Tutorial on Norm Synthesis in Normative Multi-Agent Systems

Maite López-Sánchez
maite_lopez@ub.edu

Volume Visualization and Artificial Intelligence Research Group (WAI)
Dept Matemàtica Aplicada i Anàlisi (MAiA), Facultat de Matemàtiques
Universitat de Barcelona (UB)
Tutor:
Dr. Maite López-Sánchez
University of Barcelona
• Teaching material based on
  – SOAS subject from the interuniversitary master on Artificial Intelligence (UPC-UB-URV) http://www.fib.upc.edu/en/masters/mai.html
  – Related Research papers
  – Co-authored research work:
    • Ph.D. students: Eva Bou, Jordi Campos, Javier Morales and Master students: Patricio Petruzzi, Pedro Avila, David Sanchez, Iosu Mendizábal. Co-supervised with: Dr. Juan Antonio Rodríguez- Aguilar and Dr. Marc Esteva (IIIA-CSIC)
    • Research collaborations: Dr. Jaime S. Sichman (Univ. Sao Paulo), Dr. Wamberto Vasconcelos (Univ. of Abeerdeen), Prof. Michael Wooldridge (Univ. of Oxford).
• Tutorial material available online at:
  — Tutorial slides:
    • http://www.maia.ub.es/~maite/Teaching.html
  — On-line Norm Synthesis source code:
    http://normsynthesis.github.io/NormLabSimulators/
    http://normsynthesis.github.io/NormSynthesisMachine/
1. Introduction to Normative MAS

2. On-line automatic norm synthesis.

3. Demo and hands-on activity.
1. Introduction to Normative MAS
   – 30’
2. On-line automatic norm synthesis.
   – 30’
3. Demo and hands-on activity.
   – 60’
Tutorial Outline

Objectives

1. Introduction to Normative MAS
   - Consider design questions
2. On-line automatic norm synthesis.
   - Learn one approach
3. Demo and hands-on activity.
   - Get familiar with a framework
   - Put it in practice
1. **Introduction to Normative MAS and norm synthesis approaches.**
   - Off-line norm synthesis.
   - Norm emergence
   - Other

2. **On-line automatic norm synthesis.**

3. **Demo and hands-on activity**
• Coordination by norms and social laws:
  – In our everyday lives, we use a range of techniques for **coordinating activities**. One of the most important is the use of norms and social laws (Lewis, 1969).
Norm categories

Tuomela:
- Rule norms (e.g. to pay taxes),
- Social norms (e.g. not litter),
- Moral norms: (e.g. not steal),
- Prudential norms: (e.g. max. expected utility).

Elster:
- Consumption norms (e.g. manners of dress),
- Behaviour norms (e.g. the norm against cannibalism),
- Norms of reciprocity (e.g. gift-giving norms),
- Norms of cooperation (e.g. voting and tax compliance).
A norm is an **established, expected pattern of behaviour** (Wooldridge).

- May not be enforced
- Related to authority
A norm is an *established, expected pattern of behaviour* (Wooldridge).

- May not be enforced
- Related to authority

**Alternative defs.:**

- Constraints + punishment
- Deontic Logic (DL)
  - Normative propositions
  - [Des/Pres]criptive obligations
- Game Theory (GT):
  - Violation games,…
  - Decision Theoretic GT vs DL
Norms in MAS

Norms are key for social processes:
- Simplify agent's decision-making process (templates)
- **Balance** between:
  - Individual freedom (autonomy)
  - The goal of the agent society
Norm Categories (Boella and van der Torre):

- **Regulative norms:**
  - Obligations (O),
  - Prohibitions and
  - Permissions.

- **Constitutive norms:**
  - Create institutional facts (e.g. property or marriage) and
  - Modify normative system itself.
Norms in MAS

Norm Categories (Boella and van der Torre):

- **Regulative norms:**
  - Obligations (O),
  - Prohibitions and
  - Permissions.

- **Constitutive norms:**
  - Create institutional facts (e.g. property or marriage) and
  - Modify normative system itself.

Noms and BDI agents:

- **Norm-based behaviour: BOID**
  - Meneguzzi and Luck,
  - Dignum et al. ...
- **Normative MAS:** MAS + normative system

- Agents can **decide** whether to follow explicitly represented norms,
- Normative systems specify how agents can **modify norms**.
- **Sociological theories** from sociology, economics, legal science,..
Design questions
Design questions

Example: Answers for a Traffic scenario?

- How do we represent norms?
- Who dictates norms?
- How agents decide norm fulfillment?
- Who/how detects if agents comply with norms?
- Should a norm change?
How do we represent norms?
  • Are norms implicit, hierarchical, local, imprecise,..?
  • Are there norm exceptions, contradictions?
Who dictates norms?
  • Are norms related to organisations?
  • Who spreads them?
How agents decide norm fulfillment?
  • What norms apply to an agent?
  • Do agents internalise norms?
Who/how detects if agents comply with norms?
  • If other agents do not comply with a norm, should an agent bother?
  • Are there infringement consequences?
Should a norm change?
  • Do we need additional incentives? (rewards, environment,..)
Exercise: Answers for a regulated scenario?

- How do we represent norms?
- Who dictates norms?
- How agents decide norm fulfillment?
- Who/how detects if agents comply with norms?
- Should a norm change?
**Calvinball** is a game invented by Calvin in which one makes the rules up as one goes along. Rules cannot be used twice. No Calvinball game is like another. The game may involve wickets, mallets, volleyballs, and additional equipment as well as masks.

There is only one permanent rule in Calvinball: One can't play it in the same way twice. For example, in one game of Calvinball, the goal was to capture one's opponent's flag, whereas in a different game of Calvinball, the goal was to score points by hitting badminton shuttles against trees using a croquet mallet. An apparent rule in Calvinball is that one must wear a black mask.
Applications

- Contracts (e-commerce)
- International trade
- Social norms in 3D VW (e.g. NPC in Second Life)
- Human Computer Interaction
- “What if” scenarios for policy makers
- Organizations
- What else?
How do norms come to exist within a society?

- Off-line design
- Emergence
- Other ways:
  - Norm agreement
  - Norm Learning
  - On-line generation
1. Introduction to Normative MAS and norm synthesis approaches.
   - Off-line norm synthesis.
   - Norm emergence
   - Other
2. On-line automatic norm synthesis.
3. Demo and hands-on activity
Off-line norm design


Which norm would you define?

- Each robot is required to move constantly. The direction of motion is fixed as follows. On even rows each robot must move left, while in odd rows it must move right. It is required to move up when it is in the right-most column. Finally, it is required to move down when it is on either the leftmost column of even rows or on the second rightmost column of odd rows. The movement is therefore in a 'snake-like' Structure, and defines a Hamiltonian cycle on the grid.
• Shoham and Tennenholtz (1996): Traffic law for preventing robot collisions in 2D a grid.
  
  Each robot is required to move constantly. The direction of motion is fixed as follows. On even rows each robot must move left, while in odd rows it must move right. It is required to move up when it is in the right-most column. Finally, it is required to move down when it is on either the leftmost column of even rows or on the second rightmost column of odd rows. The movement is therefore in a 'snake-like' Structure, and defines a Hamiltonian cycle on the grid.

- Determines uniquely the next movement of agents
- Provides paths to any destination cell
- Does not require perceptual capabilities of the robots
- Is effective but not very efficient (fixed directions)
Off-line norm design

Abstract Model of Environment & Agents

- $E$ a finite set of environment discrete states: $E = \{e, e', \ldots\}$.
- Agent actions transform the environment:

$$Ac = \{\alpha, \alpha', \ldots\}$$

$$r^E : e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_{u-1}} e_u$$

- A **constraint** is then a pair $<E', \alpha>$ where
  - $E' \subseteq E$ set of environment states, $\alpha \in Ac$ an action
    - “IF environment is in some state $e \in E'$, THEN action $\alpha$ is forbidden”

- A **social law** is a set of constraints
  - Useful social law: Disallows (& ensures) access to undesirable (& goal) states in the state space.
  - An **agent is legal** respect a social law if it never attempts to perform a forbidden action in this law.
Offline norm design

- Formal, exhaustive, **NP-complete**
- Norms are hardwired in agents
- Designer has more control
- But:
  - Some characteristics may not be known at design time
  - Agent goals may be constantly changing: requires agent reprogramming
  - Complex systems are hard to predict (and to design norms)
1. Introduction to Normative MAS and norm synthesis approaches.
   - Off-line norm synthesis.
   - Norm emergence
   - Other

2. On-line automatic norm synthesis.

3. Demo and hands-on activity
Norm emergence

- Norm Emergence:
  - Agents reach **global agreement** on social conventions by using only **locally available information**:

- Global: all agents adopt norms
- Local: each agent decides to adopt one based solely on its own experiences
Norm emergence

The tee shirt game: Let’s play it!

- All agents have a blue and a red T-shirt
- They should end up wearing the same colour
  - Colour adoption as a strategy or convention to adopt

- Agents:
  - Decide what to dress
  - based on their memory about encountered agents (initially, can be random)

- Form agent populations, select a monitor and play in rounds:
  - Monitor agent detects convergence (same colour)
  - Each round:
    » Form pairs of agents: each one sees the t-shirt colour of the other agent.
    » Agents can change colour (dress again) after each round.
Norm emergence issues

• Search space:
  – Agents choose a solution from a space of alternative solutions (known at design time).
  – Repeated two-player games.
  – Agents open to new ideas can periodically forget everything.

• Convergence:
  – Initial conditions.
  – Efficiency measure: time to norm convergence.

