Reflections on the design and application of eFLINT

L. Thomas van Binsbergen

Informatics Institute, University of Amsterdam
ltvanbinsbergen@acm.org

January 16, 2022
Programming Languages and the Law (ProLaLa)

UNIVERSITY OF AMSTERDAM

Contributors

L. Thomas van Binsbergen

Assistant Professor

Giovanni Sileno

Postdoctoral Researcher

Tom van Engers

Full Professor (FDR)

Lu-Chi Liu

PhD Candidate

Milen Girma Kebede

PhD Candidate

Mostafa Mohajeri Parizi

PhD Candidate

Christopher Esterhuyse

PhD Candidate

Damian Frolich

PhD Candidate

Regulated data exchange:

data exchange systems governed by regulations, agreements and policies

as an instance of

Regulated systems:

distributed software systems with embedded regulatory services derived from norm specifications that monitor and/or enforce compliance

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

Regulated data exchange:

data exchange systems governed by regulations, agreements and policies

as an instance of

Regulated systems:

distributed software systems with embedded regulatory services derived from norm specifications that monitor and/or enforce compliance

Regulated data exchange:

data exchange systems governed by regulations, agreements and policies

as an instance of

Regulated systems:

distributed software systems with embedded regulatory services derived from norm specifications that monitor and/or enforce compliance

NWO-funded: DL4LD – Data Logistics for Logistics Data

TKI DINALOG · NWO · Commit 2 Data · THALES · BiZZdesign · ciena · evofenedex · Gemeente Amsterdam · AIRFRANCE KLM · TRANSFIDES · ORACLE · Universiteit van Amsterdam · TNO innovation for life

Overview

1. Relating normative and computational concepts

2. The eFLINT language
   - eFLINT 1.0
   - eFLINT 2.0

3. Reflections
   - Goals for eFLINT 3.0

Section 1

Relating normative and computational concepts

Foundational, normative & computational concepts

computational

state

$parent(A, B) = true$
$\ldots$

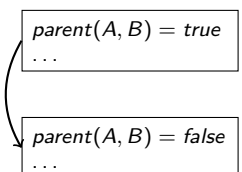
Foundational, normative & computational concepts

computational

state

$parent(A, B) = true$
...

transitions



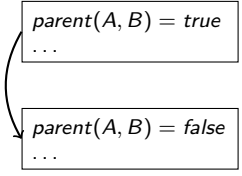
Foundational, normative & computational concepts

computational

state

$parent(A, B) = true$
...

transitions



- Violations of state and transition

Foundational, normative & computational concepts

deontic

prohibitions

permissions

obligations

computational

state

$parent(A, B) = true$
...

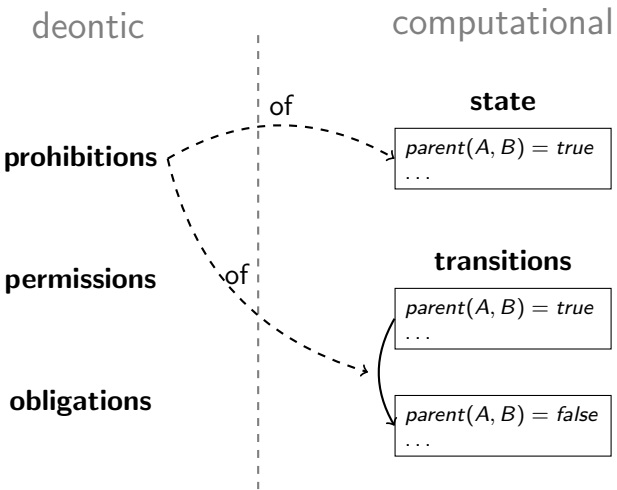
transitions

$parent(A, B) = true$
...

$parent(A, B) = false$
...

