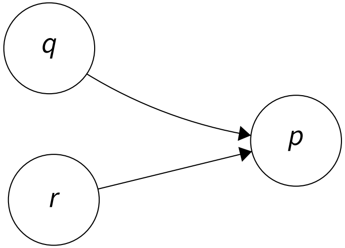
Modified derivation logic

dependency graph

Store dependencies between derivation rules in a dependency graph

1. $p \leftarrow \neg q,\ r$
2. $q \leftarrow \quad p$
3. $a \leftarrow \neg b,\ p$

A set of derivation rules containing a circular dependency.

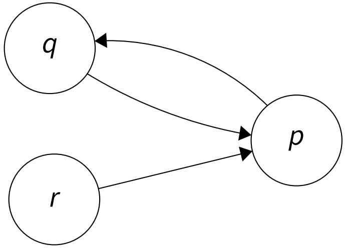


Modified derivation logic

dependency graph

Store dependencies between derivation rules in a dependency graph

1. $p \leftarrow \neg q,\ r$
2. $q \leftarrow \quad p$
3. $a \leftarrow \neg b,\ p$



A set of derivation rules containing a circular dependency.

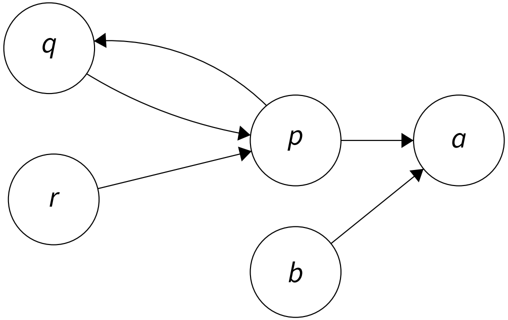
Modified derivation logic

dependency graph

Store dependencies between derivation rules in a dependency graph

1. $p \leftarrow \neg q,\ r$
2. $q \leftarrow \quad p$
3. $a \leftarrow \neg b,\ p$

A set of derivation rules containing a circular dependency.



Modified derivation logic

Keep track of ‘assumptions’

- Assume any unknown negated literal to be false
- Store assumption alongside current state of the algorithm
- Backtrack to stored state when contradicting the assumption

Modified derivation logic

Keep track of 'assumptions'

- Assume any unknown negated literal to be false
- Store assumption alongside current state of the algorithm
- Backtrack to stored state when contradicting the assumption

Weakens property 1, preserves the others

Property (Satisfaction)

Each given rule is satisfied, i.e., if each of its conditions are true in the model, then its consequent is true in the model.

Modified derivation logic

1. $r \leftarrow \top$
2. $p \leftarrow \neg q,\ r$
3. $q \leftarrow \quad p$
4. $a \leftarrow \neg b,\ p$

A set of derivation rules containing a circular dependency.

1. Derive *r*: `{r}`
2. Assume $\neg q$, derive *p*: `{r, p}`
3. Know *p*, derive *q*
4. Assumption violated, backtrack
5. Do not assume $\neg q$: `{r}`

Correctness

- 42 files tested
- 41 passes, 1 failure
- 97.6% success rate

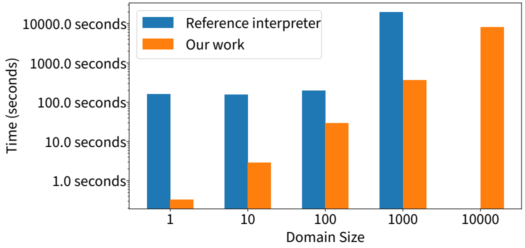
```
Event a
    Syncs with b().

Event b
    Syncs with a().

a().
```

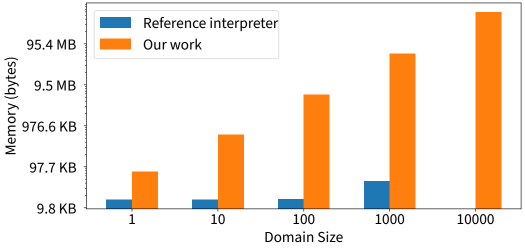
The failing test case: infinite loop

Performance (1 / 2)

Derivation rules

```
Fact x Identified by 1..<size> Holds when x(x - 1).
+x(1).
```

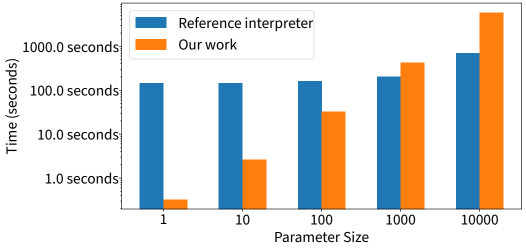


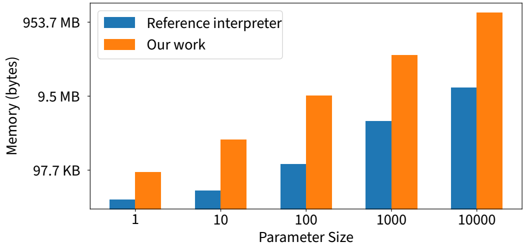


Performance (2 / 2)

Composite type dimensionality size

```
Fact parameter Identified by 1..10
Fact combined Identified by parameter1 * parameter2 * ...
?-combined.
```





Contributions

- Created a new interpreter for the eFLINT language
- Implemented a modified, modular derivation procedure
  - Can test different derivation procedures
  - Server can easily switch between procedures
- Achieved execution time speed-up in the new interpreter
- Worked on releasing a stable version of the JSON specification

Conclusions

Research questions

- Challenges:
  - Conversion of eFLINT code to JSON and IR and back
  - Derivation procedure implementation
  - Unbound variables
- Important design choices:
  - Internal representation
  - Allowing negative literals
  - Allowing unknown values
- Optimizations performed by:
  - Using hashing for efficient storage
  - Optimizing derivation process through dependency graphs

NWO-funded: SSPDDP – Secure and scalable, policy-driven data exchange

NWO-funded: DL4LD – Data Logistics for Logistics Data

EFRO-funded: AMDEX Fieldlab – neutral data-exchange infrastructure

Towards a standardised and efficient implementation of the normative specification language eFLINT

Olaf Erkemeij

Christopher A. Esterhuyse

Tim Müller

L. Thomas van Binsbergen

September 4, 2023

University of Amsterdam
Informatics Institute