• Norm changes:
  – Strategy changing cost.
Norm emergence

Research on:
- Norm adoption & internalisation (Conte et al.)
- Topology of relationships (Luck et al., Sen et al.,)
- ...
- Norm life-cycle:
  - Savarimutu and Cranefield
Norm emergence
Norm life-cycle: Savarimutu and Cranefield

Phases of norm construction:

1. Creation
   - Proposed norm

2. Spreading
   - Internalized norm

3. Enforcement
   - Punishments

4. Emergence

Mechanisms used by simulation models:
- Off-line design
- Leadership
- Entrepreneurship
- Cognition
- Learning
- Culture and Evolution
- Imitation
- Machine Learning
- Sanction
- Reputation
- Emotion
1. Introduction to Normative MAS and norm synthesis approaches.
   - Off-line norm synthesis.
   - Norm emergence
   - Other: agreement, learning, on-line

2. On-line automatic norm synthesis.

3. Demo and hands-on activity
Norm design

Norm origin

• How do norms come to exist within a society?
  – Off-line design
  – Emergence
  – Other ways:
    • Norm agreement
    • Norm Learning
    • On-line generation
Empowered members use a (meta-level) argumentation protocol to modify norms at run-time.

Democratic

Agents enriched with agreement capabilities
Norm learning:

- Genetic Algorithms: Punishment learning (Bou et al.)
- Case Based Reasoning: Norm parameter adaptation (Campos et al.)
Genetic Algorithms
Learning effective norm punishments

Agent population

Configurations

I₁
I₂
Iₖ

Fitness(A, I₁, G)
Fitness(A, I₂, G)
Fitness(A, Iₖ, G)

Configuration evaluation
Configuration adaptation
Crossover and mutation

New configurations

I₁
...
Iₖ
Learn fines

Learns norm punishments for each population

Genetic Algorithms

Learning effective norm punishments for the traffic domain
Norms:
- $\text{norm}_{FR}$: a peer cannot send data to $>\text{max}_{FR}$ simult.
- $\text{norm}_{BW}$: a peer cannot use $>\text{max}_{BW}$ bandwidth.
• How do norms come to exist within a society?
  – Off-line design
  – Emergence
  – Other ways:
    • Norm agreement
    • Norm Learning
    • On-line generation
1. Introduction to Normative MAS

2. On-line automatic norm synthesis.

3. Demo and hands-on activity.
• How to synthesise a Normative System (NS) that avoids undesirable states (i.e., conflicts) in a MAS?
  – If limited previous knowledge and/or dynamic MAS, then: on-line empirical approach.

• Is the resulting NS good enough?
  – Avoids conflicts?
  – Is it compact? (avoids overregulation and is easy to reason about)
On-line norm generation

**Architecture**

**Norm Synthesis Machine**

**MAS events**

**Norms (NS)**

**MAS Simulator**
On-line norm generation

Architecture

Norm Synthesis Machine

MAS events

Norms (NS)

MAS Simulator
On-line norm generation

Architecture

**Norm Synthesis Machine**
- Conflict detection,
- Norm synthesis

**MAS Simulator**

MAS events

**Norms (NS)**
On-line norm generation

Architecture

Norm Synthesis Machine

MAS events

Norms (NS)

MAS Simulator
On-line norm generation

Architecture

Norm Synthesis Machine

MAS events

Norms (NS)

MAS Simulator
On-line norm generation

Architectures

Norm Synthesis Machine

Conflict & norm compliance detection, Norm refinement

MAS events

Norms (NS)

MAS Simulator
On-line norm generation

Architecture

Norm Synthesis Machine

MAS events

Norms (NS)

MAS Simulator
On-line norm generation

Architecture

Norm Synthesis Machine

MAS events

Norms (NS)

MAS Simulator

Goal: conflict avoidance

Dynamicity

Division of concerns

Top-down

Bottom-up
On-line generation

Architecture

Norm Synthesis Machine

MAS events

Norms (NS)

MAS Simulator

Top-down
Bottom-up

Goal: conflict avoidance
Dynamicity
Division of concerns
Similar to human socities
On-line norm generation

Morales, López-Sánchez, Rodríguez-Aguilar, Wooldridge, Vasconcelos

— Regulatory agents propose norms to avoid conflicts in agent interactions
  • Non intrusive, preserves agent autonomy
  • Requires **conflict** detection
  • Does not search the complete state space

— Norm evaluation based on
  • Agent responses (infringements and compliances)
  • Consequences (conflicts $\approx$ system goals)
  • Normative system compactness
**Simulated** discretized traffic intersection:

- **Agents**: cars.
- **Conflicts**: car collisions.
- **MAS goal**: collision avoidance.
On-line norm generation

Norm synthesis Strategy

Norm Synthesis Machine

Strategy: 1.- Conflict detection, 2.- Norm synthesis,

MAS events

Norms (NS)

MAS Simulator
1. **Conflict detection** by MAS observation.
2. For each detected conflict $\rightarrow$ **Synthesis** of a new norm
   - to avoid that conflict in the future.
Norm creation

**Conflict Description**

Conflicting agents: \{ag_1, ag_2\}

Agent actions (t-1 → t):

\{ag_1: Go, ag_2: Go\}
**Norm creation**

**Conflict Description**

*View at time $t-1$*

- **$ag_1$**
- **$ag_2$**

*View at time $t$*

- **$ag_1$**: $Go$
- **$ag_2$**: $Go$

Conflicting agents: $\{ag_1, ag_2\}$

Agent actions $(t-1 \rightarrow t)$:

$\{ag_1: Go, ag_2: Go\}$

New norm $Prh(Go)$
In the traffic simulator, cars perceive three cells in front:

Norms are...

- **IF** … **THEN**… rules: \(<\phi, \Theta(ac)\>
  - Whenever the local perception of an agent satisfies the precondition of a norm (\(\phi\)), then the norm **applies to the agent**: the deontic operator specifies the modality of its action \(ac\)
  - \(\alpha\) : unary **predicates**: \(\alpha \in \{\text{left, front, right}\}\)
  - \(\tau_i\) : terms \(\tau_i \in \{\text{car-to-right, car-same-dir, car-to-left, car-opp-dir, nothing, wall, anything}\}\)
  - Ex.: **IF** left(car-to-right) & front(nothing) & right(nothing) **THEN** prohibition(go)
1. Conflict detection by MAS observation.
2. For each detected conflict $\rightarrow$ Synthesis of a new norm.
   - to avoid the conflict in the future.

But... are synthesised norms good enough for avoiding conflicts?
1. **Conflict detection** by MAS observation.
2. For each detected **conflict** → **Synthesis** of a new norm.
   - to avoid the conflict in the future.

   But... are synthesised norms good enough for avoiding conflicts?

3. Evaluate norms in terms of:
   - **Effectiveness**: Do norms avoid conflicts when agents comply with them?
     - If complied & no conflicts → **Effectiveness** ↑↑ (ex. Left hand side priority)
     - If complied & conflicts → **Effectiveness** ↓↓ (ex. Never give way)
   - **Necessity**: Do conflicts arise when agents infringe norms?
     - If infringed & no conflicts → **Necessity** ↓↓ (ex. Stop if no car in view)
     - If infringed & conflicts → **Necessity** ↑↑ (ex. Left hand side priority)
On-line norm generation

Architecture

Norm Synthesis Machine

3.- Conflict & norm compliance detection
4.- Norm refinement

MAS events

Norms (NS)

MAS Simulator
Normative Network (Data Structure):
- Nodes: explored norms.
- Edges: norm generalisation relationships.
Normative Network (Data Structure):
- Nodes: explored norms.
- Edges: norm generalisation relationships

n₁: Give way to police cars
n₂: Give way to fire-trucks
n₃: Give way to ambulances
n₄: Give way to emergency vehicles
Normative Network (Data Structure):
- Nodes: explored norms.
- Edges: norm generalisation relationships

A Normative Network represents a Normative System $\Omega$ as its active norms.

Ex: $\Omega = \{n_4\}$
1. **Conflict detection** by MAS observation.

2. For each detected conflict $\rightarrow$ **Synthesis** of new norms.
   - New norms are aimed to avoid the conflict in the future.
   - But... are synthesised norms good enough for avoiding conflicts?

3. Evaluate norms in terms of:
   - **Effectiveness**: Do norms avoid conflicts when agents comply with them?
   - **Necessity**: Do conflicts arise when agents infringe norms?

4. Refine norms:
   - Deactivate/Specialise norms that do not perform well
   - Generalise well performing norms (if enough evidence)
On-line norm generation

Architecture

Norm Synthesis Machine

Normative Network

Control Unit
Operators Strategy

MAS events

Norms (NS)

MAS Simulator

write
read
**create**: Synthesises a norm and adds it to the Normative Network

\[
\begin{align*}
\Omega_0 &= \{n_1\} \\
\Omega_1 &= \{n_1, n_2\}
\end{align*}
\]
**deactivate**: Deactivates a norm in the Normative Network

\[ \Omega_0 = \{ n_1, n_2 \} \]

\[ \Omega_1 = \{ n_1 \} \]
**generalise**: Generalises a set of norms into a parent norm

\[ \Omega_0 = \{n_1, n_2\} \]

\[ \Omega_1 = \{n_3\} \]
specialises: Undoing a norm generalisation

\[ \Omega_0 = \{n_3\} \quad \Rightarrow \quad \Omega_1 = \{n_1\} \]
Norm Synthesis Strategy:
1. Conflict detection
2. Norm creation
3. Norm evaluation
4. Norm Refinement:
   • Deactivate/specialise norms that do not perform well
   • Generalise well performing norms (if enough evidence)
Norm generalisation

Term taxonomy

- emergency
- car
- ambulance
- fire-brigade
- police-car
- private-car

Norm generalisation

\[ \Omega_0 = \{n_1, n_2, n_3\} \]

- \( n_1 \): Give way to ambulances
- \( n_2 \): Give way to fire brigade
- \( n_3 \): Give way to police cars
• Term taxonomy

Ambulance, fire-brigade, police-car, private-car

• Norm generalisation

Normative System \( \Omega_0 = \{n_1, n_2, n_3\} \)

- \( n_1 \): Give way to ambulances
- \( n_2 \): Give way to fire brigade
- \( n_3 \): Give way to police cars

Normative System \( \Omega_1 = \{n_4\} \)

- \( n_4 \): Give way to emergency vehicles
Norm generalisation

- Term taxonomy
  - Emergency
  - Car
  - Ambulance
  - Fire-brigade
  - Police-car
  - Private-car