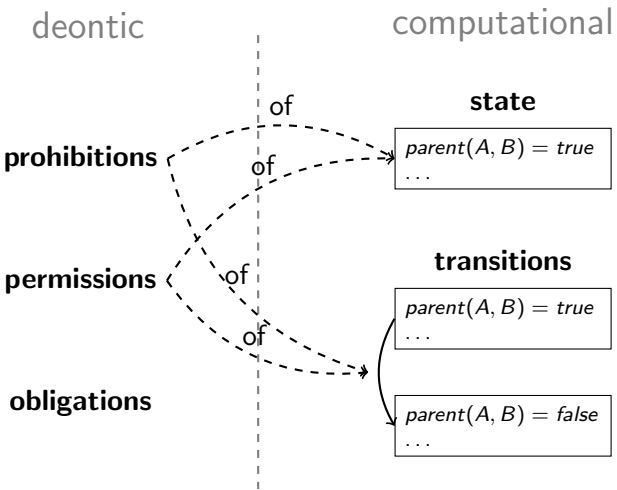
- Violations of state and transition

Foundational, normative & computational concepts



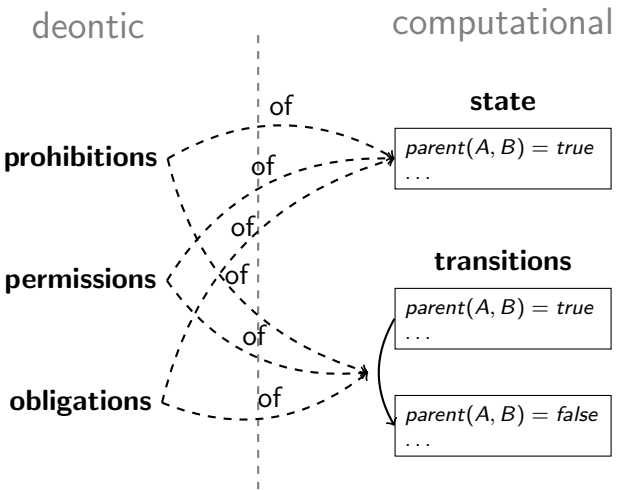
- Violations of state and transition

Foundational, normative & computational concepts



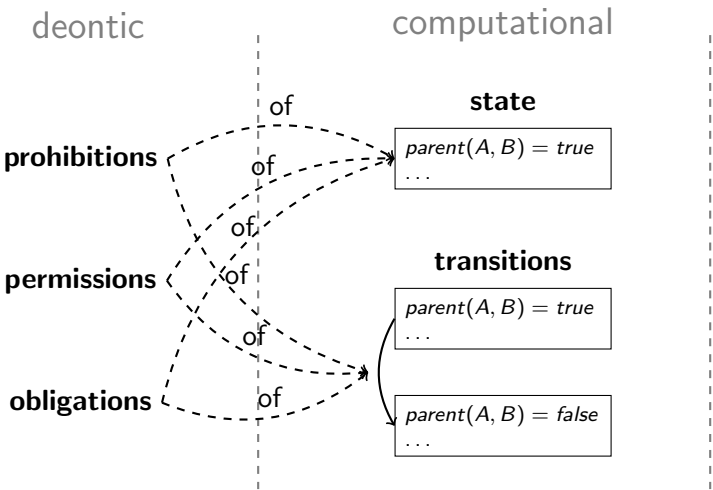
- Violations of state and transition

Foundational, normative & computational concepts



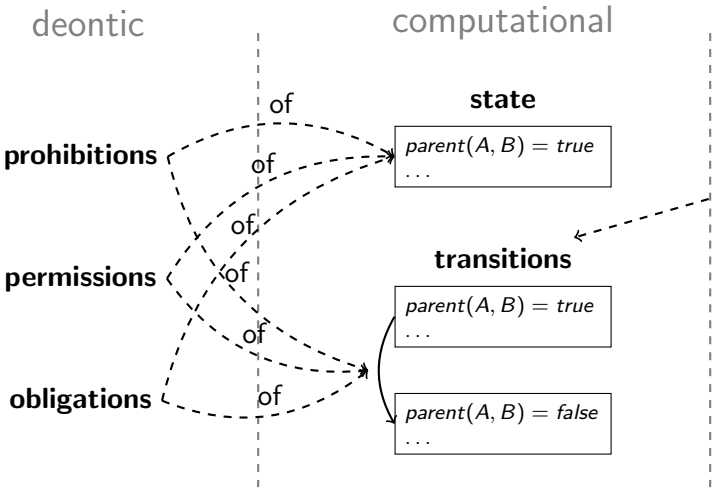
- Violations of state and transition

Foundational, normative & computational concepts

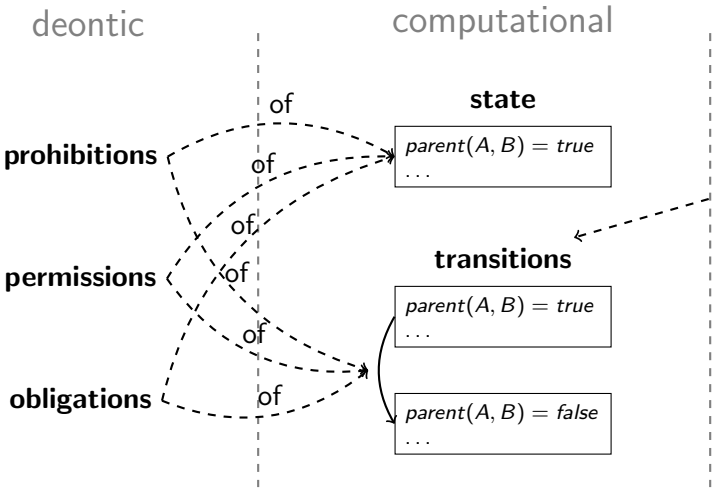


- Violations of state and transition

Foundational, normative & computational concepts



Foundational, normative & computational concepts



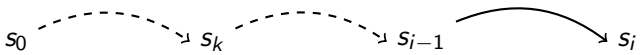
- Powers have (normative) consequences
- Deontic and potestative terms are first-class

Normative relations between actors

- A deontic term is associated with several actors:
  - The **holder** of the prohibition, obligation or prohibition
  - Zero or more **claimants** to the prohibition or obligation
  - The actor who **assigned** the prohibition, obligation or permission

- A potestative term is associated with several actors
  - The **performing** actor
  - One or more **recipients** being affected by the power
  - The actor who **assigned** the power

Normative reasoning – scenarios



Types of reasoning

A concrete **scenario** describes a single trace in the transition system

- Static ex-ante/ex-post assessment, of a given scenario (trace) {eFLINT 1.0}
- Dynamic assessment, of a given action (transition) {eFLINT 2.0 }
  - ex-post: *execute* corresponding transition and report on findings
  - ex-ante: *try* the transition and decide based on report whether to perform the action
- We are not (yet) reasoning with abstract scenarios (e.g. planning/property checking)

Section 2

The eFLINT language

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)
```

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
```

Listing 1: Domain specification

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

Listing 2: 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
```

Listing 3: Scenario

eFLINT online!

frames

```
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)

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

Fact homework-due Identified by child

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

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

domains

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

initial state

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

Examples

Knowledge representation: Vehicles | Departments | Count Votes | Cast Votes