- Norm generalisation
  
  Conservative approach
  
  Employs full evidence to generalise norms.
Norm generalisation

**Term taxonomy**
- **emergency**
- **car**
- **ambulance**
- **fire-brigade**
- **police-car**
- **private-car**

**Optimistic norm generalisation**

\[
\mathcal{NN}_0 = \{n_1, n_2\}
\]

- \(n_1\): Give way to **ambulances**
- \(n_2\): Give way to **fire brigade**
Norm generalisation

• Term taxonomy

- emergency
- car
- ambulance
- fire-brigade
- police-car
- private-car

• Optimistic norm generalisation (partial evidence)

Most specific generalisation between two terms

\[ \Omega_0 = \{n_1, n_2\} \]

\[ \Omega_1 = \{n_4\} \]

\( n_1 \): Give way to ambulances
\( n_2 \): Give way to fire brigade
\( n_4 \): Give way to emergency vehicles

Optimistic approach
Norm generalisation

Generalisation modes: shallow

• Term taxonomy

\[ \Omega_0 = \{n_1, n_2\} \]

- \( n_1 \): Give way to **ambulances**
- \( n_2 \): Give way to **fire brigade**

\[ \text{Normative System} \]

**Shallow** Optimistic norm generalisation
**Norm generalisation**

**Generalisation modes: shallow**

- **Term taxonomy**
  
  - Optimistic norm generalisation
  
  - **Shallow** Optimistic norm generalisation

  - **Directly** generalises two active norms (in $\Omega$).

  - $n_1$: Give way to **ambulances**
  - $n_2$: Give way to **fire brigade**
  - $n_4$: Give way to **emergency** vehicles
Norm generalisation

- **Term taxonomy**
  - Normative system $\Omega_0 = \{n_4, n_5\}$

- **Deep Optimistic norm generalisation**
  - $n_1$: Give way to ambulances
  - $n_2$: Give way to fire brigade
  - $n_4$: Give way to emergency vehicles
  - $n_5$: Give way to private cars
Norm generalisation

• Term taxonomy

- emergency
- car
- ambulance
- fire-brigade
- police-car
- private-car

• Deep Optimistic norm generalisation

\( \Omega_0 = \{ n_4, n_5 \} \)

\( \Omega_1 = \{ n_4, n_6 \} \)

\( n_1 \): Give way to ambulances
\( n_2 \): Give way to fire brigade
\( n_3 \): Give way to police cars
\( n_4 \): Give way to emergency vehicles
\( n_5 \): Give way to private cars
\( n_6 \): Give way to cars
Norm generalisation

Shallow vs. deep generalisation modes

**Shallow generalisation**

- More coarse

**Deep generalisation**

- More fine-grained
In this simple scenario we may synthesise many candidate norms:

1. Give way to left.
2. Give way to right.
4. Stop always.
5. Never stop.
6. ...

66 candidate norms $\rightarrow 2^{66} \approx 10^{20}$ candidate Normative Systems.

What combination of candidate norms (NS) achieves MAS goals?
1. A **typical execution** of the norm synthesis process.
   - Successful synthesis of NS that avoid collisions.

2. A **robustness analysis** w.r.t. non-compliant behaviour (norm infringements).
   - Synthesis of NS even for high norm violation rates.

3. **Analysis of the search space**
   - Different strategies explore different NS.
Tick 13: first collisions arise and the strategy synthesises first norms.

Tick 35: norm generalisations.

Tick 3349: Cardinality of the normative system reduced to 5 norms. Collisions are avoided.

Tick 13349: Simulation stops because of convergence.
Empirical evaluation

Robustness Analysis

1. **Low violation rates (up to 40%)** converges for 100% of the simulation runs.

2. **High violation rates (40%-60%)** converges between 80% and 98% of the simulation runs.

3. **Very high violation rates (70%-90%)** converges for 20% of the simulation runs despite a 70% violation rate. Norms cannot be synthesised beyond 80% violation rate.
D-SIMON focuses on an search space area with more compact NS. (D-SIMON: requires more computational effort than S-SIMON)
We have presented **SIMON**, a novel strategy for the on-line synthesis of conflict-free and compact normative systems that:

- Avoids **conflicts**.
- Avoids **over regulation**.
- **Eases** the reasoning of agents.

- Applicable to other domains.
On-line norm generation

Case study 2: Virtual Communities

- Agents model human users interacting within virtual communities
- On-line synthesis of norms to avoid conflicts (i.e. user complaints)

Ex. Norms:

IF user(1) & section(2) & contentType(porn)
THEN prh(upload(content))

MAS = Simulated virtual community
On-line norm generation

Case study 2: Virtual Communities Simulator
NORM SYNTHESIS BECOMES A PARTICIPATORY MECHANISM:

Users choose community norms by means of their complaints.
1. Introduction to Normative MAS

2. On-line automatic norm synthesis.

3. Demo and hands-on activity.
NormLab hands-on Tutorial

Javier Morales (IIIa-UB), Maite López-Sánchez(UB), Juan A. Rodríguez-Aguilar (IIIa-CISC), Michael Wooldridge (UO), Wamberto Vasconcelos (UA)
1. NormLab (Introduction)

*NormLab* is a framework to support research on norm synthesis for Multi-Agent Systems.

*NormLab* allows to:

- **Perform MAS simulations.** It incorporates two different MAS simulators: a traffic simulator, and an on-line community simulator.

- **Perform on-line norm synthesis on MAS simulations.** *NormLab* incorporates different *state-of-the-art* on-line norm synthesis strategies that can be tested on MAS simulations.

- **Develop and test custom norm synthesis strategies.** NormLab allows to develop custom on-line norm synthesis strategies to be tested on the MAS simulations.
NormLab hands-on tutorial Outline

An introduction to NormLab
1. (Introduction to NormLab)
2. NormLab architecture.
4. Traffic simulator.

Configuration of the working environment
5. NormLab download and installation.

NormLab execution:
6-8. Execution examples.
2. NormLab architecture

NormLab

Norm Synthesis Machine

Norm synthesis strategies: IRON, SIMON, ...

MAS events

Norms (NS)

MAS Simulators

Traffic junction simulator

On-line community simulator

Norm synthesis settings

Domain-dependent functions

Simulator settings
3. Norm Synthesis Machine

**Norm Synthesis Machine**

- Norm synthesis strategy: SIMON

**NormLab**

- MAS Simulator
  - Traffic junction simulator
  - MAS events
  - Norms (NS)
  - Norm synthesis settings
  - Domain-dependent functions
  - Simulator settings
3. Norm Synthesis Machine

Norm Synthesis Machine

Norm synthesis strategy: SIMON

MAS events

Norms (NS)

 MAS Simulator

Traffic junction simulator

1. Agents behave
3. Norm Synthesis Machine

NormLab

Norm Synthesis Machine

Norm synthesis strategy: SIMON

MAS events

Norms (NS)

2.- Agents are observed

MAS Simulator

Traffic junction simulator
3. Norm Synthesis Machine

Norms (NS)

Norm Synthesis Machine

Norm synthesis strategy: SIMON

MAS events

NormLab

MAS Simulator

Traffic junction simulator

3.- New norms required?
3. Norm Synthesis Machine

Norm Synthesis Machine

Norm synthesis strategy: SIMON

MAS Simulator

Traffic junction simulator

NormLab

Norms (NS)

MAS events

4.- Provide Normative System
4. Traffic simulator

- Based on Repast Simphony 2.2
- Agents are cars, and conflicts are collisions among cars.
- The goal is to synthesise normative systems that avoid collisions between cars.
An introduction to NormLab
  1. (Introduction to NormLab)
  2. NormLab architecture.
  4. Traffic simulator.

Configuration of the working environment
  5. NormLab download and installation.

NormLab execution:
  6-8. Execution examples.
5. NormLab download

NormLab is multi-platform. You can use it either in Windows, MacOS or Linux.

Requirements

- **Java JDK 1.6** or later
  http://www.java.com
- **Eclipse IDE** (just for Linux users)
  http://www.eclipse.org/downloads
- **Repast Simphony 2.2**
  http://repast.sourceforge.net

Downloads

To use NormLab you need to download:

- **NormSynthesisMachine**: http://normsynthesis.github.io/NormSynthesisMachine
  Implements an API that allows to perform norm synthesis for MAS.
- **NormLabSimulators**: http://normsynthesis.github.io/NormLabSimulators
  Code of two MAS simulators: traffic and on-line community.

Download both projects in a ZIP or TAR.GZ file.
5. NormLab installation

Preparing the working environment

1. Unzip **NormSynthesisMachine** and **NormLabSimulators** projects to your HOME folder.
   - For instance... «/Users/Javi/NormLab»

2. Both projects will be unzipped as **NormSynthesis-«project_name»-«numbers»**. For instance...
   - **NormSynthesis-NormLabSimulators-34d43o**
   - **NormSynthesis-NormSynthesisMachine-1847fje**

3. Rename both projects, removing the «NormSynthesis» part and the numbers. After renaming them they should look like this:
   - **NormLabSimulators**
   - **NormSynthesisMachine**
5. NormLab installation

Preparing the working environment

1. Open the **Repast Symphony IDE** (in Linux, open **Eclipse IDE** with Repast installed on it).
2. Select Java view in Eclipse
3. Import both projects **NormSynthesisMachine** and **NormLabSimulators** in Eclipse.
   1. **File>New>Java Project.**
   2. Uncheck «Use default location» and click on «Browse». 
**NormLabSimulators** project is structured as follows:

- **src/traffic**: The code of the traffic simulator.
- **(src/onlineComm)**: The code of the on-line community simulator.
- **launchers**: The launchers that allow to run the two simulators.
- **repa-settings/TrafficJunction.rs**: Basic Repast settings for the traffic junction simulator.
- **(repa-settings/OnlineCommunities.rs)**: Basic Repast settings for the on-line community simulator.
NormLab hands-on tutorial Outline

An introduction to NormLab
  1. (Introduction to NormLab)
  2. NormLab architecture.
  4. Traffic simulator.

Configuration of the working environment
  5. NormLab download and installation.

NormLab execution:
  6-8. Execution examples.
NormLab execution:

6-8. Execution examples:
   7. Example strategy 2: Returns a fixed set of 1 norm.
   8. Example strategy 3: Returns a fixed set of 3 norms.

9-14. Guided development of different norm synthesis strategies:
   12. Studying example 5: A strategy with norm generation + evaluation.
NormLab execution:

6-8. Execution examples
   6. Example strategy 1: NormLab execution: Returns an empty set of norms.
   7. Example strategy 2: Returns a fixed set of 1 norm.
   8. Example strategy 3: Returns a fixed set of 3 norms.

9-14. Guided development of different norm synthesis strategies
   12. Studying example 5: A strategy with norm generation + evaluation.
**TrafficJunction norm synthesis example 1**

We are going to execute the **TrafficJunction** simulator with the simplest norm synthesis strategy:

→ *Everytime the strategy is executed, return an empty normative system.*

**Consequences:** No norms are given to the agents → collisions are never avoided.

Note: This execution assumes that file parameters.xml (in directory repast-settings/TrafficJunction.rs within NormLabSimulators project) has parameter «NormSynthesisExample» with field «defaultValue» set to «1»
6. NormLab Execution: Example 1

**TrafficJunction norm synthesis example 1**

1. In Eclipse, in NormLabSimulators project, go to directory `launchers/`
2. Do right click on the file `TrafficJunctionSimulator.launch`.
3. Click on «Run As» > «TrafficJunctionSimulator».
4. Click on button  to initialise the simulator.
6. NormLab Execution: Example 1

TrafficJunction norm synthesis example 1

1. In Eclipse, in NormLabSimulators project, go to directory launchers/
2. Do right click on the file TrafficJunctionSimulator.launch.
3. Click on «Run As» > «TrafficJunctionSimulator».
4. Click on button to initialise the simulator.
5. Click on button to start the simulator. Cars will appear as coloured balls. Collisions will appear as red stars. Cars will start to drive and they will collide.
6. You can pause the simulation with button and stop it with button
NormLab **execution**: 

6-8. Execution examples 

6. **Example** strategy 1: NormLab execution: Returns an **empty** set of norms. 

7. **Example** strategy 2: Using norms: Returns a fixed set of **1 norm**. 

8. **Example** strategy 3: Returns a fixed set of **3 norms**.

9-14. Guided development of different norm synthesis strategies 

9. **Development** of example strategy 1: **Empty** set of norms. 

10. **Development** of example strategy 2: Fixed set of **1 norm**. 

11. **Studying** example 4: A strategy with norm **generation**. 

12. **Studying** example 5: A strategy with norm **generation + evaluation**. 

13. **Studying** SIMON: A strategy with norm **generation + evaluation + refinement**.
In the traffic simulator, cars’ perceptions correspond to the three cells in front of them:

**Norms are...**

- **IF ... THEN...** rules.
- Norm precondition: Set of *predicates* with one *term* each.
  - Three predicates (**left, front, right**).
  - Terms \{<, ^, >, v, -, w, *\} represent: cars with {<, ^, >, v} headings; nothing (-), wall (w); and anything (*).
- Norm postcondition: A *modality*.

**Graphical representation**

If **left(>)** & **front(-)** & **right(-)** THEN **prohibition(go)**
We will execute the *TrafficJunction* simulator with a norm synthesis strategy that returns a normative system with only one *left-side-priority* norm:

**Norm 1**

\[
\text{IF left(>)} \& \text{ front(*) } \& \text{ right(*) } \text{ THEN prohibition(go)}
\]

It avoids some (but not all) collisions.
1. In Eclipse, in NormLabSimulators project, go to directory `repast-settings/TrafficJunction.rs`.
2. Open file `parameters.xml` by doing right click > *Open with* > *Text Editor*. This file defines the `NormLab` parameters.
3. Search for the parameter «NormSynthesisExample».
4. Set the field «defaultValue» to «2». This will indicate NormLab to launch example 2, which uses a norm synthesis strategy that always returns a normative system with the left-side-priority norm.
5. Save the file.
7. Using norms: Example 2

*TrafficJunction* norm synthesis example 2

6. Do right click on the file *TrafficJunctionSimulator.launch*.
7. Click on «Run As» > «TrafficJunctionSimulator».
8. Run the simulation with button 🎥
9. Update the norm synthesis inspector. Observe how now the normative system contains norm N1, and cars occasionally stop to conform to it.

Green circle: Norm 1 applies and car $c_1$ stops ($c_3$ has priority)

Red circle: Norm 1 applies but car $c_5$ does NOT stop

Non regulated collision (between $c_1$ - $c_2$)

Regulated collision (between $c_4$ - $c_5$) $c_6$ complies with N1 (stops)
NormLab **execution:**

6-8. Execution examples

6. **Example** strategy 1: NormLab execution: Returns an empty set of norms.
7. **Example** strategy 2: Using norms: Returns a fixed set of 1 norm.
8. **Example** strategy 3: Removing collisions: Returns a fixed set of 3 norms.

9-14. Guided development of different norm synthesis strategies

9. **Development** of example strategy 1: Empty set of norms.
10. **Development** of example strategy 2: Fixed set of 1 norm.
11. **Studying** example 4: A strategy with norm generation.
12. **Studying** example 5: A strategy with norm generation + evaluation.
13. **Studying** SIMON: A strategy with norm generation + evaluation + refinement.
TrafficJunction norm synthesis example 3

Let’s define a norm synthesis strategy that avoids all possible collisions by always returning this Normative System:

N1: IF left(*) & front(^) & right(*) THEN prohibition(go)
N2: IF left(>) & front(-) & right(*) THEN prohibition(go)
N3: IF left(<) & front(<) & right(*) THEN prohibition(go)

Set NormSynthesisExample default=«3» in parameters.xml (in NormLabSimulators project, repast-settings/TrafficJunction.rs)

Tick i Tick i+1
NormLab execution:

6-8. Execution examples
   6. **Example** strategy 1: NormLab execution: Returns an empty set of norms.
   7. **Example** strategy 2: Adding norms: Returns a fixed set of 1 norm.
   8. **Example** strategy 3: Removing collisions: Returns a fixed set of 3 norms.

9-14. Guided development of different norm synthesis strategies
   9. **Development** of example strategy 1: Empty set of norms.
   10. Executing your own strategy
   11. Development of example strategy 2: Adding norms to your strategy (1 norm)
   12. **Example** 4: A strategy with norm generation.
   13. **Example** 5: A strategy with norm generation + evaluation.
   14. **SIMON**: A complete strategy with norm generation + evaluation + refinement.
9. Developing your own strategy

How are all these examples implemented? We will now develop our own norm synthesis strategy as the one from example 1, which returns an empty normative system.

To do so, we first parameterise NormLab to use a custom norm synthesis strategy:

1. In Eclipse (NormLabSimulators project), go to directory repast-settings/TrafficJunction.rs
2. Open file parameters.xml by doing right click > Open with > Text Editor. This file defines the NormLab parameters.
3. Search for the parameter «NormSynthesisExample» and set the field defaultValue=«0». This will indicate NormLab that we do not want to load a pre-designed example.
4. Search for the parameter «NormSynthesisStrategy» and set the field defaultValue=«0». This will indicate NormLab that we will provide a custom norm synthesis strategy.
5. Save the file

```
<parameter
    name="NormSynthesisExample" isReadOnly="false" displayName="NSM: Norm synthesis example" type="int"
    converter="repast.symphony.parameter.StringConverterFactory$IntConverter"
    defaultValue="0" />
<parameter
    name="NormSynthesisStrategy" isReadOnly="false"
    displayName="NSM: Norm synthesis strategy (CUSTOM/IRON/SIMON/XSIMON)" type="int"
    converter="repast.symphony.parameter.StringConverterFactory$IntConverter"
    defaultValue="0" />
```
9. Developing your own strategy

Now, **create your own norm synthesis strategy** *MyFirstStrategy.java:*

- In NormLabSimulators project, go to package `es.csic.iiia.normlab.traffic.custom` in src/traffic.
- There, right-click **New > Class** to create a new Java class *MyFirstStrategy.java* that implements NormSynthesisStrategy interface by:

  1.- Naming it MyFirstStrategy
9. Developing your own strategy

Now, create your own norm synthesis strategy *MyFirstStrategy.java*:

- In NormLabSimulators project, go to package `es.csic.iiia.normlab.traffic.custom` in `src/traffic`.
- There, right-click **New > Class** to create a new Java class *MyFirstStrategy.java* that implements `NormSynthesisStrategy` interface by:

1. Naming it `MyFirstStrategy`
9. Developing your own strategy

Now, **create your own norm synthesis strategy** `MyFirstStrategy.java`:

- In NormLabSimulators project, go to package `es.csic.iiia.normlab.traffic.custom` in `src/traffic`.
- There, right-click **New > Class** to create a new Java class `MyFirstStrategy.java` that implements `NormSynthesisStrategy` interface by:

  1.- Naming it `MyFirstStrategy`
  2.- Adding interface `es.csic.iiia.nsm.strategy.NormSynthesisStrategy`
  3.- Checking the constructor creation
9. Developing your own strategy

Now, create your own norm synthesis strategy *MyFirstStrategy.java*:

- In NormLabSimulators project, go to package `es.csic.iiia.normlab.traffic.custom` in `src/traffic`.
- There, right-click **New > Class** to create a new Java class *MyFirstStrategy.java* that implements `NormSynthesisStrategy` interface by:
  1. Naming it *MyFirstStrategy*
  3. Checking the constructor creation
  4. Creating inherited abstract method `execute()` (check “Inherited abstract methods”)

```java
package es.csic.iiia.normlab.traffic.custom;
import es.csic.iiia.nsm.norm.NormativeSystem;
import es.csic.iiia.nsm.strategy.NormSynthesisStrategy;

public class MyFirstStrategy implements NormSynthesisStrategy {
    public MyFirstStrategy() {
        // TODO Auto-generated constructor stub
    }

    @Override
    public NormativeSystem execute() {
        // TODO Auto-generated method stub
        return null;
    }
}
```
9. Developing your own strategy