GPCE2020 paper examples: Help with homework | GDPR

Various: Buyer/Seller (v1) | Buyer/Seller (v2) | Buyer/Seller (v3) | Permit Applications | Permit Applications (v2) | Multiple taxpayers | Voting

Load file: Browse... No file selected.

scenario

```
ask-for-help(Bob, Alice).
+homework-due(Bob).  // homework deadline passed
?Violated(help-with-homework(Alice,Bob)).
help(Alice,Bob).
```

Run | Reset | Save | model name

response

```
* Duty violated at step 2
    ("Alice":person,"Bob":person):help-with-homework
```

output

Step 0: initial state

Step 1: ("Bob":person,"Alice":person):ask-for-help
  +("Alice":person,"Bob":person):help-with-homework

Step 2: ("Bob":person):homework-due
  +("Bob":person):homework-due

Step 3: query

Step 4: ("Alice":person,"Bob":person):help
  -("Alice":person,"Bob":person):help-with-homework

eFLINT 2.0

1. Extensions to the eFLINT syntax
    - The `Extends` keyword to modularly extend existing declarations
      Enables rule-based formalisation of articles and proper separation across files
    - The `Syncs with` keyword to trigger multiple transitions simultaneously
      Enables the qualification of one action as an instance of another

Modular GDPR specification

Dynamic generation of access control policies from social policies

L. Thomas van Binsbergen[1,a], Milen G. Kebede[a], Joshua Baugh[b], Tom van Engers[a], Dannis G. van Vuurden[b]

[a]*Informatics Institute, University of Amsterdam, 1090GH Amsterdam, The Netherlands*
[b]*Princess Maxima Center for Pediatric Oncology, Department of Neuro-oncology, Utrecht, The Netherlands*

```
Act collect-personal-data
  Actor controller
  Recipient subject
  Related to data, processor, purpose
    Where subject-of(subject, data)
  Creates processes(processor, data, controller, purpose)
```

Article 5 – processing conditions

Article 5

Principles relating to processing of personal data

1. Personal data shall be:

(a) processed lawfully, fairly and in a transparent manner in relation to the data subject ('lawfulness, fairness and transparency');

(b) collected for specified, explicit and legitimate purposes and not further processed in a manner that is incompatible with those purposes; further processing for archiving purposes in the public interest, scientific or historical research purposes or statistical purposes shall, in accordance with Article 89(1), not be considered to be incompatible with the initial purposes ('purpose limitation');

(c) adequate, relevant and limited to what is necessary in relation to the purposes for which they are processed ('data minimisation');

(d) accurate and, where necessary, kept up to date; every reasonable step must be taken to ensure that personal data that are inaccurate, having regard to the purposes for which they are processed, are erased or rectified without delay ('accuracy');

```
Fact minimal-for-purpose Identified by processes
Extend Act collect-personal-data Conditioned by minimal-for-purpose(data, purpose)
```

Listing 4: Member (1c)

```
Fact accurate-for-purpose Identified by data * purpose
Extend Act collect-personal-data Conditioned by accurate-for-purpose(data, purpose)
```

Listing 5: Member (1d)

Article 6 – legal processing

Article 6

Lawfulness of processing

1. Processing shall be lawful only if and to the extent that at least one of the following applies:

(a) the data subject has given consent to the processing of his or her personal data for one or more specific purposes;

(b) processing is necessary for the performance of a contract to which the data subject is party or in order to take steps at the request of the data subject prior to entering into a contract;

(c) processing is necessary for compliance with a legal obligation to which the controller is subject;

```
Fact consent Identified by subject * controller * purpose * data
Extend Act collect-personal-data
  Holds when consent(subject, controller, purpose, data)
```

Listing 6: Member (1a)

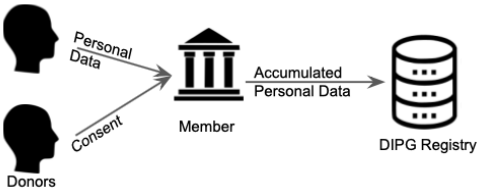
```
Fact has-legal-obligation Identified by processes
Extend Act collect-personal-data
  Holds when has-legal-obligation(controller, purpose)
```

Listing 7: Member (1c)

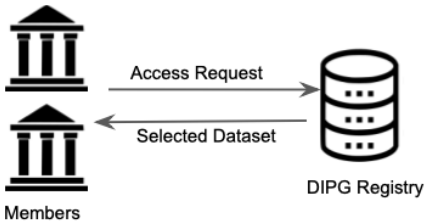
The DIPG case – Compliance questions

According to the GDPR and the DIPG regulatory document:

1. What conditions need to be fulfilled by a member before making data available?



2. What conditions need to be fulfilled when accessing data from the registry?



Compliance Question 1

DIPG Regulatory document – Article 4(2):

Members should transfer data to the DIPG registry in a coded form only

```
Fact coded Identified by dataset

Act make-data-available
  Actor institution
  Recipient dcog
  Related to dataset
  Conditioned by coded(dataset)
  Holds when member(institution)
```

Compliance Question 1

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

An institution can make a dataset available when (for each donor (subject) in the dataset):

- The institution is a member (DIPG Regulatory Document – Article 4(2))
- Data is coded (DIPG Regulatory Document – Article 4(2))
- Consent is given by the donor for data processing by the DCOG for the purpose of DIPGResearch (GDPR – Article 6)
- Data should be accurate for the purpose DIPGResearch (GDPR – Article 5)

Section 3

Reflections

Bounded vs open-ended domains

Static analyses

- eFLINT 1.0 enabled automated assessment of concrete scenarios in finite domain
- Future work: applying model checking, and/or property-based testing

Dynamic enforcement

- eFLINT 2.0 enabled dynamic interpretation, qualification and assessment
- Norms and scenario established at runtime, based on the contents of the knowledge base

Bounded vs open-ended domains

Static analyses

- eFLINT 1.0 enabled automated assessment of concrete scenarios in finite domain
- Future work: applying model checking, and/or property-based testing

Dynamic enforcement

- eFLINT 2.0 enabled dynamic interpretation, qualification and assessment
- Norms and scenario established at runtime, based on the contents of the knowledge base

Design decision: *when enumerating instances, check domain of type, then knowledge base*

```
?(Forall person: Not(homework-due(person)))

// opt1: Fact person Identified by Alice, Bob, Chloe, David
// opt2: Fact person Identified by String.
         +person(Alice). +person(Bob). +person(Chloe). +person(David).
```

Bounded vs open-ended domains

Static analyses

- eFLINT 1.0 enabled automated assessment of concrete scenarios in finite domain
- Future work: applying model checking, and/or property-based testing

Dynamic enforcement

- eFLINT 2.0 enabled dynamic interpretation, qualification and assessment
- Norms and scenario established at runtime, based on the contents of the knowledge base

Design decision: *when enumerating instances, check domain of type, then knowledge base*

```
?(Forall person: Not(homework-due(person)))

// opt1: Fact person Identified by Alice, Bob, Chloe, David
// opt2: Fact person Identified by String.
         +person(Alice). +person(Bob). +person(Chloe). +person(David).
```