And implement the **norm synthesis strategy** class:

1. In the class, add a Normative Network attribute:
   ```java
   private NormativeNetwork normativeNetwork;
   ```

The Norm Synthesis Machine contains the **Normative Network** which includes the Normative System:
- Normative Network: contains all synthesised norms.
- Normative System: set of (active) norms given to the agents.
9. Developing your own strategy

And **implement** the **norm synthesis strategy** class:

1. In the class, add a Normative Network attribute:
   ```java
   private NormativeNetwork normativeNetwork;
   ```
2. In the constructor, add the parameter `es.csic.iii.a.nsm.NormSynthesisMachine nsm` and use it to initialize (to empty) the Normative Network attribute:
   ```java
   this.normativeNetwork = nsm.getNormativeNetwork();
   ```

The Norm Synthesis Machine contains the **Normative Network** which includes the Normative System:
- Normative Network: contains all synthesised norms.
- Normative System: set of (active) norms given to the agents.
9. Developing your own strategy

And **implement** the **norm synthesis strategy** class:

1. In the class, add a Normative Network attribute:
   ```java
   private NormativeNetwork normativeNetwork;
   ```
2. In the constructor, add the parameter `es.csic.iitia.nsm.NormSynthesisMachine nsm` and use it to initialize (to empty) the Normative Network attribute:
   ```java
   this.normativeNetwork = nsm.getNormativeNetwork();
   ```

The Norm Synthesis Machine contains the **Normative Network** which includes the Normative System:
- Normative Network: contains all synthesised norms.
- Normative System: set of (active) norms given to the agents.

3. **Strategy execution**: return the empty **Normative System** in method `execute()`:
   ```java
   return this.normativeNetwork.getNormativeSystem();
   ```
Congratulations! You have created your first norm synthesis strategy, which returns an empty normative system. Your code should now look like this:

```java
package es.csic.i3ia.normlab.traffic.custom;

import es.csic.i3ia.nsm.net.norm.NormativeNetwork;
import es.csic.i3ia.nsm.norm.NormativeSystem;

/**
 *
 */
public class MyFirstStrategy implements es.csic.i3ia.nsm.strategy.NormSynthesisStrategy {

    /* The normative network, a data structure to keep track of norms */
    private NormativeNetwork normativeNetwork;

    /**
     * Constructor of the strategy
     *
     * @param nsm
     */
    public MyFirstStrategy(es.csic.i3ia.nsm.NormSynthesisMachine nsm) {
        /* Get normative network */
        this.normativeNetwork = nsm.getNormativeNetwork();
    }

    /**
     * Executes your strategy
     *
     */
    @Override
    public NormativeSystem execute() {
        return normativeNetwork.getNormativeSystem();
    }
}
Tutorial outline

NormLab execution:

6-8. Execution examples
   6. Example strategy 1: NormLab execution: Returns an empty set of norms.
   7. Example strategy 2: Adding norms: Returns a fixed set of 1 norm.
   8. Example strategy 3: Removing collisions: Returns a fixed set of 3 norms.

9-14. Guided development of different norm synthesis strategies
   10. Invoking your strategy
   11. Development of example strategy 2: Adding norms to your strategy (1 norm)
   14. SIMON: A complete strategy with norm generation + evaluation + refinement.
But, how does NormLab invoke our new norm synthesis strategy? The Traffic Simulator includes (in package es.csic.iiia.normlab.traffic.agent) an agent DefaultTrafficNormSynthesisAgent whose:

A. Constructor creates the Norm Synthesis Machine and configures it to use our strategy
B. step() method invokes our strategy at every simulation tick.

```java
public DefaultTrafficNormSynthesisAgent(List<TrafficCamera> cameras,
  PredicatesDomains predDomains, DomainFunctions dmFunctions,
  NormSynthesisSettings nsSettings, long randomSeed) {

public void step() throws IncorrectSetupException {
  this.addedNorms.clear();
  this.removedNorms.clear();

  // Execute strategy and obtain new normative system */
  NormativeSystem newNormativeSystem = nsm.executeStrategy();
```
10. Invoking your strategy (A)

Specifically, the constructor **(A) DefaultTrafficNormSynthesisAgent()** is in charge of:

1. Creating the norm synthesis machine.
2. Adding a set of sensors to the norm synthesis machine in order to perceive the scenario.
3. Setting the norm synthesis strategy.

```java
public DefaultTrafficNormSynthesisAgent(List<TrafficCamera> cameras, 
PredicatesDomains predDomains, DomainFunctions dmFunctions, 
NormSynthesisSettings nsSettings, long randomSeed) {
    this.nsSettings = nsSettings;
    this.normativeSystem = new NormativeSystem();
    this.addedNorms = new ArrayList<Norm>();
    this.removedNorms = new ArrayList<Norm>();

    /* 1. Create norm synthesis machine */
    this.nsm = new NormSynthesisMachine(nsSettings, predDomains, 
        dmFunctions, !RunEnvironment.getInstance().isBatch(), randomSeed);

    /* 2. Add sensors to the monitor of the norm synthesis machine */
    for(TrafficCamera camera : cameras) {
        this.nsm.addSensor(camera);
    }

    /* 3. Set the norm synthesis strategy */
    this.setNormSynthesisStrategy();
}
```
10. Invoking your strategy (A.1)

The invocation to the constructor of the NormSynthesisMachine (A.1) requires:

i. **NormSynthesisSettings**: The settings for the norm synthesis machine.

ii. **PredicatesDomains**: Agents’ language: predicates and terms describing the scenario from the agents’ local point of view.

iii. **DomainFunctions**: Some domain-dependent functions that the Norm Synthesis Machine requires to synthesise norms (e.g., conflict detection, norm applicability).

```java
public DefaultTrafficNormSynthesisAgent(List<TrafficCamera> cameras,
    PredicatesDomains predDomains, DomainFunctions dmFunctions,
    NormSynthesisSettings nsSettings, long randomSeed) {
    this.nsSettings = nsSettings;
    this.normativeSystem = new NormativeSystem();
    this.addedNorms = new ArrayList<Norm>();
    this.removedNorms = new ArrayList<Norm>();

    // 1. Create norm synthesis machine
    this.nsm = new NormSynthesisMachine(nsSettings, predDomains,
        dmFunctions, !RunEnvironment.getInstance().isBatch(), randomSeed);

    // 2. Add sensors to the monitor of the norm synthesis machine */
    for (TrafficCamera camera : cameras) {
        this.nsm.addSensor(camera);
    }

    // 3. Set the norm synthesis strategy */
    this.setNormSynthesisStrategy();
}
```
10. Invoking your strategy (A.1.i)

**NormSynthesisSettings (A.1.i)**: An interface to be implemented (located in package `es.csic.iiia.nsm.config` in `NormSynthesisMachine` project)

- **getNormSynthesisStrategy()**: Returns the norm synthesis strategy to use.
- **getSystemGoals()**: A list of system goals. In traffic, the only goal is “to avoid collisions”.
- **isNormGenerationReactiveToConflicts()**: True if NSM tries to add a new norm upon the detection of each non-regulated conflict. False if it creates the norm but does not add it to the Normative System immediately.
- **getNormsDefaultUtility()**: Norms’ default utility (0.5 by default).
- **getNormEvaluationLearningRate()**: The α rate in IRON and SIMON to evaluate norms (0.1 recom.).
- **getNormsPerformanceRangesSize()**: The size of the window to compute norms’ performance ranges.
- **getNormGeneralisationMode()**: SIMON’s norm generalisation mode (Shallow/Deep).
- **public int getNormGeneralisationStep()**: SIMON’s norm generalisation step: number of norm predicates that can be simultaneously generalised.
- **getGeneralisationBoundary(Dimension dim, Goal goal)**: Minimum value of effectiveness/necessity that a norm’s performance must reach to be generalised. It corresponds to the threshold $\alpha_{gen}$ in [1].
- **getSpecialisationBoundary(Dimension dim, Goal goal)**: Value of Effectiveness/necessity under which a norm can be specialised. It corresponds to the threshold $\alpha_{spec}$ described in [1].
- **getSpecialisationBoundaryEpsilon(Dimension dim, Goal goal)**: LION’s epsilon to create, together with the specialisation boundaries, a norm deactivation band.
- **getNumTicksOfStabilityForConvergence()**: Number of simulation ticks without conflicts nor changes in the normative system to converge.

An implementation of these settings for the traffic simulator is located in (NormLabSimulators project, `src/traffic`) package `es.csic.iiia.normlab.traffic.normsynthesis`, in class `TrafficNormSynthesisSettings`

10. Invoking your strategy (A.1.ii)

**PredicatesDomains (A.1.ii)**: Contains the predicates and terms that the agents employ to describe the MAS from their local point of view. Located in package `es.csic.iiia.nsm.agent.language` (NormSynthesisMachine project, src/).

The traffic simulator creates predicates and their domains in class `TrafficSimulator` (NormLabSimulators project, src/traffic) from package `es.csic.iiia.normlab.traffic`, method `createPredicatesDomains()`.

- Three different predicates \((l, f, r)\) that represent the left, front and right positions in front of a car.
- Seven different terms \(\{<, ^, >, v, -, *, w\}\) representing: cars with different headings \(\{<, ^, >, v\}\), nothing (-), anything (*), and wall (w).

Car perception:
\[ l(>) & f(-) & r(-) \]

Reference car
10. Invoking your strategy (A.1.ii)

**PredicatesDomains (A.1.ii)**: class TrafficSimulator, method createPredicatesDomains():

```java
private void createPredicatesDomains() {
    /* Predicate "left" domain */
    TaxonomyOfTerms leftPredTaxonomy = new TaxonomyOfTerms("l");
    leftPredTaxonomy.addTerm("=");
    leftPredTaxonomy.addTerm("<");
    leftPredTaxonomy.addTerm(">");
    leftPredTaxonomy.addTerm("-");
    leftPredTaxonomy.addRelationship("<", "=");
    leftPredTaxonomy.addRelationship(">", "=");
    leftPredTaxonomy.addRelationship("-", "=");

    /* Predicate "front" domain*/
    TaxonomyOfTerms frontPredTaxonomy = new TaxonomyOfTerms("f", leftPredTaxonomy);
    frontPredTaxonomy.addTerm("^");
    frontPredTaxonomy.addRelationship("^", "=");

    /* Predicate "right" domain*/
    TaxonomyOfTerms rightPredTaxonomy = new TaxonomyOfTerms("r", leftPredTaxonomy);
    rightPredTaxonomy.addTerm("w");
    rightPredTaxonomy.addRelationship("w", "=");

    this.predDomains = new PredicatesDomains();
    this.predDomains.addPredicateDomain("l", leftPredTaxonomy);
    this.predDomains.addPredicateDomain("f", frontPredTaxonomy);
    this.predDomains.addPredicateDomain("r", rightPredTaxonomy);
}
```
10. Invoking your strategy (A.1.iii)

**DomainFunctions (A.1.iii)**: An interface to be implemented. Located in package `es.csic.iiia.nsm.config` (NormSynthesisMachine project, src/).

- **isConsistent(SetOfPredicatesWithTerms agentContext)**: Returns true if a set of predicates with terms is consistent with the domain scenario. E.g.: `(left(>),front(-),right(-))` is consistent (possible) but `(left(>),front(<),right(-))` is not consistent, since two cars can not drive in opposite directions in the same lane.

- **agentContextFunction(long agentId, View view)**: Returns the local perception of a given agent (i.e., its context) from the observation (view) of the state of the simulated scenario.

- **agentActionFunction(long agentId, ViewTransition viewTransition)**: Returns a list of actions performed by an agent in the transition from a state $s_t$ to a state $s_{t-1}$.

- **getConflicts(Goal goal, ViewTransition viewTransition)**: Receives a transition between two states, a system goal (e.g., to avoid collisions) and returns the conflicts that have arisen in that transition with respect to the system goal (e.g., returns the collisions).

- **hasConflict(View view, long agentId, Goal goal)**: Returns true if a given agent is in conflict in a given system state (i.e., View).

An implementation of the domain functions for the traffic simulator is located on (NormLabSimulators project, src/traffic) `es.csic.iiia.normlab.traffic.normsynthesis`, `TrafficDomainFunctions` class.
10. Invoking your strategy (recap)

The Traffic Simulator includes `DefaultTrafficNormSynthesisAgent` agent whose:

**A. Constructor**
1. Creates the Norm Synthesis Machine (NSM).
2. Adds a set of sensors to SNM to perceive the scenario.
3. Sets the norm synthesis strategy in the NSM.

**B. step()** method invokes our strategy at every simulation **tick**.

```java
public DefaultTrafficNormSynthesisAgent(...)
{
  ... 
  /* 1. Create norm synthesis machine */
  this.nsm = new NormSynthesisMachine(nsSettings, predDomains,
  dmFunctions, !RunEnvironment.getInstance().isBatch(), randomSeed);
  /* 2. Add sensors to the monitor of the norm synthesis machine */
  for(TrafficCamera camera : cameras)
  { 
    this.nsm.addSensor(camera);
  }
  /* 3. Set the norm synthesis strategy */
  this.setNormSynthesisStrategy();

  public void step() throws IncorrectSetupException {
  this.addedNorms.clear();
  this.removedNorms.clear();
  /* Execute strategy and obtain new normative system */
  NormativeSystem newNormativeSystem = nsm.executeStrategy();
```
10. Invoking your strategy (A.3, B)

The Traffic Simulator includes `DefaultTrafficNormSynthesisAgent` agent whose:

**A. Constructor**
1. Creates the Norm Synthesis Machine (NSM).
2. Adds a set of sensors to SNM to perceive the scenario.
3. Sets the norm synthesis strategy in the NSM: Method `SetNormSynthesisStrategy()` invokes method `createCustomNormSynthesisStrategy()` (located in the same class `DefaultTrafficNormSynthesisAgent`):
   - **Implement** this method by creating and returning your norm synthesis strategy:

```java
/**
 * Sets a custom norm synthesis strategy
 */
protected NormSynthesisStrategy createCustomNormSynthesisStrategy() {
    return new MyFirstStrategy(nsm);
}
```

**B. step()** method invokes our strategy at every simulation **tick**.
- Execute the simulation as you did for examples 1, 2 and 3 (NormLabSimulators project, launchers/: TrafficJunctionSimulator.launch > Run As ...)

**Congratulations! You are using your own strategy!**
NormLab **execution:**

6-8. Execution examples

6. **Example** strategy 1: NormLab execution: Returns an **empty** set of norms.
7. **Example** strategy 2: Adding norms: Returns a fixed set of **1 norm**.
8. **Example** strategy 3: Removing collisions: Returns a fixed set of **3 norms**.

9-14. Guided development of different norm synthesis strategies

9. **Development** of example strategy 1: **Empty** set of norms.
10. Executing your own strategy

11. **Development** of example strategy 2: **Adding norms** to your strategy (1 norm)
12. **Example 4**: A strategy with norm **generation**.
13. **Example 5**: A strategy with norm **generation** + **evaluation**.
14. **SIMON**: A complete strategy with norm **generation** + **evaluation** + **refinement**.
11. Adding norms to your strategy

Let’s now **add some norms**. We will add the left-side-priority norm from **example 2**.

1. **Create** a new norm synthesis strategy `MySecondStrategy.java` by **Copying** (cut&paste+rename) your first strategy `MyFirstStrategy.java`
   
   Your code should look like this:

   ```java
   /**
    * My second strategy
    */
   public class MySecondStrategy implements es.csic.iia.nsm.strategy.NormSynthesisStrategy {

   /* The normative network, a data structure to keep track of norms */
   private NormativeNetwork normativeNetwork;

   /**
    * Constructor of the strategy
    *
    * @param nsm the norm synthesis machine
    */
   public MySecondStrategy(es.csic.iia.nsm.NormSynthesisMachine nsm) {
       this.normativeNetwork = nsm.getNormativeNetwork();
   }

   @Override
   public NormativeSystem execute() {
       return normativeNetwork.getNormativeSystem();
   }
   ```
11. Adding norms to your strategy

2. Implement a method `createNormativeSystem()` in `MySecondStrategy.java` to create norms with:
   - Preconditions: a set of predicate-term pairs and
   - Postconditions: a modality (prohibition/obligation) over an action
   i. Create a new norm precondition: \( \text{IF } \text{l}(>) \& \text{f}(* \& \text{r}(*)) \)

```java
private void createNormativeSystem() {
    /* Create norm preconditions */
    SetOfPredicatesWithTerms n1Precondition = new SetOfPredicatesWithTerms();
    n1Precondition.add("l", ">");
    n1Precondition.add("f", "*");
    n1Precondition.add("r", "*");

    /* Create norms */
    Norm n1 = new Norm(n1Precondition,
                        NormModality.Prohibition,
                        CarAction.Go);

    /* Add the norms to the normative network and activate them */
    this.normativeNetwork.add(n1);
    normativeNetwork.setState(n1, NetworkNodeState.ACTIVE);
}
```
11. Adding norms to your strategy

2. Implement a method `createNormativeSystem()` in `MySecondStrategy.java` to create norms with:
   - Preconditions: a set of predicate-term pairs and
   - Postconditions: a modality (prohibition/obligation) over an action
   i. Create a new norm precondition: `IF l(>) & f(*) & r(*)`
   ii. Create a new norm `n1` with this precondition and as postcondition: `THEN Prohibition(Go)`

```java
private void createNormativeSystem() {
    /* Create norm preconditions */
    SetOfPredicatesWithTerms n1Precondition = new SetOfPredicatesWithTerms();
    n1Precondition.add("l", ">");
    n1Precondition.add("f", "*");
    n1Precondition.add("r", "*");

    /* Create norms */
    Norm n1 = new Norm(n1Precondition,
                       NormModality.Prohibition, CarAction.Go);

    /* Add the norms to the normative network and activate them */
    this.normativeNetwork.add(n1);
    normativeNetwork.setState(n1, NetworkNodeState.ACTIVE);
}
```
11. Adding norms to your strategy

2. Implement a method `createNormativeSystem()` in `MySecondStrategy.java` to create norms with:
   - Preconditions: a set of predicate-term pairs and
   - Postconditions: a modality (prohibition/obligation) over an action
   i. Create a new norm precondition: `IF l(>) & f(*) & r(*)`
   ii. Create a new norm `n1` with this precondition and as postcondition: `THEN Prohibition(Go)`
   iii. Add norm `n1` to the Normative Network and activate it so it becomes part of the Normative System

```java
private void createNormativeSystem() {
    /* Create norm preconditions */
    SetOfPredicatesWithTerms n1Precondition = new SetOfPredicatesWithTerms();
    n1Precondition.add("l", ">");
    n1Precondition.add("f", "*");
    n1Precondition.add("r", "*");

    /* Create norms */
    Norm n1 = new Norm(n1Precondition,
                       NormModality.Prohibition, CarAction.Go);

    /* Add the norms to the normative network and activate them */
    this.normativeNetwork.add(n1);
    normativeNetwork.setState(n1, NetworkNodeState.ACTIVE);
}
```
11. Adding norms to your strategy


```java
public class MySecondStrategy implements es.csic.iiia.nsm.strategy.NormSynthesisStrategy {
    private NormativeNetwork normativeNetwork;

    public MySecondStrategy(es.csic.iiia.nsm.NormSynthesisMachine nsm) {
        this.normativeNetwork = nsm.getNormativeNetwork();
        this.createNormativeSystem(); // Create a default normative system
    }

    @Override
    public NormativeSystem execute() {
        return normativeNetwork.getNormativeSystem();
    }

    private void createNormativeSystem() {
        SetOfPredicatesWithTerms n1Precondition = new SetOfPredicatesWithTerms();
        n1Precondition.add("l", ">");
        n1Precondition.add("f", "=");
        n1Precondition.add("r", "=");

        Norm n1 = new Norm(n1Precondition, NormModality.Prohibition, CarAction.Go);
        this.normativeNetwork.add(n1);
        normativeNetwork.setState(n1, NetworkNodeState.ACTIVE);
    }
}
```

At each tick, the strategy will return the norms that are active in the normative network (i.e., the normative system).
11. Adding norms to your strategy

4. Change method `createCustomNormSynthesisStrategy()` from `DefaultTrafficNormSynthesisAgent` (in package `es.csic.iiia.normlab.traffic.agent`, NormLabSimulators project, src/traffic) to use your new strategy.

   ```java
   /**
    * Sets a custom norm synthesis strategy
    */
   protected NormSynthesisStrategy createCustomNormSynthesisStrategy() {
     return new MySecondStrategy(nsm);
   }
   ``

   • Recall that the traffic norm synthesis agent in the traffic simulator creates the norm synthesis machine and executes the strategy at every simulation tick.