Design decision: *the effects of actions* **can** *manifest independent of conditions*

eFLINT 2.0 – Enabling dynamic assessment

1. Extensions to the eFLINT syntax
   - Declarations and statements can be mixed freely
     Enables dynamic scenario and dynamic policy construction in **eflint actors**
   - The `Extends` keyword to modularly extend existing declarations
     Enables rule-based formalisation of articles and proper separation across files
   - The `Syncs with` keyword to trigger multiple transitions simultaneously
     Enables the qualification of one action as an instance of another

eFLINT 2.0 – Enabling dynamic assessment

1. Extensions to the eFLINT syntax
   - Declarations and statements can be mixed freely
     Enables dynamic scenario and dynamic policy construction in **eflint actors**
   - The `Extends` keyword to modularly extend existing declarations
     Enables rule-based formalisation of articles and proper separation across files
   - The `Syncs with` keyword to trigger multiple transitions simultaneously
     Enables the qualification of one action as an instance of another
   - `Open` and `Closed` modifiers in a (fact-)type declaration
     Akin to dynamic binding; enables externalised state, and a means to open terms

eFLINT 2.0 – Enabling dynamic assessment

1. Extensions to the eFLINT syntax
   - Declarations and statements can be mixed freely
     Enables dynamic scenario and dynamic policy construction in **eflint actors**
   - The `Extends` keyword to modularly extend existing declarations
     Enables rule-based formalisation of articles and proper separation across files
   - The `Syncs with` keyword to trigger multiple transitions simultaneously
     Enables the qualification of one action as an instance of another
   - `Open` and `Closed` modifiers in a (fact-)type declaration
     Akin to dynamic binding; enables externalised state, and a means to open terms
2. Changes to eFLINT semantics:
   - Actions manifest their effects, independent of conditions
     Violations are still reported

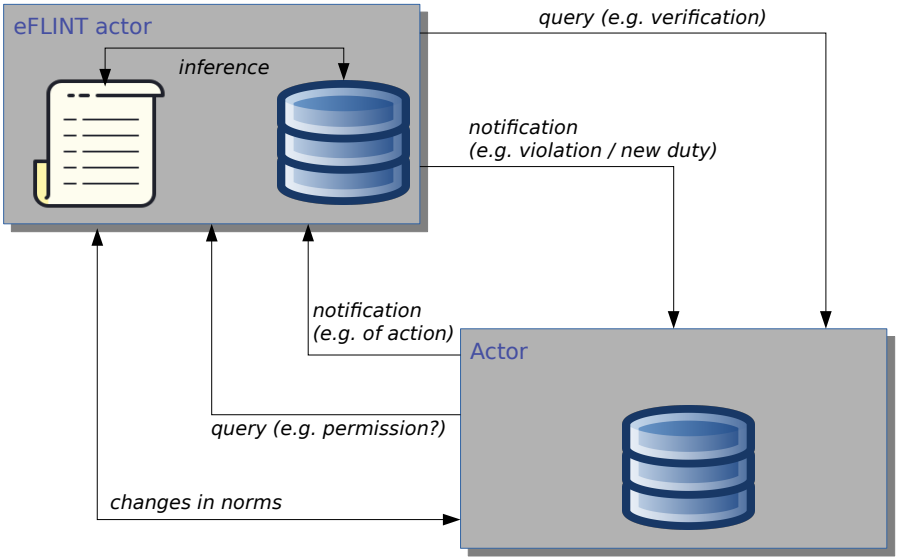
eFLINT 2.0 – Enabling dynamic assessment

1. Extensions to the eFLINT syntax
   - Declarations and statements can be mixed freely
     Enables dynamic scenario and dynamic policy construction in **eflint actors**
   - The `Extends` keyword to modularly extend existing declarations
     Enables rule-based formalisation of articles and proper separation across files
   - The `Syncs with` keyword to trigger multiple transitions simultaneously
     Enables the qualification of one action as an instance of another
   - `Open` and `Closed` modifiers in a (fact-)type declaration
     Akin to dynamic binding; enables externalised state, and a means to open terms
2. Changes to eFLINT semantics:
   - Actions manifest their effects, independent of conditions
     Violations are still reported
   - Iteration of unbounded types based on knowledge base contents
     Enables reusing specifications for both static and dynamic assessment