5. Execute the Traffic Simulator (NormLabSimulators project, launchers/: `TrafficJunctionSimulator.launch` > Run As ...) to observe that this second strategy works as example 2.
   • The normative system contains a single norm N1.
NormLab execution:

6-8. Execution examples
   6. Example strategy 1: NormLab execution: Returns an empty set of norms.
   7. Example strategy 2: Adding norms: Returns a fixed set of 1 norm.
   8. Example strategy 3: Removing collisions: Returns a fixed set of 3 norms.

9-14. Guided development of different norm synthesis strategies
   10. Executing your own strategy
   11. Development of example strategy 2: Adding norms to your strategy (1 norm)
   14. SIMON: A complete strategy with norm generation + evaluation + refinement.
12. Your strategy with automatic norm generation

How can we automatically generate norms on-line?

Example 4 (*TrafficNSExample4_NSStrategy* in package *es.csic.iiia.normlab.traffic.examples.ex4*, NormLabSimulators project) uses operators (methods defined in *TrafficNSExample4_NSOperators*) to create, add and activate norms the Normative Network:

- **Activate (norm):** sets the state of norm to «Active»
- **Add (norm):** adds norm into the Normative Network and activates it.
- **Create (Conflict, Goal):**
  - Applies Case-Based Reasoning (CBR) to create a norm aimed at avoiding future conflicts.
  - If the norm does not exist in the Normative Network, then it adds (and activates) it. Otherwise, if the norm is not active (nor represented )in the NN , then it activates it.
12. Your strategy with automatic norm generation

*TrafficNSExample4_NSStrategy* uses operators to synthesize norms:

**Everytime the strategy is executed, it:**

1. Generates norms
2. Returns the Normative System.
12. Your strategy with automatic norm generation

*TrafficNSExample4_NSStrategy* uses operators to synthesize norms:

Everytime the strategy is executed, it:
1. Generates norms
   1. Perceives the scenario
2. Returns the Normative System.

```java
/**
 * Executes IRON's strategy
 * @return the normative system resulting from the norm synthesis cycle
 */
public NormativeSystem execute() {
    this.normAdditions.clear();
    this.normDeactivations.clear();
    this.createdNorms.clear();
    this.activatedNorms.clear();

    /* Norm generation */
    this.normGeneration();

    /* Return the current normative system */
    return normativeNetwork.getNormativeSystem();
}

/**
 * Executes the norm generation phase
 */
private void normGeneration() {
    /* Obtain monitor perceptions */
    obtainPerceptions(viewTransitions);

    /* Conflict detection */
    conflicts = conflictDetection(viewTransitions);

    /* Norm generation */
    for (Goal goal : conflicts.keySet()) {
        for (Conflict conflict : conflicts.get(goal)) {
            operators.create(conflict, goal);
        }
    }
}
```

**ViewTransition:** description of partial scenario transition from time \(t-1\) to time \(t\) (current tick)
12. Your strategy with automatic norm generation

TrafficNSExample4_NSStrategy uses operators to synthesize norms:

**Everytime the strategy is executed, it:**
1. Generates norms
   1. Perceives the scenario
   2. Detects non regulated conflicts
2. Returns the Normative System.

Conflict detection through `getConflicts()` domain function.
Each conflict has a `ViewTransition` with a conflict at tick t and an involved (responsible) agent.
TrafficNSExample4_NSStrategy uses operators to synthesize norms:

1. Perceives the scenario
2. Detects non regulated conflicts
3. Creates norms for each conflict.

Everytime the strategy is executed, it:
1. Generates norms
2. Returns the Normative System.
12. Your strategy with automatic norm generation

Execute the strategy:

1. Set NormSynthesisExample `defaultValue=«4»` in `parameters.xml` (in NormLabSimulators project, `repaст-settings/TrafficJunction.rs`) and save the file.

2. Execute the simulator
   - NormLabSimulators project, launchers/: TrafficJunctionSimulator.launch > Run As ...

3. Observe how, as long as cars collide, it generates norms to avoid these collisions
   - Norms are never evaluated (select a norm and click on button `Show` performance ranges).

Example:

- 16 norms generated so far (4943 ticks)
- Current tick: norms 7, 8, 9, and 11 apply.
NormLab **execution:**

6-8. Execution examples

6. **Example** strategy 1: NormLab execution: Returns an **empty** set of norms.
7. **Example** strategy 2: Adding norms: Returns a fixed set of **1 norm**.
8. **Example** strategy 3: Removing collisions: Returns a fixed set of **3 norms**.

9-14. Guided development of different norm synthesis strategies

9. **Development** of example strategy 1: **Empty** set of norms.
10. Executing your own strategy
11. **Development** of example strategy 2: **Adding norms** to your strategy (1 norm)
12. **Example 4**: A strategy with automatic norm **generation**.

13. **Example 5**: A strategy with norm **generation** + **evaluation**.
14. **SIMON**: A complete strategy with norm **generation** + **evaluation** + **refinement**.
13. Automatic norm generation + evaluation

Are generated norms good enough?

Let’s see example 5: TrafficNSExample5_NSStrategy (in NormLabSimulators project, src/traffic es.csic.iia.normlab.traffic.examples.ex5 package):

Whenever the strategy is executed:
- It generates norms (as example 4)
- It evaluates norms: how?

```java
public NormativeSystem execute() {
    this.normAdditions.clear();
    this.normDeactivations.clear();
    this.createdNorms.clear();
    this.activatedNorms.clear();
    this.normGeneration();
    this.normEvaluation();

    /* Return the current normative system */
    return normativeNetwork.getNormativeSystem();
}
```
13. Automatic norm generation + evaluation

Norm Evaluation (*TrafficNSExample5_NSStrategy*):

```java
private void normEvaluation() {
    /* Compute norm applicability */
    this.normApplicability = this.normApplicability(viewTransitions);

    /* Detect norm applicability and compliance */
    this.normCompliance(this.normApplicability);

    /* Update utilities and performances */
    this.updateUtilitiesAndPerformances(this.normCompliance);
}
```

1. **Retrieve** the norms that applied to each agent in the simulation at time t-1:

```java
protected Map<ViewTransition, NormsApplicableInView> normApplicability(
    List<ViewTransition> vTransitions) {

    /* Clear norm applicability from previous tick */
    this.normApplicability.clear();

    /* Get applicable norms of each viewTransition (of each sensor) */
    for (ViewTransition vTrans : vTransitions) {
        NormsApplicableInView normApplicability;
        normApplicability = this.normReasoner.getNormsApplicable(vTrans);
        this.normApplicability.put(vTrans, normApplicability);
    }

    return this.normApplicability;
}
```

For each viewTransition, normReasoner computes the norms that apply to each agent by using DomainFunctions.
13. Automatic norm generation + evaluation

Norm Evaluation (*TrafficNSExample5_NSStrategy*):

```java
private void normEvaluation() {
    /* Compute norm applicability */
    this.normApplicability = this.normApplicability(viewTransitions);

    /* Detect norm applicability and compliance */
    this.normCompliance(this.normApplicability);

    /* Update utilities and performances */
    this.updateUtilitiesAndPerformances(this.normCompliance);
}
```

2. **Norm compliance**: Did agents **complied** with their applicable norms? Did that lead to conflicts?

```java
protected void normCompliance(Map<ViewTransition,
                                NormsApplicableInView> normApplicability) {
    /* Check norm compliance in the view in terms of each system goal */
    for(Goal goal : this.nsmSettings.getSystemGoals()) {
        /* Clear norm compliance of previous tick */
        this.normCompliance.get(goal).clear();

        /* Evaluate norm compliance and conflicts in each
         * view transition with respect to each system goal */
        for(ViewTransition vTrans : normApplicability.keySet()) {
            NormsApplicableInView vNormAppl = normApplicability.get(vTrans);

            /* If there is no applicable norm in the view, continue */
            if(vNormAppl.isEmpty()) {
                continue;
            }

            NormComplianceOutcomes nCompliance = this.normReasoner.
                                                 checkNormComplianceAndOutcomes(vNormAppl, goal);

            this.normCompliance.get(goal).put(vTrans, nCompliance);
        }
    }
}
```
13. Automatic norm generation + evaluation

Norm Evaluation (*TrafficNSExample5_NSStrategy*):

```csharp
private void normEvaluation() {
    /* Compute norm applicability */
    this.normApplicability = this.normApplicability(viewTransitions);

    /* Detect norm applicability and compliance */
    this.normCompliance(this.normApplicability);

    /* Update utilities and performances */
    this.updateUtilitiesAndPerformances(this.normCompliance);
}
```