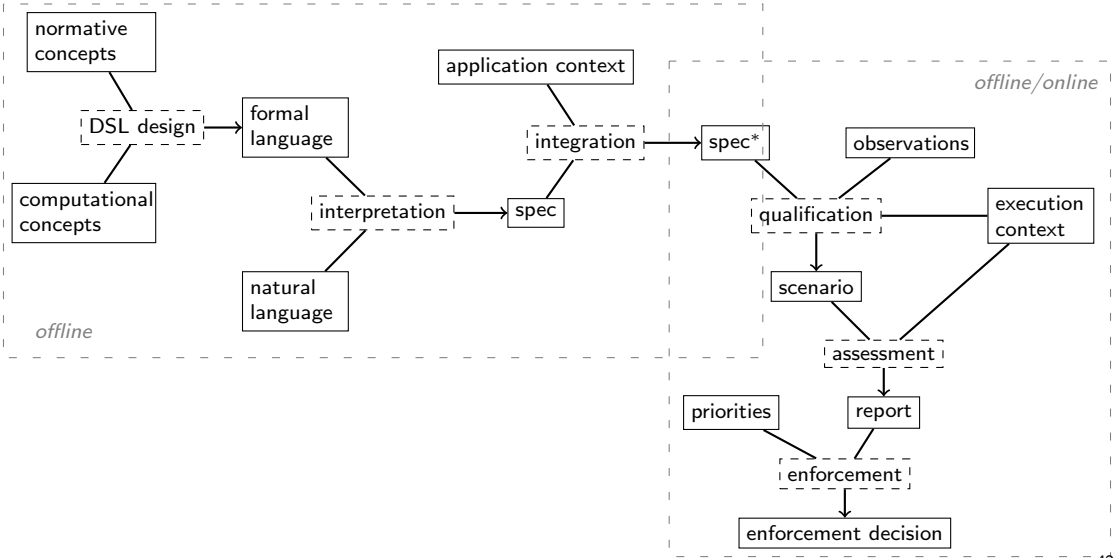
eFLINT 2.0 – Enabling dynamic assessment

1. Extensions to the eFLINT syntax
   - Declarations and statements can be mixed freely
     Enables dynamic scenario and dynamic policy construction in **eflint actors**
   - The `Extends` keyword to modularly extend existing declarations
     Enables rule-based formalisation of articles and proper separation across files
   - The `Syncs with` keyword to trigger multiple transitions simultaneously
     Enables the qualification of one action as an instance of another
   - `Open` and `Closed` modifiers in a (fact-)type declaration
     Akin to dynamic binding; enables externalised state, and a means to open terms
2. Changes to eFLINT semantics:
   - Actions manifest their effects, independent of conditions
     Violations are still reported
   - Iteration of unbounded types based on knowledge base contents
     Enables reusing specifications for both static and dynamic assessment
   - Types can be replaced and extended
     Enables reusing high-level specifications across varying application and execution contexts

eFLINT actors



Normative reasoning – information flow



State of development

1. `haskell-implementation`
    - Reference implementation of eFLINT DSL
    - `eflint-repl`: interpreter (debugging, running scenarios and tests)
    - `eflint-server`: TCP server (dynamic assessment)
    - Formal syntax / semi-formal operational semantics

State of development

1. `haskell-implementation`
   - Reference implementation of eFLINT DSL
   - `eflint-repl`: interpreter (debugging, running scenarios and tests)
   - `eflint-server`: TCP server (dynamic assessment)
   - Formal syntax / semi-formal operational semantics
2. `java-implementation`
   - TCP client
   - HTTP server
   - rudimentary EDSL for accessing `eflint-server`

State of development

1. `haskell-implementation`
   - Reference implementation of eFLINT DSL
   - `eflint-repl`: interpreter (debugging, running scenarios and tests)
   - `eflint-server`: TCP server (dynamic assessment)
   - Formal syntax / semi-formal operational semantics
2. `java-implementation`
   - TCP client
   - HTTP server
   - rudimentary EDSL for accessing `eflint-server`
3. `scala-implementation`
   - eFLINT actors in the actor-oriented Akka framework

State of development

1. `haskell-implementation`
   - Reference implementation of eFLINT DSL
   - `eflint-repl`: interpreter (debugging, running scenarios and tests)
   - `eflint-server`: TCP server (dynamic assessment)
   - Formal syntax / semi-formal operational semantics
2. `java-implementation`
   - TCP client
   - HTTP server
   - rudimentary EDSL for accessing `eflint-server`
3. `scala-implementation`
   - eFLINT actors in the actor-oriented Akka framework
4. Development environments
   - Jupyter notebooks
   - Various experimental web-applications
   - FLINT editor