3. Update norms’ utilities based on norm compliance

```csharp
protected void updateUtilitiesAndPerformances(
    Map<Goal, Map<ViewTransition, NormComplianceOutcomes>> normCompliance) {
    for (Goal goal : this.nsmSettings.getSystemGoals()) {
        for (ViewTransition vTrans : normCompliance.get(goal).keySet()) {
            for (Dimension dim : this.nsm.getNormEvaluationDimensions()) {
                this.utilityFunction.evaluate(dim, goal,
                    normCompliance.get(goal).get(vTrans), normativeNetwork);
            }
        }
    }
}
```

`evaluate(...) method in *TrafficNSExample5_NSUtilityFunction*
(in NormLabSimulators project, src/traffic.es.csic.iii.a.normlab.traffic.examples.ex5 package)
13. Automatic norm generation + evaluation

Norm Evaluation (*TrafficNSExample5_NSStrategy*):

```java
private void normEvaluation() {
/* Compute norm applicability */
this.normApplicability = this.normApplicability(viewTransitions);

/* Detect norm applicability and compliance */
this.normCompliance(this.normApplicability);

/* Update utilities and performances */
this.updateUtilitiesAndPerformances(this.normCompliance);
}
```

3. **Update norms’ utilities** based on norm compliance

```java
protected void updateUtilitiesAndPerformances(
Map<Goal, Map<ViewTransition, NormComplianceOutcomes>> normCompliance) {
    for(Goal goal : this.nsmSettings.getSystemGoals()) {
        for(ViewTransition vTrans : normCompliance.get(goal).keySet()) {
            for(Dimension dim : this.nsm.getNormEvaluationDimensions()) {
                this.utilityFunction.evaluate(dim, goal,
                                               normCompliance.get(goal).get(vTrans), normativeNetwork);
            }
        }
    }
}
```

Evaluates each norm in terms of system goals: Is it useful to **avoid conflicts**? (e.g. traffic: avoids car collisions?). Two dimensions:

- **Effectiveness**: when complied, is it effective to avoid conflicts?
  - If complied + no conflicts → **Effective**
  - If complied + conflicts → **Ineffective**
- **Necessity**: when infringed, did some conflicts actually arise?
  - If infringed + no conflicts → **Unnecessary**
  - If infringed + conflicts → **Necessary**
13. Automatic norm generation + evaluation

Execute the strategy:
1. Set NormSynthesisExample `defaultValue=«5»` in `parameters.xml` (in NormLabSimulators project, `repast-settings/TrafficJunction.rs`) and save the file.
2. Execute the simulator
   • NormLabSimulators project, launchers/: TrafficJunctionSimulator.launch > Run As ...
3. Observe how it generates norms and evaluates them.
   • Effectiveness and necessity of each norm change along time (select a norm and click on button `Show` performance ranges).
Tutorial outline

NormLab **execution**:

6-8. Execution examples
   6. **Example** strategy 1: NormLab execution: Returns an **empty** set of norms.
   7. **Example** strategy 2: Adding norms: Returns a fixed set of **1 norm**.
   8. **Example** strategy 3: Removing collisions: Returns a fixed set of **3 norms**.

9-14. Guided development of different norm synthesis strategies
   9. **Development** of example strategy 1: **Empty** set of norms.
   10. Executing your own strategy
   11. **Development** of example strategy 2: **Adding norms** to your strategy (1 norm)
   12. **Example 4**: A strategy with automatic norm **generation**.
   13. **Example 5**: A strategy with norm **generation + evaluation**.
   14. **SIMON**: A complete strategy with norm **generation + evaluation + refinement**.
14. SIMON: generation + evaluation + refinement

SIMON is a **complete norm synthesis strategy** that uses norm evaluation to refine norms

**SIMONStrategy** (in NormSynthesisMachine project, src `es.csic.iiia.nsm.strategy.simon` package):

Whenever the strategy is **executed**:
- It generates norms
- It evaluates norms
- It **refines them**: how?

```java
public NormativeSystem execute() {
    this.nsMetrics.resetNonRegulatedConflicts();
    this.visitedNorms.clear();

    /* Norm generation */
    List<Norm> normsActivated = this.normGenerator.step(viewTransitions, conflicts);

    /* Norm evaluation */
    this.normEvaluator.step(viewTransitions, normApplicability, normCompliance, normGroupCompliance);

    /* Norm refinement */
    this.normRefiner.step(normApplicability, normsActivated);

    /* Manage lists that control new additions to the normative network, * 
    * normative system, as well as norms that have been removed */
    this.manageNormControlsList();

    /* Return the current normative system */
    return normativeSystem.getNormativeSystem();
}
```

**step(...)** method in **SIMONNormRefiner** (in NormSynthesisMachine project, src `es.csic.iiia.nsm.strategy.simon` package)
Norm refinement:

1. Norms are **generalised** if their (effectiveness and necessity) > threshold.
14. SIMON: generation + evaluation + refinement

Norm refinement:

1. Norms are **generalised** if their (effectiveness and necessity) $\geq$ gen. **threshold**.

**n_1**: Give way to ambulances
**n_2**: Give way to fire brigade
**n_3**: Give way to police cars
14. SIMON: generation + evaluation + refinement

Norm refinement:

1. Norms are **generalised** if their (effectiveness **and** necessity) \( \geq \) gen. threshold.

\[ \text{New Normative system } NS_1 = \{n_4\} \]

- \( n_1 \): Give way to **ambulances**
- \( n_2 \): Give way to **fire brigade**
- \( n_3 \): Give way to **police cars**
- \( n_4 \): Give way to **emergency vehicles**

Increases Compactness
14. SIMON: generation + evaluation + refinement

Norm refinement:

1. Norms are **generalised**
2. Norms are **specialised**
   if their (effectiveness or necessity) < esp. **threshold**

```java
public void step(Map<ViewTransition, NormsApplicableInView> normApplicability,
                 List<Norm> normsActivatedDuringGeneration) {

    List<Norm> processed = new ArrayList<Norm>();
    List<Norm> visited = new ArrayList<Norm>();

    /* Compute norms that must be revised */
    List<Norm> normsToRevise = this.checkNormsToRevise(normApplicability);

    /* Classify norms */
    this.normClassifications = this.normClassifier.step(normsToRevise);

    /* Refine norms based on norm classifications */
    for(Norm norm : normClassifications.keySet()) {
        if(processed.contains(norm)) {
            continue;
        }
        List<NormAttribute> attributes = normClassifications.get(norm);

        boolean isIneffective = attributes.contains(NormAttribute.INEFFECTIVE);
        boolean isUnnecessary = attributes.contains(NormAttribute.UNNECESSARY);
        boolean isGeneralisable = attributes.contains(NormAttribute.GENERALISABLE);

        /* If the norm is whether ineffective or unnecessary, then deactivate */
        /* it (specialise it into its children) */
        if(isIneffective || isUnnecessary) {
            visited.clear();
            specialiseDown(norm, NetworkNodeState.DISCARDED, visited);
        }

        /* If the norm has enough utility to be generalised, */
        /* then try to generalise it */
        else if(isGeneralisable) {
            generaliseUp(norm, genMode, genStep);
        }

        /* Update complexities metrics */
        this.nsMetrics.incNumNodesVisited();
    }
}
```
14. SIMON: generation + evaluation + refinement

Norm refinement:

1. Norms are **generalised**
2. Norms are **specialised**
   if their (effectiveness or necessity) < esp. **threshold**

```java
class Norm {
    public void step(Map<ViewTransition, NormsApplicableInView> normApplicability, List<Norm> normsActivatedDuringGeneration) {
        List<Norm> processed = new ArrayList<Norm>();
        List<Norm> visited = new ArrayList<Norm>();

        /* Compute norms that must be revised */
        List<Norm> normsToRevise = this.checkNormsToRevise(normApplicability);

        /* Classify norms */
        }
    }
}
```

**Normative System**

```
NS_1 = \{n_4\}
```

- **n_1**: Give way to **ambulances**
- **n_2**: Give way to **fire brigade**
- **n_3**: Give way to **police cars**
- **n_4**: Give way to **emergency** vehicles

```
/* If the norm is whether ineffective or unnecessary, then deactivate
   * it (specialise it into its children) */
if(isIneffective || isUnnecessary) {
    visited.clear();
    specialiseDown(norm, NetworkNodeState.DISCARDED, visited);
}
```

```
/* If the norm has enough utility to be generalised,
   * then try to generalise it */
else if(isGeneralisable) {
    generaliseUp(norm, genMode, genStep);
}
```

```
/* Update complexities metrics */
this.nsMetrics.incNumNodesVisited();
```
14. SIMON: generation + evaluation + refinement

Norm refinement:

1. Norms are **generalised**
2. Norms are **specialised**
   if their (effectiveness or necessity) < esp. threshold

**Removes Under-performing norms**

**Normative System**

$NS_1 = \{n_4\}$

$NN_1 = \{n_4, n_1, n_2, n_3\}$

**New Normative System**

$NS_2 = \{n_1, n_2\}$

$NN_2 = \{n_1, n_2, n_3, n_4\}$

$n_1$: Give way to **ambulances**

$n_2$: Give way to **fire brigade**

$n_3$: Give way to **police cars**

$n_4$: Give way to **emergency vehicles**
14. SIMON. A complete norm synthesis strategy

Execute SIMON strategy:
1. In **parameters.xml** (in NormLabSimulators project, repast-settings/TrafficJunction.rs) set:
   - **NormSynthesisExample** defaultValue=«0»
   - **NormSynthesisStrategy** defaultValue=«2» (2 stands for SIMON strategy)
   - **NormGeneralisationMode** defaultValue=«1» (Deep norm generalisation)
   - **NormGeneralisationStep** defaultValue=«1» (generalises 1 predicate at a time)
   - Save the file.
2. Execute the simulator
   - NormLabSimulators project, launchers/:
     TrafficJunctionSimulator.launch > Run As ...
3. Observe how it generates norms, evaluates, and refines them.
   - Compact Normative System.

Normative System: 6 norms
Normative Network: 55 norms
Generalisations: 98 relationships
- Ex: n41 generalises n38, n10, n7 and n39
Coverage at tick 9428
Challenge: Can you improve the synthesis strategy?

Participate!