Goals for eFLINT 3.0

Language design

- Clear separation between:
  - Computational concepts: actions, events, synchronisation
  - Normative concepts: prohibition, obligation, permission, power
- (eFLINT 2.0 can serve as a core/inner language to eFLINT 3.0)
- A module system, introducing namespaces and a versioning mechanism
- Modular, rule-based specification as the default through implicit extensions

Language engineering

- Additional static analyses to detect inconsistencies and possible errors
- Detailed reports as part of reasoning output to improve explainability
- User-friendly programming environment for writing and testing specifications
- Interoperability, e.g. with linked data / semantic web

FAQ

eFLINT is just defining a transition system with some extra conditions lying on top

– PL expert

Law is subject to interpretation and has (deliberate) open terms

– Legal expert

eFLINT is still too difficult to use

– Legal expert

Takeaway messages

At the University of Amsterdam, we are experimenting with approaches to enforcing laws, regulations, agreements and contracts in (distributed) systems

Takeaway messages

At the University of Amsterdam, we are experimenting with approaches to enforcing laws, regulations, agreements and contracts in (distributed) systems

The eFLINT DSL serves as a tool to demonstrate and experiment with various aspects of our approach, with a focus on runtime enforcement using 'regulatory services'

Takeaway messages

At the University of Amsterdam, we are experimenting with approaches to enforcing laws, regulations, agreements and contracts in (distributed) systems

The eFLINT DSL serves as a tool to demonstrate and experiment with various aspects of our approach, with a focus on runtime enforcement using 'regulatory services'

We are currently working on a prototype to demonstrate our approach in data exchange systems such as the Amsterdam Data Exchange (AMdEX)

Takeaway messages

At the University of Amsterdam, we are experimenting with approaches to enforcing laws, regulations, agreements and contracts in (distributed) systems

The eFLINT DSL serves as a tool to demonstrate and experiment with various aspects of our approach, with a focus on runtime enforcement using 'regulatory services'

We are currently working on a prototype to demonstrate our approach in data exchange systems such as the Amsterdam Data Exchange (AMdEX)

These experiments highlight the importance of software engineering concepts such as modularity, reuse, version control, overriding/overloading mechanisms and inheritance

Takeaway messages

At the University of Amsterdam, we are experimenting with approaches to enforcing laws, regulations, agreements and contracts in (distributed) systems

The eFLINT DSL serves as a tool to demonstrate and experiment with various aspects of our approach, with a focus on runtime enforcement using 'regulatory services'

We are currently working on a prototype to demonstrate our approach in data exchange systems such as the Amsterdam Data Exchange (AMdEX)

These experiments highlight the importance of software engineering concepts such as modularity, reuse, version control, overriding/overloading mechanisms and inheritance

The next phase is to improve the practicality and usability of eFLINT through higher-level abstractions, (domain-specific) editors, static analyses, and explainability

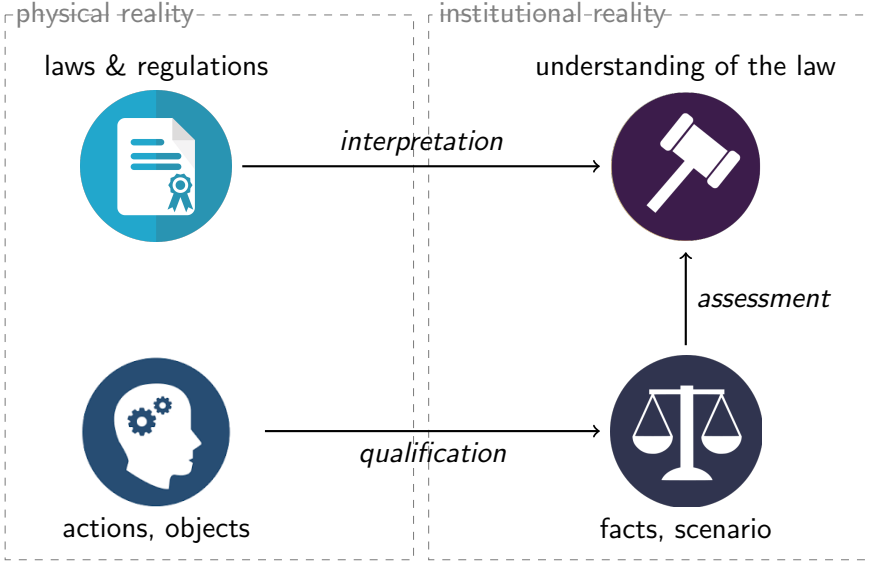
Reflections on the design and application of eFLINT

L. Thomas van Binsbergen

Informatics Institute, University of Amsterdam
ltvanbinsbergen@acm.org

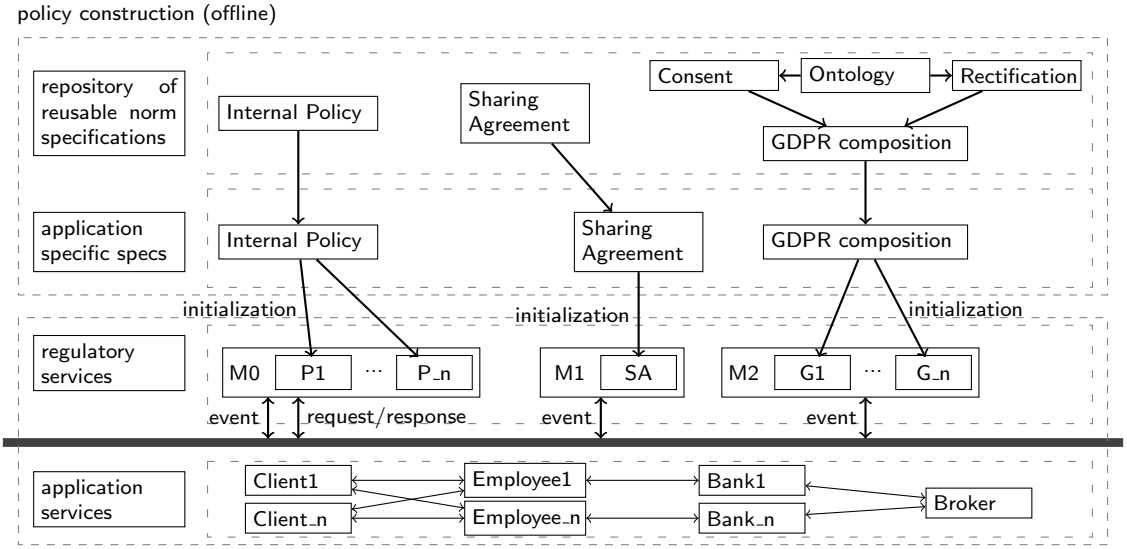
January 16, 2022
Programming Languages and the Law (ProLaLa)

UNIVERSITY OF AMSTERDAM

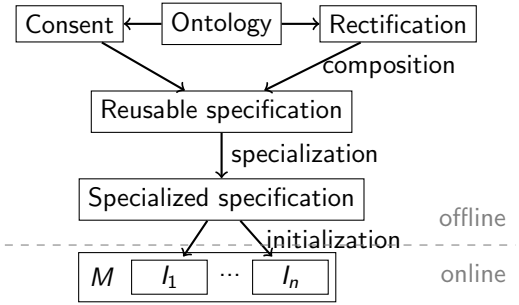


"If the facts are against you, argue the law. If the law is against you, argue the facts. If the law and the facts are against you, pound the table ..." -Carl Sandburg

Regulated systems for a banking case study



eFLINT integration – overview (GDPR example)



eFLINT integration – example

Reusable GDPR concepts

```
Fact controller
Fact subject

Fact data
Fact subject-of
 Identified by subject * data
```

Specialisation to application

```
Fact bank   //exactly one
Fact client //exactly one

Fact controller
 Derived from bank
Fact subject
 Derived from client

Fact data
 Identified by Int

Event data-change
 Terminates data
 Creates data(data + 1)

Fact subject-of
 Derived from
  subject-of(client,processed)
 ,subject-of(client,data)

Fact processed
 ...
```

Instantiation at run-time

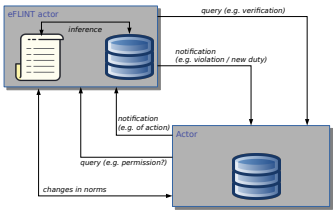
```
+bank(GNB).
+client(Alice).
+data(0).
```

Derived after instantiation

```
+controller(GNB).
+subject(Alice).
+subject-of(Alice,0).
```

Two approaches to enforcing norms

Embedding eFLINT specifications as eFLINT actors, akin to ‘policy decision point’:



Generating system-level policies, akin to ‘policy administration point’

Dynamic generation of access control policies from social policies

L. Thomas van Binsbergen[1,a], Milen G. Kebede[a], Joshua Baugh[b], Tom van Engers[a], Dennis G. van Vuurden[b]

[a]*Informatics Institute, University of Amsterdam, 1090GH Amsterdam, The Netherlands*
[b]*Princess Maxima Center for Pediatric Oncology, Department of Neuro-oncology, Utrecht, The Netherlands*